Table of Contents

Introduction..........................................................................................................1
  New to Splunk..................................................................................................1
  Welcome to the Machine Learning Toolkit.....................................................3
  About the Machine Learning Toolkit............................................................13
  Share data in the Machine Learning Toolkit.................................................15
  Install the Machine Learning Toolkit..........................................................18
  Upgrade the Machine Learning Toolkit.......................................................24

Getting started...................................................................................................28
  Using the Splunk Machine Learning Toolkit...............................................28
  Showcase examples.......................................................................................30
  Understanding the fit and apply commands...............................................37
  Splunk Machine Learning Toolkit workflow..............................................46

Custom search commands..................................................................................49
  Search commands for machine learning.....................................................49
  Search macros in the Machine Learning Toolkit.........................................55
  Configure permissions for ML-SPL commands..........................................60
  Configure the fit and apply commands......................................................62
  Using the score command............................................................................69

Algorithms..........................................................................................................126
  Algorithms....................................................................................................126
  Import a machine learning algorithm from Splunkbase.............................182

Smart Assistants...............................................................................................184
  Smart Forecasting Assistant......................................................................184

Experiments.......................................................................................................195
  Experiments overview.................................................................................195
  Predict Numeric Fields Experiment workflow...........................................200
  Predict Categorical Fields Experiment workflow......................................207
  Detect Numeric Outliers Experiment workflow.......................................213
  Detect Categorical Outliers Experiment workflow...................................219
  Forecast Time Series Experiment workflow..............................................223
  Cluster Numeric Events Experiment workflow.........................................231
# Table of Contents

## Models
- Models ......................................................... 237
  - Manage models ........................................... 239

## Custom visualizations
- Custom visualizations .................................... 241

## Classic Assistants
- Classic Assistants overview ............................ 249
  - Predict Numeric Fields Classic Assistant ........ 250
  - Predict Categorical Fields Classic Assistant .... 255
  - Detect Numeric Outliers Classic Assistant ....... 258
  - Detect Categorical Outliers Classic Assistant ... 262
  - Forecast Time Series Classic Assistant .......... 264
  - Cluster Numeric Events Classic Assistant ....... 270

## Preprocessing
- Preparing your data for machine learning ........... 274
- Preprocessing machine data using Assistants ...... 295

## Troubleshooting
- Frequently Asked Questions ......................... 301

## Release notes
- What's new .................................................. 311
  - Known issues ............................................. 320
  - Fixed issues ............................................. 321
  - Third-party software .................................. 321
  - Dataset credits .......................................... 346
Introduction

New to Splunk

If this is the first time you have used Splunk, then read on...this topic introduces the most important Splunk concepts you need to understand when installing and using Splunk apps.

Splunk and Splunk apps work together

The key points to come away with are:

- All Splunk apps run on the Splunk platform.
- Understanding how Splunk works will greatly help you understand how Splunk apps work.
- Installing and configuring the app is only part of the experience - you might need to prepare Splunk before installing your app.
- Careful planning helps achieve a successful app deployment experience.

Splunk basics

Splunk is a software platform that accepts data from many different sources, such as files or network streams. Splunk stores a unique copy of this data in what's called an index. Once the data is there, you can connect to Splunk with your web browser and run searches across that data. You can even make reports or graphs on the data, from within the browser.

You can extend Splunk's capability by installing apps. Splunk apps come with searches, reports, and graphs about specific products that are common to most IT departments. These searches, reports, and graphs reduce the amount of time it takes to get real value from installing and running the Splunk platform.

Before you can really understand how Splunk apps work, get to know how Splunk works. Fortunately, we've got you covered in that respect.

If you're new to Splunk, then the best place to learn more about it is in the Search Tutorial. It helps you learn what Splunk is and what it does, as well as what you need to run it and get step-by-step walk-throughs on how to set it up, get data into it, search with it, and saving and sharing reports and dashboards on it.
**Configuration**

Much of Splunk's extensibility is in how configurable it is. You must configure Splunk before it can collect data and extract knowledge. All Splunk apps use configuration files to determine how to collect, transform, display, and provide alerts for data. The Admin Manual shows you how to configure those files and includes a reference topic for each configuration file that Splunk uses. In some cases, you can also use Splunk Web or the CLI to make changes to a Splunk app's configuration.

Splunk also uses configuration files to configure itself. When Splunk initializes, it finds all of the configuration files located in the Splunk directory and merges them to build a final "master" configuration, which it then runs on. When you install a Splunk app on a Splunk instance, Splunk determines which configuration files to use if it encounters a conflict. This is where configuration file precedence comes in.

It's important to understand how precedence works. In many cases, if there is a configuration file conflict, Splunk Enterprise gives priority to an app's configuration file. In some situations, installing an app can inadvertently override a setting in a configuration file in Splunk Enterprise, which can lead to undesired results in data collection. Read the previously mentioned topic thoroughly for details.

**Splunk Search**

Splunk provides the ability to look through all of the data it indexes and create dashboards, reports, and even alerts. All Splunk apps rely on Splunk search, so it's a good idea to read the overview on search in the Search Manual to learn how powerful Splunk's search engine is (the Tutorial is also a good place to learn about Splunk search.)

Have an understanding of the Splunk search language. Splunk apps use the search language extensively to put together search results and knowledge objects which drive their dashboards, reports, charts, and tables.

Finally, familiarize yourself with the search commands in the Search Reference. That manual describes the commands that both Splunk and your Splunk app can use.
Sources and source types

When Splunk indexes data, it does so from a source - an entity that provides data for Splunk to extract, for example, Windows event logs, or *nix syslogs. Splunk tags incoming data with a "source" field as it gets indexed. The source type is an indicator for the type of data, so that Splunk knows how to properly format and extract it as it comes in. It's also - conveniently enough - a way to categorize data, as you can use Splunk search to display all data of a certain source type.

Splunk apps use sources and source types to extract knowledge from the data they index. Many views in an application depend on searches with specific sources and source types defined in them. Splunk apps sometimes use the source types that come with Splunk, and sometimes they define their own.

What's next?

From this point, you are ready to plan your app deployment. Continue reading for information about how this app fits into the Splunk picture, platform and hardware requirements, and other deployment considerations.

Welcome to the Machine Learning Toolkit

The Machine Learning Toolkit (MLTK) is an app available for both Splunk Enterprise and Splunk Cloud users through Splunkbase. The Machine Learning Toolkit acts like an extension to the Splunk platform and includes new Search Processing Language (SPL) search commands, macros, and visualizations. On top of the platform extensions meant for machine learning, the MLTK has guided modeling dashboards called Assistants. Assistants walk you through the process of performing particular analytics.

The Machine Learning Toolkit is not a default solution, but a way to create custom machine learning. You must have have domain knowledge, SPL knowledge, Splunk experience, and data science skills or experience to use the toolkit.

Types of machine learning

Machine learning is a process for generalizing from examples. You can use these generalizations, typically called models, to perform a variety of tasks, such as predicting the value of a field, forecasting future values, identifying patterns in
data, and detecting anomalies from new (never-before-seen) data. Without data and correct examples, it is difficult for machine learning to work at all.

The machine learning process starts with a question. You might ask one of these questions:

- Am I being hacked?
- How hot are the servers?
- How many visits to my site do I expect in the next hour?
- What is the price range of houses in a particular neighborhood?

There are different types of machine learning, including:

**Regression**

Regression modeling predicts a number from a variety of contributing factors. Regression is a predictive analytic. For example, you might have utilization metrics on a machine, such as CPU percentage and amount of disk reads and writes. You can use regression modeling to predict the amount of power that that machine is likely to draw both now and in the future.

**Example**

```python
... | fit DecisionTreeRegressor temperature from date_month date_hour into temperature_model
```

**Classification**

Classification modeling predicts a category or a class from a number of contributing factors. Classification is a predictive analytic. For example, you might have data on user behavior on a website or within a software product. You can use classification modeling to predict whether that customer is going to churn.

**Example**

```python
... | fit RandomForestClassifier SLA_violation from * into sla_model
```

**Forecasting**

Forecasting is a predictive analytic that predicts the future of a single value moving through time. Forecasting looks at past measurements for a single value, like profit per day or CPU usage per minute, to predict future values. For example, you might have sales results by quarter for the past 5 years. Use
forecast modeling to predict sales for the upcoming quarter.

**Example**

```bash
... | fit ARIMA Voltage order=4-0-1 holdback=10 forecast_k=10
```

**Clustering**

Clustering groups similar points of data together. For example, you might want to group customers together based on their buying behaviors, such as how much they tend to spend or how many items they buy at one time. Use cluster modeling to group together the features you specify.

**Example**

```bash
... | fit XMeans * | stats count by cluster
```

**Anomaly detection**

Anomaly detection finds outliers in your data by computing an expectation based on one of the machine learning types, comparing it with reality, and triggering an alert when the discrepancies between the two values is large.

**Example**

```bash
... | fit OneClassSVM * kernel="poly" nu=0.5 coef0=0.5 gamma=0.5 tol=1 degree=3 shrinking=f into Model_Name
```

**How the toolkit fits into the Splunk platform**

Different machine learning options exist in the Splunk platform.
- The Splunk platform has machine learning capabilities integrated in the SPL.

- Solutions including IT Service Intelligence (ITSI), Splunk Enterprise Security, and Splunk User Behavior Analytics offer managed machine learning options.

- The Machine Learning Toolkit is an app in the Splunkbase ecosystem that allows you to build custom machine learning solutions for any use case.

**The machine learning process**

Machine learning is a process for generalizing from examples. Ideally, the machine learning process follows a series of steps, beginning collecting data and ending with deploying your machine learning model.

1. Collect available data like CPU percentages, memory utilization, server temperatures, disk space, or sales values.
2. Clean and transform that data. All machine learning expects a matrix of numbers as input. If you're collecting data that is missing values, then you need to clean and transform that data until it's in the form machine learning requires.
3. Explore and visualize the data to make sure it is encoding what you expect it to encode.
4. Build a model on training data.
5. Evaluate the model test data.
6. Deploy the model on un-seen data.
The machine learning process follows those steps in theory, but in practice it’s rarely linear:

![Machine Learning Process Diagram]

You might evaluate your model, discover the performance is not generating the results you expect, and go back to further clean and transform the data. Maybe the data has too many missing values, lacks completeness, is improperly weighted, or has unit disagreement. Iterate around the machine learning process until the model is providing desired machine learning outcomes.

The machine learning process is potentially time consuming and could mean a need for different tools, different team members, and context switching. But with the Machine Learning Toolkit, this entire process, from ingesting the data to building reports, can all occur inside the Splunk platform.

**Layers that make up the Machine Learning Toolkit**

The Splunk Machine Learning Toolkit is comprised of several layers, each of which aid in the process of building a generalization from your data.

*Python for Scientific Computing Library*

Access to over 300 open source algorithms.

The MLTK is built on top of the Python for Scientific Computing Library. This ecosystem includes the most popular machine learning library called sci-kit learn, as well as other supporting libraries like NumPy and Statsmodels.
**ML-SPL API**

Extensibility to easily support any algorithm, whether proprietary or open source.

The toolkit includes access to an extensibility API that not only allows you to expose the 300+ algorithms from the Python app, but also custom algorithms you can write yourself. You can share and reuse your own custom algorithms in the Splunk Community for MLTK on GitHub.

**ML Commands**

New SPL commands to fit, test, score, and operationalize models.

The toolkit uses the PSC app to expose new SPL commands that let you do machine learning. These are SPL commands for building (fit), testing (apply), and validating (score) models and more.

**Algorithms**

Over 30 standard algorithms to accommodate both supervised and unsupervised learning.

These SPL commands expose different algorithms. There are more than 3 standard algorithms, which are the most commonly used machine learning algorithms.

**Showcases**

Interactive examples for typical IT, security, business, and IoT use cases.

The toolkit includes many interactive examples from different domains like IoT and business analytics. This is called the Showcase. The Showcase is the initial page you land on when you open the MLTK, and it lets you see different examples from end to end.

**Experiments and Assistants**

Guided model building, testing, and deployment for common objectives.

On top of the tools for machine learning are guided modeling dashboards called Assistants and a management framework called Experiments. Assistants are a guided walk-through of the process for performing particular analytics with your own data. The Experiment Management Framework (EMF) brings all aspects of
a monitored machine learning pipeline integrated to one interface with automated model versioning and lineage.

What is included in the toolkit

The MLTK includes several platform extensions. This includes a set of macros that allow you to compute validation statistics as well as a set of custom visualizations. The Machine Learning Toolkit provides the key commands of fit and apply:

- **fit** builds the generalization or model
- **apply** lets you use that generalization later

**Commands**

- fit
- apply
- summary
- listmodels
- deletemodel
- sample
- score

**Macros**

- regressionstatistics
- classificationstatistics
- classificationreport
- confusionmatrix
- forecastviz
- histogram
- modvizpredict
- splitby (1-5)

**Visualizations**

- Outliers Chart
- Forecast Chart
- Scatter Line Chart
- Histogram Chart
- Downsampled Line Chart
- Scatterplot Matrix
Other commands, including `summary`, `listmodels`, and `deletemodel`, manage or inspect the models you've created. The toolkit also offers the `sample` command that lets you take random partitions of your data. The `score` command is available to run statistical tests to validate model outcomes.

Fit (ie. train) a model from search results:
```
... | fit <algorithm> <target> from <variables>
    |     into <model>
... | fit LinearRegression "pc_power" from "total-unhalted_core_cycles" "total-cpu-usage" "total-disk-blocking" "total-disk-utilization"
    |     into "example_server_power"
```

Apply a model to get predictions from (new) search results:
```
... | apply <model>
... | apply "example_server_power"
```

Inspect a model (eg. display coefficients):
```
... | summary <model>
... | summary "example_server_power"
```

Validate model outcomes:
```
... | score <method of command>
... | score confusion_matrix
```

To learn more about the `fit` and `apply` commands, see Understanding the fit and apply commands.
To learn more about the `score` command see, Using the score command.

**Introduction to the Assistants**

The Assistants guide you through the process of performing an analytic in the toolkit. Assistants use all tools and extensions, including both SPL and ML-SPL. You can click **Show SPL** at any time within an Assistant to see the SPL.
### Predict Numeric Fields

Predict the value of a numeric field using a weighted combination of the values of other fields in the dataset.

**Assistant Settings**

Enter a search:

```
| inputlookup server_power.csv
```

- 31,272 results (1/1/70 12:00:00.000 AM to 2/11/19 8:05:23.000 PM)

**Preprocessing Steps**

No steps added.

+ Add a step

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Field to predict</th>
<th>Fields to use for predicting</th>
</tr>
</thead>
<tbody>
<tr>
<td>LinearRegression</td>
<td>ac_power</td>
<td>total-unhalted_core... (8)</td>
</tr>
</tbody>
</table>

**Fit Intercept**

- **estimate the intercept**

**Save the model as**

```
example_server_power
```

**Buttons**

- Fit Model
- Schedule Training
- Open in Search
- Show SPL
The Assistants walk you through the entire machine learning process:

- Preparing your data. Examples include replacing or computing missing values, and re-scaling fields with unit disagreement.
- Building the model.
- Validating the model. For example, when you're building a regression model, the preferred way to validate that kind of model is looking at the R2 (squared).
- Deploying the model. If you're happy with the model and want to ensure it continues to make accurate predictions, you might fit it on a schedule, and then inspect that schedule later.

Each type of machine learning has an accompanying Assistant:

<table>
<thead>
<tr>
<th>Type of machine learning</th>
<th>Assistant</th>
</tr>
</thead>
<tbody>
<tr>
<td>Regression</td>
<td>Predict Numeric Fields Assistant</td>
</tr>
<tr>
<td>Classification</td>
<td>Predict Categorical Fields Assistant</td>
</tr>
<tr>
<td>Anomaly detection</td>
<td>Detect Numeric Outliers Assistant</td>
</tr>
<tr>
<td>Anomaly detection</td>
<td>Detect Categorical Outliers Assistant</td>
</tr>
<tr>
<td>Forecasting</td>
<td>Forecast Time Series Assistant</td>
</tr>
<tr>
<td></td>
<td>Smart Forecasting Assistant</td>
</tr>
</tbody>
</table>
Type of machine learning  Assistant

| Clustering          | Cluster Numeric Events Assistant |

Learn more

Continue learning about the MLTK by working through this user guide or through the following links:

- To learn about companion apps, cheat-sheets, videos, courses, and support, see About the Machine Learning Toolkit.
- To learn about installing the Machine Learning Toolkit, see Installing the MLTK.
- To see the Assistants in action using pre-populated use cases, see the Showcase examples.
- To learn about workflow options within the MLTK, see the Splunk Machine Learning Toolkit workflow.

About the Machine Learning Toolkit

The Splunk Machine Learning Toolkit (MLTK) enables users to create, validate, manage, and operationalize machine learning models through a guided user interface. The MLTK extends the Splunk platform functions and lets you build custom analytics for any use case. The toolkit includes over 30 common algorithms and gives you access to more than 300 popular open-source algorithms through the Python for Scientific Computing library.

The MLTK has two main components:

- A Showcase, where you can see the six modeling assistants in action with example use cases. Through the Showcase you can inspect the Search Processing Language (SPL) and underlying code on the sample datasets, to see how it all works.
- Assistants to guide you through the process of applying an analytic from the Showcase to your own data. Follow your preferred workflow to detect incidents, reduce resolution times, optimize business events, and predict and prevent costly events in your organization.

ML-SPL Quick Reference Guide

Download the Machine Learning Toolkit Quick Reference Guide for a handy cheat sheet of ML-SPL commands and machine learning algorithms used in the
ML-SPL Performance App

Download the ML-SPL Performance App for the Machine Learning Toolkit to use performance results for guidance and benchmarking purposes in your own environment.

Extend the algorithms you can use for your models

The algorithms listed here and in the ML-SPL Quick Reference Guide are available natively in the Splunk Machine Learning Toolkit. You can also base your algorithm on over 300 open source Python algorithms from scikit-learn, pandas, statsmodel, numpy and scipy libraries available through the Python for Scientific Computing add-on in Splunkbase.

For information on how to import an algorithm from the Python for Scientific Computing add-on into the Splunk Machine Learning Toolkit, see the ML-SPL API Guide.

Add algorithms through GitHub

On-prem customers looking for solutions that fall outside of the 30 native algorithms can use GitHub to add more algorithms. Solve custom uses cases through sharing and reusing algorithms in the Splunk Community for MLTK on GitHub. Here you can also learn about new machine learning algorithms from the Splunk open source community, and help fellow users of the toolkit.

Cloud customers can also use GitHub to add more algorithms via an app. The Splunk GitHub for Machine learning app provides access to custom algorithms and is based on the Machine Learning Toolkit open source repo. Cloud customers need to create a support ticket to have this app installed.

Additional resources and materials


The toolkit ships with several example datasets meaning you can practice machine learning concepts, or re-create the Showcase examples in your own instance before working with your own data.
Watch Splunk Machine Learning Videos

Read more about machine learning tools in Splunk Machine Learning Blogs

Join the Splunk user group Slack channel.

Sign up to learn more via Splunk Education. We recommend the Splunk course on Analytics and Data Science once you have mastered the fundamentals.

Be part of the conversation on the Splunk Community page.

If you are building a custom app using the Machine Learning Toolkit and want to install it in your cloud environment, see About vetting for Splunk Cloud.

**Support**

Ask questions and get answers through community support at Splunk Answers.

If you have a support contract, submit a case using the Splunk Support Portal.

For general Splunk platform support, see the Splunk Support Programs page.

---

**Share data in the Machine Learning Toolkit**

When the Machine Learning Toolkit is deployed on Splunk Enterprise, the Splunk platform sends anonymized usage data to Splunk Inc. ("Splunk") to help improve the toolkit in future releases. For information about how to opt in or out, and how the data is collected, stored, and governed, see Share data in Splunk Enterprise.

**What data is collected**

The Splunk Machine Learning Toolkit collects the following basic usage information:

<table>
<thead>
<tr>
<th>Component</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>algo_name</td>
<td>Name of algorithm used in fit or apply.</td>
<td><code>{ &quot;algo_name&quot;: &quot;StandardScaler&quot; }</code></td>
</tr>
<tr>
<td>apply_time</td>
<td>Time the apply command took.</td>
<td></td>
</tr>
<tr>
<td>Component</td>
<td>Description</td>
<td>Example</td>
</tr>
<tr>
<td>-------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>-------------------------------------------------------------------------</td>
</tr>
<tr>
<td>app_context</td>
<td>Name of the app from which search is run.</td>
<td>`{ &quot;app_context&quot;: &quot;Splunk_ML_Toolkit&quot; }</td>
</tr>
<tr>
<td>columns</td>
<td>The number of columns being run through <code>fit</code> command.</td>
<td>`{ &quot;columns&quot;: 10 }</td>
</tr>
<tr>
<td>command</td>
<td><code>fit</code> or <code>apply</code></td>
<td>`{ &quot;command&quot;: &quot;apply&quot; }</td>
</tr>
<tr>
<td>csv_parse_time</td>
<td>CSV parse time.</td>
<td>`{ &quot;csv_parse_time&quot;: 0.019296 }</td>
</tr>
<tr>
<td>csv_read_time</td>
<td>CSV read time.</td>
<td>`{ &quot;csv_read_time&quot;: 0.019296 }</td>
</tr>
<tr>
<td>csv_render_time</td>
<td>CSV render time.</td>
<td>`{ &quot;csv_render_time&quot;: 0.01162 }</td>
</tr>
<tr>
<td>example_name</td>
<td>Name of the Showcase example being run.</td>
<td>`{ 'example_name': &quot;'Predict Server Power Consumption'&quot; }</td>
</tr>
<tr>
<td>experiment_id</td>
<td>ID of the <code>fit</code> and <code>apply</code> run on the Experiments page. All preprocessing</td>
<td>`{ &quot;experiment_id&quot;: &quot;6c47bca2776d4b6cb82685461d918180&quot; }</td>
</tr>
<tr>
<td></td>
<td>steps and final <code>fit</code> have the same ID.</td>
<td></td>
</tr>
<tr>
<td>fit_time</td>
<td>Amount of time it took to run the <code>fit</code> command.</td>
<td>`{ &quot;fit_time&quot;: 39.87447 }</td>
</tr>
<tr>
<td>Component</td>
<td>Description</td>
<td>Example</td>
</tr>
<tr>
<td>------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>-------------------------------------------------------------------------</td>
</tr>
</tbody>
</table>
| full_punct       | The punct of the data during fit or apply.                                  | { "full_punct": [ "...s-s[//:::--/s//s//."sss"://://@:://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://::://
<p>| handle_time      | Time for the handler to handle the data.                                  | { &quot;handle_time&quot;: 0.274072                                               |
| num_fields       | Total number of fields.                                                    | { &quot;num_fields&quot;: 4 }                                                     |
| num_fields_fs    | Number of fields that have the fs for Field Selector prefix.              | { &quot;num_fields_fs&quot;: 9 }                                                  |
| num_fields_PC    | Number of fields that have the PC for preprocessed prefix.                | { &quot;num_fields_PC&quot;: 70 }                                                 |
| num_fields_prefixed | Total number of preprocessed fields.                                     | { &quot;num_fields_prefixed&quot;: 28                                              |
| num_fields_RS    | Number of fields that have the RS for Robust Scaler prefix.               | { &quot;num_fields_RS&quot;: 17 }                                                 |
| num_fields_SS    | Number of fields that have the SS for Standard Scaler prefix.             | { &quot;num_fields_SS&quot;: 30 }                                                 |
| num_fields_tfidf | Number of fields that have used term frequency-inverse document frequency| { &quot;num_fields_tfidf&quot;: 9 }                                               |</p>
<table>
<thead>
<tr>
<th>Component</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>orig_sourcetype</td>
<td>The original sourcetype of the machine data.</td>
<td>{ &quot;orig_sourcetype&quot; : &quot;access_combined_wcookie&quot; }</td>
</tr>
<tr>
<td>params</td>
<td>Optional parameters used in fit step.</td>
<td>{ &quot;params&quot;: &quot;{&quot;with_std&quot;: &quot;true&quot;, &quot;with_mean&quot;: &quot;true&quot;}&quot; }</td>
</tr>
<tr>
<td>PID</td>
<td>Process identifier associated with the command.</td>
<td>{ &quot;PID&quot; : 63654 }</td>
</tr>
<tr>
<td>pipeline_stage</td>
<td>Each preprocessing step on the Experiments page is assigned a number starting from 0. This helps determine the order of the preprocessing steps and length of the pipeline.</td>
<td>{ &quot;pipeline_stage&quot; : 0 }</td>
</tr>
<tr>
<td>rows</td>
<td>The number of rows being run through fit command.</td>
<td>{ 'rows': 15627 }</td>
</tr>
<tr>
<td>UUID</td>
<td>Universally unique identifier associated with command. This is 128-bit and used to keep each fit/apply unique.</td>
<td>{ &quot;UUID&quot; : &quot;7e0828e7-3059-4a43-8419-acc0e81f2f2d&quot; }</td>
</tr>
</tbody>
</table>

Install the Machine Learning Toolkit
The Machine Learning Toolkit (MLTK) enables users to create, validate, manage, and operationalize machine learning models through a guided user interface.

Use the following directions to install the MLTK on to your system(s).

**Requirements**

The current version of the Splunk Machine Learning Toolkit is 4.3.0. In order to successfully run this version, the following is required:

- Splunk Enterprise 7.0 or later or Splunk Cloud
- Installation of the Machine Learning Toolkit app from Splunkbase
- Installation of the Python for Scientific Computing add-on version 1.3 or 1.4

  - Mac OS
  - Linux 64-bit
  - Linux 32-bit
  - Windows 64-bit

In order to save models, users need the `upload_lookup_files` capability included in their role.

You cannot access new features in the MLTK without upgrading to the latest version. Versions 3.4.0 and above of the MLTK require upgrading to versions 1.3 or 1.4 of the PSC add-on. See the version dependencies table for the specific requirements between MLTK and PSC add-on versions.

Linux 32-bit support is not available should you upgrade to version 1.4 of the Python for Scientific Computing add-on.

**Specific version dependencies**

<table>
<thead>
<tr>
<th>MLTK Version</th>
<th>PSC Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.3</td>
<td>1.3 or 1.4</td>
</tr>
<tr>
<td>4.2</td>
<td>1.3 or 1.4</td>
</tr>
<tr>
<td>4.1</td>
<td>1.3</td>
</tr>
<tr>
<td>4.0</td>
<td>1.3</td>
</tr>
<tr>
<td>3.4</td>
<td>1.3</td>
</tr>
<tr>
<td>3.3</td>
<td>1.2 or 1.3</td>
</tr>
</tbody>
</table>
If you have written any custom algorithms that rely on the PSC libraries, upgrading to version 1.3 or 1.4 the PSC library add-on will impact those algorithms. You will need to re-train any models (re-run the search that used the `fit` command) using those algorithms after you upgrade the PSC add-on.

### Splunk Cloud deployments

Follow the appropriate directions below for your instance of self-service or managed Splunk Cloud.

#### Splunk Cloud trial and self-service Splunk Cloud


1. Log in to your Splunk Cloud instance.
2. From the Splunk Web home screen, click on the gear icon next to Apps in the left navigation bar.
3. Click **Browse more apps**.
5. Search for the Machine Learning Toolkit app and install it.

#### Managed Splunk Cloud

Open a ticket with support and request the Python for Scientific Computing add-on and the Machine Learning Toolkit app to be installed for you.

### Splunk Enterprise deployments

Follow these directions for either single or distributed deployments.

#### Single instance deployment
Install the Python for Scientific Computing add-on and the Machine Learning Toolkit app onto your single instance Splunk Enterprise

1. Install the Python for Scientific Computing add-on first (required).
2. Install the Splunk Machine Learning Toolkit app.

Install an app or add-on in Splunk Web

1. In Splunk Web, click on the gear icon next to Apps in the left navigation bar.
2. On the Apps page, click **Install app from file**.
3. Click **Choose File**, navigate to and select the package file for the app or add-on, then click **Open**.
4. Click **Upload**.

Install an app or add-on from the command line

At the command line, enter the following content, depending on your operating system.

Unix/Linux: `./splunk install app <path/packagename>`
Windows: `splunk install app <path\packagename>`

Alternatively, unpack/unzip the file then copy the app directory to `$SPLUNK_HOME/etc/apps` on Unix based systems or `%SPLUNK_HOME%\etc\apps` on Windows systems.

**Distributed deployment**

Use the following tables to determine where and how to install the Machine Learning Toolkit and Python for Scientific Computing add-on in a distributed deployment of Splunk Enterprise. Depending on your environment, you may need to install the Machine Learning Toolkit and Python for Scientific Computing add-on in multiple places.

**Where to install Splunk Machine Learning Toolkit and Python for Scientific Computing**

This table provides a reference for installing the Machine Learning Toolkit and Python for Scientific Computing to a distributed deployment of Splunk Enterprise.
Install Python for Scientific Computing and the Splunk Machine Learning Toolkit to all search heads where the Splunk Machine Learning Toolkit is used. Search heads must be running Splunk Enterprise 6.6 or greater.

If you want to use the distributed apply feature of the Splunk Machine Learning Toolkit, install Python for Scientific Computing on all of your indexers. This feature is disabled by default. See Use your indexers to apply models for information. Indexers must be running Splunk Enterprise 6.6 or greater. The Splunk Machine Learning Toolkit does not need to be installed on the indexers to enable this functionality.

These apps do not contain a data collection component.

## Distributed deployment feature compatibility

This table describes the compatibility of the Splunk Machine Learning Toolkit and Python for Scientific Computing add-on with Splunk distributed deployment features.

<table>
<thead>
<tr>
<th>Distributed deployment feature</th>
<th>Supported</th>
<th>Actions required</th>
</tr>
</thead>
<tbody>
<tr>
<td>Search Head Clusters</td>
<td>Yes</td>
<td>Search heads must be running Splunk Enterprise 6.6 or greater.</td>
</tr>
<tr>
<td>Distributed deployment feature</td>
<td>Supported</td>
<td>Actions required</td>
</tr>
<tr>
<td>--------------------------------</td>
<td>-----------</td>
<td>-----------------</td>
</tr>
<tr>
<td>Indexer Clusters</td>
<td>Yes</td>
<td>If you want to use the distributed apply feature of the Splunk Machine Learning Toolkit, install Python for Scientific Computing on the indexers in your cluster. This feature is disabled by default. See Use your indexers to apply models for information. Indexers must be running Splunk Enterprise 6.6 or greater. The Splunk Machine Learning Toolkit does not need to be installed on the indexers in your cluster to enable this functionality.</td>
</tr>
</tbody>
</table>

**Use your indexers to apply models**

If you have more than one Splunk indexer and want to take advantage of the parallel computing power available on your standalone Splunk indexers or Splunk indexing cluster, you can configure your indexers to run the apply command, a CPU-intensive task that applies machine-learning models.

Follow these steps to use your indexers to apply model:

1. Install the Python for Scientific Computing add-on on all of your indexers.
2. On each search head in your deployment, open the local mlspl.conf configuration file in a text editor.
   - For Unix based systems:
     
     ```
     $SPLUNK_HOME/etc/apps/Splunk_ML_Toolkit/local/mlspl.conf
     ```
   - For Windows systems:
     
     ```
     %SPLUNK_HOME%\etc\apps\Splunk_ML_Toolkit\local\mlspl.conf
     ```

   Create the mlspl.conf in the local directory if one does not exist.

3. Copy the **[default]** stanza from the default mlspl.conf configuration file to the local version of the configuration file if this stanza is not present.
   - Location of default mlspl.conf file on Unix based systems:
     
     ```
     $SPLUNK_HOME/etc/apps/Splunk_ML_Toolkit/default/mlspl.conf
     ```
   - Location of default mlspl.conf file on Windows systems:
     
     ```
     %SPLUNK_HOME%\etc\apps\Splunk_ML_Toolkit\default\mlspl.conf
     ```

4. Change the **streaming_apply** command to be true as follows:

   ```
   streaming_apply = true
   ```
Use the deployment methodology of your choice to make these configuration changes.

- To learn about updating search head cluster members, see Use the deployer to distribute apps and configuration updates in the Distributed Search manual.
- To learn about updating peers in an indexer cluster, see Manage app deployment across all peers in the Managing Indexers and Clusters of Indexers manual.

**Bundle replication**

Permanent model files, sometimes referred to as learned models or encoded lookups, are saved on disk. These files follow Splunk knowledge object rules, including permissions and bundle replication. Bundle replication is the process by which knowledge objects on the search head are distributed to the indexers.

The Machine Learning Toolkit includes a number of example model files that support the Showcase page. These examples are powered by .csv lookup files. To prevent performance issues, these .csv lookup files are not included in the toolkit’s bundle replication process.

**Upgrade the Machine Learning Toolkit**

The Machine Learning Toolkit releases new features and enhancements regularly. Refer to this document to learn how to keep your iteration of the toolkit up to date, as well as of any release related dependencies.

To learn about the latest toolkit features and enhancements, see What's new.

**Requirements**

Running version 4.3.0 of the toolkit requires Splunk Enterprise 7.0 or later or Splunk Cloud. The Splunk Machine Learning Toolkit requires the Python for Scientific Computing add-on. Upgrading to version 3.4.0 or above (4.0.0, 4.1.0, 4.2.0, and 4.3.0 currently) of the toolkit requires upgrading to version 1.3 of the Python for Scientific Computing add-on.

In order to save models, users need the upload_lookup_files capability included in their role.
Choose to upgrade to version 1.4 of the Python for Scientific Computing add-on to access all the new features in version 4.3 of the toolkit.

Linux 32-bit support is not available should you upgrade to version 1.4 of the Python for Scientific Computing add-on.

You cannot access new features in the MLTK without upgrading to the latest version of the toolkit. Versions 3.4.0 and above of the toolkit require upgrading to versions 1.3 or 1.4 of the PSC add-on. See the version dependencies table for the specific requirements between toolkit and PSC add-on versions.

**Specific version dependencies**

<table>
<thead>
<tr>
<th>MLTK Version</th>
<th>PSC Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.3</td>
<td>1.3 or 1.4</td>
</tr>
<tr>
<td>4.2</td>
<td>1.3 or 1.4</td>
</tr>
<tr>
<td>4.1</td>
<td>1.3</td>
</tr>
<tr>
<td>4.0</td>
<td>1.3</td>
</tr>
<tr>
<td>3.4</td>
<td>1.3</td>
</tr>
<tr>
<td>3.3</td>
<td>1.2 or 1.3</td>
</tr>
<tr>
<td>3.2</td>
<td>1.2 or 1.3</td>
</tr>
<tr>
<td>3.1</td>
<td>1.2</td>
</tr>
</tbody>
</table>

If you have written any custom algorithms that rely on the PSC libraries, upgrading to version 1.3 or 1.4 the PSC library add-on will impact those algorithms. You will need to re-train any models (re-run the search that used the `fit` command) using those algorithms after you upgrade the PSC add-on.

Any algorithms that have been imported from the Python for Scientific Computing add-on into the MLTK are overwritten when the MLTK app is updated to a new version. Prior to upgrading the MLTK, save your custom algorithms and re-import them manually after the upgrade.

Algorithms are stored in `$SPLUNK_HOME/etc/apps/Splunk_ML_Toolkit/bin/algos` on Unix-based systems and `%SPLUNK_HOME%\etc\apps\Splunk_ML_Toolkit\bin\algos` on Windows systems.
**Splunk Cloud deployments**

For Splunk Cloud trial, self-service Splunk Cloud, or Managed Splunk Cloud, open a ticket with support and request the Python for Scientific Computing add-on and Machine Learning Tooklit app be upgraded to the latest version for you.

**Splunk Enterprise deployments**

Follow these directions for either single or distributed deployments.

*Single instance deployment*

Upgrade the Machine Learning Toolkit app on your single instance Splunk Enterprise

If a newer version of the Python for Scientific Computing add-on is required for the newer version of the Machine Learning Toolkit, a message will display when you run the Machine Learning Toolkit after the upgrade instructing you to install a newer version of the Python for Scientific Computing add-on.

**Update an app or add-on in Splunk Web**

In Splunk Web, click the **Update** option on the app icon in the left-hand Apps bar. The **Update** option appears when a new version of an app is available on Splunkbase.

Alternatively, you can do the following:

1. Download the latest version of the app from Splunkbase.
2. In Splunk Web, click on the gear icon next to Apps in the left navigation bar.
3. On the Apps page, click **Install app from file**.
4. Click **Choose File**, navigate to and select the package file for the app or add-on, then click **Open**.
5. Check the **Upgrade app** box.
6. Click **Upload**.

**Update an existing app on your Splunk instance using the CLI**

Run the following from the command line.

Unix/Linux:
./splunk install app <app_package_filename> -update 1 -auth <username>::<password>

Windows:

splunk install app <app_package_filename> -update 1 -auth <username>::<password>

Alternatively, unpack/unzip the file then copy the app directory to $SPLUNK_HOME/etc/apps on Unix based systems or %SPLUNK_HOME%\etc\apps on Windows systems.

**Distributed deployment**

In a distributed deployment of Splunk Enterprise, update the Machine Learning Toolkit, and Python for Scientific Computing add-on if necessary, on every Splunk instance where the application is installed. Python for Scientific Computing and the Machine Learning Toolkit should be installed on all search heads where the Machine Learning Toolkit is used.

If Python for Scientific Computing is installed on your indexers in order to use the distributed apply feature of the Machine Learning Toolkit, you need to update the Python for Scientific Computing add-on on your indexers as well as your search heads if an update is required. If an update for Python for Scientific Computing is required, you will receive a message indicating this when you run the Machine Learning Toolkit after upgrading. For information about the distributed apply feature, see Use your indexers to apply models.

If you use search head clusters or indexer clusters, use the deployment methodology of your choice to make the updates.

- To learn about updating search head cluster members, see Use the deployer to distribute apps and configuration updates in the Distributed Search manual.
- To learn about about updating peers in an indexer cluster, see Manage app deployment across all peers in the Managing Indexers and Clusters of Indexers manual.
Getting started

Using the Splunk Machine Learning Toolkit


Get started by exploring interactive examples that step you through the entire process for IT, security, business and IoT use cases. When ready choose an Experiment Assistant to guide in creating your own custom built model. You also have complete access to the underlying SPL commands generated by the toolkit. This gives you the freedom to further customize your model and to operationalize it in any way desired.

The Splunk Machine Learning Toolkit provides the following features:

- A Showcase of different sample datasets to help new users explore machine-learning concepts. Each end-to-end example pre-populates an Assistant to demonstrate how to perform different types of machine learning analysis and prediction using best practices, including what ideal results would look like when using your own data. For details see Showcase examples.
- Experiments manage your data source, selected algorithm, and additional parameters used to configure that algorithm. The Experiment Management Framework (EMF) brings all aspects of a monitored machine learning pipeline into one interface with automated model versioning and lineage baked in. Add notes to your Experiment to better track your model adjustments, and look back at previous changes through the Experiments History tab. The Assistants that live within an Experiment make it easy for you to create machine learning models through a guided workflow interface. Each Assistant offers a choice of algorithms to fit and apply a model, with visualizations to help you interpret the results. Assistants are used with your own data, and generate Splunk SPL for you. For details, see Experiments.
- Search command extensions that have been added to the Splunk Search Processing Language (SPL) to perform machine learning analytics on data such as fitting and applying a model. In addition, commands to list, summarize, and delete learned models. For details, see Search commands for machine learning.
• Custom visualizations, which are reusable information graphics for viewing and analyzing data in a particular format. For details, see Custom visualizations.

**The MLTK navigation bar**

You will find seven tabs to select from along the MLTK navigation bar including:

- **Showcases**: End-to-end examples that pre-populate the chosen assistant with a sample dataset, and demonstrate the results.
- **Experiments**: An Experiment is an exclusive knowledge object in Splunk that keeps track of its settings and history, as well as its affiliated alerts and scheduled trainings.
- **Search**: Use your SPL knowledge to perform machine learning analytics on your chosen data.
- **Models**: Access any models created using the `fit` command on the Search tab, or those made through the classic assistants. Model related details such as Model Name, Algorithm Used and Sharing settings are visible.
- **Classic**: Click here for alerts and scheduled trainings that were created in the MLTK version 3.1 or below, as well as the legacy layout of the 6 guided model building Assistants.
- **Settings**: Users with admin access can work within this interface to configure the settings of the `fit` and `apply` commands. Make changes for all algorithms, or for an individual algorithm.

The default settings will be applied to each algorithm unless it has its own value for a particular setting. To ensure you know the impact of making changes to these settings, we recommend adding the ML-SPL Performance App for the Machine Learning Toolkit to your setup via Splunkbase.

- **Docs**: Clicking here takes you out of the toolkit, and over to the documentation manual on the MLTK.
- **Video Tutorials**: Clicking takes you out of the toolkit, and over to a great series of videos on the MLTK.

**Splunk Machine Learning Toolkit files**

You can view the source code for the Splunk Machine Learning Toolkit app.

For Unix-based systems see `$SPLUNK_HOME/etc/apps/Splunk_ML_Toolkit`
For Windows systems see `%SPLUNK_HOME%\etc\apps\Splunk_ML_Toolkit`
Please note: MLTK is not open source. The code is provided as an example and for educational purposes only.

<table>
<thead>
<tr>
<th>Subdirectory</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>appserver/static</td>
<td>Contains the underlying code files (Python, JavaScript, CSS, and images).</td>
</tr>
<tr>
<td>and /bin</td>
<td></td>
</tr>
<tr>
<td>/default</td>
<td>Contains configuration and dashboard files.</td>
</tr>
<tr>
<td>/lookups</td>
<td>Contains the sample datasets used in the Showcase examples, along with more information about the datasets and their licenses.</td>
</tr>
</tbody>
</table>

**Showcase examples**

The Showcase is comprised of pre-populated use cases for the Machine Learning Toolkit. The page is organized around the different guided modeling Assistants within the toolkit. Each modeling Assistant provides a guided machine-learning experience.

Watch and learn from examples drawn from IT, Security, IoT, and Business Analytics. Optionally choose to filter by one of these sub-groups in the drop-down menu that is available in the top left of the screen.

The toolkit ships with all of the example datasets used in the Showcase. Use these datasets to practice machine learning concepts, or to re-create the Showcase examples in your own instance before working with your own data.

The Showcase contains the following examples, grouped by Assistant.

**Predict Numeric Fields**

**Algorithm**: Linear regression

Predict the value of a numeric field using a weighted combination of the values of other fields in that event. A common use of these predictions is to identify anomalies: predictions that differ significantly from the actual value may be considered anomalous.

<table>
<thead>
<tr>
<th>Example</th>
<th>Dataset</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Predict Server Power</td>
<td>Server power (server_power.csv)</td>
<td>Predicts the power usage of a machine</td>
</tr>
<tr>
<td>Example</td>
<td>Dataset</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>--------------------------------</td>
<td>------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Consumption based on other</td>
<td></td>
<td>based on other metrics such as CPU utilization and memory transactions.</td>
</tr>
<tr>
<td>Predict VPN Usage</td>
<td>App statistics (apps.csv)</td>
<td>Predicts the VPN usage of employees based on the frequency of use of other apps.</td>
</tr>
<tr>
<td>Predict Median House Value</td>
<td>Housing (housing.csv)</td>
<td>Predicts median home value in a region based on housing value-related predictor fields.</td>
</tr>
<tr>
<td>Predict Power Plant Energy</td>
<td>Power plant humidity</td>
<td>Predicts the power output of the power plant given other measured variables, such as ambient temperature and humidity.</td>
</tr>
<tr>
<td>Predict Future VPN Usage</td>
<td>App usage (app_usage.csv)</td>
<td>Predict future VPN usage with sinusodial time data.</td>
</tr>
<tr>
<td>Predict Future VPN Usage</td>
<td>App usage (app_usage.csv)</td>
<td>Predict future VPN usage with categorical time data.</td>
</tr>
</tbody>
</table>

**Predict Categorical Fields**

**Algorithm:** Logistic regression

Predict the value of a categorical field using the values of other fields in that
A common use of these predictions is to identify anomalies: predictions that differ significantly from the actual value may be considered anomalous.

<table>
<thead>
<tr>
<th>Example</th>
<th>Dataset</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Predict Hard Drive Failure</td>
<td>Disk failures (disk_failures.csv)</td>
<td>Predicts whether the hard drive is going to fail based on various indicators of drive reliability.</td>
</tr>
<tr>
<td>Predict the Presence of Malware</td>
<td>Firewall traffic (firewall_traffic.csv)</td>
<td>Predicts whether the firewall is going to be affected by malware or has a vulnerability or not based on various traffic indicators on the firewall.</td>
</tr>
<tr>
<td>Predict Telecom Customer Churn</td>
<td>Churn (churn.csv)</td>
<td>Predicts whether a customer will change providers (denoted as churn) based on the usage pattern of customers.</td>
</tr>
<tr>
<td>Predict the Presence of Diabetes</td>
<td>Diabetes (diabetes.csv)</td>
<td>Predicts response in diabetes</td>
</tr>
</tbody>
</table>
### Example Dataset Description

<table>
<thead>
<tr>
<th>Example</th>
<th>Dataset</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Diabetes</td>
<td></td>
<td>data.</td>
</tr>
<tr>
<td>Predict Vehicle Make and Model</td>
<td>Track day (track_day.csv)</td>
<td>Predicts the vehicle type given other onboard metrics.</td>
</tr>
</tbody>
</table>

#### Detect Numeric Outliers

**Algorithm**: Distribution statistics
Find values that differ significantly from previous values.

<table>
<thead>
<tr>
<th>Example</th>
<th>Dataset</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Detect Outliers in Number of Logins (vs. Predicted Value)</td>
<td>Employee logins (logins.csv)</td>
<td>Forecasts the number of logins by hour and identify when the actual number of logins differs significantly from our forecast.</td>
</tr>
<tr>
<td>Detect Outliers in Supermarket Purchases</td>
<td>Supermarket purchases (supermarket.csv)</td>
<td>Detects outliers in the quantity of purchases at a supermarket.</td>
</tr>
<tr>
<td>Detect Outliers in Power Plant Humidity</td>
<td>Power plant humidity (power_plant.csv)</td>
<td>Detects outliers in humidity of a power plant.</td>
</tr>
<tr>
<td>Detect Outliers in Call Center</td>
<td>Call center data (call_center.csv)</td>
<td>Detects cyclical outliers in call center</td>
</tr>
</tbody>
</table>
Detect Categorical Outliers

**Algorithm**: Probabilistic measures
Find events that contain unusual combinations of values.

<table>
<thead>
<tr>
<th>Example</th>
<th>Dataset</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Detect Outliers in Disk Failures</td>
<td>Disk failures (disk_failures.csv)</td>
<td>Detects categorical outliers in disk failure data.</td>
</tr>
<tr>
<td>Detect Outliers in Bitcoin Transactions</td>
<td>Bitcoin transactions (bitcoin_transactions.csv)</td>
<td>Detects outliers in bitcoin transactions that may reflect unusual activity.</td>
</tr>
<tr>
<td>Detect Outliers in Supermarket Purchases</td>
<td>Supermarket purchases (supermarket.csv)</td>
<td>Detects outliers in the whole transaction at a supermarket.</td>
</tr>
<tr>
<td>Detect Outliers in Mobile Phone Activity</td>
<td>Phone usage (phone_usage.csv)</td>
<td>Detects outliers in the number of calls that are incoming, outgoing, or missed from various phones.</td>
</tr>
</tbody>
</table>
Example | Dataset | Description |
--- | --- | --- |
Forecast Internet Traffic | Internet traffic (internet_traffic.csv) | Forecasts the peak and off-peak times of internet usage given a few full cycles of internet traffic history. |
Forecast the Number of Employee Logins | Employee logins (logins.csv) | Forecasts the number of logins by hour. |
Forecast Monthly Sales | Souvenir sales (souvenir_sales.csv) | Forecasts the number of souvenir sales by month for a Souvenir Shop. |
Forecast the Number of Bluetooth Devices | Bluetooth devices (bluetooth.csv) | Forecasts the number of distinct Bluetooth contacts that are made to the access points placed in the busiest lecture halls on the campus of the National University of Singapore. |
Forecast Exchange Rate TWI using ARIMA | Exchange Rate TWI (exchange.csv) | Forecasts the trade weighted index of a currency |

**Smart Forecasting**

**Algorithm:** StateSpaceForecast  
Forecast future numeric time series data using a step-by-step guided workflow with the option to bring in data from different sources and account for calendar specific "special days" such as holidays, company-specific event days.

<table>
<thead>
<tr>
<th>Example</th>
<th>Dataset</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Forecast the Number of Calls to a Call Center</td>
<td>Call center data (call_center.csv)</td>
<td>Forecasts the number of calls to a call center.</td>
</tr>
<tr>
<td>Forecast App Logons with Special Days</td>
<td>Apps logon count (applogonscount.txt) and Special days (specialdays.txt)</td>
<td>Forecasts the logons to an app while accounting for special calendar days.</td>
</tr>
<tr>
<td></td>
<td>App usage (app_usage.csv)</td>
<td></td>
</tr>
</tbody>
</table>
Cluster Numeric Events

**Example** | **Dataset** | **Description**
---|---|---
Forecast App Expenses |  | Forecasts the expenses for an app.

### Cluster Numeric Events

**Algorithms:** K-means, DBSCAN, Spectral Clustering, Birch

Partition events with multiple numeric fields into clusters.

<table>
<thead>
<tr>
<th>Example</th>
<th>Dataset</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cluster Hard Drives by SMART Metrics</td>
<td>Disk failures (disk_failures.csv)</td>
<td>Clusters hard drives based on the self-monitoring metrics they generate.</td>
</tr>
<tr>
<td>Cluster Behavior by App Usage</td>
<td>App usage (app_usage.csv)</td>
<td>Clusters the behavior of employees based on how frequently they use business applications like Webmail or VPN.</td>
</tr>
<tr>
<td>Cluster Neighborhoods by Properties</td>
<td>Housing (housing.csv)</td>
<td>Clusters neighborhoods based on properties like crime rate and median house value.</td>
</tr>
<tr>
<td>Cluster Vehicles by Onboard Metrics</td>
<td>Track day (track_day.csv)</td>
<td>Clusters vehicles driven on a racetrack by onboard metrics like engine temperature and G-forces.</td>
</tr>
<tr>
<td>Cluster Power Plant Operating Regimes</td>
<td>Power plant humidity (power_plant.csv)</td>
<td>Clusters the operating regimes of a power plant based on ambient...</td>
</tr>
</tbody>
</table>
**Understanding the fit and apply commands**

The Splunk Machine Learning Toolkit (MLTK) uses the `fit` and `apply` search commands to train and fit a machine learning model, sometimes referred to as a learned model.

At the highest level:

- The `fit` command produces a learned model based on the behavior of a set of events.
- The `fit` command then applies that model to the current set of search results in the search pipeline.
- The `apply` command repeats the field selection of the `fit` command steps.

This document explains how the `fit` and `apply` commands train and apply the machine learning model based on an algorithm. These commands modify the events from the sample data you input into Splunk. All the sample data ingested serves as examples for the machine learning model to train on.

Before training the model, data may need to be cleaned up in order to remove gaps in data, clean up data or remove erroneous data.

This document walks through the `fit` and `apply` commands with example data. In this example, we want to predict `field_A` from all the other fields in the data available.

**What the fit command does**

The Machine Learning Toolkit performs these steps when the `fit` command is run:

1. Search results are pulled into memory.
2. The `fit` command transforms the search results in memory through these data preparation actions:
   1. Discard fields that are null throughout all the events.
   2. Discard non-numeric fields with more than (> 100) distinct values.
   3. Discard events with any null fields.
   4. Convert non-numeric fields into "dummy variables" by using one-hot encoding.
   5. Convert the prepared data into a numeric matrix representation and run the specified machine learning algorithm to create a model.
3. Apply the model to the prepared data and produce new (predicted) columns.
4. Learned model is encoded and saved as a knowledge object.

**Search results are pulled into memory**

When you run a search, the `fit` command pulls the search results into memory, and parses them into Pandas DataFrame format. The originally ingested data is not changed.

**Discard fields that are null throughout all the events**

In the search results copy, the `fit` command discards fields that contain no values.

The following example demonstrates how the `fit` command looks for incidents of fraud within a dataset. It shows a simplified visual representation of the search results. As part of data preparation, `field_C` is highlighted to be removed because there are no values in this field.

<table>
<thead>
<tr>
<th>field_A</th>
<th>field_B</th>
<th>field_C</th>
<th>field_D</th>
<th>field_E</th>
</tr>
</thead>
<tbody>
<tr>
<td>ok</td>
<td>41</td>
<td>red</td>
<td>172.24</td>
<td>16.5</td>
</tr>
<tr>
<td>ok</td>
<td>32</td>
<td>green</td>
<td>192.168</td>
<td>0.2</td>
</tr>
<tr>
<td>FRAUD</td>
<td>1</td>
<td>blue</td>
<td>10.6</td>
<td>6.6</td>
</tr>
<tr>
<td>ok</td>
<td>43</td>
<td></td>
<td>171</td>
<td>64  72.1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>192.168</td>
<td>0.2</td>
</tr>
</tbody>
</table>

If you do not want null fields to be removed from the search results, you must change your search. For example, to replace the null values with 0 in the results for `field_C`, you can use the SPL `fillnull` command. You must specify the `fillnull` command before the `fit` command, as shown in the following search example:

```plaintext
... | fillnull field_C | fit LogisticRegression field_A from field_*
```
Discard non-numeric fields with more than (>) 100 distinct values

In the search results copy, the fit command discards non-numeric fields if the fields have more than 100 distinct values. In machine learning many algorithms do not perform well with high-cardinality fields, because every unique, non-numeric entry in a field becomes an independent feature. A high-cardinality field leads to an explosion in feature space very quickly.

In the fit command example, none of the fields have a non-numeric field with more than 100 distinct values, so no action is taken.

Should the search results have had more than 100 distinct Internet Protocol (IP) addresses in field_E it would qualify as high-cardinality. The IP numbers get interpreted as non-numeric or string values. The fit command would remove field_E from the search results copy in that case.

One alternative is to use the values in a high-cardinality field to generate a usable feature set. For example, by using SPL commands such as streamstats or eventstats, you can calculate the number of times an IP address occurs in your search results. You must generate these calculations in your search before the fit command. The high-cardinality field is removed by the fit command, but the field that contains the generated calculations remains.

The limit for distinct values is set to 100 by default. You can change the limit by changing the max_distinct_cat_values attribute in your local copy of the mlspl.conf file. See Configure the fit and apply commands for details on updating the mlspl.conf file attributes.

Prerequisites

- Only users with file system access, such as system administrators, can make changes to the mlspl.conf file.
- Review the steps in How to edit a configuration file in the Admin Manual.

Never change or copy the configuration files in the default directory. The files in the default directory must remain intact and in their original location. Make
the changes in the local directory.

Steps

1. Open the local mlspl.conf file. For example, $SPLUNK_HOME/etc/apps/Splunk_ML_Toolkit/local/.
2. Under the [default] stanza, specify a different value for the max_distinct_cat_values setting.

Discard events with any null fields

To train a model, the machine learning algorithm requires all of the search results in memory to have a value. Any null value means the entire event will not be contributed to the learned model. The fit command example dropped every column that is entirely null, and now it drops every row (event) that has one or more nulls.

Alternatively, you can specify that any search results with null values be included in the learned model. Choose to replace null values if you want the algorithm to learn from an example with a null value, and to return an empty collection. Or choose to replace null values if you want the algorithm to learn from an example with a null value, and to throw an exception.

To include the results with null values in the model, you must replace the null values before using the fit command in your search. You can replace null values by using SPL commands such as fillnull, filldown, or eval.

Convert non-numeric fields into "dummy variables" by using one-hot encoding

In the search results copy, the fit command converts fields that contain strings or characters into numbers. Algorithms expect numerically scored data, not strings or characters. The fit command converts non-numeric fields to binary indicator variables (1 or 0) using one-hot encoding.
When the `fit` command converts non-numeric fields using one-hot encoding, strings, characters, and other non-numeric values for each field are encoded as binary values (1 or 0). In this example `field_D`, which previously contained strings and characters, is converted to three fields: `field_D=red`, `field_D=green`, `field_D=blue`. The values for these new fields are either 1 or 0. The value of 1 is placed in the search results where the color name appeared previously. For example, in the new field `field_D=red`, a 1 appears in the first result, and zeros appear in the other results.

The following example shows the results of one-hot encoding. This change occurs internally, within the search results copy in memory.

<table>
<thead>
<tr>
<th>Target</th>
<th>Explanatory Variables...</th>
</tr>
</thead>
<tbody>
<tr>
<td>field_A</td>
<td>field_B</td>
</tr>
<tr>
<td>ok</td>
<td>41</td>
</tr>
<tr>
<td>ok</td>
<td>32</td>
</tr>
<tr>
<td>FRAUD</td>
<td>1</td>
</tr>
</tbody>
</table>

If you want more than 100 values per field, you can use one-hot encoding with SPL commands before using the `fit` command.

**Example:**

In the following example, SPL is used to code search results without limiting values to 100 values per field:

```
| eval {field_D}=1? | fillnull 0
```

**Convert the prepared data into a numeric matrix representation and run the specified machine learning algorithm to create a model**

The `fit` command has prepared the search results copy for the algorithm. This dense numeric matrix is used to train the machine learning model. A temporary model is created in memory.
**Apply the model to the prepared data and produce new (predicted) columns**

The `fit` command applies the model to the now prepared data. These are the results that are returned from the search before the `fit` command is run. The model is applied to each search result to predict values for the search results, including the search results with null values.

The `fit` command appends one or more fields (columns) to the results. The appended search results are then returned to the search pipeline.

The following image shows the original search results with the appended field. The name of the appended field starts with Predicted and includes the name of the `<field_to_predict>`, as specified with the `fit` command. In this example, the `<field_to_predict>` is `field_A`. The name of the appended field is `Predicted (field_A)`. This field contains predicted values for all of the results. Although there is an empty field in our target column, a predicted result is still generated. This works because the predicted value is generated from all the other available fields, not from the target field value.

![Image showing original search results with appended field](image)

**Learned model is encoded and saved as a knowledge object**

If the algorithm supports saved models, and the `into` clause is included in the `fit` command, the learned model is saved as a knowledge object.

When the temporary model file is saved, it becomes a permanent model file. These permanent model files are sometimes referred to as learned models or encoded lookups. The learned model is saved on disk. The model follows all of the Splunk knowledge object rules, including permissions and bundle replication.

If the algorithm does not support saved models, or the `into` clause is not included, the temporary model is deleted.
What the apply command does

The apply command, similar to the fit command, goes through a series of steps to prepare the data for the saved model. The apply command uses the learned model to generate predictions on the supplied data.

The apply command is fast to run because the coefficients created through the fit process and the resulting model artifact are computed and saved. Think of apply like a streaming command that can be applied to data.

The Machine Learning Toolkit performs these steps when the apply command is run:

1. Load the learned model.
2. The apply command transforms the search results in memory through these data preparation actions:
   1. Discard fields that are null throughout all the events.
   2. Discard non-numeric fields with more than (>100) distinct values.
   3. Convert non-numeric fields into "dummy variables" by using one-hot encoding.
   4. Discard dummy variables that are not present in the learned model.
   5. Fill missing dummy variables with zeros.
   6. Convert the prepared data into a numeric matrix representation.
3. Apply the model to the prepared data and produce new (predicted) columns.

Load the learned model

The learned model specified by the search command is loaded in memory. Normal knowledge object permission parameters apply.

Examples

...| apply temp_model

...| apply user_behavior_clusters

Discard fields that are null throughout all the events

This step removes all fields that have no values for this search results set.
Discard non-numeric fields with more than (>100) distinct values

Next, apply removes all fields that have more than 100 distinct values.

The limit for distinct values is set to 100 by default. You can change the limit by changing the max_distinct_cat_values attribute in your local copy of the mlspl.conf file. See Configure the fit and apply commands for details on updating the mlspl.conf file attributes.

Prerequisites

- Only users with file system access, such as system administrators, can make changes to the mlspl.conf file.
- Review the steps in How to edit a configuration file in the Admin Manual.

Never change or copy the configuration files in the default directory. The files in the default directory must remain intact and in their original location. Make the changes in the local directory.

Steps

1. Open the local mlspl.conf file. For example, $SPLUNK_HOME/etc/apps/Splunk_ML_Toolkit/local/.
2. Under the [default] stanza, specify a different value for the max_distinct_cat_values setting.

Convert non-numeric fields into "dummy variables" by using one-hot encoding

The apply command converts fields that contain strings or characters into numbers. Algorithms expect numerically scored data, not strings or characters. This step converts non-numeric fields to binary indicator variables using one-hot encoding.

When converting categorical variables, a new value might come up in the data. In this example, there is new color data for yellow. This new data requires the one-hot encoding step, which converts column D into a binary value (1 or 0). In the graphic below, there is a new column for field_D=yellow.
Discard dummy variables that are not present in the learned model

The apply command removes data that is not part of the learned (saved) model. The data for the color yellow did not appear during the fit process. As such, the column created in the convert non-numeric fields step is discarded.

Fill missing dummy variables with zeros

Any result with missing dummy variables are automatically filled with the value 0 at this step. Doing so is a standard machine learning practice, in order for the algorithm to be applied.

Convert the prepared data into a numeric matrix representation

The apply command creates a dense numeric matrix. The model file is applied to the copy of the event and the results are calculated.

Apply the model to the prepared data and produce new (predicted) columns

The apply command returns to the prepared data and adds the results columns to the search pipeline. Although there is an empty field in our target column, a predicted result still generated. This works because the predicted value is generated from all the other available fields, not from the target field value.
Splunk Machine Learning Toolkit workflow

Overview

Machine learning is a process for generalizing from examples. These generalizations, typically called models, are used to perform a variety of tasks such as predicting the value of a field, forecasting future values, identifying patterns in data, and detecting anomalies from new (never-before-seen) data. Machine learning isn't magic, it's a process that starts with a question. For example:

- Am I being hacked?
- How hot are the servers?
- How many visits to my site do I expect in the next hour?
- What is the price range of houses in a particular neighborhood?

The Splunk Machine Learning Toolkit lets users create analytics in 6 useful areas:

- Predict Numeric Fields
- Predict Categorical Fields
- Detect Numeric Outliers
- Detect Categorical Outliers
- Forecast Time Series
- Cluster Numeric Events

The toolkit includes over 30 common algorithms, and gives you access to over 300 popular open source algorithms through the Python for Scientific Computing library.

Get started by exploring interactive examples that step you through the entire process for IT, security, business and IoT use cases. When ready, choose a guided modeling assistant to step you through creating your own custom built model.

You also have complete access to the underlying SPL commands generated by the toolkit. This gives you the freedom to further customize your model and to operationalize it in any way desired.

Workflow options

The MLTK's guided modeling assistants live within the six different areas listed above. There are 2 main workflows for these areas:
Both the Classic and Experiment workflows follow the same major steps:

1. Specify a data source via the search bar.
2. Select an algorithm and algorithm parameters.
   - Click here to see a list of all supported algorithms
   - Some of the assistants provide the ability to apply multiple sequential transformations to your data. See preprocessing methods for information.
3. Select the fields for the algorithms to analyze, and set training/test data splits.
4. Instruct the assistant to fit the algorithm to the selected training data and generate results, including visualizations and statistical analysis. Commands can vary depending upon which Assistant is selected:
   - Predict Numeric and Predict Categorical Fields use the fit and apply commands
   - Forecast Time Series can use the predict command
   - Cluster Numeric Events uses the fit and apply commands
5. Once complete, optionally schedule regular re-training of the model and deploy it in your production environment as a scheduled alert.

Not all of the Assistants, nor all of the algorithms result in a model being created.

**Training data examples**

The MLTK Assistants, regardless of Experiment or Classic framework, support both supervised and unsupervised learning:
In supervised learning, the model learns from labeled examples through prediction, regression and forecasting methods. In unsupervised learning, the model learns from unlabeled examples through clustering methods.

**The MLTK guided modeling Assistants**

Select from the following links to see specific workflows for each guided modeling Assistant within both the experiment and classic frameworks:

**Experiment framework**

- Predict Numeric Fields Experiment Assistant
- Predict Categorical Fields Experiment Assistant
- Detect Numeric Outliers Experiment Assistant
- Detect Categorical Outliers Experiment Assistant
- Forecast Time Series Experiment Assistant
- Cluster Numeric Events Experiment Assistant

**Classic framework**

- Predict Numeric Fields Classic Assistant
- Predict Categorical Fields Classic Assistant
- Detect Numeric Outliers Classic Assistant
- Detect Categorical Outliers Classic Assistant
- Forecast Time Series Classic Assistant
- Cluster Numeric Events Classic Assistant
Custom search commands

Search commands for machine learning

The Splunk Machine Learning Toolkit contains several custom search commands, referred to as ML-SPL commands, that implement classic machine learning and statistical learning tasks:

- **fit**: Fit and apply a machine learning model to search results.
- **apply**: Apply a machine learning model that was learned using the `fit` command.
- **summary**: Return a summary of a machine learning model that was learned using the `fit` command.
- **listmodels**: Return a list of machine learning models that were learned using the `fit` command.
- **deletemodel**: Delete a machine learning model that was learned using the `fit` command.
- **sample**: Randomly sample or partition events.
- **score**: Run statistical tests to validate model outcomes.

You can use these custom search commands on any Splunk platform instance on which the Splunk Machine Learning Toolkit is installed.


**fit**

Use the `fit` command to fit and apply a machine learning model to search results.

**Supervised Syntax**

```plaintext
fit <algorithm> (<option_name>=<option_value>)* (<response-field>) from (<explanatory-field>)*
```

**Unsupervised Syntax**

```plaintext
fit <algorithm> (<option_name>=<option_value>)* (from)? (<explanatory-field>)*
```
The first argument, which is required, is the algorithm to use. There are a number of available algorithms, which are documented here: Algorithms

All algorithms require a list of fields to use when learning a model. For classification and regression algorithms, follow the response field with the from keyword. Subsequent fields are the fields to use when making predictions (i.e. explanatory fields). The from keyword separates the response field from the explanatory fields. When using unsupervised algorithms, there is no response field, and the from keyword is optional. In the unsupervised case, all fields listed will be treated as explanatory fields when training the model (i.e. for clustering, which fields to cluster over).

Use the as keyword to rename the field added to search results by the model.

Use the into keyword to store the learned model in an artifact that can later be applied to new search results with the apply command. Not all algorithms support saved models.

Some algorithms support options that can be given as name = value arguments. For example, KMeans and PCA both support a k option that specifies how many clusters or how many principal components to learn.

You can also configure the fit command. See Configure the fit and apply commands.

**Examples**

Fit a LinearRegression model to predict errors using _time:

```
... | fit LinearRegression errors from _time
```

Fit a LinearRegression model to predict errors using _time and save it into a model named errors_over_time:

```
... | fit LinearRegression errors from _time into errors_over_time
```

Fit a LogisticRegression model to predict a categorical response from numerical measurements:

```
... | fit LogisticRegression species from petal_length petal_width sepal_length sepal_width
```
**apply**

Use the `apply` command to compute predictions for the current search results based on a model that was learned by the `fit` command. The `apply` command can be used on different search results than those used when fitting the model, but the results should have an identical list of fields.

You can also configure the `apply` command. See Configure the `fit` and `apply` commands.

**Syntax**

```
apply <model_name> (as <output_field>)?
```

Use the `as` keyword to rename the field added to search results by the model.

**Examples**

Apply a learned `LinearRegression` model, "errors_over_time":

```
... | apply errors_over_time
```

Rename the output of the model to "predicted_errors":

```
... | apply errors_over_time as predicted_errors
```

**summary**

Use the `summary` command to return a summary of a machine learning model that was learned using the `fit` command. The summary is algorithm specific. For example, the summary for the `LinearRegression` algorithm is a list of coefficients. The summary for the `LogisticRegression` algorithm is a list of coefficients for each class.

**Syntax**

```
summary <model_name>
```

**Examples**

Inspect a learned `LinearRegression` model "errors_over_time":

```
| summary errors_over_time
```
listmodels

Use the listmodels command to return a list of machine learning models that were learned using the fit command. The algorithm and arguments given when fit was invoked are displayed for each model.

Syntax

    listmodels

Examples

List all models:

    | listmodels

deletemodel

Use the deletemodel command to delete a machine learning model learned using the fit command.

Syntax

    deletemodel <model_name>

Examples

Delete the "errors_over_time" model:

    | deletemodel errors_over_time

cmmand

Use the sample command to randomly sample or partition events.

Sampling modes:

- ratio: A float between 0 and 1 indicating the probability as a percentage that each event has of being included in the result set. For example, a ratio of 0.01 means that events have a 1% probability of being included in the results. Use ratio when you want an approximation.
You can omit the `ratio` keyword, for example use `sample ratio=0.01` or `sample 0.01`.

- **count**: A number that indicates the exact number of randomly-chosen events to return. If the sample count exceeds the total number of events in the search, all events are returned.

You can omit the `count` keyword, for example use `sample count=10` or `sample 10`.

- **proportional**: The name of a numeric field to use to determine the sampling probability of each event, which yields a biased sampling. Each event is sampled with a probability specified by this field value.

Partitioning mode:

- **partitions**: The number of partitions in which to randomly divide events, approximately split. Use `partitions` when you want to divide your results into groups for different purposes, such as using results for testing and training.

Additional options:

- **seed**: A number that specifies a random seed. Using `seed` ensures reproducible results. If unspecified, a pseudorandom value is used.
- **by <field>**: Used with `count`. Specifies a field by which to split events, returning the `count` number of events for each value of the specified field. If there are more events than `count`, all events are included in the results.
- **inverse**: Used with `proportional`. Inverts the probability, returning samples with one minus the probability specified in the proportional field.
- **fieldname**: The name of the field in which to store the partition number. Defaults to `partition_number`.

This `sample` command is not identical to using sampling options on the Event Sampling menu on the Search page in Splunk Web:

- Options from the Event Sampling menu perform sampling before the data is collected from indexes, at the beginning of the search pipeline.
- The `sample` command is applied after data is collected, accessing everything in the search pipeline.

Using the Event Sampling menu options is faster, but the `sample` command can be used anywhere in the search command and provides several modes that are
not available to the **Event Sampling** feature. For example, the `sample` command supports partitioning, biased sampling, and the ability to retrieve an exact number of results.

**Syntax**

```plaintext
sample (ratio=<float between 0 and 1>)? (count=<positive integer>)? (proportional=<name of numeric field> (inverse)?)? (partitions=<natural number greater than 1> (fieldname=<string>)?)? (seed=<number>)? (by <split_by_field>)?
```

**Examples**

Retrieve approximately 1% of events at random:

```plaintext
... | sample ratio=0.01
... | sample 0.01
```

Retrieve exactly 20 events at random:

```plaintext
... | sample count=20
... | sample 20
```

Retrieve exactly 20 events at random from each host:

```plaintext
... | sample count=20 by host
```

Return each event with a probability determined by the value of "some_field":

```plaintext
... | sample proportional="some_field"
```

Partition events into 7 groups, with the chosen group returned in a field called "partition_number":

```plaintext
... | sample partitions=7 fieldname="partition_number"
```

**score**

The `score` command runs statistical tests to validate model outcomes. Use the `score` command to validate models and statistical tests for any use case.

**Syntax**
... | score <scoring_method> (<field> | <selector>=<field>)* [as <outputfield>] [by <splitfield>] [<arg> | <param>=<value>]* | ...

Example

... | score confusion_matrix true="species" pred="predicted(species)"

The Splunk Machine Learning Toolkit includes the following classes of the score command, each with their own sets of methods (ie. Accuracy, F1-score, T-test etc):

- Classification
- Clustering scoring
- Pairwise distances scoring
- Regression scoring
- Statistical functions (statsfunctions)
- Statistical testing (statstest)

Score commands are not customizable within the Splunk Machine Learning Toolkit.

The Splunk Machine Learning Toolkit also helps you test for model overfitting through the K-fold scoring option.

**Search macros in the Machine Learning Toolkit**

The Machine Learning Toolkit (MLTK) ships with several search macros. **Search macros** are reusable chunks of Search Processing Language (SPL) that you can insert into other searches. Search macros can be any part of a search, such as an `eval` statement or search term and do not need to be a complete command. You can also specify whether the macro field takes any arguments.

Use these macros to save time writing common SPL queries and to validate models. This document details the following three particular macros:

- Classification statistics
- Confusion matrix
- Regression statistics
View the MLTK search macros

In your own Splunk instance view the available macros via the Settings drop down menu in the Splunk bar, and by selecting Advanced Settings.

On the resulting page, choose Search macros.

From the App menu, choose the Splunk Machine Learning Toolkit for MLTK search macros. Listed information includes the name, definition, and status. Search macros that take arguments are identified by a bracketed number following the name. For example, confusionmatrix(2) and regressionstatistics(2). Confusion matrix and regression statistics are the search macro names, each of which can take two (2) arguments.
Insert search macros into search strings

To include a search macro in your saved or ad hoc searches, place a back tick character (`) before and after the macro name. You can also reference a search macro within other search macros using this same syntax.

For search macros that take arguments, define those arguments when you insert the macro into the search string. The following example shows a search macro with the arguments defined.

```
... | `classificationstatistics("DiskFailure", "predicted(DiskFailure)`)`
```

**Classification statistics macro**

Use the classification statistics macro to save time when measuring the statistics of your classification model.

**Syntax**

```
... | `classificationstatistics(response, prediction)`
```

**Example**

The following example shows the classification statistics macro on a test set. The first code block shows the passing of the `fit` command with the LogisticRegression algorithm.

```
| inputlookup disk_failures.csv | eventstats max(SMART_1_Raw) as max1
min(SMART_1_Raw) as min1 | eventstats max(SMART_2_Raw) as max2
min(SMART_2_Raw) as min2 | eventstats max(SMART_3_Raw) as max3
min(SMART_3_Raw) as min3 | eventstats max(SMART_4_Raw) as max4
min(SMART_4_Raw) as min4 | eventstats max(SMART_5_Raw) as max5
min(SMART_5_Raw) as min5 | eval SMART_1_Transformed = (SMART_1_Raw - min1)/(max1-min1) | eval SMART_2_Transformed = (SMART_2_Raw - min2)/(max2-min2) | eval SMART_3_Transformed = (SMART_3_Raw - min3)/(max3-min3) | eval SMART_4_Transformed = (SMART_4_Raw - min4)/(max4-min4) | eval SMART_5_Transformed = (SMART_5_Raw - min5)/(max5-min5) | table Date Model CapacityBytes SerialNumber
DiskFailure SMART_1_Raw SMART_1_Transformed SMART_2_Raw SMART_2_Transformed SMART_3_Raw SMART_3_Transformed SMART_4_Raw SMART_4_Transformed SMART_5_Raw SMART_5_Transformed | fit LogisticRegression fit_intercept=true "DiskFailure" from "Model"
"SMART_1_Transformed" "SMART_2_Transformed" "SMART_3_Transformed"
"SMART_4_Transformed" "SMART_5_Transformed" into "example_disk_failures"
```
The second code block shows the passing of the `apply` command, followed by the macro.

```
| inputlookup disk_failures.csv | eventstats max(SMART_1_Raw) as max1
min(SMART_1_Raw) as min1 | eventstats max(SMART_2_Raw) as max2
min(SMART_2_Raw) as min2 | eventstats max(SMART_3_Raw) as max3
min(SMART_3_Raw) as min3 | eventstats max(SMART_4_Raw) as max4
min(SMART_4_Raw) as min4 | eventstats max(SMART_5_Raw) as max5
min(SMART_5_Raw) as min5 | eval SMART_1_Transformed = (SMART_1_Raw - min1)/(max1-min1)
| eval SMART_2_Transformed = (SMART_2_Raw - min2)/(max2-min2)
| eval SMART_3_Transformed = (SMART_3_Raw - min3)/(max3-min3)
| eval SMART_4_Transformed = (SMART_4_Raw - min4)/(max4-min4)
| eval SMART_5_Transformed = (SMART_5_Raw - min5)/(max5-min5)
| table Date Model CapacityBytes SerialNumber DiskFailure SMART_1_Raw SMART_1_Transformed SMART_2_Raw
SMART_2_Transformed SMART_3_Raw SMART_3_Transformed SMART_4_Raw
SMART_4_Transformed SMART_5_Raw SMART_5_Transformed | apply "example_disk_failures" | `classificationstatistics("DiskFailure", "predicted(DiskFailure)")`
```

**Example output**

---

**Classification report macro option**

You have the option to view the classification statistics results by class using the classification report macro. The classification report macro gives you the weighted average for each of the classification statistics classes.

**Example output**
Confusion matrix macro

Use the confusion matrix macro to save time when assessing the performance of your classification model.

Syntax

```
... | `confusionmatrix(response, prediction)`
```

Example

The following example shows the confusion matrix macro on a test set. The first code block shows the passing of the `fit` command with the LogisticRegression algorithm.

```
| inputlookup diabetes.csv
| sample partitions=3 seed=42
| search partition_number < 2
| fit LogisticRegression response from BMI age into LogisticRegressionClassifier
```

The second code block shows the passing of the `apply` command, followed by the macro.

```
| inputlookup diabetes.csv
| sample partitions=3 seed=42
| search partition_number = 2
| apply LogisticRegressionClassifier as prediction
| `confusionmatrix(response, prediction)`
```

Example output

![Confusion Matrix](image)

Classification report macro option

You have the option to view the confusion matrix results by class using the classification report macro. The classification report macro gives you the weighted average for each of the confusion matrix classes.

Example output
Regression statistics macro

Use the regression statistics macro to save time when measuring the statistics of your regression model.

Syntax

```
... | `regressionstatistics(response, prediction)`
```

Example

The following example shows the regression statistics macro on a test set. The first code block shows the passing of the `fit` command with the `LinearRegression` algorithm.

```
| inputlookup server_power.csv | fit LinearRegression fit_intercept=true
"ac_power" from "total-unhalted_core_cycles"
"total-instructions_retired" "total-last_level_cache_references"
"total-memory_bus_transactions" "total-cpu-utilization"
"total-disk-accesses" "total-disk-blocks" "total-disk-utilization" into
"example_server_power"
```

The second code block shows the passing of the `apply` command, followed by the macro.

```
| inputlookup server_power.csv | apply "example_server_power" | `regressionstatistics("ac_power", "predicted(ac_power)")`
```

Example output

Configure permissions for ML-SPL commands

You can configure permissions to restrict access to ML-SPL commands.
Restricting access can help you limit `fit` and `apply` command resource utilization to only those users who require the commands.

You can change ML-SPL command permissions in Splunk Web or in `default.meta.conf`.

**Change permissions in Splunk Web**

1. In Splunk Web, select **Settings > Advanced search**.
2. Click **Search commands**.
3. Under the Sharing column for the search command, click **Permissions**.
4. In the Permissions view, specify:
   1. Whether the command appears in the current app or all apps.
   2. Which roles have read and write access to the command.
5. Click **Save**.

**Change permissions in default.meta.conf**

To change permissions, edit the `default.meta.conf` file in `$SPLUNK_HOME/etc/apps/Splunk_ML_Toolkit/metadata/`.

**Example**
This example shows the default access for `commands.conf` and access permissions for the `fit` command. In this case, running `fit` is restricted to users with the admin role.

```
[commands]
access = read : [ * ], write : [ admin ]
export = system

[commands/fit]
access = read : [ admin ], write : [ admin ]
```

**Additional information**

For more details on configuring user access to search commands, see the following topics in the Search Manual:

- Security responsibilities
- Control access to the custom command
Configure the fit and apply commands

The MLSPL.conf file controls the resources used by the Machine Learning Toolkit. Here, users with Admin level access can configure the default settings for all algorithms or for specific ones. You can change the MLSPL.conf file within the MLTK itself, under the tab called Settings. The file settings can also still be changed through the command line interface. This document covers both methods.

The MLSPL.conf file default settings are set to prevent the overloading of a search head. Users are encouraged to understand these MLSPL.conf file settings before making any changes.

Machine learning requires compute resources and disk space. Each algorithm has a different performance cost, complicated by the number of selected input fields and events processed. Ensure you know the impact of making changes to the algorithm settings by adding the ML-SPL Performance App for the Machine Learning Toolkit to your setup via Splunkbase.

Configure through the Settings tab

Configure the fit and apply commands by setting properties in the mlspl.conf configuration file via the Settings tab within the Machine Learning Toolkit.

1. Select the name of the algorithm of which you wish to view or alter settings.

   The Edit Default Settings button option changes the master list of settings, rather than those of any individual algorithm. Use this option judiciously.
2. From the specific algorithm page, adjust the available fields as desired. See the table below for setting details, or hover over any setting name to view more information from within the toolkit.
3. Click the green **Save** button when done.

Repeat these steps for any of the other listed algorithms.

**Setting descriptions**

<table>
<thead>
<tr>
<th>Setting</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>handle_new_cat</td>
<td>default</td>
<td>Action to perform when new value(s) for categorical variable/explanatory variable is encountered in <code>partial_fit</code>. Default sets all values of the column that correspond to the new categorical value to zeroes. Skip skips over rows that contain the new value(s) and raises a warning. Stop stops the operation by raising an error.</td>
</tr>
<tr>
<td>max_distinct_cat_values</td>
<td>100</td>
<td>The maximum number of distinct values in a categorical feature field, or input field, that will be used in one-hot encoding. One-hot encoding is when you convert categorical values to numeric values. If the number of distinct values exceeds this limit, the field will be dropped, or excluded from analysis, and a warning appears.</td>
</tr>
<tr>
<td>max_distinct_cat_values_for_classifiers</td>
<td>100</td>
<td>The maximum number of distinct values in a categorical field that is the target, or output,</td>
</tr>
<tr>
<td>Setting</td>
<td>Default</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------------------------------</td>
<td>---------</td>
<td>-----------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>max_distinct_cat_values_for_scoring</td>
<td>100</td>
<td>Determines the upper limit for the number of distinct values in a categorical field that is the target (or response) variable in a scoring method. If the number of distinct values exceeds this limit, the field will be dropped (with an appropriate error message).</td>
</tr>
<tr>
<td>max_fit_time</td>
<td>600</td>
<td>The maximum time, in seconds, to spend in the “fit” phase of an algorithm. This setting does not relate to the other phases of a search such as retrieving events from an index.</td>
</tr>
<tr>
<td>max_inputs</td>
<td>100000</td>
<td>The maximum number of events an algorithm considers when fitting a model. If this limit is exceeded and use_sampling is true, the fit command downsamples its input using the Reservoir Sampling algorithm before fitting a model. If use_sampling is false and this limit is exceeded, the fit command throws an error.</td>
</tr>
<tr>
<td>max_memory_usage_mb</td>
<td>1000</td>
<td>The maximum allowed memory usage, in megabytes, by the fit command.</td>
</tr>
<tr>
<td>Setting</td>
<td>Default</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------</td>
<td>---------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>max_model_size_mb</td>
<td>15</td>
<td>The maximum allowed size of a model, in megabytes, created by the fit command. Some algorithms (e.g. SVM and RandomForest) might create unusually large models, which can lead to performance problems with bundle replication.</td>
</tr>
<tr>
<td>max_score_time</td>
<td>600</td>
<td>The maximum time, in seconds, to spend in the &quot;score&quot; phase of an algorithm.</td>
</tr>
<tr>
<td>streaming_apply</td>
<td>false</td>
<td>Setting streaming_apply to true allows the execution of apply command at indexer level. Otherwise apply is done on search head.</td>
</tr>
<tr>
<td>use_sampling</td>
<td>true</td>
<td>Indicates whether to use Reservoir Sampling for data sets that exceed max_inputs or to instead throw an error.</td>
</tr>
</tbody>
</table>

A reboot of Splunk is required in order to put any setting changes into effect.

**Configure using the command line interface**

Configure the fit and apply commands by setting properties in the mlspl.conf configuration file located in the default directory:

$SPLUNK_HOME/etc/apps/Splunk_ML_Toolkit/default/mlspl.conf

In this file, you can specify default settings for all algorithms, or for an individual algorithm. To apply global settings, use the [default] stanza and algorithm-specific settings in a stanza named for the algorithm, for example,
[LinearRegression] for the LinearRegression algorithm. Be aware that not all global settings can be set or overwritten in an algorithm-specific section. For details, see How to copy and edit a configuration file.

To avoid losing your configuration file changes when you upgrade the app, create a copy of the mlspl.conf file with only the modified stanzas and settings, then save it to $SPLUNK_HOME/etc/apps/Splunk_ML_Toolkit/local/

**Setting descriptions**

<table>
<thead>
<tr>
<th>Setting</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>max_inputs</td>
<td>100000</td>
<td>The maximum number of events an algorithm considers when fitting a model. If this limit is exceeded and <code>use_sampling</code> is true, the <code>fit</code> command downsamples its input using the Reservoir Sampling algorithm before fitting a model. If <code>use_sampling</code> is false and this limit is exceeded, the <code>fit</code> command throws an error.</td>
</tr>
<tr>
<td>use_sampling</td>
<td>true</td>
<td>Indicates whether to use Reservoir Sampling for data sets that exceed <code>max_inputs</code> or to instead throw an error.</td>
</tr>
<tr>
<td>max_fit_time</td>
<td>600</td>
<td>The maximum time, in seconds, to spend in the &quot;fit&quot; phase of an algorithm. This setting does not relate to the other phases of a search such as retrieving events from an index.</td>
</tr>
<tr>
<td>max_memory_usage_mb</td>
<td>1000</td>
<td>The maximum allowed memory usage, in</td>
</tr>
<tr>
<td>Setting</td>
<td>Default</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------------------</td>
<td>---------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>max_model_size_mb</td>
<td>15</td>
<td>The maximum allowed size of a model, in megabytes, created by the <code>fit</code> command. Some algorithms (e.g. SVM and RandomForest) might create unusually large models, which can lead to performance problems with bundle replication.</td>
</tr>
<tr>
<td>max_distinct_cat_values</td>
<td>100</td>
<td>The maximum number of distinct values in a categorical feature field, or input field, that will be used in one-hot encoding. One-hot encoding is when you convert categorical values to numeric values. If the number of distinct values exceeds this limit, the field will be dropped, or excluded from analysis, and a warning appears.</td>
</tr>
<tr>
<td>max_distinct_cat_values_for_classifiers</td>
<td>100</td>
<td>The maximum number of distinct values in a categorical field that is the target, or output, variable in a classifier algorithm.</td>
</tr>
</tbody>
</table>
Using the score command

In the Splunk Machine Learning Toolkit (MLTK), the `score` command runs statistical tests to validate model outcomes. You can use the `score` command for robust model validation and statistical tests in any use case.

The `score` command is only available on versions 4.0.0 or above of the MLTK. You need version 1.3 of the Python for Scientific Computing add-on for version 4.0 or above of the MLTK. For more on version dependencies, see Upgrading the MLTK.

The MLTK uses the following classes of the `score` command, each with their own sets of methods:

- Classification
- Clustering
- Pairwise distances scoring
- Regression scoring
- Statistical functions (statsfunctions)
- Statistical testing (statstest)

The Splunk Machine Learning Toolkit also enables the examination of how well your model might generalize on unseen data by using folds of the training set. This method is known as k-fold scoring. The `kfold` command does not use the `score` command, but operates as a type of scoring.

Score commands cannot be customized within the Splunk Machine Learning Toolkit.

Classification

You can use classification scoring metrics to evaluate the predictive power of a classification learning algorithm.

Classification scoring in the Splunk Machine Learning Toolkit includes the following methods:

- Accuracy
- Confusion matrix
- F1-score
- Precision
- Precision-Recall-F1-Support
Overview

The most common use of classification scoring is to evaluate how well a classification model performs on the test set. The inputs to the classification scoring methods are actual and predicted fields, corresponding to ground-truth-labels and predicted-labels, respectively. The syntax also supports the comparison of multiple fields, allowing for multi-field comparisons. This is useful for evaluating which classification model is best suited for your data.

Classification scoring methods only work on categorical data such as integers and string-types, but not on floats. These methods are used to evaluate the output of classification algorithms, such as logistic regression. You may see an error message if you attempt to use the comparison scoring method on numeric float-type data.

Preprocessing

All classification scoring methods follow the same preprocessing steps:

1. Search commands are pulled into memory.
2. The data is prepared:
   1. All rows containing NAN values are removed prior to computing the score.
   2. You may receive an error if any categorical fields are found.

Parameters

- The `pos_label` parameter must be an element of all actual or ground truth data.
- An error will display if the a valid value for `pos_label` is not found in an actual field.
- Use the `pos_label` parameter to specify the positive class when `average=binary`.
- The `pos_label` parameter is ignored if the average is not binary.
- The `average` parameter includes several options including None, Binary, Micro, Macro and Weighted.

<table>
<thead>
<tr>
<th>Parameter option for</th>
<th>Use cases</th>
</tr>
</thead>
</table>

70
<table>
<thead>
<tr>
<th>average</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td>Returns the scoring metric for each unique class in the union of actual_field + predicted_field.</td>
</tr>
<tr>
<td>Binary</td>
<td>Reports results for the class specified by the pos_label parameter. This parameter only works for binary data and will display an error if applied to a multiclass problem.</td>
</tr>
<tr>
<td>Micro</td>
<td>Calculates metrics globally by counting the total true-positives, false-negatives, and false-positives.</td>
</tr>
<tr>
<td>Macro</td>
<td>Calculates metrics for each label and finds their unweighted mean. Does not take label imbalance into account.</td>
</tr>
<tr>
<td>Weighted</td>
<td>Calculates metrics for each label and finds their average weighted by support as in the number of true instances for each label. This alters the Macro to account for label imbalance and can result in an F-score that is not between Precision and Recall.</td>
</tr>
</tbody>
</table>

**Syntax**

As with all scoring methods, classification methods support pairwise comparisons between two sets of fields or arrays. The general syntax is as follows:

```
.. | score <scoring-method-name> array_a against array_b [options]
```

- The `against` parameter separates the ground-truth fields (on the left) from the predicted fields. `~` is equivalent.
- `array_a` represents the ground-truth fields, and is specified by fields `actual_field_1 ... actual_field_n`
- `array_b` represents the predicted fields, and is specified by fields `predicted_field_1 ... bpredicted_field_n`

**SPL syntax**

```
.. | score <scoring-method-name> <actual_field_1> ... <actual_field_n> against <predicted_field_1> ... <predicted_field_n> [options]
```

**Syntax constraints**

Classification scoring supports the wildcard (*) character in cases of 1-to-n only.

**Examples**
The following example shows the loaded data split into training (<=70 partitions) and testing (>70 partitions) sets. Classification scoring is used, and the model saved as a knowledge object.

The training set is selected, and the model is applied to get predictions on unseen data, perform scoring, and analysis of the results.

The following syntax example is training multiple models on the same field.

```plaintext
| inputlookup iris.csv
| sample partitions=100 seed=1234 search partition_number <= 70
| fit LogisticRegression species from * into LR_model
| fit RandomForestClassifier species from * into RF_model
| fit GaussianNB species from * into GNB_model
| fit DecisionTreeClassifier species from * into DT_model
```

The following syntax example is evaluating the ground truth field against multiple predictions.

```plaintext
| inputlookup iris.csv
| sample partitions=100 seed=1234 search partition_number > 70
| apply LR_model as LR_species
```
| apply RF_model as RF_species |
| apply GNB_model as GNB_species |
| apply DT_model as DT_species |

| score precision_recall_fscore_support species ~ LR_species RF_species GNB_species DT_species average=weighted |

The following visualization shows the evaluation of the ground truth field against multiple predictions.

**Accuracy scoring**

You can use accuracy scoring to get the prediction accuracy between actual-labels and predicted-labels.

Accuracy scoring implements `sklearn.metrics.accuracy_score`. Learn more here: http://scikit-learn.org/stable/modules/generated/sklearn.metrics.accuracy_score.html

Further reading: https://en.wikipedia.org/wiki/Accuracy_and_precision

**Parameters**

- The `normalize` parameter default is True.
- The `normalize` parameter dictates whether to return the raw count of correctly classified samples (normalize=False) or the fraction of correctly classified samples (normalize=True).
- When the `pos_label` parameter `average=binary` and the combined cardinality of the actual or predicted field is <= 2, the report results for class=pos_label only.

**Syntax**

```python
...|score accuracy_score <actual_field_1> ... <actual_field_n> against <predicted_field_1> ... <predicted_field_n> normalize=<True|False>
```

**Syntax constraints**

- Accuracy scoring supports 1-to-1, n-to-n, and 1-to-n comparison syntaxes.
- Accuracy scoring supports the wildcard (*) character in cases of 1-to-n only.

**Example**
You manually specify fields because a predicted field exists in the data. In particular, manually specify fields for second call to the `fit` command and onwards.

```plaintext
| inputlookup track_day.csv
| sample partitions=100 seed=1234
| search partition_number <= 70

| fit LogisticRegression vehicleType from batteryVoltage engineCoolantTemperature engineSpeed into LR_model
| fit DecisionTreeClassifier vehicleType from batteryVoltage engineCoolantTemperature engineSpeed into DT_model
```

After training a classifier to predict vehicle type, you can analyze your test set accuracy.

```plaintext
| inputlookup track_day.csv
| sample partitions=100 seed=1234
| search partition_number > 70

| apply LR_model as LR_prediction
| apply DT_model as DT_prediction
| score accuracy_score vehicleType against LR_prediction DT_prediction
```

**Example output**

![Confusion matrix example](image)

**Confusion matrix**

You can use a confusion matrix to get the prediction accuracy between actual-labels and predicted-labels.


**Parameters**

The confusion matrix takes no parameters.
Syntax

score confusion_matrix <actual_field> against <predicted_field>

Syntax constraints

- The ground-truth-labels map along the vertical event axis, and the predicted-labels map along the horizontal field-axis.
- Works only for 1-1 comparisons, because the output of confusion_matrix is already 2d.
- Confusion matrix scoring does not support the wildcard (*) character.

Although order is not preserved in the output fields and events, the correspondence of fields and events is preserved.

Example

The following example uses a confusion matrix to test actual vehicle type against predicted vehicle type.

```
| inputlookup track_day.csv
| sample partitions=100 seed=1234
| search partition_number > 70
| apply DT_model as DT_prediction
| score confusion_matrix vehicleType against DT_prediction
```

Example output

The following visualization of the confusion matrix shows which classes were most and least successfully predicted, as well as what they were mistaken for.
**F1-score**

You can use the F1-score to get the prediction accuracy between true-labels and predicted-labels.

Implements sklearn.metrics.f1_score. Learn more here: http://scikit-learn.org/stable/modules/generated/sklearn.metrics.f1_score.html

Further reading: https://en.wikipedia.org/wiki/F1_score

**Parameters**

- The `pos_label` parameter default is 1.
- When the `pos_label` parameter `average=binary` and the combined cardinality of the actual or predicted field is <= 2, the report results for `class=pos_label` only.
- The `average` parameter default is binary.

**Syntax**

```
|score f1_score <actual_field_1> ... <actual_field_n> against <predicted_field_1> ... <predicted_field_n> average=<binary(default) | micro | macro | weighted> pos_label=<str | int>
```

**Syntax constraints**

- F1-score supports 1-to-1, n-to-n, and 1-to-n comparison syntaxes.
- F1-score supports the wildcard (*) character in cases of 1-to-n only.

**Example**

The following example tests the prediction of vehicle type using F1-score.

```
| inputlookup track_day.csv
| sample partitions=100 seed=1234
| search partition_number <= 70
| fit LogisticRegression vehicleType from batteryVoltage engineCoolantTemperature engineSpeed into LR_model
| fit DecisionTreeClassifier vehicleType from batteryVoltage engineCoolantTemperature engineSpeed into DT_model
```

After training a classifier to predict vehicle type, you can evaluate your model's precision on the training set for each vehicle type.
inputlookup track_day.csv

sample partitions=100 seed=1234
search partition_number > 70
apply LR_model as LR_prediction
apply DT_model as DT_prediction

score f1_score vehicleType against LR_prediction DT_prediction
average=micro

Example output

The following visualization shows the F1-score model on a test set for each vehicle type with LogisticRegression results on the left and DecisionTree results on the right. The visualization also shows the average across all vehicle types.

**Precision**

You can use precision scoring to get the prediction accuracy between actual-labels and predicted-labels.

Implements sklearn.metrics.precision_score. Learn more here: http://scikit-learn.org/stable/modules/generated/sklearn.metrics.precision_score.html

Further reading: https://en.wikipedia.org/wiki/Accuracy_and_precision

**Parameters**

- The `pos_label` parameter default is 1.
- When the `pos_label` parameter average=binary and the combined cardinality of the actual or predicted field is <= 2, the report results for class=pos_label only.
- The `average` parameter default is binary.
Syntax

```
... | score precision_score <actual_field_1> ... <actual_field_n> against <predicted_field_1> ... <predicted_field_n>
average=<binary(default)|micro|macro|weighted> pos_label=<str|int>
```

Syntax constraints

- Precision scoring supports 1-to-1, n-to-n and 1-to-n. comparison syntaxes.
- Precision scoring supports the wildcard (*) character in cases of 1-to-n only.

Example

The following example tests the prediction of vehicle type using precision scoring.

```
| inputlookup track_day.csv

| sample partitions=100 seed=1234
| search partition_number <= 70

| fit LogisticRegression vehicleType from batteryVoltage engineCoolantTemperature engineSpeed into LR_model
| fit DecisionTreeClassifier vehicleType from batteryVoltage engineCoolantTemperature engineSpeed into DT_model
```

After training a classifier to predict vehicle type, you can evaluate the model's precision on the training set for each vehicle type.

```
| inputlookup track_day.csv

| sample partitions=100 seed=1234
| search partition_number > 70

| apply LR_model as LR_prediction
| apply DT_model as DT_prediction

| score precision_score vehicleType against LR_prediction DT_prediction
average=None
```

Example output

The following visualization shows the precision model on a test set for each vehicle type with LogisticRegression results on the left and DecisionTree results on the right. A warning shows that rows containing NAN values and have been removed.
**Precision-Recall-F1-Support**

You can use Precision-Recall-F1-Support scoring to get the precision, recall, F1-score and support prediction accuracy between actual-fields and predicted-fields.


**Parameters**

- The `pos_label` parameter default is 1.
- When the `pos_label` parameter `average=binary` and the combined cardinality of the actual or predicted field is <= 2, the report results for class=pos_label only.
- The `average` parameter default is None.
- The `beta` parameter default is 1.0.
- The `beta` parameter shows the strength of recall versus precision in f-score.

**Syntax**

```
score precision_recall_fscore_support <actual_field_1> ... <actual_field_n> against <predicted_field_1> ... <predicted_field_n>
pos_label=<str> average=<str> beta=<float>
```

**Syntax constraints**

You can refer to the following table to distinguish your results when average=None and when average=Not None.

<table>
<thead>
<tr>
<th>Average</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

79
<table>
<thead>
<tr>
<th>Average</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>Not None</td>
<td>Works for all syntax constraints including 1-to-1 and 1-to-n.</td>
</tr>
<tr>
<td>None</td>
<td>Only works for 1-1 comparisons because the output of \texttt{precision_recall_f_score_support} is already 2d. Support scoring is only defined when \texttt{average=None} because averaged values are not generated for support.</td>
</tr>
</tbody>
</table>

Precision-Recall-F1-Support supports the wildcard (*) character in cases of 1-to-n only.

**Example**

The following example tests the prediction of vehicle type using Precision-Recall-F1-Support scoring.

```plaintext
| inputlookup track_day.csv |
| sample partitions=100 seed=1234 |
| search partition_number <= 70 |
| fit LogisticRegression vehicleType from batteryVoltage engineCoolantTemperature engineSpeed into LR_model |
| fit DecisionTreeClassifier vehicleType from batteryVoltage engineCoolantTemperature engineSpeed into DT_model |

After training a classifier to predict vehicle type, you can evaluate your model's precision on the training set for each vehicle type.

```plaintext
| inputlookup track_day.csv |
| sample partitions=100 seed=1234 |
| search partition_number > 70 |
| apply LR_model as LR\_prediction |
| apply DT_model as DT\_prediction |
| score precision\_recall\_f\_score\_support vehicleType against LR\_prediction DT\_prediction average=weighted |
```

**Example output**

The following visualization shows the precision, recall, and f_beta scores for the prediction of vehicle type, under a weighted averaging scheme.
Recall

You can use recall scoring to get the prediction accuracy between actual-labels and predicted-labels.


Further reading: https://en.wikipedia.org/wiki/Precision_and_recall

Parameters

- The pos_label parameter default is 1.
- When the pos_label parameter average=binary and the combined cardinality of the actual or predicted field is <= 2, the report results for class=pos_label only.
- The average parameter default is binary.

Syntax

|score recall <actual_field_1> ... <actual_field_n> against <predicted_field_1> ... <predicted_field_n> average=<binary(default) | micro | macro | weighted> pos_label=<str | int>

Syntax constraints

- Recall supports 1-to-1, n-to-n, and 1-to-n comparison syntaxes.
- Recall partially supports the wildcard (*) character in cases of 1-to-n only.

Example

The following example tests the prediction of vehicle type using recall scoring.
| inputlookup track_day.csv
| sample partitions=100 seed=1234
| search partition_number <= 70

| fit LogisticRegression vehicleType from batteryVoltage engineCoolantTemperature engineSpeed into LR_model
| fit DecisionTreeClassifier vehicleType from batteryVoltage engineCoolantTemperature engineSpeed into DT_model

After training a classifier to predict vehicle type you can evaluate your model's precision on the training set for each vehicle type.

| inputlookup track_day.csv
| sample partitions=100 seed=1234
| search partition_number > 70

| apply LR_model as LR_prediction
| apply DT_model as DT_prediction

| score recall_score vehicleType against LR_prediction DT_prediction average=weighted

**Example output**

The following visualization shows the recall model on a test set with LogisticRegression results on the left and DecisionTree results on the right. The visualization shows the average across all vehicle types where the average is weighted by the support.
**ROC-AUC-score**

You can use ROC-AUC scoring to get the prediction accuracy between actual-labels and predicted-scores.


Further reading: https://en.wikipedia.org/wiki/Receiver_operating_characteristic

**Parameters**

- Although sklearn.metrics.roc_auc_score supports an average parameter, this parameter is currently disabled as the toolkit does not support the label-indicator format.
- The pos_label parameter default is 1.
- You can use the pos_label label when data is multi class but you want to apply binary scoring methods. The pos_label label allows multiclass data to cast to binary by specifying the class identified as pos_label as the positive class, and all other classes as negative. For the original multiclass data of a, b, c, a, e when pos_label=a the resulting binary data is 1, 0, 0, 1, 0.
- When the predicted field contains target scores, that field can either be probability estimates of the positive class, confidence values, or a non-thresholded measure of decisions.

**Requirements**

- ROC-AUC-score only applies to binary data. To support multi-class problems, binarize the data using the pos_label parameter.
- The predicted field must be numeric. The numeric data must be float or integer type, corresponding to probability estimates of the positive class, confidence values, or a non-thresholded measure of decisions as returned by the decision_function parameter on some classifiers.
- If the predicted field does not meet the numeric criteria, an error message will display.

<table>
<thead>
<tr>
<th>If</th>
<th>Then</th>
</tr>
</thead>
<tbody>
<tr>
<td>Binary data is given</td>
<td>The data must be true binary such as {0,1} or {-1,1}.</td>
</tr>
<tr>
<td>Binary is not data such as multiclass</td>
<td>The pos_label parameter must be specified and contained in the ground_truth field.</td>
</tr>
<tr>
<td>If</td>
<td>Then</td>
</tr>
<tr>
<td>-----------------------------------------</td>
<td>----------------------------------------------------------------------</td>
</tr>
<tr>
<td>Binary is not true binary</td>
<td>The pos_label parameter must be specified and contained in the ground_truth field.</td>
</tr>
</tbody>
</table>

If the pos_label parameter is not in the ground_truth field, an error message will display.

If the ground truth data is multiclass and the pos_label parameter is properly specified, you may see an error message.

Syntax

```
score roc_auc_score <actual_field_1> ... <actual_field_n> against <predicted_field_1> ... <predicted_field_n> pos_label=<str | int>
```

Syntax constraints

- ROC-AUC-score supports 1-to-1, n-to-n, and 1-to-n comparison syntaxes.
- ROC-AUC-score does not support the wildcard (*) character.

Example

The following example shows how you can obtain the area under the ROC curve for predicting the vehicle type of 2013 Audi RS5.

```
| inputlookup track_day.csv
| sample partitions=100 seed=1234
| search partition_number <= 70

| fit LogisticRegression vehicleType from * probabilities=True into LR_model
| inputlookup track_day.csv
| sample partitions=100 seed=1234
| search partition_number > 70

| score roc_auc_score vehicleType against "probability(vehicleType=2013 Audi RS5)" pos_label="2013 Audi RS5"
```

Example output

The following visualization shows the results of the ROC-AUC scoring on a test set.
ROC-curve

You can use ROC-curve scoring to get the prediction accuracy between actual-fields and predicted-fields.


Further reading: https://en.wikipedia.org/wiki/Receiver_operating_characteristic

Parameters

- The pos_label parameter default is 1.
- You can use the pos_label label when data is multiclass but you want to apply binary scoring methods. The pos_label label allows multiclass data to cast to binary by specifying the class identified as pos_label as the positive class, and all other classes as negative. The original multiclass data of a, b, c, a, e when pos_label=a results in the binary data of 1, 0, 0, 1, 0.
- The drop_intermediate parameter default is True. Whether to drop suboptimal thresholds which would not appear on a plotted ROC-curve. This is useful when creating lighter ROC curves.

Requirements

- ROC-curve only applies to binary data. To support multiclass problems, convert the data into binary with the pos_label parameter.
- The predicted field must be numeric. The numeric data must be float or integer type, corresponding to probability estimates of the positive class, confidence values, or non-thresholded measure of decisions.
- If the predicted field does not meet the numeric criteria, an error message will display.

<table>
<thead>
<tr>
<th>If</th>
<th>Then</th>
</tr>
</thead>
<tbody>
<tr>
<td>Binary data is given</td>
<td>It must be true-binary such as {0,1} or {-1,1}.</td>
</tr>
</tbody>
</table>
**If** | **Then**
---|---
Binary is not data such as multiclass | The `pos_label` parameter must be specified and contained in the `ground_truth` field.
Binary is not true binary | The `pos_label` parameter must be specified and contained in the `ground_truth` field.

If the `pos_label` parameter is not in the `ground_truth` field, an error message displays.

If the ground truth data is multiclass and the `pos_label` is properly specified, you are warned of the conversion.

**Syntax**

```
score roc_curve <actual_field> against <predicted_field>
pos_label=<str|int> drop_intermediate=<True|False>
```

**Syntax constraints**

- ROC-curve scoring only works for 1-1 comparisons.
- ROC-curve scoring does not support the wildcard (*) character.

**Example**

The following example tests the probability of churn using ROC-curve scoring.

```
| inputlookup churn.csv
| sample partitions=100 seed=1234
| search partition_number <= 70
| fit LogisticRegression Churn? from * probabilities=True into LR_model
| inputlookup churn.csv
| sample partitions=100 seed=1234
| search partition_number > 70
| apply LR_model probabilities=True
| score roc_curve Churn? against "probability(Churn?=True.)" pos_label='True.'
```

**Example output**

The following visualization shows how the true positive rate (tpr) varies with the false positive rate (fpr), along with the corresponding probability thresholds.
Clustering scoring

You can use clustering scoring to evaluate the predicted value of a clustering model. The inputs to the clustering scoring methods are arrays of data specified by an ordered sequence of fields.

Clustering scoring in the Splunk Machine Learning Toolkit includes the following methods:

- Silhouette score

Overview

Clustering scoring methods can operate on two arrays. The label and features fields are specified by the ordered sequence of the fields `<label_field>` and `feature_field_1 feature_field_2 ... feature_field_n` respectively.

You can use the `against` clause to separate the arrays where `label_field against feature_field_1 feature_field_2 ... feature_field_n` correspond to label (ground truth or predicted labels) against features (features used by clustering algorithm), respectively.

Clustering scoring methods will only work on numerical data, and are expected to be used to evaluate the output of clustering models such as KMeans and Spectral Clustering. Attempting to score on categorical data will display an error.

Neither parameters that take a list or array as input or metrics that calculate the distance between categorical arrays are supported.

Preprocessing
Clustering scoring methods perform the following preprocessing steps:

1. Search commands are pulled into memory.
2. The data is prepared:
   1. All rows containing NAN values are removed prior to computing the score.
   2. The label field is converted to categorical.
   3. An error message displays if any categorical fields are found in the feature fields.

Parameters

- The metric parameter default is euclidean.
- Supported metric values include: cityblock, cosine, euclidean, l1, l2, manhattan, braycurtis, canberra, chebyshev, correlation, hamming, matching, minkowski, and sqeuclidean.
- The wildcard (*) character is disabled for clustering scoring methods.
- The number of fields in the label_array parameter is limited to one and it must be categorical.

**Silhouette score**

You can use the silhouette score to calculate the prediction accuracy between label_array and feature_array.

Implements sklearn.metrics.silhouette_score. Learn more here: http://scikit-learn.org/stable/modules/generated/sklearn.metrics.silhouette_score.html


Parameters

Silhouette score supports clustering scoring parameters.

Syntax

... | score silhouette_score <label_field> against <feature_field_1> ... <feature_field_n> metric=<euclidean(default) | cityblock | cosine | l1 | l2 | manhattan | braycurtis | canberra | chebyshev | correlation | hamming | matching | minkowski | sqeuclidean>

**Syntax constraints**

Silhouette score supports the following syntax constraints:
• `label_field` parameter must have only single field.
• `feature_fields` parameter can have single or multiple fields.
• Silhouette score supports the wildcard (*) character in cases of 1-to-n only.

**Example**

The following example uses silhouette scoring to calculate species prediction on a test data set.

```
| inputlookup iris.csv
| fit StandardScaler petal_length petal_width sepal_length sepal_width as scaled
| fit KMeans scaled_* k=3 as KMeans_predicted_species
| score silhouette_score KMeans_predicted_species against scaled_petal_length scaled_petal_width scaled_sepal_length scaled_sepal_width
```

**Example output**

The following visualization shows the results of silhouette scoring on the iris dataset.

![Visualization of silhouette scoring on the iris dataset](image)

**Pairwise distances scoring**

You can use pairwise distances scoring to calculate the distances between different fields.

Pairwise distances scoring in the Splunk Machine Learning Toolkit includes the following methods:

• Pairwise distances score

**Overview**

The inputs to the pairwise distances scoring methods are array(s) of data specified by an ordered sequence of fields. The arrays for pairwise distances scoring methods are `a_array` and `b_array`. 
Pairwise distances scoring methods support pairwise comparisons between two sets of fields or arrays. The general syntax is as follows:

```
..| score pairwise_distances a_field_1 a_field_2 ... a_field_n against b_field_1 b_field_2 ... b_field_m
```

The `against` clause separates arrays. The "~" symbol is equivalent.

In general, statistical methods are commutative such that `a_field against b_field` is equivalent to `b_field against a_field`. The arrays `a_array` and `b_array` are specified by a sequence of fields: `a_field_1 ... a_field_n` and `b_field_1 ... b_field_m`.

Pairwise distances scoring methods only work on numerical data. Attempting to score on categorical data will display an error.

**Preprocessing**

All pairwise distance scoring methods follow the same preprocessing steps:

- Search commands are pulled into memory.
- The data is prepared:
  - All rows containing NAN values are removed prior to computing the score.
  - An error message displays if any categorical fields are found in both arrays.

**Parameters**

- The `metric` parameter default = euclidean.
- Supported metric values include: cityblock, cosine, euclidean, l1, l2, manhattan, braycurtis, canberra, chebyshev, correlation, hamming, matching, minkowski, sqeuclidean, Kolmogorov-Smirnov (2 samples), and Wasserstein distance.
- The `output` parameter default = matrix.
- Pairwise distances scoring supports the wildcard (*) character.
- Supported output values are matrix and list.

Using metric values of Kolmogorov-Smirnov (2 samples) or Wasserstein distance requires running version 1.4 of the Python for Scientific Computing add-on.

Some parameters have not been supported:

- Parameters that take a list or array as input.
metrics that calculate the distance between categorical arrays.

Pairwise distances score

Calculate the pairwise distances score between a_array and b_array.

Implements sklearn.metrics.pairwise.pairwise_distances. Learn more here: http://scikit-learn.org/0.19/modules/generated/sklearn.metrics.pairwise.pairwise_distances.html

Parameters

Pairwise distances score score supports pairwise distances scoring parameters.

- The metric parameter default = euclidean.
- Supported metric values include: cityblock, cosine, euclidean, l1, l2, manhattan, braycurtis, canberra, chebyshev, correlation, hamming, matching, minkowski, sqeuclidean Kolmogorov-Smirnov (2 samples), and Wasserstein distance.
- The output parameter default = matrix.
- Supported output values are matrix and list.
- In cases of event-vs-event distances Pairwise distances support only field-vs-field (column-wise) distances. If you need to calculate the distances between events you can transpose the matrix first and then use the pairwise_distances score command.
- Pairwise distances scoring supports the wildcard (*) character.

Using metric values of Kolmogorov-Smirnov (2 samples) or Wasserstein distance requires running version 1.4 of the Python for Scientific Computing add-on.

Syntax

...|score pairwise_distances <a_field_1> ... <a_field_n> against <b_field_l> ... <b_field_m> metric=<euclidean(default) | cityblock | cosine | l1 | l2 | manhattan | braycurtis | canberra | chebyshev | correlation | hamming | matching | minkowski | sqeuclidean | ks_2samp | wasserstein_distance | list> output=<matrix(default) | list>

Syntax constraints

- a_field can have single or multiple fields with numbers that may be equal to each other, or differ from each other.
- b_field can have single or multiple fields with numbers that may be equal to each other, or differ from each other.

Examples
The following example uses pairwise distances scoring on a test set.

| inputlookup iris.csv | score pairwise_distances petal_length petal_width AGAINST sepal_length sepal_width |

The following visualization shows pairwise distance scoring on a test set.

The following example uses the `output=list` parameter.

| inputlookup iris.csv | score pairwise_distances petal_length petal_width AGAINST sepal_length sepal_width output=list |

The following visualization shows pairwise distance scoring on a test set including the `output=list` parameter.

The following example uses event-vs-event distances on a test set.

| inputlookup iris.csv | table petal* sepal* | transpose 0 |

The following visualization shows event-vs-event distances on a test set.

The following example uses the `output=list` parameter on a test set.

| inputlookup iris.csv | table petal* sepal* | transpose 0 | score pairwise_distances "row 1" "row 2" AGAINST "row 3" "row 4" output=list | fields *_fields pairwise* |
The following visualization shows the output=list parameter on a test set.

Regression scoring

Use regression scoring metrics to evaluate the predictive power of a regression learning algorithm. The most common use of regression scoring is to evaluate how well a regression model performs on the test set.

Regression scoring in the Splunk Machine Learning Toolkit includes the following methods:

- Explained variance score
- Mean absolute error score
- Mean squared error score
- R2 score

The inputs to the regression scoring methods are arrays of data specified by an ordered sequence of fields.

Regression scoring methods can operate on two arrays. The actual and predicted fields are specified by an ordered sequence of fields actual_field_1 .. actual_field_n and predicted_field_1 ... predicted_field_n, respectively.

You can use the against clause to separate the arrays where actual_field_1 ... actual_field_n against predicted_field_1 ... predicted_field_n correspond to to actual (ground truth target values) against predicted (predicted target values), respectively.

These scoring methods only work on numerical data, and are used to evaluate the output of regression algorithms such as Gradient Boosting Regression and Linear Regression.

Attempting to score on categorical data, or having no numerical fields in any of the arrays displays an error.

Preprocessing
Regression scoring methods follow the same preprocessing steps:

1. Search commands are pulled into memory.
2. The data is prepared:
   1. All rows containing NAN values are removed prior to computing the score.
   2. An error message displays if any categorical fields are found in the feature fields.

Parameters

- The `multioutput` parameter default is `raw_values`.
- The `raw_values` parameter returns a full set of regressions scores or errors between each field in `fields_a` and `field_b` respectively.
- The wildcard (*) character is supported in cases of 1-to-n only.
- The number of fields in `actual_fields` and `predicted_fields` must be either equal to each other or one of them must have only one field.
- If one of the arrays has a single field and the other array has multiple fields, the `multioutput` parameter is set to `raw_values`.
- If one of the arrays has a single field and the other array has multiple fields, the regression score is calculated between each field of the array which has multiple fields and the one field of the array that has a single field.
- If the `multioutput` parameter was set to a different value by the user beforehand, an error displays.

Parameters that take a list or array as input are not supported.

**Explained variance score**

You can use explained variance score to calculate the explained variance regression score between predicted and actual fields.

Implements `sklearn.metrics.explained_variance_score`. Learn more here: [http://scikit-learn.org/stable/modules/generated/sklearn.metrics.explained_variance_score.html#sklearn.metrics.explained_variance_score](http://scikit-learn.org/stable/modules/generated/sklearn.metrics.explained_variance_score.html#sklearn.metrics.explained_variance_score)


Parameters:

- The `multioutput` parameter default is `raw_values`.
- The `variance_weighted` parameter is the scores of all outputs averages with the weights of each individual output's variance.
To see each explained variance score compared to the actual score, set the multioutput parameter to `raw_values`.

**Syntax**

```plaintext
... | score explained_variance_score <actual_field_1> ...<actual_field_n> against <predicted_field_1> ... <predicted_field_n> multioutput=<raw_values (default) | uniform_average | variance_weighted>
```

**Syntax constraints**

- Explained variance score supports 1-to-1, n-to-n, and 1-to-n comparison syntaxes.
- Explained variance score supports the wildcard (*) character in 1-to-n cases.

Explained variance score is not symmetric.

**Example**

The following example shows manually specified fields in particular for the second call to the `fit` command and onwards because a predicted field exists in the data.

| inputlookup server_power.csv
| fit LinearRegression ac_power from total-cpu-utilization total-disk-accesses total-disk-utilization as ac_power_LR
| fit RandomForestRegressor ac_power from total-cpu-utilization total-disk-accesses total-disk-utilization as ac_power_RFR
| score explained_variance_score ac_power_LR against ac_power_RFR

To see each explained variance score compared to the actual score, set the `multioutput` parameter to `raw_values`.

**Example output**

The following visualization shows the results of explained variance score on a test set.
Mean absolute error score

You can use mean absolute error scoring to calculate regression loss between actual_fields and predicted_fields.

Implements sklearn.metrics.mean_absolute_error. Learn more here: http://scikit-learn.org/stable/modules/generated/sklearn.metrics.mean_absolute_error.html#sklearn.metrics.mean_absolute_error

Further reading: https://en.wikipedia.org/wiki/Mean_absolute_error

Parameters

Mean absolute error score supports regression scoring parameters.

Syntax

... | score mean_absolute_error <actual_field_1> ... <actual_field_n> against <predicted_field_1> ... <predicted_field_n> multioutput=<raw_values(default) | uniform_average>

Syntax constraints

- Mean absolute error score supports 1-to-1, n-to-n, and 1-to-n comparison syntaxes.
- Mean absolute error score supports the wildcard (*) character in 1-to-n cases.

Example

The following example shows manually specified fields particularly for the second call to the fit command and onwards because a predicted field exists in the data.

To see each mean absolute score compared to the actual score the multioutput parameter must be set to raw_values. If set to another value a warning message displays.

| inputlookup power_plant.csv |
| fit LinearRegression Energy_Output from Temperature Pressure Humidity Vacuum fit_intercept=true as energy_output_LR |
| fit Lasso Energy_Output from Temperature Pressure Humidity Vacuum as energy_output_LASSO |
| score mean_absolute_error Energy_Output against energy_output_LR energy_output_LASSO multioutput=uniform_average |

Example output

The following visualization shows mean absolute error scoring on a test set.
Mean squared error

You can use mean squared error score to calculate regression loss between actual_fields and predicted_fields.

Implements sklearn.metrics.mean_squared_error. Learn more here: http://scikit-learn.org/stable/modules/generated/sklearn.metrics.mean_squared_error.html#sklearn.metrics.mean_squared_error

Further reading: https://en.wikipedia.org/wiki/Mean_squared_error

Parameters

Mean squared error score supports regression scoring parameters.

Syntax

...|score mean_squared_error <actual_field_1> ... <actual_field_n> against <predicted_field_1> ... <predicted_field_n> multioutput=<raw_values (default) | uniform_average>

Syntax constraints

- Mean squared error score supports 1-to-1, n-to-n, and 1-to-n comparison syntaxes.
- Mean squared error score supports the wildcard (*) character in 1-to-n cases.

Example

The following example shows manually specified fields particularly for the second call to the fit command and onwards because a predicted field exists in the data.

| inputlookup power_plant.csv |
| fit LinearRegression Energy_Output from Temperature Pressure Humidity Vacuum fit_intercept=true as energy_output_LR |
| fit Lasso Energy_Output from Temperature Pressure Humidity Vacuum as energy_output_LASSO |
| score mean_squared_error Energy_Output against energy_output_LR energy_output_LASSO multioutput=raw_values |

Example output

97
The following visualization shows mean squared error scoring on a test set.

**R2 score**

You can use this method to calculate the R2 score between actual_fields and predicted_fields.

Implements sklearn.metrics.r2_score. Learn more here: http://scikit-learn.org/stable/modules/generated/sklearn.metrics.r2_score.html#sklearn.metrics.r2_score


**Parameters**

- The `multioutput` parameter default is `raw_values`.
- The `variance_weighted` parameter is the scores of all outputs, averaged with weights of each individual output's variance.
- The `none` parameter acts the same as the `uniform_average` parameter.

**Syntax**

```
...|score r2_score <actual_field_1> ... <actual_field_n> against 
<predicted_field_1> ... <predicted_field_n>
multioutput=<raw_values(default) | uniform_average | variance_weighted | None >
```

**Syntax constraints**

- R2 score supports 1-to-1, n-to-n, and 1-to-n comparison syntaxes.
- R2 score supports the wildcard (*) character in 1-to-n cases.

R2 score is not symmetric.

**Example**

The following example shows manually specified fields particularly for the second call to the `fit` command and onwards because a predicted field exists in the data.
| inputlookup server_power.csv
| fit LinearRegression ac_power from total-cpu-utilization
total-disk-accesses total-disk-utilization into LR_model
| fit RandomForestRegressor ac_power from total-cpu-utilization
total-disk-accesses total-disk-utilization into RFR_model
| fit Lasso ac_power from total-cpu-utilization total-disk-accesses
total-disk-utilization into LASSO_model
| fit DecisionTreeRegressor ac_power from total-cpu-utilization
total-disk-accesses total-disk-utilization into DTR_model

After training several regressors to predict ac_power, you can analyze their predictions compared to the ground truth.

| inputlookup server_power.csv
| apply LR_model as LR_prediction
| apply RFR_model as RFR_prediction
| apply LASSO_model as LASSO_prediction
| apply DTR_model as DTR_prediction
| score r2_score ac_power against LR_prediction RFR_prediction
LASSO_prediction DTR_prediction

Example output

The following visualization shows R2 scoring on a test set.

Statistical functions (statsfunctions)

Statistical functions are general statistical methods that either provide statistical information about data or perform a statistical test on data. A statistic/p-value is not returned.

Statistical functions scoring in the Splunk Machine Learning Toolkit include the following methods:

- Describe
- Moment
- Pearson
- Spearman
- Tmean
- Trim
- Tvar
Preprocessing

All statistical functions scoring methods follow the same preprocessing steps:

1. Search commands are pulled into memory.
2. The data is prepared:
   1. All rows containing NAN values are removed prior to computing the score.
   2. An error message displays if any categorical fields are found in the feature fields.

Parameters

Statistical functions support the wildcard (*) character in single array cases only.

Describe

You can use Describe scoring to compute several descriptive statistics of the passed array.

Implements scipy.stats.describe. Learn more here: https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.describe.html

Parameters

- The ddof parameter default is 1.
- The ddof parameter stands for delta degrees of freedom.
- The ddof parameter is only used for variance.
- If the bias parameter is False, then the skewness and kurtosis calculations are corrected for statistical bias.

Syntax

|score describe <a_field_1> <a_field_2> ... <a_field_n> ddof=<int> bias=<true|false>

Syntax constraints

- Describe scoring supports the wildcard (*) character.
- A single sequence of fields.

Example

The following example uses Describe scoring on a test set.
| inputlookup diabetes.csv
| score describe blood_pressure diabetes_pedigree glucose_concentration

Example output

The following visualization shows Describe scoring on a test set.

---

**Moment**

A Moment is a specific quantitative measure of the shape of a set of points. It is often used to calculate coefficients of skewness and kurtosis due to its close relationship with them.


Further reading: https://en.wikipedia.org/wiki/Moment_(mathematics)

**Parameters**

The `moment` parameter default is 1.

**Syntax**

|score moment  <a_field_1> <a_field_2> ... <a_field_n> moment=<int>

**Syntax constraints**

- Moment scoring supports the wildcard (*) character.
- A sequence of fields.

**Example**

The following example calculates the third Moment of the given data.

| inputlookup diabetes.csv
| score moment blood_pressure diabetes_pedigree glucose_concentration moment=3

**Example output**

The following visualization shows Moment scoring on a test set.
**Pearson**

You can use pearson scoring to calculate a pearson correlation coefficient and the p-value for testing non-correlation.


Further reading: https://en.wikipedia.org/wiki/Pearson_correlation_coefficient

**Parameters**

Pearson scoring has no parameters.

**Syntax**

|score pearsonr <a_field> against <b_field>

**Syntax constraints**

- A pair of fields such as a 1-to-1 comparison.
- Pearson scoring does not support the wildcard (*) character.

**Returns**

Pearson scoring returns the correlation coefficient and the p-value for testing non-correlation.

**Example**

The following example uses Pearson scoring on a test set.

| inputlookup track_day.csv
| score pearsonr engineSpeed against speed

**Example output**

The following visualization shows Pearson scoring on a test set.
**Spearman**

You can use Spearman scoring to calculate the rank-order correlation coefficient and the p-value to test for non-correlation.

**Implements** `scipy.stats.spearmanr`. Learn more here:

Further reading:
https://en.wikipedia.org/wiki/Spearman%27s_rank_correlation_coefficient

**Parameters**

Spearman scoring has no parameters.

**Syntax**

`|score spearmanr <a_field> against <b_field>`

**Syntax constraints**

- A pair of fields such as a 1-to-1 comparison.
- Spearman scoring does not support the wildcard (*) character.

**Returns**

Spearman scoring returns the correlation coefficient and the p-value to test for non-correlation.

**Example**

The following example uses Spearman scoring on a test set.

```
| inputlookup track_day.csv
| score spearmanr engineSpeed against speed
```

**Example output**

The following visualization shows Spearman scoring on a test set.
**Tmean**

The Tmean function finds the arithmetic mean of given values, and ignores values outside the given limits.


Further reading: https://en.wikipedia.org/wiki/Truncated_mean

**Parameters**

- The optional `lower_limit` parameter default is None.
- The `lower_limit` parameter represents the lower bound of data to include. If None, there is no lower bound.
- The optional `upper_limit` parameter default is None.
- The `upper_limit` parameter represents the upper bound of data to include. If None, there is no upper bound.

**Syntax**

```
score tmean <a_field_1> ... <a_field_n> lower_limit=<float|None> upper_limit=<float|None>
```

A global trimmed mean is calculated across all fields.

**Syntax constraints**

- Tmean supports the wildcard (*) character.
- A sequence of fields.

**Returns**

The Tmean function returns a single value representing the trimmed mean of the data as in the mean ignoring samples outside of the given bounds.

**Example**

The following example shows the Tmean function on a test set.
inputlookup diabetes.csv
score tmean blood_pressure diabetes_pedigree glucose_concentration
lower_limit=-1 upper_limit=1

Example output

The following visualization shows the trimmed mean result for the test set.

Trim
The Trim function slices off a proportion of items from both ends of an array.

Implements scipy.stats.trim. Learn more here: https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.trimboth.html

Parameters

- The tail parameter default is both.
- The tail parameter determines whether to cut off data from the left, right or both sides of the distribution.
- In the proportiontocut parameter you must specify float.

Syntax

|score trim <a_field_1> ... <a_field_n> proportiontocut=<float>
tail=<left|right|both>

Syntax constraints

- Trim supports the wildcard (*) character.
- A sequence of fields.

Returns

The Trim function returns a shortened version of the data where the order of the trimmed content is undefined.

Example

The following example uses the Trim function on a test set.
Example output

The following visualization shows the Trim function on a test set.

![](image)

**Tvar**

You can use the Tvar function to compute the sample variance of an array of values, while ignoring values that are outside of given limits.

Implements `scipy.stats.ttest_ind`. Learn more here: https://docs.scipy.org/doc/scipy-0.14.0/reference/generated/scipy.stats.tvar.html

**Parameters**

- The optional `lower_limit` parameter default is None.
- The `lower_limit` parameter represents the lower bound of data to include. If None, there is no lower bound.
- The optional `upper_limit` parameter default is None.
- The `upper_limit` parameter represents the upper bound of data to include. If None, there is no upper bound.

**Syntax**

```
|score tvar <a1> <a2> ... <an>  lower_limit=<float|None> 
upper_limit=<float|None>
```

A global trimmed variance is calculated across all fields.

**Syntax constraints**

- Tvar supports the wildcard (*) character.
- A sequence of fields.

**Returns**

106
The Tvar function returns a single value representing the trimmed variance of the data such as the variance while ignoring samples outside of the given bounds.

Example

The following example uses the Tvar function on a test set.

```
| inputlookup diabetes.csv
| score tvar blood_pressure diabetes_pedigree glucose_concentration lower_limit=-1 upper_limit=1
```

Example output

The following visualization shows the Tvar function on a test set.

Statistical testing (statstest)

Statistical testing (statstest) scoring is used to validate or invalidate a statistical hypothesis. The output of statstest scoring methods is a test-specific statistic and a corresponding p-value.

All statistical-testing methods support the parameter alpha, which indicates the alpha-level or significant-level for the statistical test. The default value is 0.05.

Statistical testing in the Splunk Machine Learning Toolkit includes the following methods:

- Augmented Dickney-Fuller (Adfuller)
- Kolmogorov-Smirnov (KS) test (1 sample)
- Kolmogorov-Smirnov (KS) test (2 samples)
- Kwiatkowski-Phillips-Schmidt-Shin (KPSS)
- Mannwhitneyu
- Normal test
- One-way ANOVA
- T-test (1 sample)
- T-test (2 independent samples)
- T-test (2 related samples)
- Wasserstein distance
- Wilcoxon
Overview

The inputs to the statstest scoring methods are arrays of data specified by an ordered sequence of fields. The arrays for statistical testing methods are referred to as `a_array` and `b_array`.

In general, statistical testing methods are commutative as in `a_field against b_field` being equivalent to `b_field against a_field`. Arrays `array_a` and `array_b` are specified by a sequence of fields: `a_field_1 ... a_field_n` and `b_field_1 ... b_field_n`.

Statstest scoring methods can operate on a single array or two arrays.

<table>
<thead>
<tr>
<th>Array count</th>
<th>Example syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>One</td>
<td>...</td>
</tr>
<tr>
<td>Two</td>
<td>...</td>
</tr>
</tbody>
</table>

Preprocessing

All statistical testing scoring methods follow the same preprocessing steps:

1. Search commands are pulled into memory.
2. The data is prepared:
   1. All rows containing NAN values are removed prior to computing the score.
   2. An error message displays if any categorical fields are found in the feature fields.

For scoring methods requiring 2 arrays, use the `against` clause to separate the arrays. You can use `~` as an equivalent to `against`.

**Augmented Dickey-Fuller (Adfuller)**

You can use the Augmented Dickey-Fuller test to test for a unit root in a univariate process in the presence of serial correlation.

Implements `statsmodels.tsa.stattools.adfuller`. Learn more here: https://www.statsmodels.org/dev/generated/statsmodels.tsa.stattools.adfuller.html

Further reading:
https://en.wikipedia.org/wiki/Augmented_Dickey%E2%80%93 Fuller_test

Parameters
- The maxlag parameter default is 10.
- The maxlag parameter determines the maximum lag included in the test.
- The regression parameter default is c.
- The regression parameter determines the constant and trend order to include in the regression.
  - c: constant only (default)
  - ct: constant and trend
  - ctt: constant, and linear and quadratic trend
  - nc: no constant, no trend
- The autolag parameter default is AIC.
  - If None, then maxlag tags are used.
  - If AIC or BIC, then the number of lags is chosed to minimize the corresponding information criterion.
  - The parameter stat starts with maxlag and drops a lag until the t-statistic on the last lag length is significant using a 5%-sized test.
- The alpha parameter default is 0.05.

Null hypothesis

The null hypothesis of the Augmented Dickey-Fuller is that there is a unit root, with the alternative that there is no unit root.

Syntax

```
|score adfuller <field> autolag=<aic|bic|t-stat|none>
regression=<c|ct|ctt|nc> maxlag=<int> alpha=<float>
```

Syntax constraints

A single field.

Example

The following examples uses Augmented Dickey-Fuller on a test set.

```
| inputlookup app_usage.csv
| score adfuller HR1
```

Example output

The following visualization shows Augmented Dickey-Fuller on a test set.
**Kolmogorov-Smirnov (KS) test (1 sample)**

You can use Kolmogorov-Smirnov (KS) test (1 sample) to test whether the specified field is statistically identical to the specified cumulative distribution function (cdf).

Implements `scipy.stats.kstest`. Learn more here: https://docs.scipy.org/doc/scipy-0.14.0/reference/generated/scipy.stats.kstest.html

Further reading: https://en.wikipedia.org/wiki/Kolmogorov%E2%80%93Smirnov_test

**Parameters**

Each cdf has a required set of parameters that must be specified.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Required information</th>
</tr>
</thead>
<tbody>
<tr>
<td>cdf = chi2</td>
<td>df &lt;int&gt;</td>
</tr>
<tr>
<td></td>
<td>loc &lt;float&gt;</td>
</tr>
<tr>
<td></td>
<td>scale &lt;float&gt;</td>
</tr>
<tr>
<td>cdf = longnorm</td>
<td>s &lt;float&gt;</td>
</tr>
<tr>
<td></td>
<td>loc &lt;float&gt;</td>
</tr>
<tr>
<td></td>
<td>scale &lt;float&gt;</td>
</tr>
<tr>
<td>cdf = norm</td>
<td>loc &lt;float&gt;</td>
</tr>
<tr>
<td></td>
<td>scale &lt;float&gt;</td>
</tr>
</tbody>
</table>

**Null hypothesis**

The sample distribution is identical to the specified distribution (cdf, with cdf parameters).

**Syntax**

```
|score kstest <field> cdf=<norm | lognorm | chi2> <required_cdf_parameters> alpha=<int>
```

All required cdf parameters must be supplied.

**Syntax constraints**

- A single field.
- Kolmogorov-Smirnov (KS) test (1 sample) does not support the wildcard (*) character.

**Example**
The following example uses Kolmogorov-Smirnov (KS) test (1 sample) on a test set.

```
| inputlookup power_plant.csv
| score ktest Humidity cdf=norm loc=65 scale=2
```

**Example output**

The following visualization shows that you can reject the hypothesis that the field `Humidity` is identical to a q-function with mean 65 and standard deviation 2.

**Kolmogorov-Smirnov (KS) test (2 samples)**

Use the Kolmogorov-Smirnov statistic on two samples to test if two independent samples are drawn from the same distribution.

Implements `scipy.stats.ttest_ind`. Learn more here: https://docs.scipy.org/doc/scipy-0.14.0/reference/generated/scipy.stats.ks_2samp.html

Further reading: https://en.wikipedia.org/wiki/Kolmogorov%E2%80%93Smirnov_test#Two-sample%20Kolmogorov%E2%80%93Smirnov

**Parameters**

The `alpha` parameter default is 0.05.

**Null hypothesis**

Kolmogorov-Smirnov (KS) test (2 samples) is a two-sided test for the null hypothesis that two independent samples are drawn from the same continuous distribution.

**Syntax**

```
|score ks_2samp <a_field> against <b_field> alpha=<int>
```

**Syntax constraints**

A single pair of fields or a 1-to-1 comparison.

**Example**
The following example shows the two measurements of the \textit{HR} field are drawn from the same distribution.

```
| inputlookup app_usage.csv
| score ks_2samp HR1 against HR2
```

**Example output**

The following example visualization rejects the null hypothesis that the two samples were drawn from the same distribution.

**Kwiatkowski-Phillips-Schmidt-Shin (KPSS)**

Use the Kwiatkowski-Phillips-Schmidt-Shin (KPSS) test to compute for the null hypothesis that a selected field is level or trend stationary.

Implements \texttt{statsmodels.tsa.stattools.kpss}. Learn more here: https://www.statsmodels.org/dev/generated/statsmodels.tsa.stattools.kpss.html

Further reading: https://en.wikipedia.org/wiki/KPSS_test

**Parameters**

- The \texttt{regression} parameter default is \texttt{c}.
  - The \texttt{regression} parameter indicates the null hypothesis for the KPSS test.
  - The \texttt{c} parameter indicates that the data is stationary around a constant.
  - The \texttt{cf} parameter indicates that the data is stationary around a trend.
- The \texttt{lags} parameter default is None.
  - The \texttt{lags} parameter indicates the number of lags to be used. If None, set to \texttt{int (12 * (n / 100)**(1 / 4))}, where \(n\) is the number of samples.
- The \texttt{alpha} default is 0.05.

**Null hypothesis**

The null hypothesis of the KPSS test is that the selected field (\texttt{field}) is level or trend stationary.
Syntax

|score kpss <field> regression=<c | ct> lags=<int> alpha=<float>

Syntax constraints

A single field.

Example

The following example uses KPSS test on a test set.

| inputlookup app_usage.csv
| score kpss HR1

Example output

The following visualization shows KPSS test on a test set.

MannWhitneyU

You can use MannWhitneyU to test whether a randomly selected value from one sample is less than or greater than a randomly selected value from another sample.

Implements scipy.stats.mannwhitneyu. Learn more here: https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.mannwhitneyu.html

Further reading: https://en.wikipedia.org/wiki/Mann%E2%80%93Whitney_U_test

Parameters

- The use continuity parameter determines whether a continuity correction (1/2) must be taken into account.
- The use continuity parameter default is True.
- The alternative parameter determines whether to get the p-value for the one-sided hypothesis (less or greater) or for the two-sided hypothesis (two-sided).
- The alternative parameter default is two-sided.
- The alpha parameter default is 0.05.
Null hypothesis

MannWhitneyU is a test of the null hypothesis that it is equally likely that a randomly selected value from one sample is less than or greater than a randomly selected value from another sample.

Syntax

|score mannwhitneyu <a_field> against <b_field>
use_continuity=<true|false> alternative=<less|two-sided|greater>
alpha=<int>

Syntax constraints

A single pair of fields or a 1-to-1 comparison.

Example

The following example uses MannWhitneyU on a test set.

| inputlookup churn.csv
| score mannwhitneyu "Day Charge" against "Eve Charge"
alternative=greater

Example output

The following visualization shows that the random sample from Day Charge is likely greater than a random sample in Eve Charge.

Normal-test

you can use Normal-test to test whether a sample differs from a normal distribution.


Further reading:

• D'Agostino, R. B. (1971), "An omnibus test of normality for moderate and large sample size", Biometrika, 58, 341-348

Parameters

The alpha parameter is 0.05.

Null hypothesis

The sample \(<a_field_1>, ..., <a_field_n>\) comes from a normal distribution.

Syntax

...|score normaltest <field_1> <field_2> ... <field_n> alpha=<int>

An array (array_a) is specified by the ordered sequence of fields )<a1>, <a2>,...,<an>).

Syntax constraints

- Normal-test supports the wildcard (*) character.
- A single array or set of fields.

Example

The following example uses Normal-test on a test set.

```
| inputlookup diabetes.csv
| score normaltest BMI age blood_pressure diabetes_pedigree glucose_concentration skin_thickness
| table field pvalue
```

Example output

The following visualization tests whether fields are different from that of a normal distribution. From the plot you can deduce that glucose_concentration is the most likely value to come from a normal distribution.
One-way ANOVA

You can use One-way ANOVA to test the null hypothesis that two or more groups have the same population mean.

Implements scipy.stats.f_oneway. Learn more here: https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.f_oneway.html


Parameters

The alpha parameter default is 0.05.

Null hypothesis

The specified groups field_1, ..., field_n have the same population mean.

Syntax

|score f_oneway <field_1> <field_2> ... <field_n> alpha=<int>

Syntax constraints

- One-way ANOVA supports the wildcard (*) character.
- A single array or set of fields.

Example

The following example uses One-way ANOVA on a test set.

| inputlookup app_usage.csv
| score f_oneway HR1 HR2

Example output
The following visualization shows One-way ANOVA on a test set.

**T-test (1 sample)**

You can use T-test (1 sample) to test whether the expected value (mean) of a sample of independent observations is equal to the specified population mean.

Implements `scipy.stats.ttest_1samp`. Learn more here: https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.ttest_1samp.html

Further reading: http://www.biostathandbook.com/onesamletest.html

**Parameters:**

- The `popmean` parameter has no default, and must be specified.
- The `popmean` parameter represents the population mean in the null hypothesis.
- The `alpha` parameter default is 0.05.

**Null hypothesis**

The expected value (mean) of the specified samples of independent observations (`field_1 ... ,field_n`) are equal to the given population mean (`popmean`).

**Syntax**

```
...|score ttest_1samp <field_1> ... <field_n> popmean=<float> alpha=<int>
```

**Syntax constraints**

- T-test (1 sample) supports the wildcard (*) character.
- A single array or set of fields.

**Example**

The following example tests whether the sample mean differs from an expected population mean.
Example output

The following visualization shows the negative statistic indicating that the sample mean is less than the hypothesized mean of 20.

T-test (2 independent samples)

You can use T-test (2 independent samples) to test whether two independent samples come from the same distribution.


Further reading:
https://en.wikipedia.org/wiki/Student%27s_t-test#Independent_two-sample_t-test

Parameters

- The `equal_var` parameter default is True.
- If the `equal_var` parameter is True, perform a standard independent 2 sample test that assumes equal population variances.
- If the `equal_var` parameter is False, perform Welch’s T-test, which does not assume equal population variance.
- The `alpha` default is 0.05.

Null hypothesis

The null hypothesis is that the pairs `a_field_i` and `b_field_i` (independent) samples have identical average (expected) values. This test assumes that the fields have identical variances by default.

Syntax

```
score ttest_ind <a_field_1> ... <a_field_n> against <b_field_1>...<b_field_n> equal_var=<true|false> alpha=<int>
```

Syntax constraints
• Two arrays specified by two ordered sequences of fields (1-to-1, n-to-n, and 1-to-n comparison syntaxes).
• T-test (2 independent samples) supports the wildcard (*) character in 1-to-n cases.

Example

The following example analyzes disk failures to see if disks are equally likely to fail, or if some disks are more likely to cause failure.

| inputlookup disk_failures.csv | score ttest_ind SMART_1_Raw against SMART_2_Raw SMART_3_Raw SMART_4_Raw |

Disk failures are assumed to be independent across disks.

Example output

The following visualization shows that with an alpha of 0.05 you cannot reject the null hypothesis. It does not appear that disks 2, 3, and 4 are failing more than disk 1. All are close to each other.

T-test (2 related samples)

You can use T-test (2 related samples) to test if two related samples come from the same distribution.

Implements scipy.stats.ttest_ind. Learn more here: https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.ttest_rel.html

Further reading:
https://en.wikipedia.org/wiki/Student%27s_t-test#Dependent%20t-test%20for%20paired%20samples

Parameters

The alpha parameter default is 0.05.

Null hypothesis
The null hypothesis is that the pairs \( a_{\text{field}_i} \) and \( b_{\text{field}_i} \) (related, as in two measurements of the same thing) samples have identical average (expected) values.

**Syntax**

|score ttest_rel <a_field_1> ... <a_field_n> against <b_field_1> ... <b_field_n> alpha=<int>

**Syntax constraints**

- Two arrays specified by two ordered sequences of fields (1-to-1, n-to-n and 1-to-n comparison syntaxes).
- T-test (2 related samples) supports the wildcard (*) character in 1-to-n cases.

**Example**

The following example tests if the two measurements of the HR field taken at the same time are statistically identical or not.

| inputlookup app_usage.csv
| score ttest_rel HR1 against HR2

**Example output**

The following visualization shows that you can reject the null hypothesis and conclude that the two measurements are statistically different, potentially indicating a shift from equilibrium.

**Wasserstein distance**

You can use Wasserstein distance to compute the first wasserstein distance between two one-dimensional distributions.

Using Wasserstein distance scoring requires running version 1.4 of the Python for Scientific Computing add-on.

Null hypothesis

The null hypothesis of the Wasserstein distance is that the `a_field` and `b_field` are probability distributions.

Syntax

`| score wasserstein_distance <a_field> against <b_field>`

Syntax constraints

- A single pair of fields or 1-to-1 comparison.
- Wasserstein distance does not support the wildcard (*) character.

Example

The following example shows the distance between two measurements of the HR field.

`| inputlookup app_usage.csv`  
`| score wasserstein_distance HR1 against HR2`

Example output

The following visualization shows Wasserstein distance on a test set.

Wilcoxon

You can use Wilcoxon to test if two related samples come from the same distribution.

Implements `scipy.stats.wilcoxon`. Learn more here: https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.wilcoxon.html

Further reading: https://en.wikipedia.org/wiki/Wilcoxon_signed-rank_test

Parameters
• The zero-method parameter default is Wilcox.
• Use the pratt parameter to include zero differences in ranking process (more conservative).
• Use the wilcox parameter to discard zero-differences.
• Use the zsplit parameter to split zero ranks between positive and negative.
• The correction parameter default is False.
• If the correction parameter is True, apply continuity correction by adjusting the Wilcoxon rank statistic by 0.5 towards the mean value when computing the z-statistic.
• The alpha parameter default is 0.05.

Null hypothesis

The null hypothesis is that two related paired samples come from the same distribution. In particular, Wilcoxon tests whether the distribution of the differences \( x - y \) is symmetric about zero.

Syntax

```
score wilcoxon <a_field> against <b_field> zero_method=<pratt | wilcox | zsplit> correction=<True | False> alpha=<int>
```

Syntax constraints

A single pair of fields or 1-to-1 comparison.

Example

The following example shows you if the distribution of nighttime minutes used differs from the distribution of evening minutes used.

```
| inputlookup churn.csv
| score wilcoxon "Night Mins" against "Eve Mins"
```

Example output

The following visualization shows the Wilcoxon test on a test set.
**K-fold scoring**

Cross-validation assesses how well a statistical model generalizes on an independent dataset. Cross-validation tells you how well your machine learning model is expected to perform on data that it has not been trained on. The scores obtained from K-fold cross-validation are generally a less biased and less optimistic estimate of the model performance than a standard training and testing split.

There are many types of cross-validation, but K-fold cross-validation (kfold.cv) is one of the most common.

The kfold.cv parameter does not use the score command, but operates like a scoring method.

Cross-validation is typically used for the following machine learning scenarios:

- Comparing two or more algorithms against each other for selecting the best choice on a particular dataset.
- Comparing different choices of hyper-parameters on the same algorithm for choosing the best hyper-parameters for a particular dataset.
- An improved method over a train/test split for quantifying model generalization.

Cross-validation is **not** well suited for time-series charts:

- In situations where the data is ordered such as time-series, cross-validation is not well suited because the training data is shuffled. In these situations, other methods such as Forward Chaining are more suitable.
- The most straightforward implementation is to wrap sklearn’s Time Series Split. Learn more here: https://en.wikipedia.org/wiki/Forward_chaining

With the kfold.cv parameter, the training set is randomly partitioned into k equal-sized subsamples. Then, each sub-sample takes a turn at becoming the validation (test) set, predicted by the other k-1 training sets. Each sample is used exactly once in the validation set, and the variance of the resulting estimate is reduced as k is increased. The disadvantage of the kfold.cv parameter is that k different models have to be trained, leading to long execution times for large datasets and complex models.
You can obtain k performance metrics, one for each training and testing split. These k performance metrics can then be averaged to obtain a single estimate of how well the model generalizes on unseen data.

**Syntax**

The $kfold_cv$ parameter is applicable to all classification and regression algorithms, and you can append the parameter to the end of an SPL search.

Here $kfold_cv=<int>$ specifies that $k=<int>$ folds is used. When you specify a classification algorithm, stratified k-fold is used instead of k-fold. In stratified k-fold, each fold contains approximately the same percentage of samples for each class.

```
.. | fit <classification | regression algo> <targetVariable> from <featureVariables> [options] kfold_cv=<int>
```

The $kfold_cv$ parameter cannot be used when saving a model.

**Output**

The $kfold_cv$ parameter returns performance metrics on each fold using the same model specified in the SPL - including algorithm and hyper parameters. Its only function is to give you insight into how well you model generalizes. It does not perform any model selection or hyper parameter tuning. In this way, the current implementation is seen as a scoring method.

**Examples**
The first example shows the `kfold_cv` parameter used in classification. Where the output is a set of metrics for each fold including accuracy, f1_weighted, precision_weighted, and recall_weighted.

This second example shows the `kfold_cv` parameter used in classification. Where the output is a set of metrics for each the neg_mean_squared_error and r^2 folds.
Algorithms

Algorithms

The Splunk Machine Learning Toolkit supports the algorithms listed here. In addition to the examples included in the Splunk Machine Learning Toolkit, you can find more examples of these algorithms on the scikit-learn website.

The following algorithms use the fit and apply commands within the Splunk Machine Learning Toolkit. For information on the steps taken by these commands, please review the Understanding the fit and apply commands document.

Looking for information on using the score command? Please navigate to the score command documentation for details.

**ML-SPL Quick Reference Guide**


**ML-SPL Performance App**

Download the ML-SPL Performance App for the Machine Learning Toolkit to use performance results for guidance and benchmarking purposes in your own environment.

**Extend the algorithms you can use for your models**

The algorithms listed here and in the ML-SPL Quick Reference Guide are available natively in the Splunk Machine Learning Toolkit. You can also base your algorithm on over 300 open source Python algorithms from scikit-learn, pandas, statsmodel, numpy and scipy libraries available through the Python for Scientific Computing add-on in Splunkbase.

For information on how to import an algorithm from the Python for Scientific Computing add-on into the Splunk Machine Learning Toolkit, see the ML-SPL API Guide.
Add algorithms through GitHub

On-prem customers looking for solutions that fall outside of the 30 native algorithms can use GitHub to add more algorithms. Solve custom uses cases through sharing and reusing algorithms in the Splunk Community for MLTK on GitHub. Here you can also learn about new machine learning algorithms from the Splunk open source community, and help fellow users of the toolkit.

Cloud customers can also use GitHub to add more algorithms via an app. The Splunk GitHub for Machine learning app provides access to custom algorithms and is based on the Machine Learning Toolkit open source repo. Cloud customers need to create a support ticket to have this app installed.

Anomaly Detection

Anomaly detection algorithms detect anomalies and outliers in numerical or categorical fields.

DensityFunction

The DensityFunction algorithm provides a consistent and streamlined workflow to create and store density functions and utilize them for anomaly detection. DensityFunction allows for grouping of the data using a by clause, where for each group a separate density function is fitted and stored.

The DensityFunction algorithm supports the following three continuous probability density functions: Normal, Exponential, and Gaussian Kernel Density Estimation (Gaussian KDE).

Using the DensityFunction algorithm requires running version 1.4 of the Python for Scientific Computing add-on.

The accuracy of the anomaly detection for DensityFunction depends on the quality and the size of the training dataset, how accurately the fitted distribution models the underlying process that generates the data, and the value chosen for the threshold parameter.

Follow these guidelines to make your models perform more accurately:

- Aim for fitted distributions to have a cardinality (training dataset size) of at least 50. If you cannot collect more training data, create fewer groups of data using the by clause, giving you more datapoints per group.
• The threshold parameter has a default value, but ideally the value for threshold, lower_threshold, or upper_threshold are chosen based on experimentation as guided by domain knowledge.
• Continue tuning the threshold parameter until you are satisfied with the results.
• Always inspect the model using the summary command.
• If the distribution of the data changes through time, re-train your models frequently.

Parameters

• Valid values for the dist parameter include: norm (normal distribution), expon (exponential distribution), gaussian_kde (Gaussian KDE distribution), and auto (automatic selection).
• The dist parameter default is auto. When set to auto, norm (normal distribution), expon (exponential distribution), and gaussian_kde (Gaussian KDE distribution) all run, with the best results returned.
• The metric parameter calculates the distance between the sampled dataset from the density function and the training dataset.
• Valid metrics for the metric parameter include: kolmogorov_smirnov and wasserstein.
• The metric parameter default is wasserstein.
• The sample parameter can be used during fit or apply stages.
• The sample parameter default is False.
• If sample parameter is set to True, samples are taken from the fitted density function.
• If sample parameter is set to True, results output in a new column called SampledValue.
• If sample parameter is set to True during the fit stage, the size of the samples will be equal to the training dataset.
• If sample parameter is set to True during the apply stage, the size of the samples will be equal to the testing dataset.
• The cardinality value generated by the summary command represents the number of data points used when fitting the selected density function.
• The distance value generated by the summary command represents the metric type used when calculating the distance as well as the distance between the sampled data points from the density function and the training dataset.
• The mean value is the mean of the density function.
• The value for std represents the standard deviation of the density function.
• A value under other represents any parameters other than mean and std as applicable. In the case of Gaussian KDE, other could show parameter
size or bandwidth.

- The `type` field shows both the chosen density function as well as if the `dist` parameter is set to auto.
- The `show_density` parameter default is `False`. If the parameter is set to `True`, the density of each data point will be provided as output in a new field called `ProbabilityDensity`.
- The output for `ProbabilityDensity` is the probability density of the data point according to the fitted probability density. This output is provided when the `show_density` parameter is set to `True`.
- The `fit` command will fit a probability density function over the data, optionally store the resulting distribution's parameters in a model file, and output the outlier in a new field called `IsOutlier`.
- The output for `IsOutlier` is a list of labels. Number 1 represents outliers, and 0 represents inliers, assigned to each data point. Outliers are detected based on the values set for the `threshold` parameter. Inspect the `IsOutlier` results column to see how well the outlier detection is performing.
- The parameters `threshold`, `lower_threshold`, and `upper_threshold` control the outlier detection process.
- The `threshold` parameter is the center of the outlier detection process. It represents the percentage of the area under the density function and has a value between 0.000000001 (refers to ~0%) and 1 (refers to 100%). The `threshold` parameter guides the `DensityFunction` algorithm to mark outlier areas on the fitted distribution. For example, if `threshold=0.01`, then 1% of the fitted density function will be set as the outlier area.
- The `threshold` parameter default value is 0.01.
- The output for `BoundaryRanges` is the boundary ranges of outliers on the density function which are set according to the values of the `threshold` parameter.
- Each boundary region has three values: boundary opening point, boundary closing point, and percentage of boundary region.
  - The syntax follows the convention of `[[first_boundary_region], [second_boundary_region], ?, [n_th boundary_region]]`.
  - In cases of a single boundary region, the value for the percentage of boundary region is equal to the `threshold` parameter value.
  - In some distributions (for example Gaussian KDE), the sum of outlier areas might not add up to the exact value of `threshold` parameter value, but will be a close approximation.
- `BoundaryRanges` is calculated as an approximation and will be empty in the following two cases:
  - Where the density function has a sharp peak from low standard deviation.
  - When there are a low number of data points.
• Data points that are exactly at the boundary opening or closing point are assigned as inliers. An opening or closing point is determined by the density function in use.
• Normal density function has left and right boundary regions. Data points on the left of the left boundary closing point, and data points on the right of the right boundary opening point are assigned as outliers.
• Exponential density function has one boundary region. Data points on the right of the right boundary opening point are assigned as outliers.
• Gaussian KDE density function can have one or more boundary regions, depending on the number of peaks and dips within the density function. Data points in these boundary regions are assigned as outliers. In cases of boundary regions to the left or right, guidelines from Normal density function apply. As the shape for Gaussian KDE density function can differ from dataset to dataset, you do not consistently observe left and right boundary regions.

Syntax

| fit DensityFunction <field> [by "<field1>,<field2>,....<field5>"] [into <model name>] [dist=<str>] [show_density=true|false] [sample=true|false][threshold=<float>|lower_threshold=<float>|upper_threshold=<float>] [metric=<str>]

You can apply the saved model to new data with the apply command, with the option to update the parameters for threshold, lower_threshold, upper_threshold, and show_density. Parameters for dist and metric cannot be applied at this stage, and any new values provided will be ignored.

apply <model name>

[threshold=<float>|lower_threshold=<float>|upper_threshold=<float>][show_density=true|false][sample=true|false]

You can inspect the model learned by DensityFunction with the summary command.

| summary <model name>

Syntax constraints

• Fields within the by clause must be given in quotation marks.
• The maximum number of fields within the by clause is 5.
• The total number of groups calculated with the by clause can not exceed 1024. In an example clause of by "DayOfWeek,HourOfDay" there are two fields: one for DayOfWeek and one for HourOfDay. As there are seven days in a week, there are seven groups for DayOfWeek. As there are twenty-four hours in a day, there are twenty-four groups for HourOfDay. Meaning the
The total number of groups calculated with the by clause is $7 \times 24 = 168$.

- The limited number of groups prevents model files from growing too large. You can increase the limit by changing the value of `max_groups` in the DensityFunction settings. Larger limits mean larger model files and longer load times when running `apply`.
- Decrease `max_kde_parameter_size` to allow for the increase of `max_groups`. This change keeps model sizes small while allowing for increased groups.

- The parameters `threshold`, `lower_threshold`, and `upper_threshold` must be within the range of $0.00000001$ to $1$.
- If the parameters of `lower_threshold` and `upper_threshold` are both provided, the summation of these parameters must be less than $1$ (100%).
- The `threshold` and `lower_threshold`/`upper_threshold` parameters cannot be specified together.
- Exponential density function and Gaussian KDE density function only support the `threshold`.
- Exponential density function and Gaussian KDE density function do not support `lower_threshold` or `upper_threshold`.
- Normal density function supports either `threshold` or `lower_threshold`/`upper_threshold`.
- The parameters `lower_threshold` and `upper_threshold` can be used with only Normal density function.

**Examples**

The following example shows DensityFunction on a dataset with the `fit` command.

```plaintext
| inputlookup call_center.csv |
| eval _time=strptime(_time, "%Y-%m-%dT%H:%M:%S") |
| bin _time span=15m |
| eval HourOfDay=strftime(_time, "%H") |
| eval BucketMinuteOfHour=strftime(_time, "%M") |
| eval DayOfWeek=strftime(_time, "%A") |
| stats max(count) as Actual by HourOfDay,BucketMinuteOfHour,DayOfWeek,source,_time |
| fit DensityFunction Actual by "HourOfDay,BucketMinuteOfHour,DayOfWeek" into mymodel |
```
The following example shows DensityFunction on a dataset with the `apply` command.

```
| inputlookup call_center.csv
| eval _time=strptime(_time, "%Y-%m-%dT%H:%M:%S")
| bin _time span=15m
| eval HourOfDay=strftime(_time, "%H")
| eval BucketMinuteOfHour=strftime(_time, "%M")
| eval DayOfWeek=strftime(_time, "%A")
| stats max(count) as Actual by HourOfDay,BucketMinuteOfHour,DayOfWeek,source,_time
| apply mymodel show_density=True sample=True
```

The following example shows DensityFunction on a dataset with the `summary` command.

```
| summary mymodel
```
The following example shows **BoundaryRages** on a test set. In this example the threshold is set to 30% (0.3). The first row has a left boundary range which starts at -Infinity and goes up to the number 44.6912. The area of the left boundary range is 15% of the total area under the density function. It has also a right boundary range which starts at a number 518.3088 and goes up to Infinity. Again, the area of the right boundary range is the same as the left boundary range with 15% of the total area under the density function. The areas of right and left boundary ranges add up to the threshold value of 30%. The third row has only one boundary range which starts at number 300.0943 and goes up to Infinity. The area of the boundary range is 30% of the area under the density function.

```
| inputlookup call_center.csv
| eval _time=strptime(_time, "%Y-%m-%dT%H:%M:%S")
| bin _time span=15m
| eval HourOfDay=strftime(_time, "%H")
| eval BucketMinuteOfHour=strftime(_time, "%M")
| eval DayOfWeek=strftime(_time, "%A")
| stats max(count) as Actual by HourOfDay, BucketMinuteOfHour, DayOfWeek, source, _time
| fit DensityFunction Actual by "HourOfDay, BucketMinuteOfHour, DayOfWeek" threshold=0.3 into mymodel
```
**LocalOutlierFactor**

The LocalOutlierFactor algorithm uses the scikit-learn Local Outlier Factor (LOF) to measure the local deviation of density of a given sample with respect to its neighbors. LocalOutlierFactor is an unsupervised outlier detection method. The anomaly score depends on how isolated the object is with respect to its neighbors.

For descriptions of the `n_neighbors`, `leaf_size` and other parameters, see the sci-kit learn documentation: http://scikit-learn.org/stable/modules/generated/sklearn.neighbors.LocalOutlierFactor.html

Using the LocalOutlierFactor algorithm requires running version 1.3 or above of the Python for Scientific Computing add-on.

**Parameters**

- The `anomaly_score` parameter default is True. Disable this default by adding the `False` keyword to the command.
- The `n_neighbors` parameter default is 20
- The `leaf_size` parameter default is 30
- The `p` parameter is limited to $p \geq 1$
- The `contamination` parameter must be within the range of 0.0 (not included) to 0.5 (included)
- The `contamination` parameter default is 0.1
- Options for the `algorithm` parameter include: brute, kd_tree, ball_tree, and auto. The default is auto.
- The brute, kd_tree, ball_tree, and auto `algorithm` options have respective valid metrics. The default `metric` for each is algorithm is minkowski.
  - Valid metrics for brute include: cityblock, euclidean, $l_1$, $l_2$, manhattan, chebyshev, minkowski, braycurtis, canberra, dice, hamming, jaccard, kulsinski, matching, rogerstanimoto, russellrao, sokalmichener, sokalsneath, cosine, correlation, sqeuclidean, and yule.
  - Valid metrics for kd_tree include: cityblock, euclidean, $l_1$, $l_2$, manhattan, chebyshev, and minkowski.
  - Valid metrics for ball_tree include: cityblock, euclidean, $l_1$, $l_2$, manhattan, chebyshev, minkowski, braycurtis, canberra, dice, hamming, jaccard, kulsinski, matching, rogerstanimoto, russellrao, sokalmichener, and sokalsneath.
- The output for LocalOutlierFactor is a list of labels titled `is_outlier`, assigned 1 for outliers, and -1 for inliers.
Syntax

fit LocalOutlierFactor <fields>
[n_neighbors=<int>] [leaf_size=<int>] [p=<int>]
[contamination=<float>]
[metric=<str>] [algorithm=<str>] [anomaly_score=<true|false>]

Syntax constraints

- You cannot save LocalOutlierFactor models using the into keyword. This algorithm does not support saving models.
- LocalOutlierFactor does not include the predict method.

Example

The following example uses LocalOutlierFactor on a test set.

| inputlookup iris.csv | fit LocalOutlierFactor petal_length petal_width n_neighbors=10 algorithm=kd_tree metric=minkowski p=1 contamination=0.14 leaf_size=10

OneClassSVM

The OneClassSVM algorithm uses the scikit-learn OneClassSVM to fit a model from a set of features or fields for detecting anomalies and outliers, where features are expected to contain numerical values. OneClassSVM is an unsupervised outlier detection method.

For further information, see the sci-kit learn documentation:

Parameters

- The kernel parameter specifies the kernel type for using in the algorithm, where the default value is kernel is rbf.
  - Kernel types include: linear, rbf, poly, and sigmoid.
- You can specify the upper bound on the fraction of training error as well as the lower bound of the fraction of support vectors using the nu parameter, where the default value is 0.5.
- The degree parameter is ignored by all kernels except the polynomial kernel, where the default value is 3.
- gamma is the kernel co-efficient that specifies how much influence a single data instance has, where the default value is \( \frac{1}{\text{number of features}} \).
- The independent term of coef0 in the kernel function is only significant if you have polynomial or sigmoid function.
- The term tol is the tolerance for stopping criteria.
The **shrinking** parameter determines whether to use the shrinking heuristic.

### Syntax

```bash
fit OneClassSVM <fields> [into <model name>] [kernel=<str>] [nu=<float>] [coef0=<float>]
[gamma=<float>] [tol=<float>] [degree=<int>] [shrinking=<true|false>]
```

- You can save OneClassSVM models using the `into` keyword.
- You can apply the saved model later to new data with the `apply` command.

### Syntax constraints

- After running the `fit` or `apply` command, a new field named `isNormal` is generated. This field defines whether a particular record (row) is normal (`isNormal=1`) or anomalous (`isNormal=-1`).
- You cannot inspect the model learned by OneClassSVM with the `summary` command.

### Example

The following example uses OneClassSVM on a test set.

```bash
... | fit OneClassSVM * kernel="poly" nu=0.5 coef0=0.5 gamma=0.5 tol=1
degree=3 shrinking=f into
TESTMODEL_OneClassSVM
```

### Classifiers

Classifier algorithms predict the value of a categorical field.

The `kfold` cross-validation command can be used with all Classifier algorithms. [Learn more here.](#)

#### BernoulliNB

The BernoulliNB algorithm uses the scikit-learn BernoulliNB estimator to fit a model to predict the value of categorical fields where explanatory variables are assumed to be binary-valued. BernoulliNB is an implementation of the Naive Bayes classification algorithm. This algorithm supports incremental fit.

### Parameters
The alpha parameter controls Laplace/ Lidstone smoothing. The default value is 1.0.

The binarize parameter is a threshold that can be used for converting numeric field values to the binary values expected by BernoulliNB. The default value is 0.

- If binarize=0 is specified, the default, values > 0 are assumed to be 1, and values <= 0 are assumed to be 0.

The fit_prior Boolean parameter specifies whether to learn class prior probabilities. The default value is True. If fit_prior=f is specified, classes are assumed to have uniform popularity.

Syntax

```
fit BernoulliNB <field_to_predict> from <explanatory_fields> [into <model name>] [alpha=<float>] [binarize=<float>] [fit_prior=<true|false>] [partial_fit=<true|false>]
```

You can save BernoulliNB models using the into keyword and apply the saved model later to new data using the apply command.

```
... | apply TESTMODEL_BernoulliNB
```

You can inspect the model learned by BernoulliNB with the summary command as well as view the class and log probability information as calculated by the dataset.

```
... | summary My_Incremental_Model
```

Syntax constraints

- The partial_fit parameter controls whether an existing model should be incrementally updated or not. The default value is False, meaning it will not be incrementally updated. Choosing partial_fit=True allows you to update an existing model using only new data without having to retrain it on the full training data set.
- Using partial_fit=True on an existing model ignores the newly supplied parameters. The parameters supplied at model creation are used instead. If partial_fit=False or partial_fit is not specified (default is False), the model specified is created and replaces the pre-trained model if one exists.
- If My_Incremental_Model does not exist, the command saves the model data under the model name My_Incremental_Model. If My_Incremental_Model exists and was trained using BernoulliNB, the command updates the existing model with the new input.
My_Incremental_Model exists but was not trained by BernoulliNB, an error message displays.

Example

The following example uses BernoulliNB on a test set.

```plaintext
... | fit BernoulliNB type from * into TESTMODEL_BernoulliNB alpha=0.5 binarize=0 fit_prior=f
```

**DecisionTreeClassifier**

The DecisionTreeClassifier algorithm uses the scikit-learn DecisionTreeClassifier estimator to fit a model to predict the value of categorical fields. For further information, see the sci-kit learn documentation: [http://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html](http://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html).

**Parameters**

To specify the maximum depth of the tree to summarize, use the `limit` argument. The default value for the `limit` argument is 5.

```plaintext
... | summary model_DTC limit=10
```

**Syntax**

```plaintext
fit DecisionTreeClassifier <field_to_predict> from <explanatory_fields> [into <model_name>] [max_depth=<int>] [max_features=<str>] [min_samples_split=<int>] [max_leaf_nodes=<int>] [criterion=<gini|entropy>] [splitter=<best|random>] [random_state=<int>]
```

You can save DecisionTreeClassifier models by using the `into` keyword and apply it to new data later by using the `apply` command.

```plaintext
... | apply model_DTC
```

You can inspect the decision tree learned by DecisionTreeClassifier with the `summary` command.

```plaintext
... | summary model_DTC
```

See a JSON representation of the tree by giving `json=t` as an argument to the `summary` command.

```plaintext
... | summary model_DTC json=t
```
Example

The following example uses DecisionTreeClassifier on a test set.

```plaintext
... | fit DecisionTreeClassifier SLA_violation from * into sla_model | ...
```

**GaussianNB**

The GaussianNB algorithm uses the scikit-learn GaussianNB estimator to fit a model to predict the value of categorical fields, where the likelihood of explanatory variables is assumed to be Gaussian. GaussianNB is an implementation of Gaussian Naive Bayes classification algorithm. This algorithm supports incremental fit.

**Parameters**

- The `partial_fit` parameter controls whether an existing model should be incrementally updated or not. This allows you to update an existing model using only new data without having to retrain it on the full training data set.
- The `partial_fit` parameter default is False.

**Syntax**

```plaintext
fit GaussianNB <field_to_predict> from <explanatory_fields> [into <model name>] [partial_fit=<true|false>]
```

You can save GaussianNB models using the `into` keyword and apply the saved model later to new data using the `apply` command.

```plaintext
... | apply TESTMODEL_GaussianNB
```

You can inspect models learned by GaussianNB with the `summary` command.

```plaintext
... | summary My_Incremental_Model
```

**Syntax constraints**

- If `My_Incremental_Model` does not exist, the command saves the model data under the model name `My_Incremental_Model`. If `My_Incremental_Model` exists and was trained using GaussianNB, the command updates the existing model with the new input. If `My_Incremental_Model` exists but was not trained by GaussianNB, an error message is thrown.
- If `partial_fit=False` or `partial_fit` is not specified the model specified is created and replaces the pre-trained model if one exists.
Example

The following example uses GaussianNB on a test set.

```
... | fit GaussianNB species from * into TESTMODEL_GaussianNB
The following example includes the partial_fit command.

| inputlookup iris.csv | fit GaussianNB species from * partial_fit=true into My_Incremental_Model
```

**GradientBoostingClassifier**

This algorithm uses the GradientBoostingClassifier from scikit-learn to build a classification model by fitting regression trees on the negative gradient of a deviance loss function. For further information, see the sci-kit learn documentation:

**Syntax**

```
fit GradientBoostingClassifier <field_to_predict> from <explanatory_fields>[into <model name>]
[loss=<deviance | exponential>] [max_features=<str>]
[learning_rate=<float>] [min_weight_fraction_leaf=<float>]
[n_estimators=<int>]
[max_depth=<int>] [min_samples_split =<int>] [min_samples_leaf=<int>]
[max_leaf_nodes=<int>] [random_state=<int>]
```

You can apply the saved model later to new data using the apply command.

```
... | apply TESTMODEL_GradientBoostingClassifier
```

You can inspect features learned by GradientBoostingClassifier with the summary command.

```
... | summary TESTMODEL_GradientBoostingClassifier
```

Example

The following example uses GradientBoostingClassifier on a test set.

```
... | fit GradientBoostingClassifier target from * into TESTMODEL_GradientBoostingClassifier
```
**LogisticRegression**

The LogisticRegression algorithm uses the scikit-learn LogisticRegression estimator to fit a model to predict the value of categorical fields.

**Parameters**

- The `fit_intercept` parameter specifies whether the model includes an implicit intercept term.
- The default value of the `fit_intercept` parameter is True.
- The `probabilities` parameter specifies whether probabilities for each possible field value should be returned alongside the predicted value.
- The default value of the `probabilities` parameter is False.

**Syntax**

```python
fit LogisticRegression <field_to_predict> from <explanatory_fields>
[into <model name>]
[fit_intercept=<true|false>] [probabilities=<true|false>]
```

You can save LogisticRegression models using the `into` keyword and apply new data later using the `apply` command.

```python
... | apply sla_model
```

You can inspect the coefficients learned by LogisticRegression with the `summary` command.

```python
... | summary sla_model
```

**Example**

The following examples uses LogisticRegression on a test set.

```python
... | fit LogisticRegression SLA_violation from IO_wait_time into sla_model | ...
```

**MLPClassifier**

The MLPClassifier algorithm uses the scikit-learn Multi-layer Perceptron estimator for classification. MLPClassifier uses a feedforward artificial neural network model that trains using backpropagation. This algorithm supports incremental fit.

For descriptions of the `batch_size`, `random_state` and `max_iter` parameters, see the scikit-learn documentation at

141
Using the MLPClassifier algorithm requires running version 1.3 or above of the Python for Scientific Computing add-on.

**Parameters**

- The `partial_fit` parameter controls whether an existing model should be incrementally updated or not. This allows you to update an existing model using only new data without having to retrain it on the full training data set.
- The `partial_fit` parameter default is `False`.
- The `hidden_layer_sizes` parameter format (`int`) varies based on the number of hidden layers in the data.

**Syntax**

```
fit MLPClassifier <field_to_predict> from <explanatory_fields> [into <model name>] [batch_size=<int>] [max_iter=<int>] [random_state=<int>] [hidden_layer_sizes=<int>-<int>-<int>] [activation=<str>] [solver=<str>] [learning_rate=<str>] [tol=<float>] [momentum=<float>]
```

You can save MLPClassifier models by using the `into` keyword and apply it to new data later by using the `apply` command.

You can inspect models learned by MLPClassifier with the `summary` command.

```
... | summary My_Example_Model
```

**Syntax constraints**

- If `My_Example_Model` does not exist, the model is saved to it.
- If `My_Example_Model` exists and was trained using MLPClassifier, the command updates the existing model with the new input.
- If `My_Example_Model` exists but was not trained using MLPClassifier, an error message displays.

**Example**

The following example uses MLPClassifier on a test set.

```
... | inputlookup diabetes.csv | fit MLPClassifier response from * into MLP_example_model hidden_layer_sizes='100-100-80' |...
```
The following example includes the partial_fit command.

```
| inputlookup iris.csv | fit MLPClassifier species from * partial_fit=true into My_Example_Model
```

**RandomForestClassifier**

The RandomForestClassifier algorithm uses the scikit-learn RandomForestClassifier estimator to fit a model to predict the value of categorical fields.

For descriptions of the `n_estimators`, `max_depth`, `criterion`, `random_state`, `max_features`, `min_samples_split`, and `max_leaf_nodes` parameters, see the scikit-learn documentation at http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html.

**Syntax**

```
fit RandomForestClassifier <field_to_predict> from <explanatory_fields>
[into <model name>]
[n_estimators=<int>] [max_depth=<int>] [criterion=<gini | entropy>]
[random_state=<int>]
[max_features=<str>] [min_samples_split=<int>] [max_leaf_nodes=<int>]
```

You can save RandomForestClassifier models using the into keyword and apply new data later using the apply command.

```
... | apply sla_model
```

You can list the features that were used to fit the model, as well as their relative importance or influence with the summary command.

```
... | summary sla_model
```

**Example**

The following example uses RandomForestClassifier on a test set.

```
... | fit RandomForestClassifier SLA_violation from * into sla_model |
```

**SGDClassifier**

The SGDClassifier algorithm uses the scikit-learn SGDClassifier estimator to fit a model to predict the value of categorical fields. This algorithm supports incremental fit.

**Parameters**

143
The `partial_fit` parameter controls whether an existing model should be incrementally updated or not. This allows you to update an existing model using only new data without having to retrain it on the full training data set.

- The `partial_fit` parameter default is False.
- `n_iter=<int>` is the number of passes over the training data also known as epochs. The default is 5. The number of iterations is set to 1 if using `partial_fit`.
- The `loss=<hinge|log|modified_huber|squared_hinge|perceptron>` parameter is the loss function to be used.
  - Defaults to `hinge`, which gives a linear SVM.
  - The `log` loss gives logistic regression, a probabilistic classifier.
  - `modified_huber` is another smooth loss that brings tolerance to outliers as well as probability estimates.
  - `squared_hinge` is like `hinge` but is quadratically penalized.
  - `perceptron` is the linear loss used by the perceptron algorithm.
- The `fit_intercept=<true|false>` parameter specifies whether the intercept should be estimated or not. The default is True.
- `penalty=<l1|l2|elasticnet>` is the penalty, also known as regularization term, to be used. The default is `l2`.
- `learning_rate=<constant|optimal|invscaling>` is the learning rate.
  - `constant`: $\eta = \eta_0$
  - `optimal`: $\eta = 1.0/(\alpha \times t)$
  - `invscaling`: $\eta = \eta_0 / \text{pow}(t, \text{power\_t})$
  - The default is `invscaling`
- `l1_ratio=<float>` is the Elastic Net mixing parameter, with $0 \leq l1\_ratio \leq 1$ (default 0.15).
  - `l1_ratio=0` corresponds to L2 penalty, `l1\_ratio=1` to L1.
- `alpha=<float>` is the constant that multiplies the regularization term (default 0.0001). Also used to compute `learning\_rate` when set to `optimal`.
- `eta0=<float>` is the initial learning rate. The default is 0.01.
- `power\_t=<float>` is the exponent for inverse scaling learning rate. The default is 0.25.
- `random\_state=<int>` is the seed of the pseudo random number generator to use when shuffling the data.

**Syntax**

```
fit SGDClassifier <field_to_predict> from <explanatory_fields>
[ into <model name> ] [ partial_fit=<true|false> ]
[ loss=<hinge|log|modified_huber|squared_hinge|perceptron> ]
[ fit_intercept=<true|false> ]
[ random_state=<int> ] [ n_iter=<int> ] [ l1_ratio=<float> ]
[ alpha=<float> ] [ eta0=<float> ] [ power\_t=<float> ]
[ penalty=<l1|l2|elasticnet> ]
```
You can save SGDClassifier models using the `into` keyword and apply the saved model later to new data using the `apply` command.

```
... | apply sla_model
```

You can inspect the model learned by SGDClassifier with the `summary` command.

```
... | summary sla_model
```

**Syntax constraints**

- If `My_Incremental_Model` does not exist, the command saves the model data under the model name `My_Incremental_Model`.
- If `My_Incremental_Model` exists and was trained using SGDClassifier, the command updates the existing model with the new input.
- If `My_Incremental_Model` exists but was not trained by SGDClassifier, an error displays.
- Using `partial_fit=true` on an existing model ignores the newly supplied parameters. The parameters supplied at model creation are used instead.
- If `partial_fit=false` or `partial_fit` is not specified the model specified is created and replaces the pre-trained model if one exists.

**Example**

The following example uses SGDClassifier on a test set.

```
... | fit SGDClassifier SLA_violation from * into sla_model
```

The following example includes the `partial_fit=<true|false>` command.

```
partial_fit | inputlookup iris.csv | fit SGDClassifier species from *
partial_fit=true into My_Incremental_Model
```

**SVM**

The SVM algorithm uses the scikit-learn kernel-based SVC estimator to fit a model to predict the value of categorical fields. It uses the radial basis function (rbf) kernel by default. For descriptions of the `C` and `gamma` parameters, see the scikit-learn documentation at http://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html.

Kernel-based methods such as the scikit-learn SVC tend to work best when the data is scaled, for example, using our StandardScaler algorithm:

```
StandardScaler into scaling_model | fit SVM from into svm_model
```

For details, see "A Practical Guide to Support Vector Classification" at
Parameters

- The `gamma` parameter controls the width of the rbf kernel. The default value is $1 / \text{number of fields}$.
- The `C` parameter controls the degree of regularization when fitting the model. The default value is 1.0.

Syntax

```
fit SVM <field_to_predict> from <explanatory_fields> [into <model name>] [C=<float>] [gamma=<float>]
```

You can save SVM models using the `into` keyword and apply new data later using the `apply` command.

```
... | apply sla_model
```

Syntax constraints

You cannot inspect the model learned by SVM with the `summary` command.

Example

The following example uses SVM on a test set.

```
... | fit SVM SLA_violation from * into sla_model | ...
```

Clustering Algorithms

Clustering is the grouping of data points. Results will vary depending upon the clustering algorithm used. Clustering algorithms differ in how they determine if data points are similar and should be grouped. For example, the K-means algorithm clusters based on points in space, whereas the DBSCAN algorithm clusters based on local density.

**BIRCH**

The BIRCH algorithm uses the scikit-learn BIRCH clustering algorithm to divide data points into set of distinct clusters. The cluster for each event is set in a new field named `cluster`. This algorithm supports incremental fit.

Parameters
The k parameter specifies the number of clusters to divide the data into after the final clustering step, which treats the sub-clusters from the leaves of the CF tree as new samples.

- By default, the cluster label field name is `cluster`. Change that behavior by using the `as` keyword to specify a different field name.
- The `partial_fit` parameter controls whether an existing model should be incrementally updated on not. This allows you to update an existing model using only new data without having to retrain it on the full training data set.
- The `partial_fit` parameter default is False.

**Syntax**

```plaintext
fit BIRCH <fields> [into <model name>] [k=<int>] [partial_fit=<true|false>] [into <model name>]
```

You can save BIRCH models using the `into` keyword and apply new data later using the `apply` command.

```plaintext
... | apply BIRCH_model
```

**Syntax constraints**

- If `My_Incremental_Model` does not exist, the command saves the model data under the model name `My_Incremental_Model`.
- If `My_Incremental_Model` exists and was trained using BIRCH, the command updates the existing model with the new input.
- If `My_Incremental_Model` exists but was not trained by BIRCH, an error message displays.
- Using `partial_fit=true` on an existing model ignores the newly supplied parameters. The parameters supplied at model creation are used instead.
- If `partial_fit=false` or `partial_fit` is not specified the model specified is created and replaces the pre-trained model if one exists.
- You cannot inspect the model learned by BIRCH with the `summary` command.

**Examples**

The following example uses BIRCH on a test set.

```plaintext
... | fit BIRCH * k=3 | stats count by cluster
```

The following example includes the `partial_fit` command.

```plaintext
| inputlookup track_day.csv | fit BIRCH * k=6 partial_fit=true into My_Incremental_Model
```
**DBSCAN**

The DBSCAN algorithm uses the scikit-learn DBSCAN clustering algorithm to divide a result set into distinct clusters. The cluster for each event is set in a new field named `cluster`. DBSCAN is distinct from K-Means in that it clusters results based on local density, and uncovers a variable number of clusters, whereas K-Means finds a precise number of clusters. For example, \( k=5 \) finds 5 clusters.

**Parameters**

- The `eps` parameter specifies the maximum distance between two samples for them to be considered in the same cluster.
  
  - By default, the cluster label field name is `cluster`. Change that behavior by using the `as` keyword to specify a different field name.

- The `min_samples` parameter defines the number of samples, or the total weight, in a neighborhood for a point to be considered as a core point - including the point itself. You can choose the `min_samples` parameter's best value based on preference for cluster density or noise in your dataset.

- The `min_samples` parameter is optional.
- The `min_samples` default value is 5.

- The minimum value for the `min_samples` parameter is 3.
- If `min_samples=8` you need at least 8 data points to form a dense cluster.

If you choose the `min_samples` parameter's best value based on noise in your dataset, it's recommended to have a larger data set to pull from.

**Syntax**

```
| fit DBSCAN <fields> [eps=<number>] [min_samples=<integer>]
```

**Syntax constraints**

You cannot save DBSCAN models using the `into` keyword. To predict cluster assignments for future data, combine the DBSCAN algorithm with any classifier algorithm. For example, first cluster the data using DBSCAN, then fit RandomForestClassifier to predict the cluster.

**Examples**

The following example uses DBSCAN without the `min_samples` parameter.

```
... | fit DBSCAN * | stats count by cluster
```

The following example uses DBSCAN with the `min_samples` parameter.
K-means clustering is a type of unsupervised learning. It is a clustering algorithm that groups similar data points, with the number of groups represented by the variable \( k \). The K-means algorithm uses the scikit-learn K-means implementation. The cluster for each event is set in a new field named `cluster`. Use the K-means algorithm when you have unlabeled data and have at least approximate knowledge of the total number of groups into which the data can be divided.

Using the K-means algorithm has the following advantages:

- Computationally faster than most other clustering algorithms.
- Simple algorithm to explain and understand.
- Normally produces tighter clusters than hierarchical clustering.

Using the K-means algorithm has the following disadvantages:

- Difficult to determine optimal or true value of \( k \). See X-means.
- Sensitive to scaling. See StandardScaler.
- Each clustering may be slightly different, unless you specify the `random_state` parameter.
- Does not work well with clusters of different sizes and density.

For descriptions of default value of \( k \), see the scikit-learn documentation at http://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html

**Parameters**

The \( k \) parameter specifies the number of clusters to divide the data into. By default, the cluster label field name is `cluster`. Change that behavior by using the `as` keyword to specify a different field name.

**Syntax**

```
fit KMeans <fields> [into <model name>] [k=<int>] [random_state=<int>]
```

You can save K-means models using the `into` keyword when using the `fit` command.

You can apply the model to new data using the `apply` command.

```
... | apply cluster_model
```
You can inspect the model using the `summary` command.

```plaintext
... | summary cluster_model
```

**Example**

The following example uses K-means on a test set.

```plaintext
... | fit KMeans * k=3 | stats count by cluster
```

**SpectralClustering**

The SpectralClustering algorithm uses the scikit-learn SpectralClustering clustering algorithm to divide a result set into set of distinct clusters. SpectralClustering first transforms the input data using the Radial Basis Function (rbf) kernel, and then performs K-Means clustering on the result. Consequently, SpectralClustering can learn clusters with a non-convex shape. The cluster for each event is set in a new field named `cluster`.

**Parameters**

The `k` parameter specifies the number of clusters to divide the data into after kernel step. By default, the cluster label field name is `cluster`. Change that behavior by using the `as` keyword to specify a different field name.

**Syntax**

```
fit SpectralClustering <fields> [k=<int>] [gamma=<float>] [random_state=<int>]
```

**Syntax constraints**

You cannot save SpectralClustering models using the `into` keyword. If you want to be able to predict cluster assignments for future data, you can combine the SpectralClustering algorithm with any clustering algorithm. For example, first cluster the data using SpectralClustering, then fit a classifier to predict the cluster using RandomForestClassifier.

**Example**

The following example uses SpectralClustering on a test set.

```plaintext
... | fit SpectralClustering * k=3 | stats count by cluster
```
Use the X-means algorithm when you have unlabeled data and no prior knowledge of the total number of labels into which that data could be divided. The X-means clustering algorithm is an extended K-means that automatically determines the number of clusters based on Bayesian Information Criterion (BIC) scores. Starting with a single cluster, the X-means algorithm goes into action after each run of K-means, making local decisions about which subset of the current centroids should split themselves in order to fit the data better.

Using the X-means algorithm has the following advantages:

- Eliminates the requirement of having to provide the value of $k$.
- Normally produces tighter clusters than hierarchical clustering.

Using the X-means algorithm has the following disadvantages:

- Sensitive to scaling. See StandardScaler.
- Different initializations might result in different final clusters.
- Does not work well with clusters of different sizes and density.

**Parameters**

- The $k$ is the total number of labels/clusters in the data.
- The splitting decision is done by computing the BIC.
- The cluster for each event is set in a new field named cluster, and the total number of clusters is set in a new field named n_clusters.
- By default, the cluster label field name is cluster. Change that behavior by using the as keyword to specify a different field name.

**Syntax**

```
fit XMeans <fields> [into <model name>]
```

You can apply new data to the saved X-means model using the apply command.

```
... | apply cluster_model
```

You can save X-means models using the into command. You can inspect the model learned by X-means with the summary command.

```
...| summary cluster_model
```

**Example**
The following example uses X-means on a test set.

... | fit XMeans * | stats count by cluster

**Cross-validation**

Cross-validation assesses how well a statistical model generalizes on an independent dataset. Cross-validation tells you how well your machine learning model is expected to perform on data that it has not been trained on. There are many types of cross-validation, but K-fold cross-validation (kfold_cv) is one of the most common.

Cross-validation is typically used for the following machine learning scenarios:

- Comparing two or more algorithms against each other for selecting the best choice on a particular dataset.
- Comparing different choices of hyper-parameters on the same algorithm for choosing the best hyper-parameters for a particular dataset.
- An improved method over a train/test split for quantifying model generalization.

Cross-validation is **not** well suited for time-series charts:

- In situations where the data is ordered such as time-series, cross-validation is not well suited because the training data is shuffled. In these situations, other methods such as Forward Chaining are more suitable.
- The most straightforward implementation is to wrap sklearn's Time Series Split. Learn more here: https://en.wikipedia.org/wiki/Forward_chaining

**K-fold cross-validation**

In the kfold_cv parameter, the training set is randomly partitioned into k equal-sized subsamples. Then, each sub-sample takes a turn at becoming the validation (test) set, predicted by the other k-1 training sets. Each sample is used exactly once in the validation set, and the variance of the resulting estimate is reduced as k is increased. The disadvantage of the kfold_cv parameter is that k different models have to be trained, leading to long execution times for large datasets and complex models.

The scores obtained from K-fold cross-validation are generally a less biased and less optimistic estimate of the model performance than a standard training and testing split.
You can obtain k performance metrics, one for each training and testing split. These k performance metrics can then be averaged to obtain a single estimate of how well the model generalizes on unseen data.

**Syntax**

The `kfold_cv` parameter is applicable to all classification and regression algorithms, and you can append the command to the end of an SPL search.

Here `kfold_cv=<int>` specifies that k=<int> folds is used. When you specify a classification algorithm, stratified k-fold is used instead of k-fold. In stratified k-fold, each fold contains approximately the same percentage of samples for each class.

```
..| fit <classification | regression algo> <targetVariable> from <featureVariables> [options] kfold_cv=<int>
```

The `kfold_cv` parameter cannot be used when saving a model.

**Output** The `kfold_cv` parameter returns performance metrics on each fold using the same model specified in the SPL - including algorithm and hyper parameters. Its only function is to give you insight into how well you model generalizes. It does not perform any model selection or hyper parameter tuning.

**Examples**

The first example shows the `kfold_cv` parameter used in classification. Where the output is a set of metrics for each fold including accuracy, f1_weighted, precision_weighted, and recall_weighted.
This second example shows the `kfold_cv` parameter used in classification. Where the output is a set of metrics for each the `neg_mean_squared_error` and `r^2` folds.

**Feature Extraction**

Feature extraction algorithms transform fields for better prediction accuracy.

**FieldSelector**

The FieldSelector algorithm uses the scikit-learn GenericUnivariateSelect to select the best predictor fields based on univariate statistical tests. For descriptions of the `mode` and `param` parameters, see the scikit-learn documentation at http://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.GenericUnivariateSelect.html.

**Parameters**

The `type` parameter specifies if the field to predict is categorical or numeric.

**Syntax**

```plaintext
fit FieldSelector <field_to_predict> from <explanatory_fields> 
[into <model name>] [type=<categorical, numeric>] 
[mode=k_best, fpr, fdr, fwe, percentile] [param=<int>]
```

You can save FieldSelector models using the `into` keyword and apply new data later using the `apply` command.

```plaintext
... | apply sla_model
```

You can inspect the model learned by FieldSelector with the `summary` command.

```plaintext
| summary sla_model
```

**Example**

154
The following example uses FieldSelector on a test set.

```plaintext
... | fit FieldSelector type=categorical SLA_violation from * into sla_model | ...
```

**HashingVectorizer**

The HashingVectorizer algorithm converts text documents to a matrix of token occurrences. It uses a feature hashing strategy to allow for hash collisions when measuring the occurrence of tokens. It is a stateless transformer, meaning that it does not require building a vocabulary of the seen tokens. This reduces the memory footprint and allows for larger feature spaces.

HashingVectorizer is comparable with the TFIDF algorithm, as they share many of the same parameters. However HashingVectorizer is a better option for building models with large text fields provided you do not need to know term frequencies, and only want outcomes.

For descriptions of the `ngram_range`, `analyzer`, `norm`, and `token_pattern` parameters, see the scikit-learn documentation at https://scikit-learn.org/0.19/modules/generated/sklearn.feature_extraction.text.HashingVectorizer.html

### Parameters

- The `reduce` parameter is either True or False and determines whether or not to reduce the output to a smaller dimension using TruncatedSVD.
- The `reduce` parameter default is True.
- The `k=<int>` parameter sets the number of dimensions to reduce when the `reduce` parameter is set to `true`. Default is 100.
- The default for the `max_features` parameter is 10,000.
- The `n_iters` parameter specifies the number of iterations to use when performing dimensionality reduction. This is only used when the `reduce` parameter is set to `True`. Default is 5.

### Syntax

```plaintext
fit HashingVectorizer <field_to_convert> [max_features=<int>] [n_iters=<int>] [reduce=<bool>] [k=<int>] [ngram_range=<int>-<int>] [analyzer=<str>] [norm=<str>] [token_pattern=<str>] [stop_words=english]
```

### Syntax constraints

HashingVectorizer does not support saving models, incremental fit, or K-fold cross validation.
Example

The following example uses HashingVectorizer to hash the text dataset and applies KMeans clustering (where $k=3$) on the hashed fields.

| inputlookup authorization.csv | fit HashingVectorizer Logs ngram_range=1-2 k=50 stop_words=english | fit KMeans Logs_hashed* k=3 | fields cluster* Logs | sample 5 by cluster | sort by cluster

ICA

ICA (Independent component analysis) separates a multivariate signal into additive sub-components that are maximally independent. Typically, ICA is not used for separating superimposed signals, but for reducing dimensionality. The ICA model does not include a noise term for the model to be correct, meaning whitening must be applied. Whitening can be done internally using the whiten argument, or manually using one of the PCA variants.

Parameters

- The `n_components` parameter determines the number of components ICA uses.
- The `n_components` parameter is optional.
- The `n_components` parameter default is `None`. If `None` is selected, all components are used.
- Use the `algorithm` parameter to apply parallel or deflation algorithm for FastICA.
- The `algorithm` parameter default is `algorithm='parallel'`.
- Use the `whiten` parameter to set a noise term.
- The `whiten` parameter is optional.
- If the `whiten` parameter is `False` no whitening is performed.
- The `whiten` parameter default is `True`.
- The `max_iter` parameter determines the maximum number of iterations during the running of the `fit` command.
- The `max_iter` parameter is optional.
- The `max_iter` parameter default is 200.
- The `fun` parameter determines the functional form of the G function used in the approximation to neg-entropy.
- The `fun` parameter is optional.
- The `fun` parameter default is `logcosh`. Other options for this parameter are `exp` or `cube`.
- The `tol` parameter sets the tolerance on update at each iteration.
- The `tol` parameter is optional.
- The `tol` parameter default is 0.0001.
The `random_state` parameter sets the seed value used by the random number generator.

- The `random_state` parameter default is `None`.
- If `random_state=None` then a random seed value is used.

**Syntax**

```plaintext
fit ICA n_components=<int>, algorithm="parallel"|"deflation", whiten=<bool>, fun="logcosh"|"exp"|"cube", max_iter=<int>, tol=<float>, random_state=<int> <explanatory_fields> [into <model name>]
```

You can save ICA models using the `into` keyword and apply new data later using the `apply` command.

**Syntax constraints**

You cannot inspect the model learned by ICA with the `summary` command.

**Example**

The following example shows how ICA is able to find the two original sources of data from two measurements that have mixes of both. As a comparison, PCA is used to show the difference between the two?PCA is not able to identify the original sources.

```plaintext
| makeresults count=2
| streamstats count as count
| eval time=case(count=2,relative_time(now(),"+2d"),count=1,now())
| makecontinuous time span=15m
| eval _time=time
| eval s1 = sin(2*time)
| eval s2 = sin(4*time)
| eval m1 = 1.5*s1 + .5*s2, m2 = .1*s1 + s2
| fit ICA m1, m2 n_components=2 as IC
| fit PCA m1, m2 k=2 as PC
| fields _time, *
| fields - count, time
```

**KernelPCA**

The KernelPCA algorithm uses the scikit-learn KernelPCA to reduce the number of fields by extracting uncorrelated new features out of data. The difference between KernelPCA and PCA is the use of kernels in the former, which helps with finding nonlinear dependencies among the fields. Currently, KernelPCA only supports the Radial Basis Function (rbf) kernel.

Kernel-based methods such as KernelPCA tend to work best when the data is scaled, for example, using our StandardScaler algorithm: `fit StandardScaler into scaling_model | fit KernelPCA into kpca_model`. For details, see "A Practical Guide to Support Vector Classification" at https://www.csie.ntu.edu.tw/~cjlin/papers/guide/guide.pdf.

**Parameters**

The k parameter specifies the number of features to be extracted from the data. The other parameters are for fine tuning of the kernel.

**Syntax**

```
fit KernelPCA <fields> [into <model name>] [degree=<int>] [k=<int>] [gamma=<int>] [tolerance=<int>] [max_iteration=<int>]
```

You can save KernelPCA models using the into keyword and apply new data later using the apply command.

```
... | apply user_feedback_model
```

**Syntax constraints**

You cannot inspect the model learned by KernelPCA with the summary command.

**Example**

The following example uses KernelPCA on a test set.

```
... | fit KernelPCA * k=3 gamma=0.001 | ...
```

**NPR**

The Normalized Perich Ratio (NPR) algorithm converts high cardinality categorical field values into numeric field entries while intelligently handling space optimization. NPR offers low computational costs to perform feature extraction on variables with high cardinalities such as ZIP codes or IP addresses.

NPR does not perform one-hot encoding unlike other algorithms that leverage the fit and apply commands.
Parameters

- Use the summary command to inspect the variance information of the saved model.
- After running NPR the transformed dataset has calculated ratios for all feature variables (feature_field). Based on the training data NPR calculates a variable of X_unobserved which can be used as a replacement value in the following two scenarios:
  - In conjunction with the fit command NPR initially replaces missing values in the dataset for feature_field with the keyword unobserved which is then replaced by the calculated NPR value of X_unobserved.
  - In conjunction with the apply command, any new value for target_field that was not visible during model training but is encountered in the test dataset.
- The number of transformed columns created after running NPR is equal to the number of distinct values for feature_field within the search string.
- From the saved model, use the variance output field to examine the contribution of a particular feature towards the accuracy of the prediction. Higher variance indicates highly important categorical values whereas low variance indicates the value being of lower importance towards the target prediction. Variance may assist in the process of discarding irrelevant feature variables.

Syntax

fit NPR <target_field> from <feature_field> [into <model name>]
You can couple NPR with existing MLTK algorithms to feed the transformed results to the model as a means to enhance predictions.

| fit NPR <target_field> from <feature_field> | fit SGDClassifier <target_field> from NPR

You can save NPR models using the into keyword and apply new data later using the apply command.

| input lookup disk_failures.csv | tail 1000 | apply npr_disk

You can inspect the model learned by NPR with the summary command.

| summary npr_disk

Syntax constraints
• The wildcard (*) character is not supported.
• The maximum matrix size calculated from |X| * |Y| where X is the feature_field and Y is the target_field is 1000000. For example, if number of distinct categorical feature values are 1000 and distinct categorical target values are 100 then the matrix size is 100000.

Examples

The following example uses NPR on a test set.

| inputlookup disk_failures.csv | head 5000 | fit NPR DiskFailure from Model into npr_disk |

The following example couples NPR with another MLTK algorithm on a test set.

| inputlookup disk_failures.csv | head 5000 | fit NPR DiskFailure from Model | fit SGDClassifier DiskFailure from NPR_* random_state=42 n_iter=2 | score accuracy_score DiskFailure against predicted* |

The following example uses NPR over multiple fields with additional uses of the fit command.

| inputlookup disk_failures.csv | head 5000 | fit NPR DiskFailure from Model into npr_disk_1 | fit NPR DiskFailure from SerialNumber into npr_disk_2 |

PCA

The Principal Component Analysis (PCA) algorithm uses the scikit-learn PCA algorithm to reduce the number of fields by extracting new, uncorrelated features out of the data.

Parameters

• The k parameter specifies the number of features to be extracted from the data.
• The variance parameter is short for percentage variance ratio explained. This parameter determines the percentage of variance ratio explained in the principal components of the PCA. It computes the number of principal components dynamically by preserving the specified variance ratio.
• The variance parameter defaults to 1 if k is not provided.
• The variance parameter can take a value between 0 and 1.
• The component name parameter represents the name of the selected components from the value specified in n_components.
• The explained_variance parameter measures the proportion to which the principal component accounts for dispersion of a given dataset. A higher value denotes a higher variation.
The `explained_variance_ratio` parameter is the percentage of variance explained by each of the selected components.

- The `singular_values` parameter represents the singular values corresponding to each of the selected components. Singular values are equal to the 2-norms of the `n_components` variables in the lower-dimensional space.

**Syntax**

```
fit PCA <fields> [into <model name>] [k=<int>] [variance=<float>]
```

You can save PCA models using the `into` keyword and apply new data later using the `apply` command.

```
...into example_hard_drives_PCA_2 | apply example_hard_drives_PCA_2
```

You can inspect the model learned by PCA with the `summary` command.

```
| summary example_hard_drives_PCA_2
```

**Syntax constraints**

The `variance` parameter and `k` parameter cannot be used together. They are mutually exclusive.

**Examples**

The following example uses PCA on a test set.

```
| fit PCA "SS_SMART_1_Raw", "SS_SMART_2_Raw", "SS_SMART_3_Raw",
"SS_SMART_4_Raw", "SS_SMART_5_Raw" k=2 into example_hard_drives_PCA_2
```

The following example includes the `variance` parameter. The value `variance=0.5` tells the algorithm to choose as many principal components for the data set until able to explain 50% of the variance in the original dataset.

```
| fit PCA "SS_SMART_1_Raw", "SS_SMART_2_Raw", "SS_SMART_3_Raw",
"SS_SMART_4_Raw", "SS_SMART_5_Raw" variance=0.50 into example_hard_drives_PCA_2
```

**TFIDF**

The TFIDF algorithm uses the scikit-learn TfidfVectorizer to convert raw text data into a matrix making it possible to use other machine learning estimators on the data. For descriptions of the `max_features`, `max_df`, `min_df`, `ngram_range`, `analyzer`, `norm`, and `token_pattern` parameters, see the scikit-learn documentation at

161
TFIDF uses memory to create a dictionary of all terms including ngrams and words, and expands the Splunk search events with additional fields per event. If you are concerned with memory limits, consider using the HashingVectorizer algorithm.

**Parameters**

The default for `max_features` is 100.

To configure the algorithm to ignore common English words (for example, "the", "it", "at", and "that"), set `stop_words` to `english`. For other languages (for example, machine language) you can ignore the common words by setting `max_df` to a value greater than or equal to 0.7 and less than 1.0.

**Syntax**

```
fit TFIDF <field_to_convert> [into <model name>] [max_features=<int>] [max_df=<int>] [min_df=<int>] [ngram_range=<int>-<int>] [analyzer=<str>] [norm=<str>] [token_pattern=<str>] [stop_words=english]
```

You can save TFIDF models using the `into` keyword and apply new data later using the `apply` command.

```
... | apply user_feedback_model
```

**Syntax constraints**

You cannot inspect the model learned by TFIDF with the `summary` command.

**Example**

The following example uses TFIDF to convert the text dataset to a matrix of TF-IDF features and then applies KMeans clustering (where `k=3`) on the matrix.

```
| inputlookup authorization.csv | fit TFIDF Logs ngram_range=1-2 [max_df=0.6 min_df=0.2 stop_words=english] | fit KMeans Logs_tfidf* k=3 | *fields cluster Logs | sample 6 by cluster | sort by cluster
```

162
Preprocessing (Prepare Data)

Preprocessing algorithms are used for preparing data. Other algorithms can also be used for preprocessing that may not be organized under this section. For example, PCA can be used for both Feature Extraction and Preprocessing.

Imputer

The Imputer algorithm is a preprocessing step wherein missing data is replaced with substitute values. The substitute values can be estimated, or based on other statistics or values in the dataset. To use Imputer, the user passes in the names of the fields to impute, along with arguments specifying the imputation strategy, and the values representing missing data. Imputer then adds new imputed versions of those fields to the data, which are copies of the original fields, except that their missing values are replaced by values computed according to the imputation strategy.

Parameters

- Available imputation strategies include mean, median, most frequent, and field. The default strategy is mean.
- All but the field parameter require numeric data. The field strategy accepts categorical data.

Syntax

.. | fit Imputer <field>* [as <field prefix>] [missing_values="NaN"|integer] [strategy=<mean|median|most_frequent>] [into <model name>]

You can inspect the value (mean, median, or mode) that was substituted for missing values by Imputer with the summary command.

... | summary <imputer model name>

You can save Imputer models using the into keyword and apply new data later using the apply command.

... | apply <imputer model name>

Example

The following example uses Imputer on a test set.
RobustScaler

The RobustScaler algorithm uses the scikit-learn RobustScaler algorithm to standardize data fields by scaling their median and interquartile range to 0 and 1, respectively. It is very similar to the StandardScaler algorithm, in that it helps avoid dominance of one or more fields over others in subsequent machine learning algorithms, and is practically required for some algorithms, such as KernelPCA and SVM. The main difference between StandardScaler and RobustScaler is that RobustScaler is less sensitive to outliers.

Parameters

The `with_centering` and `with_scaling` parameters specify if the fields should be standardized with respect to their median and interquartile range.

Syntax

```
fit RobustScaler <fields> [into <model name>] [with_centering=<true|false>] [with_scaling=<true|false>]
```

You can save RobustScaler models using the `into` keyword and apply new data later using the `apply` command.

```
... | apply scaling_model
```

You can inspect the statistics extracted by RobustScaler with the `summary` command.

```
... | summary scaling_model
```

Syntax constraints

RobustScaler does not support incremental fit.

Example

The following example uses RobustScaler on a test set.
The StandardScaler algorithm uses the scikit-learn StandardScaler algorithm to standardize data fields by scaling their mean and standard deviation to 0 and 1, respectively. This preprocessing step helps to avoid dominance of one or more fields over others in subsequent machine learning algorithms. This step is practically required for some algorithms, such as KernelPCA and SVM. This algorithm supports incremental fit.

**Parameters**

- The `with_mean` and `with_std` parameters specify if the fields should be standardized with respect to their mean and standard deviation.
- The `partial_fit` parameter controls whether an existing model should be incrementally updated or not. This allows you to update an existing model using only new data without having to retrain it on the full training data set. The default is False.

**Syntax**

```plaintext
fit StandardScaler <fields> [into <model name>] [with_mean=<true|false>] [with_std=<true|false>] [partial_fit=<true|false>]
```

You can save StandardScaler models using the `into` keyword and apply new data later using the `apply` command.

```plaintext
... | apply scaling_model
```

You can inspect the statistics extracted by StandardScaler with the `summary` command.

```plaintext
... | summary scaling_model
```

**Syntax constraints**

- Using `partial_fit=true` on an existing model ignores the newly supplied parameters. The parameters supplied at model creation are used instead. If `partial_fit=false` or `partial_fit` is not specified (default is false), the model specified is created and replaces the pre-trained model if one exists.
- If `My_Incremental_Model` does not exist, the command saves the model data under the model name `My_Incremental_Model`. 
If My_Incremental_Model exists and was trained using StandardScaler, the command updates the existing model with the new input.

- If My_Incremental_Model exists but was not trained by StandardScaler, an error message is thrown.

Examples

The following example uses StandardScaler on a test set.

```plaintext
... | fit StandardScaler * | ...
```

The following example includes the `partial_fit` parameter.

```plaintext
| inputlookup track_day.csv | fit StandardScaler "batteryVoltage", "engineCoolantTemperature", "engineSpeed" partial_fit=true into My_Incremental_Model
```

Regressors

Regressors algorithms predict the value of a numeric field.

The `kfold` cross-validation command can be used with all Regressor algorithms. Learn more here.

**DecisionTreeRegressor**

The DecisionTreeRegressor algorithm uses the scikit-learn DecisionTreeRegressor estimator to fit a model to predict the value of numeric fields. For descriptions of the `max_depth`, `random_state`, `max_features`, `min_samples_split`, `max_leaf_nodes`, and `splitter` parameters, see the scikit-learn documentation at http://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeRegressor.html.

Parameters

To specify the maximum depth of the tree to summarize, use the `limit` argument. The default value for the `limit` argument is 5.

```plaintext
| summary model_DTC limit=10
```

Syntax

```plaintext
fit DecisionTreeRegressor <field_to_predict> from <explanatory_fields>
[into <model_name>]
[max_depth=<int>] [max_features=<str>] [min_samples_split=<int>]
[random_state=<int>]
```
You can save DecisionTreeRegressor models using the \texttt{into} keyword and apply it to new data later using the \texttt{apply} command.

\begin{verbatim}
... \mid apply model_DTR
\end{verbatim}

You can inspect the decision tree learned by DecisionTreeRegressor with the \texttt{summary} command.

\begin{verbatim}
... \mid summary model_DTR
\end{verbatim}

You can get a JSON representation of the tree by giving \texttt{json=t} as an argument to the \texttt{summary} command.

\begin{verbatim}
... \mid summary model_DTR json=t
\end{verbatim}

\textbf{Example}

The following example uses DecisionTreeRegressor on a test set.

\begin{verbatim}
... \mid fit DecisionTreeRegressor temperature from date_month date_hour
\quad \textbf{into temperature_model} \mid ...
\end{verbatim}

\textit{ElasticNet}

The ElasticNet algorithm uses the scikit-learn ElasticNet estimator to fit a model to predict the value of numeric fields. ElasticNet is a linear regression model that includes both L1 and L2 regularization and is a generalization of \textit{Lasso} and \textit{Ridge}.

For descriptions of the \texttt{fit_intercept}, \texttt{normalize}, \texttt{alpha}, and \texttt{l1_ratio} parameters, see the scikit-learn documentation at http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.ElasticNet.html.

\textbf{Syntax}

\begin{verbatim}
fit ElasticNet <field_to_predict> from <explanatory_fields>
[\textbf{into <model name>}] \ [\texttt{fit_intercept=<true|false>}] \ [\texttt{normalize=<true|false>}] \ [\texttt{alpha=<int>}] \ [\texttt{l1_ratio=<int>}] \n\end{verbatim}

You can save ElasticNet models using the \texttt{into} keyword and apply new data later using the \texttt{apply} command.

\begin{verbatim}
... \mid apply temperature_model
\end{verbatim}
You can inspect the coefficients learned by ElasticNet with the `summary` command.

```sh
... | summary temperature_model
```

**Example**

The following example uses ElasticNet on a test set.

```sh
... | fit ElasticNet temperature from date_month date_hour
    normalize=true alpha=0.5 | ...
```

**GradientBoostingRegressor**

This algorithm uses the GradientBoostingRegressor algorithm from scikit-learn to build a regression model by fitting regression trees on the negative gradient of a loss function. For further information see the scikit-learn documentation at http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.GradientBoostingRegressor.html

**Syntax**

```sh
fit GradientBoostingRegressor <field_to_predict> from 
    <explanatory_fields>
[into <model_name>] [loss=ls|lad|huber|quantile]
[max_features=<str>] [learning_rate=<float>]
[min_weight_fraction_leaf=<float>]
[alpha=<float>] [subsample=<float>] [n_estimators=<int>]
[max_depth=<int>]
[min_samples_split=<int>] [min_samples_leaf=<int>]
[max_leaf_nodes=<int>]
[random_state=<int>]
```

You can use the `apply` method to apply the trained model to the new data.

```sh
... | apply temperature_model
```

You can inspect the features learned by GradientBoostingRegressor with the `summary` command.

```sh
... | summary temperature_model
```

**Example**

The following example uses the GradientBoostingRegressor algorithm to fit a model and saves that model as `temperature_model`. 

168
The KernelRidge algorithm uses the scikit-learn KernelRidge algorithm to fit a model to predict numeric fields. This algorithm uses the radial basis function (rbf) kernel by default. For details, see the scikit-learn documentation at http://scikit-learn.org/stable/modules/generated/sklearn.kernel_ridge.KernelRidge.html.

Parameters

The gamma parameter controls the width of the rbf kernel. The default value is \(\frac{1}{\text{number of fields}}\).

Syntax

```
fit KernelRidge <field_to_predict> from <explanatory_fields> [into <model_name>] [gamma=<float>]
```

You can save KernelRidge models using the into keyword and apply new data later using the apply command.

```
... | apply sla_model
```

Syntax constraints

You cannot inspect the model learned by KernelRidge with the summary command.

Example

The following example uses KernelRidge on a test set.

```
... | fit KernelRidge temperature from date_month date_hour into temperature_model | ...
```

The Lasso algorithm uses the scikit-learn Lasso estimator to fit a model to predict the value of numeric fields. Lasso is like LinearRegression, but it uses L1 regularization to learn a linear models with fewer coefficients and smaller coefficients. Lasso models are consequently more robust to noise and resilient against overfitting.

Parameters

- The `alpha` parameter controls the degree of L1 regularization.
- The `fit_intercept` parameter specifies whether the model should include an implicit intercept term. The default value is True.

Syntax

```
fit Lasso <field_to_predict> from <explanatory_fields> [into <model name>] [alpha=<float>] [fit_intercept=<true|false>] [normalize=<true|false>]
```

You can save Lasso models using the `into` keyword and apply new data later using the `apply` command.

```
... | apply temperature_model
```

You can inspect the coefficients learned by Lasso with the `summary` command.

```
... | summary temperature_model
```

Example

The following example uses Lasso on a test set.

```
... | fit Lasso temperature from date_month date_hour | ...
```

LinearRegression

The LinearRegression algorithm uses the scikit-learn LinearRegression estimator to fit a model to predict the value of numeric fields.

Parameters

The `fit_intercept` parameter specifies whether the model should include an implicit intercept term. The default value is True.

Syntax

```
fit LinearRegression <field_to_predict> from <explanatory_fields> [into <model name>] [fit_intercept=<true|false>] [normalize=<true|false>]
```
You can save LinearRegression models using the `into` keyword and apply new data later using the `apply` command.

... | apply temperature_model

You can inspect the coefficients learned by LinearRegression with the `summary` command.

... | summary temperature_model

**Example**

The following example uses LinearRegression on a test set.

... | fit LinearRegression temperature from date_month date_hour into temperature_model | ..

**RandomForestRegressor**

The RandomForestRegressor algorithm uses the scikit-learn RandomForestRegressor estimator to fit a model to predict the value of numeric fields. For descriptions of the `n_estimators`, `random_state`, `max_depth`, `max_features`, `min_samples_split`, and `max_leaf_nodes` parameters, see the scikit-learn documentation at http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestRegressor.html.

**Syntax**

```
fit RandomForestRegressor <field_to_predict> from <explanatory_fields>
[into <model name>] [n_estimators=<int>] [max_depth=<int>]
[random_state=<int>]
[max_features=<str>] [min_samples_split=<int>] [max_leaf_nodes=<int>]
```

You can save RandomForestRegressor models using the `into` keyword and apply new data later using the `apply` command.

... | apply temperature_model

You can list the features that were used to fit the model, as well as their relative importance or influence with the `summary` command.

... | summary temperature_model

**Example**

The following example uses RandomForestRegressor on a test set.
Ridge

The Ridge algorithm uses the scikit-learn Ridge estimator to fit a model to predict the value of numeric fields. Ridge is like LinearRegression, but it uses L2 regularization to learn a linear models with smaller coefficients, making the algorithm more robust to collinearity. For descriptions of the fit_intercept, normalize, and alpha parameters, see the scikit-learn documentation at http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.Ridge.html.

Parameters

The alpha parameter specifies the degree of regularization. The default value is 1.0.

Syntax

fit Ridge <field_to_predict> from <explanatory_fields> [into <model name>] [fit_intercept=<true|false>] [normalize=<true|false>] [alpha=<int>]

You can save Ridge models using the into keyword and apply new data later using the apply command.

apply temperature_model

You can inspect the coefficients learned by Ridge with the summary command.

summary temperature_model

Example

The following example uses Ridge on a test set.

SGDRegressor

The SGDRegressor algorithm uses the scikit-learn SGDRegressor estimator to fit a model to predict the value of numeric fields. This algorithm supports incremental fit.

Parameters
The `partial_fit` parameter controls whether an existing model should be incrementally updated or not. This allows you to update an existing model using only new data without having to retrain it on the full training data set. The default is False.

The `fit_intercept=<true|false>` parameter determines whether the intercept should be estimated or not.

The `fit_intercept=<true|false>` parameter default is True.

The `n_iter=<int>` parameter is the number of passes over the training data also known as epochs. The default is 5.

- The number of iterations is set to 1 if using `partial_fit`.

The `penalty=<l2|l1|elasticnet>` parameter set the penalty or regularization term to be used. The default is `l2`.

The `learning_rate=<constant|optimal|invscaling>` parameter is the learning rate.

- `constant`: \( \text{eta} = \text{eta0} \)
- `optimal`: \( \text{eta} = 1.0/(\text{alpha} \times t) \)
- `invscaling`: \( \text{eta} = \text{eta0} / \text{pow}(t, \text{power}_t) \)
- default is `invscaling`.

The `l1_ratio=<float>` parameter is the Elastic Net mixing parameter, with \( 0 \leq l1\_ratio \leq 1 \). Default is 0.15.

- `l1\_ratio=0` corresponds to L2 penalty
- `l1\_ratio=1` to L1

The `alpha=<float>` parameter is the constant that multiplies the regularization term. Default is 0.0001.

- Also used to compute `learning_rate` when set to `Optimal`.

The `eta0=<float>` parameter is the initial learning rate. Default is 0.01.

The `power_t=<float>` parameter is the exponent for inverse scaling learning rate. Default is 0.25.

The `random_state=<int>` parameter is the seed of the pseudo random number generator to use when shuffling the data.

**Syntax**

```plaintext
fit SGDRegressor <field_to_predict> from <explanatory_fields>
[into <model name>] [partial_fit=<true|false>]
[fit_intercept=<true|false>]
[random_state=<int>] [n_iter=<int>] [l1_ratio=<float>]
[alpha=<float>] [eta0=<float>] [power_t=<float>]
[penalty=<l1|l2|elasticnet>]
[learning_rate=<constant|optimal|invscaling>]
```

You can save SGDRegressor models using the `into` keyword and apply new data later using the `apply` command.

```plaintext
... | apply temperature_model
```

173
You can inspect the coefficients learned by SGDRegressor with the `summary` command.

```
... | summary temperature_model
```

**Syntax constraints**

- If `My_Incremental_Model` does not exist, the command saves the model data under the model name `My_Incremental_Model`.
- If `My_Incremental_Model` exists and was trained using SGDRegressor, the command updates the existing model with the new input.
- If `My_Incremental_Model` exists but was not trained by SGDRegressor, an error message displays.
- Using `partial_fit=true` on an existing model ignores the newly supplied parameters. The parameters supplied at model creation are used instead.
- If `partial_fit=false` or `partial_fit` is not specified the model specified is created and replaces the pre-trained model if one exists.

**Examples**

The following example uses SGDRegressor on a test set.

```
... | fit SGDRegressor temperature from date_month date_hour into temperature_model | ...
```

The following example includes the `partial_fit` parameter.

```
| inputlookup server_power.csv | fit SGDRegressor "ac_power" from "total-cpu-utilization" "total-disk-accesses" partial_fit=true into My_Incremental_Model
```

**Time Series Analysis**

Forecasting algorithms, also known as time series analysis, provide methods for analyzing time series data in order to extract meaningful statistics and other characteristics of the data, and forecast its future values.

**ARIMA**

The Autoregressive Integrated Moving Average (ARIMA) algorithm uses the StatsModels ARIMA algorithm to fit a model on a time series for better understanding and/or forecasting its future values. An ARIMA model can consist of autoregressive terms, moving average terms, and differencing operations. The autoregressive terms express the dependency of the current value of time series to its previous ones.
The moving average terms, also called random shocks or white noise, model the effect of previous forecast errors on the current value. If the time series is non-stationary, differencing operations are used to make it stationary. A stationary process is a stochastic process in that its probability distribution does not change over time.

See the StatsModels documentation at http://statsmodels.sourceforge.net/devel/generated/statsmodels.tsa.arima_model.ARIMA.html for more information.

It is highly recommended to send the time series through timechart before sending it into ARIMA to avoid non-uniform sampling time. If _time is not to be specified, using timechart is not necessary.

Parameters

- The time series should not have any gaps or missing data otherwise ARIMA will complain. If there are missing samples in the data, using a bigger span in timechart or using streamstats to fill in the gaps with average values can do the trick.
- When chaining ARIMA output to another algorithm (i.e. ARIMA itself), keep in mind the length of the data is the length of the original data + forecast_k. If you want to maintain the holdback position, you need to add the number in forecast_k to your holdback value.
- ARIMA requires the order parameter to be specified at fitting time. The order parameter needs three values:
  - Number of autoregressive (AR) parameters
  - Number of differencing operations (D)
  - Number of moving average (MA) Parameters
- The forecast_k=<int> parameter tells ARIMA how many points into the future should be forecasted. If _time is specified during fitting along with the field_to_forecast, ARIMA will also generate the timestamps for forecasted values. By default, forecast_k is zero.
- The conf_interval=<1..99> parameter is the confidence interval in percentage around forecasted values. By default it is set to 95%.
- The holdback=<int> parameter is the number of data points held back from the ARIMA model. This is useful for comparing the forecast against known data points. By default, holdback is zero.

Syntax

```
fit ARIMA [ _time ] <field_to_forecast> order=<int>-<int>-<int> [ forecast_k=<int> ] [ conf_interval=<int> ] [ holdback=<int> ]
```
Syntax constraints

- ARIMA supports one time series at a time.
- ARIMA models cannot be saved and used at a later time in the current version.

Example

The following example uses ARIMA on a test set.

... | fit ARIMA Voltage order=4-0-1 holdback=10 forecast_k=10

StateSpaceForecast

StateSpaceForecast is a forecasting algorithm for time series data in the MLTK. It is based on Kalman filters. The algorithm supports incremental fit.

Advantages of StateSpaceForecast over ARIMA include:

- Persists models created using the fit command that can then be used with apply.
- A specialdays field allows you to account for the effects of a specified list of special days.
- It is automatic in that you do no need to choose parameters or mode.
- Supports multivariate forecasting.

Parameters

- By default the historical data results from running the fit command are not shown. To modify this behavior set output_fit=True.
- Use the target field to specify fields from which to forecast using historical data and other values.
- The target field is a comma-separated list of fields that can be univariate or multivariate. These fields must be specified during the fit process.
  - Optionally use the target field to fit multiple fields during the fit process but apply only a selection of those target fields during the apply process.
- If the target field is not specified, then all fields will be forecast together using historical data.
- The specialdays field specifies the field that indicates effects due to special days such as holidays.
- The specialdays field values must be numeric and are typically 0 and 1, with 1 indicating the existence of a special day effect. Null values are treated as 0.
The majority of use cases have no special days. Events that occur regularly and frequently such as weekends should not be treated as special days. Use special days to capture events such as holiday sales.

- Use special days in the apply step if it has been specified during fit. The same field(s) must be assigned.
- Use the period parameter to specify if your data has a known periodicity.
- If the period parameter is not specified it is computed automatically.
- Set period=1 to treat the time series as non-periodic.
- As with other MLTK algorithms, the partial_fit parameter controls whether a model should be incrementally updated or not. This allows you to update a model using only new data without having to retrain the model on the full dataset.
- The default for partial_fit is False.
- Use update_last to modify the behavior of partial_fit.
- The default for update_last is False.
- If partial_fit=True StateSpaceForecast will first update the model parameters and then predict.
- If partial_fit=True and update_last=True StateSpaceForecast will first predict and then update the model parameters. This allows you to review the forecast before running new data through.
- The conf_interval=<1..99> parameter is the confidence interval in percentage around forecasted values. Input an integer between 1 and 99 where a larger number means a greater tolerance for forecast uncertainty. The default integer is 95.
- The as field to gives aliases for forecasted fields.
- In univariate cases the as field field-list is a single field name.
- In multivariate cases, the as field adheres to the following conventions:
  - The list must be in double quotes, separated by either spaces or commas.
  - The aliases correspond to the original fields in the given order.
  - The number of aliases can be smaller than the number of original fields.
- The summary command lists the names of the fields used in the fit command step, the name of the special days field, and the period.
- The holdback parameter is the number of data points held back from training. This is useful for comparing the forecast against known data points. Default holdback value is 0.
- If you want to maintain the holdback position, add the position number in forecast_k to your holdback value.
- The forecast_k parameter tells StateSpaceForecast how many points into the future should be forecasted. If _time is specified during fitting along with the field_to_forecast, StateSpaceForecast also generates the timestamps for forecasted values. Default, forecast_k value is 0.
- The `holdback` and `forecast_k` values can be of two types: an integer or a time range.
  - An integer specifies a number of events. An example of `forecast_k=10` forecasts 10 events into the future. An example of `holdback=10` withholds the last 10 events from training.
  - A time range takes the form `XY` where `X` is a non-negative integer and `Y` is either empty or adheres to format in the time range table. If `Y` is empty, then the time range is instead interpreted as an integer or a number of events. An example of `holdback=3day` `forecast_k=1week` withholds 3 days of events and forecasts 1 week's worth of events.

The actual number of events withheld and forecasted using the time range option depends on the time interval between consecutive events.

<table>
<thead>
<tr>
<th>Time range</th>
<th>Acceptable formats for Y value</th>
</tr>
</thead>
<tbody>
<tr>
<td>seconds</td>
<td>s, sec, secs, second, seconds</td>
</tr>
<tr>
<td>minutes</td>
<td>m, min, minute, minutes</td>
</tr>
<tr>
<td>hours</td>
<td>h, hr, hrs, hour, hours</td>
</tr>
<tr>
<td>days</td>
<td>d, day, days</td>
</tr>
<tr>
<td>weeks</td>
<td>w, week, weeks</td>
</tr>
<tr>
<td>months</td>
<td>mon, month, months</td>
</tr>
<tr>
<td>quarters</td>
<td>q, qtr, qtrs, quarter, quarters</td>
</tr>
<tr>
<td>years</td>
<td>y, yr, yrs, year, years</td>
</tr>
</tbody>
</table>

**Syntax**

```
| fit StateSpaceForecast <fields> [from *] [specialdays=<field name>] [holdback=<int | time-range>] [forecast_k=<int | time-range>] [conf_interval=<float>] [period=<int>] [partial_fit=<true|false>] [update_last=<true|false>] [output_fit=<true|false>] [into <model name>] [as <field-list>]
```

You can apply the saved model to new data with the `apply` command.

```
| apply <model name> [specialdays=<field name>] [target=<fields>] [holdback=<int | time-range>] [forecast_k=<int | time-range>] [conf_interval=<float>]
```

You can inspect the model learned by StateSpaceForecast with the `summary` command.

```
| summary <model name>
```
Syntax constraints

- For univariate analysis the fields parameter is a single field, but for multivariate analysis it is a list of fields.
- For multivariate analysis, only one specialdays field can be specified and it applies to all the fields.
- The specialdays field values must be numeric.
- Null values in the specialdays field are treated as 0.
- Double quotes are required around field lists.

Examples

The following is a univariate example of StateSpaceForecast. The example is considered univariate as there is only a single field following fit StateSpaceForecast. The example dataset is derived from the milk.csv dataset that ships with the toolkit. The milk2.csv has a new column named holiday. This column has two values 0 and 1. The 0 value represents no holiday and 1 value represents a holiday for the associated date. The 1 values were set randomly.

```
| inputlookup milk2.csv
| fit StateSpaceForecast milk_production from * specialdays=holiday into milk_model
| apply milk_model specialdays=holiday forecast_k=30
```

The following is a multivariate example of StateSpaceForecast on a test set. The syntax is the same as that in the univariate example, except that this case has a list of fields (CRM, ERP, and Expenses) following fit StateSpaceForecast, making it multivariate.

```
| inputlookup app_usage.csv
| fields CRM ERP Expenses
| fit StateSpaceForecast CRM ERP Expenses holdback=12 into app_usage_model as "crm, erp"
```

The following example is also multivariate and includes the target field. In this example the fields of CRM and ERP are forecast using historical data and the Expenses field. The apply command is used against the model created in the fit command step, resulting in the app_usage_model model.

```
| inputlookup app_usage.csv
| fields CRM ERP Expenses
| apply app_usage_model target="CRM, ERP" forecast_k=36 holdback=36
```
The following example is again multivariate but without the target field. This
example forecasts the fields CRM, ERP, and Expenses using historical data.

| inputlookup app_usage.csv
| fields CRM ERP Expenses
| apply app_usage_model forecast_k=36 holdback=36

The following example shows how to improve your output with
StateSpaceForecast.

| inputlookup cyclical_business_process_with_external_anomalies.csv
| eval holiday=if(random()%100<98,0,1)
| fit StateSpaceForecast logons from logons into My_Model
forecast_k=3000
Adding of the SPL line period=2016 could improve the output, but would not

account for the period being seven days rather than twenty-four hours.

| inputlookup cyclical_business_process_with_external_anomalies.csv
| table _time,logons
| eval holiday=if(random()%100<98,0,1)
| eval dayOfWeek=strftime(_time,"%a")
| eval holidayWeekend=case(in(dayOfWeek,"Sat","Sun"),1,true(),0)
| apply MyBadModel specialdays=holidayWeekend forecast_k=3000
| eval old_predict='predicted(logons)'
| eval dayOfWeek=strftime(_time,"%a")
| eval holidayWeekend=case(in(dayOfWeek,"Sat","Sun"),1,true(),0)
| apply My_Model specialdays=holidayWeekend holdback=3000
forecast_k=3000

Utility Algorithms
These utility algorithms are not machine learning algorithms, but provide
methods to calculate data characteristics. These algorithms facilitate the process
of algorithm selection and parameter selection. See the StatsModels
documentation at
for more information.
ACF (autocorrelation function)
ACF (autocorrelation function) calculates the correlation between a sequence
and a shifted copy of itself, as a function of shift. Shift is also referred to as lag
or delay.
Parameters
180


• The \(k\) parameter specifies the number of lags to return autocorrelation for. By default \(k\) is 40.
• The \(fft\) parameter specifies whether ACF is computed via Fast Fourier Transform (FFT). By default \(fft\) is False.
• The \(conf\_interval\) parameter specifies the confidence interval in percentage to return. By default \(conf\_interval\) is set to 95.

Syntax

```bash
fit ACF <field> [k=<int>] [fft=true|false] [conf_interval=<int>]
```

Example

The following example uses ACF (autocorrelation function) on a test set.

```bash
... | fit ACF logins k=50 fft=true conf_interval=90
```

**PACF (partial autocorrelation function)**

PACF (partial autocorrelation function) gives the partial correlation between a sequence and its lagged values, controlling for the values of lags that are shorter than its own. See the StatsModels documentation at [http://www.statsmodels.org/stable/generated/statsmodels.tsa.stattools.pacf.html](http://www.statsmodels.org/stable/generated/statsmodels.tsa.stattools.pacf.html) for more information.

Parameters

• The \(k\) parameter specifies the number of lags to return partial autocorrelation for. By default \(k\) is 40.
• The \(method\) parameter specifies which method for the calculation to use. By default \(method\) is unbiased.
• The \(conf\_interval\) parameter specifies the confidence interval in percentage to return. By default \(conf\_interval\) is set to 95.

Syntax

```bash
fit PACF <field> [k=<int>] [method=ywunbiased|ywml|ols] [conf_interval=<int>]
```

Example

The following example uses PACF (partial autocorrelation function) on a test set.

```bash
... | fit PACF logins k=20 conf_interval=90
```
Import a machine learning algorithm from Splunkbase

The process to import a machine learning algorithm from Splunkbase is the same as downloading an application from Splunkbase. If you are unfamiliar with downloading an application from Splunkbase, see Installing Add-ons in Supported Add-ons.

Use the algorithm

To use the algorithm, reference it with the same SPL commands as the algorithms that shipped with the Splunk Machine Learning Toolkit. You can use the algorithm within the Machine Learning Toolkit, the application that hosts the algorithm, and any other add-on.

Algorithm permissions

In the Splunk Machine Learning Toolkit, you can define the read permissions for each algorithm. Read permissions dictate whether a user can access the algorithms. Administrators have write permissions by default. To change permission settings, either modify the REST API ACL endpoint in the `algos.conf` file, or modify the manual configuration of `local.meta` file.

Manage permissions in REST API

The example below uses the algorithm KMeans, although this process works the same way with any algorithm in the Splunk Machine Learning Toolkit.

1. To change the algorithm settings, modify the API ACL endpoint in the `algos.conf` file.
   1. To assign read permissions to all roles:

   ```
   $ curl -k -u admin:<admin password> \
   https://localhost:8089/servicesNS/nobody/Splunk_ML_Toolkit/configs/conf-algos/KMeans/acl \
   -d owner=nobody \
   -d sharing=app \
   -d perms.read=* 
   ```

2. To assign read permissions to the user and power role:

   ```
   $ curl -k -u admin:<admin password> \
   https://localhost:8089/servicesNS/nobody/Splunk_ML_Toolkit/configs/conf-algos/KMeans/acl \
   ```
You do not need to restart splunkd after you change the permission settings in the REST API.

The read permission in the REST API also controls the GET method. If you are unfamiliar with the REST method subset, see HTTP Status Codes in the Rest API User Manual. For more information on ACL REST API endpoint, see Access Control list in the Rest API User Manual.

**Manage permissions manually in the configuration file**

You cannot modify the default.meta file directly. Instead, you can make changes in the local.meta file, and the new content will override the default.meta file. For further information on the structure of the default.meta file, see default.meta.conf in the Admin Manual.

The example below uses the algorithm KMeans, although this process works the same way with any algorithm in the Splunk Machine Learning Toolkit.

1. If you do not have a local.meta file, create one.
2. Open the local.meta file.

   ```bash
   $SPLUNK_HOME/etc/apps/Splunk_ML_Toolkit/metadata/local.meta
   ```

   1. To assign read permissions to all roles:

      ```
      [algos/KMeans]
      access = read : [ * ]
      ```

   2. To assign read permissions to the user and power roles:

      ```
      [algos/KMeans]
      access = read : [ user,power ]
      ```

3. Restart splunkd.

The process to change permissions in the REST API endpoint also changes the content of a local.meta file in the directory.

```bash
$SPLUNK_HOME/etc/apps/Splunk_ML_Toolkit/metadata
```
Smart Assistants

Smart Forecasting Assistant

The Smart Forecasting Assistant enables machine learning outcomes for users with little to no SPL knowledge. Introduced in version 4.3.0 of the Machine Learning Toolkit, this new Assistant is built on the backbone of the Experiment Management Framework (EMF), offering enhanced time-series forecasting abilities. The Smart Forecasting Assistant offers a segmented, guided workflow with an updated user interface. Move through the stages of Define, Learn, Review, and Operationalize to load data, build your model, and put that model into production. Each stage offers a data preview and visualization panel.

This Assistant leverages the StateSpaceForecast algorithm which persists a model using the `fit` command that can be used with the `apply` command. StateSpaceForecast is based on Kalman filters, supports incremental fit, and automatically imputes any missing values in your data. To help improve the accuracy of your forecast, this algorithm includes the ability to account for the effects of specific days that need to be treated differently.

To learn more about the Smart Forecasting Assistant algorithm, see StateSpaceForecast algorithm.

The Smart Forecasting Assistant only supports univariate forecasting.

Smart Forecasting Assistant Showcase

You can gain familiarity of this new Assistant through the MLTK Showcase, accessed under its own tab. The three new Showcase examples include:

- Forecast the Number of Calls to a Call Center
- Forecast App Logons with Special Days
- Forecast App Expenses
Click the name of any Smart Forecasting Showcase to see this new Assistant and its updated interface using pre-loaded test data and pre-selected forecast parameters.

Unlike previous Showcase examples, Smart Forecasting Assistant showcases require you to click to continue the demonstration. Please refer to the following screenshots to learn where and when to take action to move the Showcase forward.

**Forecast the Number of Calls to a Call Center Showcase example**

The Showcase begins in the Define stage. The call center data set that ships with the MLTK is pre-loaded into the search panel for you. Click **Next** to continue the walk-through.
Clicking Next moves you into the Learn stage. Open the Smart forecasting menu to see the pre-selected forecasting options. You can hover over any helper icon in the Smart forecasting menu to learn more about a particular field and its function. Click Run to view these forecasting parameters on the test data and then click Next to continue the walk-through. Alternately, click Edit to view these fields in edit mode, followed by Forecast to view the these forecasting parameters on the test data, then Next.

Clicking Next moves you to the Review stage. The top of the page displays the forecasting parameters set in the Learn stage in plain English. This stage enables you to review your forecast settings prior to putting this model into production. Navigate back to the Define or Learn stages as needed to adjust the results at the Review stage. You do not see an option to save work here, as this is only a Showcase. When working in the Assistant itself, options to save as well as to operationalize the model are available.

Click Cancel to return to the main Showcase page.
Smart Forecasting Assistant workflow

Move through the stages of Define, Learn, Review, and Operationalize to draw in data, build your model, and put that model into production.

This document uses the example dataset of call_center.csv that ships with the MLTK. You can use this dataset or another of your choice to explore the Smart Forecasting Assistant and its features before building a model with your own data.

To begin, select Smart Forecasting from the Experiments landing page and the Create New Experiment button in the top right.

Enter an Experiment Title, and optionally add a Description. Click Create to move into the Assistant interface.

**Define**

Use the Define stage to select and preview the data you wish to use for the forecast. You can pull in data from anywhere in the Splunk platform. Use the Search bar as you would with any other Experiment Assistant. You can use the Search bar to modify your dataset data in advance of using it within the Learn step.
As an alternative to accessing data via Search, you can choose the Datasets option. Under Datasets, you can find any data you have ingested into Splunk, as well as any datasets that ship with Splunk Enterprise and the Machine Learning Toolkit. You can filter by type to find your preferred data faster.

As with other Experiment Assistants, the Smart Forecasting Assistant includes a time-range picker to narrow down the data time-frame to a particular date or date range. The default setting of 'All time' can be changed to suit your needs. Once data is selected, the Data Preview and Visualization tabs populate.

Following the selection of data, the Learn stage becomes the available next step. Click Learn in the left menu, or Next in the top right of the page to continue.
**Learn**

Use the Learn stage to perform any preprocessing on your data, and to create your forecasting model. Select from the interactive learning steps to customize and complete the forecasting outcome.

The Learn stage begins with the Initial data menu open, showing the successful addition of data from the Define stage with a green check mark icon. A preview and visual evaluation of the data is available in the main page body.

Choose your forecast parameters using the **Smart forecasting** menu. Click **Edit** to add or update the available fields.
You can refer to the following table for details of each available field. Certain fields are required. Hovering over the question mark helper icons beside each field also provides field descriptions.

<table>
<thead>
<tr>
<th>Field name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Field to forecast</td>
<td>Required field. Select a numeric field to forecast.</td>
</tr>
<tr>
<td>Holdback period</td>
<td>Required field. Holdback is the number of data points held back from training to compare the forecast against known values. This comparison is done using R2 and RMSE statistics.</td>
</tr>
<tr>
<td>Future timespan</td>
<td>Required field. Pick number of days you want to forecast into the future. The further into the future the forecast, the less accurate it is likely to be.</td>
</tr>
<tr>
<td>Confidence interval</td>
<td>Required field. Specify and integer between 0 and 99, where a larger value means a greater tolerance for forecast uncertainty.</td>
</tr>
<tr>
<td>Special days field</td>
<td>Optional field. Accessed using the Join special time entries preprocessing step. Special days data can improve your forecast by accounting for days which should be treated differently such as Black Friday sales or IP traffic on July 4th.</td>
</tr>
<tr>
<td>Period</td>
<td>Optional field. Specify if the data has a known periodicity. The units of the period are equal to the span of the _time field. For example, hourly data may have a period of 24 (one day), whereas daily data may have a period of 7 (one week).</td>
</tr>
<tr>
<td>Notes</td>
<td>Optional field. Use this free form block of text to track the selections made in the parameter fields. Refer back to notes to review which parameter combinations yield the best results.</td>
</tr>
</tbody>
</table>

The fields for Holdback period and Future timespan allow you to choose the measurement value from a pre-populated list. Click **Forecast** to see a preview of your parameter settings.
Clicking **Forecast** produces a preview and visualization of the forecast. A plain English description of the chosen parameters also displays at the top of the screen. Use the **Edit** button to further adjust your forecast parameters.

Leveraging the StateSpaceForecast algorithm provides the option to take company or business calendar specific days into account when building your forecasting model. Include these special days in your forecast using the **Join special time entries** preprocessing step option.

Add any special days data using joins from CSV lookups. For details on how to work with data from CSV lookups, see Define a CSV lookup in Splunk Web.
As you work through the Smart Forecasting Assistant, SPL is created for you and can be viewed via the **SPL** button.

Changes to the Experiment are also tracked and can be viewed by clicking **View History**.

When you are happy with the results from the parameter settings in the Learn stage, click **Next** in the top right of the page to continue.
**Review**

From the Review stage, you can see the forecast accuracy based on the parameters selected at the Learn stage. The panels on the Review page let you confirm you are seeing good results prior to putting this model into production.

Use the provided model statistics from $R^2$ and RMSE to assess model accuracy and error rate. You can also choose to set a threshold for the amount of time before a metric reaches a certain value. Navigate back to the Learn stage to make forecast adjustments, or click **Save and Next** to continue.

After you click **Save and Next** a modal window offers the opportunity to update the Experiment name or description. When ready, click **Save**.

**Operationalize**

The Operationalize stage provides publishing, alerting, and scheduled training in one place. Click **Done** to move to the Experiments listings page.
The Experiments listing page provides a place to publish, set up alerts, and schedule training for any of your saved Experiments across all Assistant types including Smart Forecasting.

Learn more

To learn about implementing analytics and data science projects using Splunk's statistics, machine learning, built-in and custom visualization capabilities, see the Splunk for Analytics and Data Science course.
Experiments

Experiments overview

Introduced in version 3.2 of the Splunk Machine Learning Toolkit (MLTK), the Experiment Management Framework (EMF) brings all aspects of a monitored machine learning pipeline into one interface with automated model versioning and lineage baked in.

Experiment workflow

The Experiment workflow begins with the creation of a new machine learning pipeline, based on the selected MLTK guided modeling interface or Assistant. There are six Experiment Assistants to select from including Predict Numeric Fields, Predict Categorical Fields, Detect Numeric Outliers, Detect Categorical Outliers, Forecast Time Series, and Cluster Numeric Events.

Once you select and apply Experiment parameters to your data and generate results, the workflow continues through the available visualizations and statistical analysis. In this way, Experiments are similar to the Classic Assistant workflow. Through the guided Experiment Assistant you make selections including:

• Specifying your data sources.
• Selection of an algorithm and algorithm parameters.
• Selection of the fields for the algorithms to analyze.
• Setting training/test data splits.

Every step of an Experiment has tool-tips as additional guides, the option to see the SPL as written by the Experiment with explanations for the commands, and an option to open a clone of the SPL in a new search window for further customization.

Saved Experiments

Once you save an Experiment, a new exclusive knowledge object is created in the Splunk platform that keeps track of all the settings for that pipeline, as well as its affiliated alerts and scheduled trainings.

Save your work prior to scheduling a training job for the Experiment, managing alerts for an Experiment, or deploying an Experiment.
This saved knowledge object enables you to:

- Organize your Experiment around solving a business problem with machine learning.
- Keep all of your modeling history and experimentation in one place.

Unlike most knowledge objects in the Splunk platform, Experiments are bound to the user who creates them and cannot be shared. Please use the publish or export options to share Experiments that persist models with another app or user.

You can operationalize your persisted models to other SPL workflows in the Splunk platform through the publish functionality, as well as create alerts for any Experiments saved within the framework. When creating alerts, select from standard Trigger Conditions, or from Machine Learning Conditions that are specific to your Experiment and the Experiment Assistant.

The following table lists the Machine Learning trigger conditions as available by Experiment Assistant.

<table>
<thead>
<tr>
<th>Experiment Assistant</th>
<th>Machine Learning Condition Options</th>
</tr>
</thead>
<tbody>
<tr>
<td>Predict Numeric Fields Experiment Assistant</td>
<td>Triggers based on the predicted value during a scheduled search. Triggers based on the residual value during a scheduled search. Triggers based on the R square value during a scheduled search.</td>
</tr>
</tbody>
</table>
### Experiment Assistant

<table>
<thead>
<tr>
<th>Experiment Assistant</th>
<th>Machine Learning Condition Options</th>
</tr>
</thead>
<tbody>
<tr>
<td>Predict Categorical Fields Experiment Assistant</td>
<td>Triggers based on the value of predicted field during a scheduled search. Triggers based on whether the predicted categorical value matches the actual value during a scheduled search.</td>
</tr>
<tr>
<td>Detect Numeric Outliers Experiment Assistant</td>
<td>Triggers based on the outlier number being greater than threshold during a scheduled search.</td>
</tr>
<tr>
<td>Detect Categorical Outliers Experiment Assistant</td>
<td>Triggers based on the number of outliers during a scheduled search.</td>
</tr>
<tr>
<td>Forecast Time Series Experiment Assistant</td>
<td>Triggers based on the predicted value during a scheduled search.</td>
</tr>
<tr>
<td>Cluster Numeric Events Experiment Assistant</td>
<td>Triggers based on the number of clusters during a scheduled search. Triggers based on the <code>cluster_distance</code> during a scheduled search. Triggers based on the range of cluster ID on a scheduled search.</td>
</tr>
</tbody>
</table>

### Experiment composition

Each Experiment contains the following sections. These vary slightly depending on the type of machine learning analytic being performed.

- **Create or Detect:** Follow the workflow laid out in the Experiment to create a new model or forecast, or detect outliers. The workflow depends on the type of analytic but usually includes performing a lookup on a dataset, selecting a field to predict or analyze, and selecting fields or values to use for performing different types of analysis.
- **Raw Data Preview:** This section is displayed for predictions and forecasts to show you the data that is being used.
- **Validate:** Use the tables and visualizations to determine how well the model was fitted, how well outliers were detected, or how well a forecast performed.
- **Deploy:** Click the buttons beneath the visualizations and tables to see different ways to use the analysis. For example, you can open the search in the Search app, show the SPL, or create an alert. Experiments that persist a model include the option to publish from within the EMF.
• Experiment History tab: Each time you use an Experiment, a history is captured of the settings used. Compare the effects of different searches, algorithms and parameters, and identify the best choices for your use. As a model learns over time, the Experiment monitors models to assess for an increase or decrease in accuracy. This average amount of error within the mathematical model is also captured within this history tab.

**Experiment commands**

Experiment Assistants use machine learning SPL commands. Commands use varies based on which Experiment Assistant is selected:

• Predict Numeric and Predict Categorical Fields use the `fit` and `apply` commands
• Forecast Time Series can use the `predict` command
• Cluster Numeric Events uses the `fit` and `apply` commands

**Choose the Experiment Assistant to suit your needs**

Experiments cover machine learning areas including Predict Numeric Fields, Predict Categorical Fields, Detect Numeric Outliers, Detect Categorical Outliers, Forecast Time Series, and Cluster Numeric Events. Choose an Experiment Assistant based on the type of machine learning you wish to perform on your data.
The Predict Numeric Fields Experiment Assistant uses regression algorithms to predict or estimate numeric values. Such models are useful for determining to what extent certain peripheral factors contribute to a particular metric result. After the regression model is computed, use these peripheral values to make a prediction on the metric result.

The Predict Categorical Fields Experiment Assistant displays a type of learning known as classification. A classification algorithm learns the tendency for data to belong to one category or another based on related data.

The Detect Numeric Outliers Experiment Assistant determines values that appear to be extraordinarily higher or lower than the rest of the data. Identified outliers are indicative of interesting, unusual, and possibly dangerous events. This Assistant is restricted to one numeric data field.

The Detect Categorical Outliers Experiment Assistant identifies data that indicate interesting or unusual events. This Assistant allows non-numeric and multi-dimensional data, such as string identifiers and IP addresses. To detect categorical outliers, input your data and select the fields from which to look for unusual combinations or a coincidence of rare values. When multiple fields have rare values, the result is an outlier.
• The Forecast Time Series Experiment Assistant forecasts the next values in a sequence for a single time series. Forecasting makes use of past time series data trends to make a prediction about likely future values. The result includes both the forecasted value and a measure of the uncertainty of that forecast.

• The Smart Forecasting Assistant is a new feature shipping with version 4.3.0 of the MLTK, offering a new guided interface to forecast future numeric time-series data. This new Assistant is built on the backbone of the Experiment Management Framework but offers an updated look and feel as well as the option to bring in data from different sources to build your model. This is the first in a new series of Assistants that will be delivered over the coming months.

• The Cluster Numeric Events Experiment Assistant partitions events into groups of events based on the values of those fields. As the groupings are not known in advance, this is considered unsupervised learning.

**Predict Numeric Fields Experiment workflow**

Experiments manage the data source, algorithm, and the parameters to configure that algorithm within one framework. An Experiment is an exclusive knowledge object in the Splunk platform that keeps track of its settings and history, as well as its affiliated alerts and scheduled trainings. Experiment Assistants enable machine learning through a guided user interface.

The following visualization illustrates a scatter plot of the actual versus predicted results. This visualization is from the Showcase example for Server Power Consumption.
Available algorithms

The Predict Numeric Fields Experiment Assistant uses the following algorithms:

- Linear Regression
- RandomForestRegressor
- Lasso
- KernelRidge
- ElasticNet
- Ridge
- DecisionTreeRegressor

Create an Experiment to predict a numeric field

The Predict Numeric Fields Experiment Assistant uses regression algorithms to predict numeric values from other fields in the event. Use regression models to determine to what extent certain peripheral factors contribute to a particular metric result. After the Experiment Assistant computes the regression model, use these peripheral values to make a prediction on the metric result.

Before you begin

- The Predict Numeric Fields Assistant offers the option to preprocess your data. Read up on the preprocessing algorithms available here: Preprocessing machine data using Assistants.
- The toolkit default selects the Linear Regression algorithm. Use this default if you aren't sure which one is best for you. To learn about your other algorithm options see, Algorithms.
**Assistant workflow**

Follow these steps to create a Predict Numeric Fields Experiment.

1. From the MLTK navigation bar, click **Experiments**.
   - If this is the first Experiment in the toolkit, you will land on a display screen of all six Assistants. Select the **Predict Numeric Fields** block.
   - If you have at least one Experiment in the toolkit, you will land on a list view of Experiments. Click the **Create New Experiment** button.
2. Fill in an Experiment Title, and add a description. Both the name and description can be edited later if needed.
3. Click **Create**.
4. Run a search and be sure to select a date range.
5. (Optional) Click **Add a step** to add preprocessing steps.
6. Select an algorithm from the **Algorithm** drop-down menu.
   - LinearRegression is selected by default. Another algorithm option may better fit your Experiment.

![Algorithm options](image)

7. Select a target field from the drop-down menu **Field to Predict**. When you select the **Field to predict**, the **Fields to use for predicting** drop-down menu populates with available fields to include in your model.
8. Select a combination of fields from the drop-down menu **Fields to use for predicting**. In the following server power showcase example, the drop-down menu contains a list of all the possible fields used to predict ac_power using the linear regression algorithm.
9. Use the slider bar to split your data into training and testing data. The default split is 50/50, and the data is divided randomly into two groups.

10. (Optional) Add notes to this Experiment. Use this free form block of text to track the selections made in the Experiment parameter fields. Refer back to notes to review which parameter combinations yield the best results.

The algorithm you select determines the fields available to build your model. Hover over any field name to get more information about that field.

11. Click **Fit Model**. The Experiment is now in a Draft state. Draft versions allow you to alter settings without committing or overwriting a saved Experiment. An Experiment is not stored to Splunk until it is saved.

The following table explains the differences between a draft and saved Experiment:

<table>
<thead>
<tr>
<th>Action</th>
<th>Draft Experiment</th>
<th>Saved Experiment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Create new record in Experiment history</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Run Experiment search jobs</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>(As applicable) Save and update Experiment model</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>(As applicable) Update all Experiment alerts</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>(As applicable) Update Experiment scheduled trainings</td>
<td>No</td>
<td>Yes</td>
</tr>
</tbody>
</table>

**Interpret and validate results**

After you fit the model, review the prediction results and visualizations to see how well the model predicted the numeric field. You can use the following methods to
evaluate your predictions:

<table>
<thead>
<tr>
<th>Charts and Results</th>
<th>Applications</th>
</tr>
</thead>
<tbody>
<tr>
<td>Actual vs. Predicted Scatter Chart</td>
<td>This visualization plots the predicted value against the raw actual values for the predicted field. The closer the points are to the line, the better the model. Hover over the blue dots to see actual values.</td>
</tr>
<tr>
<td>Residuals Histogram</td>
<td>This visualization shows the difference between the actual values and the predicted values. Hover over the predicted values (blue bars) to see the number of residual errors and the sample count values. Residuals commonly end on a bell curve clustered tightly around zero.</td>
</tr>
<tr>
<td>R² Statistic</td>
<td>This statistic shows how well the model explains the variability of the result. 100% (a value of 1) means the model fits perfectly. The closer the value is to 1, the better the result.</td>
</tr>
<tr>
<td>Root Mean Squared Error</td>
<td>This chart shows the variability of the result, which is the standard deviation of the residual. The formula takes the difference between the actual and predicted values, squares this value, takes an average, and then takes a square root. This value can be arbitrarily large and just gives you an idea of how close or far the model is. These values only make sense within one dataset and shouldn’t be compared across datasets.</td>
</tr>
<tr>
<td>Fit Model Parameters Summary</td>
<td>This summary displays the coefficients associated with each variable in the regression model. A relatively high coefficient value shows a high association of that variable with the result. A negative value shows a negative correlation.</td>
</tr>
<tr>
<td>Actual vs. Predicted Overlay</td>
<td>This overlay shows the actual values against the predicted values, in sequence.</td>
</tr>
<tr>
<td>Residuals</td>
<td>The residuals show the difference between predicted and actual values, in sequence.</td>
</tr>
</tbody>
</table>
Refine the Experiment

After you validate the results, refine the Experiment and run the `fit` command again. Optionally choose to track your changes in the Notes text field.

Consider the following options to better refine your Experiment:

1. Reduce the number of fields selected in the Fields to use for predicting drop-down menu. Having too many fields can generate a distraction.
2. Bring in new data sources to enrich your modeling space.
3. Build features on raw data, model on behaviors of the data instead of raw data points, using SPL, Streamstats, or eventstats.
4. Check your fields to ensure you are using categorical values correctly. For example are you using DayOfWeek as a number (0 to 6) instead of as "Monday", "Tuesday", and so forth? Make sure you have the right type of value for categorical fields.
5. Bring in context via lookups - holidays, external anomalies, etc.
6. Increase the number of fields (from additional data, feature building as above, etc) selected in the Fields to use for predicting drop-down menu.

Use the Experiment History tab to review settings and changes made as you refine the model.

The history of any scheduled model retraining is captured in the Experiment History tab.

Save the Experiment

Once you are getting valuable results from your Experiment, save it. Saving your Experiment results in the following actions:

1. Assistant settings saved as an Experiment knowledge object.
2. The Draft version saves to the Experiment Listings page.
3. Any affiliated scheduled trainings and alerts update to synchronize with the search SPL and trigger conditions.

You can load a saved Experiment by clicking the Experiment name.
Deploy the Experiment

Saved predict numeric fields Experiments include options to manage and publish.

Within the Experiment framework

From within the framework, you can both manage and publish your Experiments. To manage your Experiment, perform the following steps:

1. From the MLTK navigation bar, choose Experiments. A list of your saved Experiments populates.
2. Click the Manage drop-down menu which is available in the Actions column.

The toolkit supports the following Experiment management options:

- Create and manage Experiment-level alerts. Choose from both Splunk platform standard trigger conditions, as well as from Machine Learning Conditions related to the Experiment.
- Edit the title and description of the Experiment.
- Schedule a training job for an Experiment.
- Delete an Experiment.

Updating a saved Experiment can affect affiliated alerts. Re-validate your alerts once you complete the changes. For more information about alerts, see Getting started with alerts in the Splunk Enterprise Alerting Manual.

You can publish your Experiment through the following steps:

1. From the MLTK navigation bar, choose Experiments. A list of your saved Experiments populates.
2. Click the Publish button available under the Actions column.
   Publishing an Experiment model means the main model and any associated preprocessing models will be copied as lookup files in the user's namespace within a selected destination app. Published models
can be used to create alerts or schedule model trainings.

The Publish link will only show if you have created the Experiment and fit the model.
3. Give the model a title. It must start with letter or underscore, and only have letters, numbers, and underscores in the name.
4. Select the destination app.
5. Click Save.
6. A message will let you know whether the model published or why the action was not completed.

Experiments are always stored under the user's namespace, so changing sharing settings and permissions on Experiments is not supported.

**Outside the Experiment framework**

1. Click Open in Search to generate a New Search tab for this same dataset. This new search opens in a new browser tab, away from the Assistant. This search query uses all data, not just the training set. You can adjust the SPL directly and see the results immediately. You can also save the query as a Report, Dashboard Panel or Alert.
2. Click Show SPL to open a new modal window/overlay showing the search query you used to fit the model. Copy the SPL to use in other aspects of your Splunk instance.

**Learn more**

To learn about implementing analytics and data science projects using Splunk's statistics, machine learning, built-in and custom visualization capabilities, see the Splunk for Analytics and Data Science course.

**Predict Categorical Fields Experiment workflow**

Experiments manage the data source, algorithm, and the parameters to configure that algorithm within one framework. An Experiment is an exclusive knowledge object in the Splunk platform that keeps track of its settings and history, as well as its affiliated alerts and scheduled trainings. Experiment Assistants enable machine learning through a guided user interface.
The following classification table shows the actual state of the field versus predicted state of the field. The yellow bar highlights an incorrect prediction.

<table>
<thead>
<tr>
<th>DiskFailure</th>
<th>predicted(DiskFailure)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>No</td>
<td>No</td>
</tr>
</tbody>
</table>

**Available algorithms**

The Predict Categorical Fields assistant uses the following classification algorithms:

- LogisticRegression
- SVM
- RandomForestClassifier
- GaussianNB
- BernouliNB
- DecisionTreeClassifier

**Create an Experiment to predict a categorical field**

The Predict Categorical Fields Assistant displays a type of learning known as classification. A classification algorithm learns the tendency for data to belong to one category or another based on related data.

**Before you begin**

- The Predict Categorical Fields Assistant offers the option to preprocess your data. Read up on the preprocessing algorithms available here: Preprocessing machine data using Assistants.
- The toolkit default selects the Logistic Regression algorithm. Use this default if you aren’t sure which one is best for you. Read up on the other algorithm options here: Algorithms.
**Assistant workflow**

Follow these steps to create a Predict Categorical Fields Experiment.

1. From the MLTK navigation bar, click **Experiments**.
   - If this is the first experiment in the toolkit, you will land on a display screen of all six Assistants. Select the **Predict Categorical Fields** block.
   - If you have at least one experiment in the toolkit, you will land on a list view of experiments. Click the **Create New Experiment** button.
2. Fill in an Experiment Title, and (optionally) add a description. Both the name and description can be edited later if needed.
3. Click **Create**.
4. Run a search and be sure to select a date range.
5. (Optional) Click **+ Add a step** to add preprocessing steps.
6. Select an algorithm from the **Algorithm** drop-down menu. LogisticRegression is selected by default. Another algorithm option may better fit your Experiment.
7. Select a target field from the drop-down menu **Field to Predict**. When you select the **Field to predict**, the **Fields to use for predicting** drop-down menu populates with available fields to include in your model.
8. Select a combination of fields from the drop-down menu **Fields to use for predicting**.
9. Use the slider bar to split your data into training and testing data. The default split is 50/50, and the data is divided randomly into two groups.
10. (Optional) Add notes to this Experiment. Use this free form block of text to track the selections made in the Experiment parameter fields. Refer back to notes to review which parameter combinations yield the best results.

The algorithm you select determines the fields available to build your model. Hover over any field name to get more information about that field.

11. Click **Fit Model**. The experiment is now in a Draft state. Draft versions allow you to alter settings without committing or overwriting
a saved Experiment. An Experiment is not stored to Splunk until it is saved. The following table explains the differences between a draft and a saved Experiment.

<table>
<thead>
<tr>
<th>Action</th>
<th>Draft Experiment</th>
<th>Saved Experiment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Create new record in Experiment history</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Run Experiment search jobs</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>(As applicable) Save and update Experiment model</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>(As applicable) Update all Experiment alerts</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>(As applicable) Update Experiment scheduled trainings</td>
<td>No</td>
<td>Yes</td>
</tr>
</tbody>
</table>

**Interpret and validate results**

After you fit the model, review the prediction results and visualizations to see how well the model predicted the categorical field. In this analysis, metrics are related to mis-classifying the field, and are based on false positives and negatives, and true positives and negatives. You can use the following methods to evaluate your predictions:

<table>
<thead>
<tr>
<th>Charts and Results</th>
<th>Applications</th>
</tr>
</thead>
<tbody>
<tr>
<td>Precision</td>
<td>This statistic is the percentage of the time a predicted class is the correct class.</td>
</tr>
<tr>
<td>Recall</td>
<td>This statistic is the percentage of time that the correct class is predicted.</td>
</tr>
<tr>
<td>Accuracy</td>
<td>This statistic is the overall percentage of correct predictions.</td>
</tr>
<tr>
<td>F1</td>
<td>This statistic is the the weighted average of precision and recall, based on a scale from zero to one. The closer the statistic is to one, the better the fit of the model.</td>
</tr>
<tr>
<td>Classification Results</td>
<td>This table charts the number of actual results against predicted results, also known as a Confusion Matrix.</td>
</tr>
</tbody>
</table>
Refine the Experiment

After you validate your results, refine the Experiment and run the `fit` command again. Optionally choose to track your changes in the **Notes** text field.

Consider the following options to better refine your Experiment:

1. Reduce the number of fields selected in the `Fields to use for predicting` drop-down menu. Having too many fields can generate a distraction.
2. Bring in new data sources to enrich your modeling space.
3. Build features on raw data, model on behaviors of the data instead of raw data points, using SPL, Streamstats, or eventstats.
4. Check your fields to ensure you are using categorical values correctly. For example are you using `DayOfWeek` as a number (0 to 6) instead of as "Monday", "Tuesday", and so forth? Make sure you have the right type of value for categorical fields.
5. Bring in context via lookups - holidays, external anomalies, etc.
6. Increase the number of fields (from additional data, feature building as above, etc) selected in the `Fields to use for predicting` drop-down menu.

Use the **Experiment History** tab to review settings and changes made as you refine the model.

The history of any scheduled model retraining is captured in the Experiment History tab.

Save the Experiment

Once you are getting valuable results from your Experiment, save it. Saving your Experiment results in the following actions:

1. Assistant settings saved as an Experiment knowledge object.
2. The Draft version saves to the Experiment Listings page.
3. Any affiliated scheduled trainings and alerts update to synchronize with the search SPL and trigger conditions.

You can load a saved Experiment by clicking the Experiment name.

**Deploy the Experiment**

Saved predict categorical fields Experiments include options to manage and publish.

*Within the Experiment framework*

From within the framework, you can both manage and publish your Experiments. To manage your Experiment, perform the following steps:

1. From the MLTK navigation bar, choose **Experiments**. A list of your saved experiments populates.
2. Click the **Manage** button available under the Actions column.

The toolkit supports the following Experiment management options:

- Create and manage Experiment-level alerts. Choose from both Splunk platform standard trigger conditions, as well as from Machine Learning Conditions related to the Experiment.
- Edit the title and description of the Experiment.
- Schedule a training job for an Experiment.
- Delete an Experiment.

Updating a saved Experiment can affect affiliated alerts. Re-validate your alerts once you complete the changes. For more information about alerts, see Getting started with alerts in the Splunk Enterprise Alerting Manual.

You can publish your Experiment through the following steps:

1. From the MLTK navigation bar, choose **Experiments**. A list of saved Experiments populates.
2. Click the **Publish** button available under the Actions column. Publishing an experiment model means the main model and any associated preprocessing models will be copied as lookup files in the user's namespace within a selected destination app. Published models can be used to create alerts or schedule model trainings.

The Publish link will only show if you have created the experiment, and fit the model.

3. Give the model a title. It must start with letter or underscore, and only have letters, numbers, and underscores in the name.

4. Select the destination app.

5. Click **Save**.

6. A message will let you know whether the model has been published, or why the action was not completed.

Experiments are always stored under the user's namespace, so changing sharing settings and permissions on Experiments is not supported.

---

**Outside the Experiment framework**

1. Click **Open in Search** to generate a New Search tab for this same dataset. This new search opens in a new browser tab, away from the Assistant. This search query uses all data, not just the training set. You can adjust the SPL directly and see the results immediately. You can also save the query as a Report, Dashboard Panel or Alert.

2. Click **Show SPL** to generate a new modal window/overlay showing the search query you used to fit the model. Copy the SPL to use in other aspects of your Splunk instance.

---

**Learn more**

To learn about implementing analytics and data science projects using Splunk's statistics, machine learning, built-in and custom visualization capabilities, see the Splunk for Analytics and Data Science course.

---

**Detect Numeric Outliers Experiment workflow**

Experiments manage the data source, algorithm, and the parameters to configure that algorithm within one framework. An Experiment is an exclusive
knowledge object in the Splunk platform that keeps track of its settings and history, as well as its affiliated alerts and scheduled trainings. Experiment Assistants enable machine learning through a guided user interface.

In the following visualization, the yellow dots indicate outliers.

![Data and Outliers](image)

**Available algorithms**

The Detect Numeric Outliers Assistant is compatible with the following distribution statistics:

- Standard deviation
- Median absolute deviation
- Interquartile range

**Create an Experiment to detect numeric outliers**

The Detect Numeric Outliers Experiment Assistant determines values that appear to be extraordinarily higher or lower than the rest of the data. Identified outliers are indicative of interesting, unusual, and possibly dangerous events. This Assistant is restricted to one numeric data field.

Input the data and select the parameters you want to investigate. When a situation violates the expectations for a parameter, it results in an outlier.

**Assistant workflow**

Follow these steps to create a Detect Numeric Outliers Experiment.

1. From the MLTK navigation bar, click **Experiments**.
If this is the first Experiment in the toolkit, you will land on a display screen of all six Assistants. Select the **Detect Numeric Outliers** block.

If you have at least one Experiment in the toolkit, you will land on a list view of Experiments. Click the **Create New Experiment** button.

2. Fill in an Experiment Title, and add a description. Both the name and description can be edited later if needed.

3. Click **Create**.

4. Run a search, and be sure to select a date range.

5. In **Field to analyze**, select a numeric field.
   The list populates every time you run a search.

   In picking a method, consider both the distribution of the data, as well as how the method impacts outlier detection.
   Use the following table to guide your decision.

<table>
<thead>
<tr>
<th>Method</th>
<th>Application</th>
</tr>
</thead>
<tbody>
<tr>
<td>Standard Deviation</td>
<td>This method is appropriate if your data exhibits a normal distribution. Since the standard deviation method centers on the mean, it is more impacted by outliers.</td>
</tr>
<tr>
<td>Median Absolute Deviation</td>
<td>This method applies a stricter interpretation of outliers than standard deviation because the measurement centers on the median and uses Median Absolute Deviation (MAD) instead of standard deviation.</td>
</tr>
<tr>
<td>Interquartile Range</td>
<td>This method is appropriate when your data exhibits an asymmetric distribution. Instead of centering the measurement on a mean or median, it uses quartiles to determine whether a value is an outlier.</td>
</tr>
</tbody>
</table>

7. Specify a value for the **Threshold multiplier**.
   The larger the number, the larger the outlier envelope, and therefore, the fewer the outliers.

8. (Optional) In the **Sliding window** field, specify the number of values to use to compute each slice of the outlier envelope.
   A sliding window is useful if the distribution of your data changes frequently. If you do not specify a sliding window, the Assistant uses the whole dataset which results in an outlier envelope of uniform size.

9. Select **Include current point** to include the current point in the calculations before assessing whether it is an outlier.

10. (Optional) In **Fields to split by**, select up to 5 fields.
    In the visualizations the data points are grouped by field, or if more than one split by field is specified, by the combination of the values of the fields.
    It is better to split by a categorical field than a numeric field. For example,
if you detect outliers in grocery store purchases and analyze the `quantity` field, you could split by `store_ID` to group the `quantity` datapoints by store.

11. (Optional) Add notes to this Experiment. Use this free form block of text to track the selections made in the Experiment parameter fields. Refer back to notes to review which parameter combinations yield the best results.

12. Click **Detect Outliers**. The Experiment is now in a Draft state. Draft versions allow you to alter settings without committing or overwriting a saved Experiment. An Experiment is not stored to Splunk until it is saved.

The following table explains the differences between a draft and saved Experiment:

<table>
<thead>
<tr>
<th>Action</th>
<th>Draft Experiment</th>
<th>Saved Experiment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Create new record in Experiment history</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Run Experiment search jobs</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>(As applicable) Save and update Experiment model</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>(As applicable) Update all experiment alerts</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>(As applicable) Update Experiment scheduled trainings</td>
<td>No</td>
<td>Yes</td>
</tr>
</tbody>
</table>

### Interpret and validate results

After you detect outliers, review your results in the tables and visualizations. Results commonly have a few outliers. You can use the following methods to better evaluate the Experiment results:

<table>
<thead>
<tr>
<th>Charts and Results</th>
<th>Applications</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data and Outliers</td>
<td>Outliers, represented by yellow dots, are datapoints that fall outside of the light blue envelope. A chart to the right of the graph reports the total number of outliers. Hover over a yellow dot to see the value and quantity of an outlier. To learn more about the nature of the outlier, click it to drill down to the search query to see the base of the datapoint.</td>
</tr>
<tr>
<td>Charts and Results</td>
<td>Applications</td>
</tr>
<tr>
<td>--------------------</td>
<td>--------------</td>
</tr>
<tr>
<td>Split by fields</td>
<td>If you selected a field to split by, then the Data and Outliers chart displays up to 10 values that you can add or remove from the chart. The chart groups the datapoints based on split by field values. For example, if there are 3 split by field values, the chart will be broken out into 3 separate charts for each split value. The number of outliers for each split by field value is displayed to the right of the chart.</td>
</tr>
<tr>
<td>Outlier Count Over Time chart</td>
<td>This chart plots the outliers over time, and only appears if you use time series data. If you specify more than one split, the chart shows the outlier count for each field value. To see which values are too high or too low, check the box for Split outliers above and below threshold. If you split by field, each field contains a value for outliers above and below the threshold.</td>
</tr>
<tr>
<td>Data Distribution histogram</td>
<td>This histogram shows the distribution, and displays the number of datapoints within the threshold (the light blue area) and the number of datapoints outside the threshold.</td>
</tr>
<tr>
<td>Data and Outliers table</td>
<td>This table shows each outlier the corresponding value, as well as the the lists of values for any split by field.</td>
</tr>
<tr>
<td>Outlier Split Value Distribution</td>
<td>If you specified one or more split by fields, this table displays the number of outliers for each split value or combination of split values.</td>
</tr>
</tbody>
</table>

**Save the Experiment**

Once you are getting valuable results from your Experiment, save it. Saving your Experiment results in the following actions:

1. Assistant settings saved as an Experiment knowledge object.
2. The Draft version saves to the Experiment Listings page.
3. Any affiliated scheduled trainings and alerts update to synchronize with the search SPL and trigger conditions.

You can load a saved Experiment by clicking the Experiment name.
Deploy the Experiment

Saved detect numeric outlier Experiments include options to manage, but not to publish.

In Experiments built using the Detect Numeric Outlier Assistant a model is not persisted, meaning you will not see an option to publish. However, you can achieve the same results as publishing the Experiment through the steps below for Outside the Experiment framework.

**Within the Experiment framework**

To manage your Experiment, perform the following steps:

1. From the MLTK navigation bar, choose Experiments. A list of your saved Experiments populates.
2. Click the Manage button available under the Actions column.

The toolkit supports the following Experiment management options:

- Create and manage Experiment-level alerts. Choose from both Splunk platform standard trigger conditions, as well as from Machine Learning Conditions related to the Experiment.
- Edit the title and description of the Experiment.
- Delete an Experiment.

Updating a saved Experiment can affect affiliated alerts. Re-validate your alerts once you complete the changes. For more information about alerts, see Getting started with alerts in the Splunk Enterprise Alerting Manual.

Experiments are always stored under the user's namespace, so changing sharing settings and permissions on Experiments is not supported.

**Outside the Experiment framework**

1. Click Open in Search to generate a New Search tab for this same dataset. This new search opens in a new browser tab, away from the
2. Click **Show SPL** to open a new modal window/overlay showing the search query that you used to detect outliers. Copy the SPL to use in other aspects of your Splunk instance.

### Learn more

To learn about implementing analytics and data science projects using Splunk's statistics, machine learning, built-in and custom visualization capabilities, see the Splunk for Analytics and Data Science course.

### Detect Categorical Outliers Experiment workflow

Experiments manage the data source, algorithm, and the parameters to configure that algorithm within one framework. An Experiment is an exclusive knowledge object in the Splunk platform that keeps track of its settings and history, as well as its affiliated alerts and scheduled trainings. Experiment Assistants enable machine learning through a guided user interface.

The following image illustrates results from the **showcase example** in the Splunk Machine Learning Toolkit with Bitcoin data.
Available algorithm

The Detect Categorical Outliers Assistant uses the following algorithm:

• Probabilistic measures

Create an Experiment to detect categorical outliers

The Detect Categorical Outliers Experiment Assistant identifies data that indicate interesting or unusual events. This Assistant allows non-numeric and multi-dimensional data, such as string identifiers and IP addresses. To detect categorical outliers, input data and select the fields for which to look for unusual combinations or a coincidence of rare values. When multiple fields have rare values, the result is an outlier.

Assistant workflow

Follow these steps to create a Detect Categorical Outliers Experiment.

1. From the MLTK navigation bar, click Experiments.
   ♦ If this is the first Experiment in the toolkit, you will land on a display screen of all six assistants. Select the Detect Categorical Outliers block.
   ♦ If you have at least one Experiment in the toolkit, you will land on a list view of Experiments. Click the Create New Experiment button.
2. Fill in an Experiment Title, and add a description. Both the name and description can be edited later if needed.
3. Click Create.
4. Run a search, and be sure to select a date range.
5. Select the fields you want to analyze.
   The list populates every time you run a search.
6. (Optional) Add notes to this Experiment. Use this free form block of text to track the selections made in the Experiment parameter fields. Refer back to notes to review which parameter combinations yield the best results.
7. Click Detect Outliers. The Experiment is now in a Draft state.
   Draft versions allow you to alter settings without committing or overwriting a saved Experiment. This Experiment is not stored to Splunk until it is saved.
   The following table explains the differences between a draft and a saved Experiment:

<table>
<thead>
<tr>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
<tr>
<td>Draft Experiment</td>
</tr>
<tr>
<td>------------------</td>
</tr>
<tr>
<td>Create new record in Experiment history</td>
</tr>
<tr>
<td>Run Experiment search jobs</td>
</tr>
<tr>
<td>(As applicable) Save and update Experiment model</td>
</tr>
<tr>
<td>(As applicable) Update all Experiment alerts</td>
</tr>
<tr>
<td>(As applicable) Update Experiment scheduled trainings</td>
</tr>
</tbody>
</table>

**Interpret and validate results**

After you detect outliers, review your results and the corresponding tables. Results often have a few outliers. You can use the following methods to better evaluate your Experiment results:

<table>
<thead>
<tr>
<th>Result</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Outliers</td>
<td>This result shows the number of events flagged as outliers.</td>
</tr>
<tr>
<td>Total Events</td>
<td>This result shows the total number of events that were evaluated.</td>
</tr>
<tr>
<td>Data and Outliers</td>
<td>This table lists the events that marked as outliers, and the corresponding reason that the event is marked as an outlier.</td>
</tr>
</tbody>
</table>

**Save the Experiment**

Once you are getting valuable results from your Experiment, save it. Saving your Experiment results in the following actions:

1. Assistant settings saved as an Experiment knowledge object.
2. The Draft version saves to the Experiment Listings page.
3. Any affiliated scheduled trainings and alerts update to synchronize with the search SPL and trigger conditions.

You can load a saved Experiment by clicking the Experiment name.
Deploy the Experiment

Saved detect categorical outlier Experiments include options to manage, but not to publish.

In Experiments built using the Detect Categorical Outlier Assistant a model is not persisted, meaning you will not see an option to publish. However, you can achieve the same results as publishing the Experiment through the steps below for Outside the Experiment framework.

Within the Experiment framework

To manage your Experiment, perform the following steps:

1. From the MLTK navigation bar, choose Experiments. A list of your saved Experiments populates.
2. Click the Manage button available under the Actions column.

The toolkit supports the following Experiment management options:

- Create and manage Experiment-level alerts. Choose from both Splunk platform standard trigger conditions, as well as from Machine Learning Conditions related to the Experiment.
- Edit the title and description of the Experiment.
- Delete an Experiment.

Updating a saved Experiment can affect affiliated alerts. Re-validate your alerts once you complete the changes. For more information about alerts, see Getting started with alerts in the Splunk Enterprise Alerting Manual.

Experiments are always stored under the user's namespace, meaning that changing sharing settings and permissions on Experiments is not supported.

Outside the Experiment framework

1. Click Open in Search to generate a New Search tab for this same dataset. This new search opens in a new browser tab, away from the...
Assistant.
This search query uses all data, not just the training set. You can adjust the SPL directly and see the results immediately. You can also save the query as a Report, Dashboard Panel or Alert.
2. Click Show SPL to generate a modal window/overlay showing the search query that you used to fit the model. Copy the SPL to use in other aspects of your Splunk instance.

Learn more

To learn about implementing analytics and data science projects using Splunk's statistics, machine learning, built-in and custom visualization capabilities, see the Splunk for Analytics and Data Science course.

Forecast Time Series Experiment workflow

Experiments manage the data source, algorithm, and the parameters to configure that algorithm within one framework. An Experiment is an exclusive knowledge object in the Splunk platform that keeps track of its settings and history, as well as its affiliated alerts and scheduled trainings. Experiment Assistants enable machine learning through a guided user interface.

The following visualization shows a forecast of sales numbers using the Kalman Filter algorithm.

Available algorithms

The Forecast Time Series Experiment Assistant uses the following algorithms:

- State-space method using the Kalman filter
- Autoregressive Integrated Moving Average (ARIMA)
Create an Experiment to forecast a time series

The Forecast Time Series Assistant predicts the next value in a sequence of time series data. The result includes both the predicted value and a measure of the uncertainty of that prediction. Forecasting is one type of prediction. Forecasting refers to the use of past time series data to make predictions.

To build a forecast, input the data and select the field by which you want to forecast.

Assistant workflow

Follow these steps to create a Forecast Time Series Experiment.

1. From the MLTK navigation bar, click Experiments.
   - If this is the first Experiment in the toolkit, you will land on a display screen of all six Assistants. Select the Forecast Time Series block.
   - If you have at least one Experiment in the toolkit, you will land on a list view of Experiments. Click the Create New Experiment button.
2. Fill in an Experiment Title, and add a description. Both the name and description can be edited later if needed.
3. Click Create.
4. Run a search, and be sure to select a date range.
5. (Optional) Click + Add a step to add preprocessing steps
6. Select an algorithm from the Algorithm drop-down menu.
   - If you are not sure which algorithm to choose, start with the default Kalman filter algorithm.
7. From the Field to Forecast list, select the field you want to forecast.
   - The Field to Forecast drop-down menu populates with fields from the search.
8. Select your parameters. Use the following table to guide your choices.

<table>
<thead>
<tr>
<th>If</th>
<th>Then</th>
</tr>
</thead>
<tbody>
<tr>
<td>You chose the Kalman filter algorithm</td>
<td>These methods consider subsets of features such as local level (an average of recent values), trend (a slope of line that fits through recent values), and seasonality (repeating patterns).</td>
</tr>
<tr>
<td>You chose the ARIMA algorithm</td>
<td>Specify the values for: AR (autoregressive); p, I (integrated); d, MA (moving average); q parameters.</td>
</tr>
<tr>
<td></td>
<td>For example, AR(1) means you would forecast future values by looking at 1 past value. I(1) means it took 1</td>
</tr>
</tbody>
</table>
If each data point was subtracted from the one that follows it, to make the time series stationary. MA(1) means you would forecast future values using 1 previous prediction error.

9. Specify the **Future Timespan**, which indicates how far beyond the data you want to forecast. The size of the confidence interval is used to gauge how confident the algorithm is in its forecast.

10. In the **Holdback** field, specify the number of values to withhold. Decide how many search results to use for validating the quality of the forecast. The larger the withholding, the fewer values available to train your model.

11. Select the **Confidence Interval**, which is the percentage of the future data you expect to fall inside of the confidence envelope.

12. For the Kalman algorithm, select the **Period**, which indicates the period of any known repeating patterns in the data to assist the algorithm. For example, if your data includes monthly sales data that follows annual patterns, specify 12 for the period.

13. (Optional) Add notes to this Experiment. Use this free form block of text to track the selections made in the Experiment parameter fields. Refer back to notes to review which parameter combinations yield the best results.

14. Click **Forecast**. The Experiment is now in a Draft state. Draft versions allow you to alter settings without committing or overwriting a saved Experiment. An Experiment is not stored to Splunk until it is saved.

The following table explains the differences between a draft and saved Experiment:

<table>
<thead>
<tr>
<th>Action</th>
<th>Draft Experiment</th>
<th>Saved Experiment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Create new record in Experiment history</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Run Experiment search jobs</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>(As applicable) Save and update Experiment model</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>(As applicable) Update all Experiment alerts</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>(As applicable) Update Experiment scheduled trainings</td>
<td>No</td>
<td>Yes</td>
</tr>
</tbody>
</table>

You cannot save a model if you use the DBSCAN or Spectral Clustering algorithm.
Interpret and validate results

After you forecast a time series, review your results in the following tables and visualizations.

<table>
<thead>
<tr>
<th>Result</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Raw Data Preview</td>
<td>This displays the raw data from the search.</td>
</tr>
<tr>
<td>Forecast</td>
<td>This graphs displays the actual value as a solid line and the predicted value as a dotted line, surrounded by a confidence envelope. Values that fall outside the confidence envelope are outliers. A vertical line indicates where training data stops and test data begins. When the real data ends, forecasted values are displayed in shades of green. The larger the envelope, the less confidence we have about forecasts around that time. The size of the envelope is directly related to the specified confidence interval percentage.</td>
</tr>
<tr>
<td>R2 Statistic</td>
<td>This statistic explains how well the model explains the variability of the result. 100% (a value of 1) means the model fits perfectly. The closer the value is to 1 (100%), the better the result.</td>
</tr>
<tr>
<td>Root Mean Squared Error</td>
<td>This statistic explains the variability of the result, essentially the standard deviation of the residual. The formula takes the difference between actual and predicted values, squares this value, takes an average, and then takes a square root. The result is an absolute measure of fit, the smaller the number the better the fit. These values only apply to one dataset and are not comparable to values outside of it.</td>
</tr>
<tr>
<td>Prediction Outliers</td>
<td>This result shows the total number of outliers detected.</td>
</tr>
</tbody>
</table>

Predicting with the ARIMA algorithm

When predicting using the ARIMA algorithm, additional autocorrelation panels are present. Autocorrelation charts can be used to estimate and identify the three
main parameters for the model:

- the autoregressive component $p$
- the integrated component or order of differencing $d$
- the moving average component $q$

**ACF: Autocorrelation function chart**

The autocorrelation function chart shows the predicted field's autocorrelations at various lags, surrounded by confidence interval lines. For example, the column at lag 1 shows the amount of correlation between the time series and a lagged version of itself.

![ACF Chart](image)

**PACF: Partial autocorrelation function chart**

The partial autocorrelation function chart shows the predicted field's autocorrelations at various lags while controlling for the amount of correlation contributed by earlier lag points. This chart is also surrounded by confidence interval lines. For example, the column at lag 2 shows the amount of correlation between the time series and a lagged version of itself, while removing the correlation contributed by the lag 1 data points.
ACF Residual: Autocorrelation function residual chart

The autocorrelation function residual chart shows prediction errors. The errors are the difference between the series and the predictions. The ACF of the residuals should be close to zero. If the errors are highly correlated, the model might be poorly parameterized, or the series might not be stationary.
**PACF Residual: partial autocorrelation function residual chart**

The partial autocorrelation function chart shows prediction errors. The errors are the difference between the series and the predictions. The PACF of the residuals should be close to zero. If the errors are highly correlated, the model might be poorly parameterized or the series might not be stationary.

Refine the Experiment

After you create a forecast, you can select an alterate algorithm option to see whether a different choice yields better results.

Be advised that the quality of the forecast primarily depends on the predictability of the data.

Save the Experiment

Once you are getting valuable results from your Experiment, save it. Saving your Experiment results in the following actions:

1. Assistant settings saved as an Experiment knowledge object.
2. The Draft version saves to the Experiment Listings page.
3. Any affiliated scheduled trainings and alerts update to synchronize with the search SPL and trigger conditions.

You can load a saved Experiment by clicking the Experiment name.
Deploy the Experiment

Saved forecast time series Experiments include options to manage, but not to publish.

In Experiments built using the Forecast Time Series Assistant a model is not persisted, meaning you will not see an option to publish. However, you can achieve the same results as publishing the Experiment through the steps below for Outside the Experiment framework.

Within the Experiment framework

To manage your Experiment, perform the following steps:

1. From the MLTK navigation bar, choose Experiments. A list of your saved Experiments populates.
2. Click the Manage button available under the Actions column.

The toolkit supports the following Experiment management options:

- Create and manage Experiment-level alerts. Choose from both Splunk platform standard trigger conditions, as well as from Machine Learning Conditions related to the Experiment.
- Edit the title and description of the Experiment.
- Delete an Experiment.

Updating a saved Experiment can affect affiliated alerts. Re-validate your alerts once you complete the changes. For more information about alerts, see Getting started with alerts in the Splunk Enterprise Alerting Manual.

Experiments are always stored under the user's namespace, meaning that changing sharing settings and permissions on Experiments is not supported.

Outside the Experiment framework

1. Within the Experiment click Open in Search to generate a New Search tab for this same dataset. This new search will open in a new browser tab,
away from the Assistant.
This search query uses all data, not just the training set. You can adjust the SPL directly and see results immediately. You can also save the query as a Report, Dashboard Panel or Alert.
2. Click Show SPL to generate a new modal window/ overlay showing the search query you used to forecast. Copy the SPL to use in other aspects of your Splunk instance.

Learn more

To learn about implementing analytics and data science projects using Splunk's statistics, machine learning, built-in and custom visualization capabilities, see the Splunk for Analytics and Data Science course.

Cluster Numeric Events Experiment workflow

Experiments manage the data source, algorithm, and the parameters to configure that algorithm within one framework. An Experiment is an exclusive knowledge object in the Splunk platform that keeps track of its settings and history, as well as its affiliated alerts and scheduled trainings. Experiment Assistants enable machine learning through a guided user interface.

The following visualization illustrates a clustering of humidity data results. This visualization is from the showcase example for Power Plant Operating Regimes.
Available algorithms

The Cluster Numeric Events Experiment Assistant uses the following algorithms:

- K-means
- DBSCAN
- Birch
- Spectral Clustering

Create an Experiment to cluster numeric events

The Cluster Numeric Events Assistant partitions events with multiple numeric fields into groups of events based on the values of those fields. The groupings aren’t known in advance, therefore, the learning is unsupervised.

To cluster numeric events, input data, optionally perform preprocessing, then select the algorithm to use for clustering and other parameters as necessary.

Before you begin

- The Cluster Numeric Events Assistant offers the option to preprocess your data. Read up on the preprocessing algorithms available here: Preprocessing machine data using Assistants.
- The toolkit default selects the K-means algorithm. Use this default algorithm if you aren’t sure which one is best for you. Read up on the other algorithm options here: Algorithms.

Assistant workflow

Follow these steps to create a Cluster Numeric Events Experiment.

1. From the MLTK navigation bar, click Experiments.
   - If this is the first Experiment in the toolkit, you will land on a display screen of all six Assistants. Select the Cluster Numeric Events block.
   - If you have at least one Experiment in the toolkit, you will land on a list view of Experiments. Click the Create New Experiment button.
2. Fill in an Experiment Title, and add a description. Both the name and description can be edited later if needed.
3. Click Create.
4. Run a search, and be sure to select a date range.
5. (Optional) Click + Add a step to add preprocessing steps
6. Select an algorithm from the Algorithm drop-down menu. K-means is selected by default. Another algorithm option may better fit your Experiment.

7. Specify the Fields to use for clustering. Use the following table to guide your next steps:

<table>
<thead>
<tr>
<th>If</th>
<th>Then</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data has been preprocessed</td>
<td>Choose from the preprocessed fields.</td>
</tr>
<tr>
<td>Algorithm of K-means, Birch or Spectral Clustering</td>
<td>Specify the number of clusters to use.</td>
</tr>
<tr>
<td>Algorithm of DBSCAN</td>
<td>Specify a value between 0 and 1 for ( \text{eps} ) (the size of the neighborhood). Smaller numbers result in more clusters.</td>
</tr>
</tbody>
</table>

8. (Optional) Add notes to this Experiment. Use this free form block of text to track the selections made in the Experiment parameter fields. Refer back to notes to review which parameter combinations yield the best results.

9. Click Cluster. The experiment is now in a Draft state. Draft versions allow you to alter settings without committing or overwriting a saved Experiment. An Experiment is not stored to Splunk until it is saved.

The following table explains the differences between a draft and saved Experiment:

<table>
<thead>
<tr>
<th>Action</th>
<th>Draft Experiment</th>
<th>Saved Experiment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Create new record in Experiment history</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Run Experiment search jobs</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>(As applicable) Save and update Experiment model</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Action</td>
<td>Draft Experiment</td>
<td>Saved Experiment</td>
</tr>
<tr>
<td>-------------------------------------------------</td>
<td>------------------</td>
<td>------------------</td>
</tr>
<tr>
<td>(As applicable) Update all Experiment alerts</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>(As applicable) Update Experiment scheduled trainings</td>
<td>No</td>
<td>Yes</td>
</tr>
</tbody>
</table>

You cannot save a model if you use the DBSCAN or Spectral Clustering algorithm.

**Interpret and validate results**

After the numeric events are clustered, review the cluster visualization. The fields included in the visualization are listed on screen. You can add and remove fields, and click **Visualize** to change the visualization.

You can drag a selection rectangle around some of the points in a plot to see the corresponding points on the other plots.

![Visualization example](image)

The visualization displays a maximum of 1000 points, 20 series and 6 fields (1 label and 5 variables).

**Save the Experiment**

Once you are getting valuable results from your Experiment, save it. Saving your Experiment results in the following actions:

1. Assistant settings saved as an Experiment knowledge object.
2. The Draft version saves to the Experiment Listings page.
3. Any affiliated scheduled trainings and alerts update to synchronize with the search SPL and trigger conditions.

You can load a saved Experiment by clicking the Experiment name.
Deploy the Experiment

Saved cluster numeric events Experiments include options to manage and publish.

The options to create alerts or to schedule training are not available if you use the DBSCAN or Spectral Clustering algorithms.

**Within the Experiment framework**

From within the framework, you can both manage and publish your Experiments. To manage your Experiment, perform the following steps:

1. From the MLTK navigation bar, choose Experiments. A list of your saved Experiments populates.
2. Click the Manage button available under the Actions column.

The toolkit supports the following Experiment management options:

- Create and manage Experiment-level alerts. Choose from both Splunk platform standard trigger conditions, as well as from Machine Learning Conditions related to the Experiment.
- Edit the title and description of the Experiment.
- Schedule a training job for an Experiment.
- Delete an Experiment.

Updating a saved Experiment can affect affiliated alerts. Re-validate your alerts once you complete the changes. For more information about alerts, see Getting started with alerts in the Splunk Enterprise Alerting Manual.

You can publish your Experiment through the following steps:

1. From the MLTK navigation bar, choose Experiments. A list of your saved Experiments populates.
2. Click the Publish button available under the Actions column. Publishing an Experiment model means the main model and any associated preprocessing models will be copied as lookup files in the
user's namespace within a selected destination app. Published models can be used to create alerts or schedule model trainings.

The Publish link will only show if you have created the Experiment using KMeans or Birch algorithms, and run the cluster action.

3. Give the model a title. It must start with letter or underscore, and only have letters, numbers and underscores in the name.
4. Select the destination app.
5. Click Save.
6. A message will let you know whether the model published, or why the action was not completed.

Experiments are always stored under the user's namespace, so changing sharing settings and permissions on Experiments is not supported.

Outside the Experiment framework

1. Click Open in Search to generate a New Search tab for this same dataset. This new search opens in a new browser tab, away from the Assistant. This search query that uses all data, not just the training set. You can adjust the SPL directly and see results immediately. You can also save the query a Report, Dashboard Panel or Alert.
2. Click Show SPL to open a new modal window/overlay showing the search query you used to fit the model. Copy the SPL to use in other aspects of your Splunk instance.

Learn more

To learn about implementing analytics and data science projects using Splunk's statistics, machine learning, built-in and custom visualization capabilities, see the Splunk for Analytics and Data Science course.
Models

The Splunk Machine Learning Toolkit (MLTK) provides custom search commands for machine learning. These commands use model files to store machine learning algorithm results on a dataset. This model can then be applied to other datasets.

Models are Splunk platform knowledge objects with configurable sharing and permissions.

Under the Models tab of the MLTK navigation bar, access any models created using the `fit` command on the Search tab, or those made through the Classic layout of the guided modeling Assistants.

Creating and using models

Models are created using the `fit` command and applied to datasets using the `apply` command. For more details, see:

- Understanding the fit and apply commands
- Search commands for machine learning

Namespacing and permissions

By default, MLTK models created with the `fit` command are created in the namespace of the user who ran the search.

Managing model permissions within Lookups

Model permissions can be managed from within the Models page, or via Lookups. Navigate to Settings > Lookups to access or update MLTK knowledge object permissions. Model files on this page are prefixed with `__mlspl__`. For example, a model named `my_model` is contained in the `__mlspl_my_model.csv` knowledge object.

See Manage knowledge object permissions in the Knowledge Manager Manual for more details.
You can also prefix model names to manage permissions by using the `fit`, `apply`, `summary`, and `deletemodel` custom search commands:

<table>
<thead>
<tr>
<th>Prefix</th>
<th>SPL command(s)</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>No prefix</td>
<td><img src="fit_into.png" alt="fit ... into &lt;model_name&gt;" /></td>
<td>The <code>fit</code> command creates the model in the user's namespace.</td>
</tr>
<tr>
<td>No prefix</td>
<td><img src="apply.png" alt="apply &lt;model_name&gt;" /> <img src="summary.png" alt="summary &lt;model_name&gt;" /> <img src="deletemodel.png" alt="deletemodel &lt;model_name&gt;" /></td>
<td>These commands use the first available model with the specified <code>&lt;model_name&gt;</code>. If a model with this name is available in both the user's private namespace and the shared application namespace, the model in the user's private namespace is used. If a model with this name is available only in the shared namespace, it is used.</td>
</tr>
<tr>
<td>app:</td>
<td><img src="fit_into.png" alt="fit ... into app:&lt;model_name&gt;" /></td>
<td>The <code>fit</code> command saves the model into the shared application namespace. By default, only the admin and power roles can save models into the shared application namespace.</td>
</tr>
<tr>
<td>app:</td>
<td><img src="apply.png" alt="apply app:&lt;model_name&gt;" /> <img src="summary.png" alt="summary app:&lt;model_name&gt;" /> <img src="deletemodel.png" alt="deletemodel app:&lt;model_name&gt;" /></td>
<td>These commands use the model from the shared application namespace even if a model with the same name exists in the user's private namespace.</td>
</tr>
</tbody>
</table>

The `deletemodel` command follows standard Splunk platform namespace rules. If the specified model name exists in the shared app namespace but not in the user's private namespace, the shared model is deleted if the user has write permissions on it.

**Sharing models from other Splunk apps**

The MLTK can access pre-trained models provided by other Splunk apps, provided that:

- The model to be shared has its sharing level set to "global" using standard knowledge object access settings. See Make an object available to users.
of all apps in the Knowledge Manager Manual.

- The model to be shared does not have the same name as a model that already exists in the MLTK.

For more information about building custom Splunk apps, see the Splunk developer portal.

**Upgrading from MLTK versions 2.2 and earlier**

Prior to MLTK version 2.3, models were created in the shared application namespace. By default, all users could read from them and write to them.

Model namespaces and permissions have changed in version 2.3, as described in [Namespacing and permissions](#).

<table>
<thead>
<tr>
<th>SPL command(s)</th>
<th>Result on MLTK versions 2.3 and later</th>
<th>Result on MLTK versions 2.2 and earlier</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>fit... into &lt;new_model_name&gt;</code></td>
<td>Creates a new model in the user's private namespace</td>
<td>Creates a new model in the shared application namespace</td>
</tr>
</tbody>
</table>

While all users can read models created in earlier versions of the MLTK, only admin and power roles can write to those models.

**Manage models**

The Models page presents the models in your Splunk Machine Learning Toolkit in list form. You will only see the models that you have permission to see. To view this page, click **Models** on the MLTK navigation bar.

In the displayed row(s), you can see what types of models are present, who owns them, and how they are shared.

Click the > symbol in the first column to expand the row and view model details including:

- Date last modified
- Algorithm used
- Fields used when training the model
- Model owner
- Sharing settings
Manage model permissions

Change model permissions to change their availability to other users. You can set up read and write access by role, and you can make models globally accessible, restricted to a particular app context, or private to a single user. For more information about model permissions, see Namespacing and permissions.

By default, only the Power and Admin roles can set permissions for models.

Modify permissions for a model in the Splunk Machine Learning Toolkit app:

1. Click **Models** on the MLTK navigation bar.
2. On the Models page, select the model that requires permission edits.
3. Click the current permission (Private, App or Public) in the **Sharing** column. This will only be editable if your current user has permission to edit that model.
4. In the resulting Edit Permissions window make your changes and click **Save**.

Under the tab, you will only see models created using the `fit` command on the Search tab, or those made through the classic layout of the guided modeling assistants under the **Classic** tab. You will not see models that were created within the framework of an Experiment.

Delete models

You can also delete models through the Models page:

1. Click **Models** on the MLTK navigation bar.
2. On the Models page, select the model that needs deletion.
3. Click **Delete** in the **Actions** column.
4. In the Delete Model window, click **Delete** again to verify that you want to delete the model.
The Splunk Machine Learning Toolkit includes several reusable custom visualizations that you can use in your own dashboards. Each visualization expects data in a certain format with certain fields, that you can see in the syntax portion of the visualization descriptions.

To apply a custom visualization to your data:

1. Run a search from the Search page in the Splunk Machine Learning Toolkit or the default Search & Reporting app on the Splunk platform.

2. Click the Visualization tab, then click the menu at the top left to display available visualizations.

3. Select a visualization.

You can use these custom visualizations on any Splunk platform instance on which the Splunk Machine Learning Toolkit is installed.

Most of the visualizations are also displayed when using particular Machine Learning Toolkit Assistants including the Predict Numeric Fields Assistant, Detect Numeric Outliers Assistant, Forecast Time Series Assistant, and Cluster Numeric Events Assistant.
3D Scatter Plot

Use the 3D Scatter Plot to see patterns in data where there are clusters of similar data points or drill down to identify singular data points.

Syntax

| eval clusterColor = case(clusterId=0, "teal", clusterId=2, "purple")
| table clusterId x y z clusterColor

The `clusterColor` parameter is optional. If no `clusterColor` parameter is provided the plot takes the MLTK's default color list. The `clusterColor` parameter supports written color names, or any hex color code.

The `| table clusterId x y z` line must be provided in order for the visualization to properly render.

Example

| inputlookup firewall_traffic.csv
| eval clusterId=serial_number, x=bytes_received, y=bytes_sent, z=packets_received, clusterColor = case(clusterId="sn_0009C101998", "teal")
| table clusterId x y z clusterColor

Example output
Boxplot Chart

Use the Boxplot Chart to show the minimum, lower quartile, median, upper quartile, and maximum of each field.

Syntax

\[
\text{search\_fragment} = | \text{boxplot} \ ...
\]

Boxplot requires the input of the macro `boxplot` in order to render. Failing to include the macro displays an error.

The box plot chart visualization expects five rows corresponding to min, max, median, lower quartile and upper quartile, in any order.

- `exactperc25` is the lower quartile
- `exactperc75` is the upper quartile

Example

\[
... | \text{inputlookup app\_usage.csv} | `boxplot`
\]

Downsampled Line Chart

Use the Downsampled Line Chart to show values and trends over time implementing downsampling to show large numbers of points.

The following image shows the Actual vs. Predicted Line Chart and the Residuals Line Chart that are also available when using the Predict Numeric Fields
Syntax

`search_fragment = | table <xAxis> <yAxis1> <yAxis2> ...`

Example

`... | table _time, "median_house_value", "predicted(median_house_value)" ...`

Forecast Chart

Use the Forecast Chart to show the forecasted value for data. This visualization is available in the Forecast Time Series Assistant and Smart Forecasting Assistant, which use different macros to produce the output:

- The Forecast Time Series Assistant uses the `fit` or `predict` commands with the ARIMA algorithm.
The Smart Forecasting Assistant uses the `fit` command with the StateSpaceForecast algorithm.

The following image shows the Forecast Chart on test data.

Syntax

```plaintext
search_fragment = | fit ARIMA [_time] <field_to_forecast> 
order=<int>-<int>-<int> [forecast_k=<int>] [conf_interval=<int>] 
[holdback=<int>] | `forecastviz(<forecast_k>, <holdback>, 
<field_to_forecast>, <conf_interval>)`

search_fragment = | fit StateSpaceForecast variable_name1 
[variable_name2] [variable_name3] [variable_name4] [variable_name5] 
output_metadata=true [conf_interval=<int>] | 
`smartforecastviz(<variable_name1> [,<variable_name2>] [, <variable_name3] [, <variable_name4] [, <variable_name5>])`
```

Examples

```plaintext
| inputlookup exchange.csv | fit ARIMA _time rate holdback=5 
conf_interval=95 order=1-0-1 forecast_k=10 as prediction | 
`forecastviz(10, 5, "rate", 95)`
| inputlookup app_usage.csv | fields CRM ERP Expenses | fit 
StateSpaceForecast CRM ERP output_metadata=true holdback=0 
forecast_k=50 conf_interval=50 into app_usage_model | 
`smartforecastviz(CRM, ERP)`
```

Histogram Chart

Use the Histogram Chart to show continuous data as bucketed by the `bin` command.

The following image shows the Residuals Histogram that is available when using the Predict Numeric Fields Assistant.
Syntax

`search_fragment = | bin <field> bins=<number>`

**Example**

`... | bin residual bins=100 ...`

**Outliers Chart**

Use the Outliers Chart to show the acceptable range for a value and to highlight the points that are outside of this range.

The following image shows the Outliers Chart that is also available when using the Detect Numeric Outliers Assistant.
Syntax

```
search_fragment = | table _time, outlier_variable, lowerBound, upperBound
```

Example

```
... | table _time, quantity, lowerBound, upperBound, isOutlier ...
```

**Scatter Line Chart**

Use the Scatter Line Chart to show the relationships between discrete values in two dimensions, as well as an additional identity (x=y) line.

The following image shows the Actual vs. Predicted Scatter Chart that is also available when using the Predict Numeric Fields Assistant.
Syntax

```plaintext
search_fragment = | table <xAxis> <yAxis>
```

Example

```plaintext
... | table "median_house_value" "predicted(median_house_value)" ...
```

**Scatterplot Matrix**

![Scatterplot Matrix](image)

Use the Scatterplot Matrix to show the relationships between discrete values in multiple dimensions.

All field values must be numeric in order to render the Scatterplot Matrix.

The following example shows the Scatterplot Matrix that is also available when using the Cluster Numeric Events Assistant.

![Cluster Numeric Events Assistant](image)

Syntax

```plaintext
search_fragment = | table <name_category>, <dimension_1>, <dimension_2>,
                  <dimension_3> ...
```

Example

```plaintext
... | table cluster, "avg_rooms_per_dwelling", "business_acres",
    "median_house_value" ...
```
Classic Assistants

Classic Assistants overview

There are six Classic Assistants that live within the Machine Learning Toolkit. This is the legacy version of the guided modeling Assistant layout for the MLTK. These Assistants enable machine learning through a guided user interface.

Through the Assistants workflow, users specify data sources, select an algorithm and algorithm parameters, select the fields for the algorithms to analyze and set training/test data splits. Once the user instructs the Assistant to fit the algorithms to the selected training data and generate results, the workflow continues through the available visualizations and statistical analysis. Every step of the workflow has tooltips as additional guides, the option to see the SPL being written by the Assistant (with an explanation for the commands), and an option to open a clone of the SPL in a new search window for customization by the user.

Choose the Classic Assistant to suit your needs:

- The **Predict Numeric Fields** Classic Assistant uses regression algorithms to predict or estimate numeric values. Such models are useful for determining to what extent certain peripheral factors contribute to a particular metric result. After the regression model is computed, you can use these peripheral values to make a prediction on the metric result.

- The **Predict Categorical Fields** Classic Assistant displays a type of learning known as classification. A classification algorithm learns the tendency for data to belong to one category or another based on related data.

- The **Detect Numeric Outliers** Classic Assistant determines values that appear to be extraordinarily higher or lower than the rest of the data. Identified outliers are indicative of interesting, unusual, and possibly dangerous events. This assistant is restricted to one numeric data field.

- The **Detect Categorical Outliers** Classic Assistant identifies data that indicate interesting or unusual events. This assistant allows non-numeric and multi-dimensional data, such as string identifiers and IP addresses. To detect categorical outliers, input data and select the fields for which to look for unusual combinations or a coincidence of rare values. When multiple fields have rare values, the result is an outlier.
• The **Forecast Time Series** Classic Assistant forecasts the next values in a sequence for a single time series. The result includes both the forecasted value and a measure of the uncertainty of that forecast. Forecasting refers to the use of past time series data trends to make a prediction about likely future values.

• The **Cluster Numeric Events** Classic Assistant partitions events with multiple numeric fields into groups of events based on the values of those fields. The groupings aren't known in advance and the algorithms are often referred to as unsupervised learning.

**Assistant commands**

Commands vary depending upon which Assistant is selected:

- Predict Numeric and Predict Categorical Fields use the `fit model` command
- Detect Numeric Outliers and Detect Categorical Outliers use `detect outliers` command
- Forecast Time Series uses `forecast` command
- Cluster Numeric Events uses `cluster` command

**Further information**

For more information about alerts, see Getting started with alerts in the Splunk Enterprise Alerting Manual.
For more information about reports, see Saving and sharing reports in the Splunk Enterprise Search Tutorial Manual.

**Predict Numeric Fields Classic Assistant**

Classic Assistants enable machine learning through a guided user interface. The Predict Numeric Fields Classic Assistant uses regression algorithms to predict numeric values. Such models are useful for determining to what extent certain peripheral factors contribute to a particular metric result. After the Classic Assistant computes the regression model, you can use these peripheral values to make a prediction on the metric result.

The following visualization illustrates a scatter plot of the actual versus predicted results. This visualization is from the showcase example for Server Power Consumption.
Algorithms

The Predict Numeric Fields Classic Assistant uses the following regression algorithms to predict numeric values:

- Linear Regression
- RandomForestRegressor
- Lasso
- KernelRidge
- ElasticNet
- Ridge
- DecisionTreeRegressor

Create a model to predict a numeric field

Before you begin

- The Predict Numeric Fields Assistant offers the option to preprocess your data. Read up on the preprocessing algorithms available here: Preprocessing machine data.
- The toolkit default selects the Linear Regression algorithm. Use this default if you aren't sure which one is best for you. Read up on the other algorithm options here: Algorithms.

Workflow

Follow these steps for the Predict Numeric Fields Classic Assistant.
1. From the MLTK navigation bar select Classic > Assistants > **Predict Numeric Fields**.
2. Run a search, and be sure to select a date range.
3. (Optional) Click **+ Add a step** to add preprocessing steps.
4. Select an algorithm from the **Algorithm** drop-down menu.
5. Select a target field from the drop-down menu **Field to predict**.
   When you select **Field to predict**, the **Fields to use for predicting** drop-down populates with available fields to include in your model.
6. Select a combination of fields from the drop-down menu **Fields to use for predicting**. As seen below in the server power showcase example, the drop-down menu contains a list of all the possible fields used to predict ac_power using the linear regression algorithm.

   ![Image of field selection and algorithm options]

7. Split your data into training and testing data. The default split is 50/50, and the data is divided randomly into two groups.

   The algorithm selected determines the fields available to build your model. Hover over any field name to get more information about that field.
8. Type the name the model in **Save the model as** field.
   You must specify a name for the model in order to fit a model on a schedule or schedule an alert.
9. Click **Fit Model**.

### Interpret and validate results

After you fit the model, review the prediction results and visualizations to see how well the model predicted the numeric field. You can use the following methods to evaluate your predictions:

<table>
<thead>
<tr>
<th>Charts and Results</th>
<th>Applications</th>
</tr>
</thead>
<tbody>
<tr>
<td>Actual vs. Predicted</td>
<td>This visualization plots the predicted value against the actual values for the predicted field. The closer the points are to the line, the better the model. Hover over the blue dots to see the actual values.</td>
</tr>
</tbody>
</table>
This visualization shows the difference between the actual values and the predicted values. Hover over the predicted values (blue bars) to see the number of residual errors and the sample count values. Residuals commonly end on a bell curve clustered tightly around zero.

This statistic shows how well the model explains the variability of the result. 100% (a value of 1) means the model fits perfectly. The closer the value is to 1, the better the result.

This chart shows the variability of the result, which is the standard deviation of the residual. The formula takes the difference between the actual and predicted values, squares this value, takes an average, and then takes a square root. This value can be arbitrarily large and gives you an idea of how close or far the model is. These values only make sense within one dataset and shouldn’t be compared across datasets.

This summary displays the coefficients associated with each variable in the regression model. A relatively high coefficient value shows a high association of that variable with the result. A negative value shows a negative correlation.

This overlay shows the actual values against the predicted values in sequence.

The residuals show the difference between predicted and actual values in sequence.

Refine the model

After you validate the model, refine the model and run the fit command again.

Consider trying the following:

1. Reduce the number of fields selected in the Fields to use for predicting drop-down menu. Having too many fields can generate a distraction.
2. Bring in new data sources to enrich your modeling space.
3. Build features on raw data, model on behaviors of the data instead of raw data points, using SPL, Streamstats, eventstats, etc.
4. Check your fields - are you using categorical values correctly? For example are you using DayOfWeek as a number (0 to 6) instead of

253
"Monday", "Tuesday", etc? Make sure you have the right type of value.

5. Bring in context via lookups - holidays, external anomalies, etc.

6. Increase the number of fields (from additional data, feature building as above, etc) selected in the Fields to use for predicting drop-down menu.

**Deploy the model**

After you validate and refine the model you can deploy the model.

**Within the Classic Assistant framework**

1. Click the **Schedule Training** button to the right of Fit Model to schedule model training.

This opens a new modal window/overlay with fields to fill out including Report title, time range and trigger actions. You can set up a regular interval to fit the model.

**Outside the Classic Assistant framework**

1. Click **Open in Search** to generate a New Search tab for this same dataset. This new search opens in a new browser tab, away from the Classic Assistant.

   This search query that uses all data, not just the training set. You can adjust the SPL directly and see results immediately. You can also save the query as a Report, Dashboard Panel or Alert.

2. Click **Show SPL** to generate a new modal window/overlay showing the search query you used to fit the model. Copy the SPL to use in other aspects of your Splunk instance.

3. Click **Schedule Alert** to set up a triggered alert when the predicted value meets a threshold you specify.

Once you navigate away from the Classic Assistant page, you cannot return to it through the Classic or Models tabs. Classic Assistants are great for generating SPL, but may not be ideal for longer-term projects.

For more information about alerts, see Getting started with alerts in the Splunk Enterprise Alerting Manual.
Predict Categorical Fields Classic Assistant

Classic Assistants enable machine learning through a guided user interface. The Predict Categorical Fields Classic Assistant displays a type of learning known as classification. A classification algorithm learns the tendency for data to belong to one category or another based on related data.

The following classification table shows the actual state of the field versus predicted state of the field. The yellow bar highlights an incorrect prediction.

<table>
<thead>
<tr>
<th>DiskFailure</th>
<th>predicted(DiskFailure)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>No</td>
<td>No</td>
</tr>
</tbody>
</table>

Algorithms

The Predict Categorical Fields assistant uses the following classification algorithms to predict fields:

- LogisticRegression
- SVM
- RandomForestClassifier
- GaussianNB
- BernouliNB
- DecisionTreeClassifier

Create a model to predict a categorical field

Before you begin

- The Predict Numeric Fields Assistant offers the option to preprocess your data. Read up on the preprocessing algorithms available here: Preprocessing machine data.
• The toolkit default selects the Logistic Regression algorithm. Use this default if you aren't sure which one is best for you. Read up on the other algorithm options here: Algorithms.

Workflow

Follow these steps for the Predict Categorical Fields Classic Assistant.

1. From the MLTK navigation bar select Classic > Assistants > Predict Categorical Fields.
2. Run a search, and be sure to select a date range.
3. (Optional) Click + Add a step to add preprocessing steps.
4. Select an algorithm from the Algorithm drop-down menu.
5. Select a target field from the drop-down menu Field to predict.
   When you select the Field to predict, the Fields to use for predicting drop-down populates with available fields to include in your model.
6. Select a combination of fields from the drop-down menu Fields to use for predicting.
7. Split your data into training and testing data. The default split is 50/50, and the data is divided randomly into two groups.

   The algorithm selected determines the fields available to build your model. Hover over any field name to get more information about that field.
8. Type the name the model in Save the model as field.
   You must specify a name for the model in order to fit a model on a schedule or schedule an alert.
9. Click Fit Model.

Interpret and validate

After you fit the model, review the prediction results and visualizations to see how well the model predicted the categorical field. In this analysis, metrics are related to mis-classifying the field, and are based on false positives and negatives, and true positives and negatives.

<table>
<thead>
<tr>
<th>Result</th>
<th>Application</th>
</tr>
</thead>
<tbody>
<tr>
<td>Precision</td>
<td>This statistic is the percentage of the time a predicted class is the correct class.</td>
</tr>
<tr>
<td>Recall</td>
<td>This statistic is the percentage of time that the correct class is predicted.</td>
</tr>
</tbody>
</table>
Result | Application |
--- | --- |
Accuracy | This statistic is the overall percentage of correct predictions. |
F1 | This statistic is the weighted average of precision and recall, based on a scale from zero to one. The closer the statistic is to one, the better the fit of the model. |
Classification Results (Confusion Matrix) | This table charts the number of actual results against predicted results, also known as a Confusion Matrix. The shaded diagonal numbers should be high (closer to 100%), while the other numbers should be closer to 0. |

**Refine the model**

After you validate the model, refine the model and run the `fit` command again.

Consider trying the following:

1. Reduce the number of fields selected in the `Fields to use for predicting` drop-down menu. Having too many fields can generate a distraction.
2. Bring in new data sources to enrich your modeling space.
3. Build features on raw data, model on behaviors of the data instead of raw data points, using SPL. Streamstats, eventstats, etc.
4. Check your fields - are you using categorical values correctly? For example are you using `DayOfWeek` as a number (0 to 6) instead of "Monday", "Tuesday", etc? Make sure you have the right type of value.
5. Bring in context via lookups - holidays, external anomalies, etc.
6. Increase the number of fields (from additional data, feature building as above, etc) selected in the `Fields to use for predicting` drop-down menu.

**Deploy the model**

After you validate and refine the model, you can deploy the model.

**Within the classic assistant framework**

1. Click the **Schedule Training** button to the right of Fit Model to schedule model training.

This open a new modal/ window overlay with fields to fill out including Report title, time range and trigger actions. You can set up a regular interval to fit the
Outside the Classic Assistant framework

1. Click **Open in Search** to generate a New Search tab for this same dataset. This new search will open in a new browser tab, away from the Classic Assistant. This search query that uses all data, not just the training set. You can adjust the SPL directly and see results immediately. You can also save the query as a Report, Dashboard Panel or Alert.

2. Click **Show SPL** to generate a new window showing the search query that was used to fit the model. Copy the SPL here for use in other aspects of your Splunk instance.

3. Click **Schedule Alert** to set up an alert that is triggered when the predicted value meets a threshold you specify.

Once you navigate away from the Classic Assistant page, you cannot return to it through the Classic or Models tabs. Classic Assistants are great for generating SPL, but may not be ideal for longer-term projects.

For more information about alerts, see Getting started with alerts in the Splunk Enterprise Alerting Manual.

**Detect Numeric Outliers Classic Assistant**

Classic Assistants enable machine learning through a guided user interface. The Detect Numeric Outliers Classic Assistant determines values that appear to be extraordinarily higher or lower than the rest of the data. Identified outliers are indicative of interesting, unusual, and possibly dangerous events. The Detect Numeric Outliers Assistant is restricted to one numeric data field.

In the following visualization, the yellow dots indicate outliers.
Algorithms

The Detect Numeric Outliers Classic Assistant is compatible with the following distribution statistics:

- Standard deviation
- Median absolute deviation
- Interquartile range

Detect Numeric Outliers

Input the data and select the parameters you want to investigate. When a situation violates the expectations for a parameter, it results in an outlier.

Workflow

Follow these steps for the Detect Numeric Outliers Classic Assistant.

1. From the MLTK navigation bar select Classic > Assistants > Detect Numeric Outliers.
2. Run a search, and be sure to select a date range.
3. Select a numeric field in Field to analyze.
   The list populates every time you run a search.
   In picking a method, consider both the distribution of the data, as well as how the method impacts outlier detection. Use the following table to guide your decision:

<table>
<thead>
<tr>
<th>Method</th>
<th>Application</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
This method is appropriate if your data exhibits a normal distribution. Since the standard deviation method centers on the mean, it is more impacted by outliers.

This method applies a stricter interpretation of outliers than standard deviation because the measurement centers on the median and uses Median Absolute Deviation (MAD) instead of standard deviation.

This method is appropriate when your data exhibits an asymmetric distribution. Instead of centering the measurement on a mean or median, it uses quartiles to determine whether a value is an outlier.

Specify a value for Threshold multiplier. The larger the number, the larger the outlier envelope, and therefore, the fewer the outliers.

(Optional) In the Sliding window field, specify the number of values to use to compute each slice of the outlier envelope. A sliding window is useful if the distribution of your data changes frequently. If you do not specify a sliding window, the assistant uses the whole dataset which results in an outlier envelope of uniform size.

Select Include current point to include the current point in the calculations before assessing whether it is an outlier.

(Optional) In Fields to split by, select up to 5 fields. In the visualizations the data points are grouped by field, or if more than one split by field is specified, by the combination of the values of the fields. It is better to split by a categorical field than a numeric field. For example, if you detect outliers in grocery store purchases and analyze the quantity field, you could split by store_ID to group the quantity data points by store.

Click Detect Outliers.

### Interpret and validate

After you detect outliers, review your results in the tables and visualizations. Results commonly have a few outliers.

<table>
<thead>
<tr>
<th>Method</th>
<th>Application</th>
</tr>
</thead>
<tbody>
<tr>
<td>Standard Deviation</td>
<td>This method is appropriate if your data exhibits a normal distribution. Since the standard deviation method centers on the mean, it is more impacted by outliers.</td>
</tr>
<tr>
<td>Median Absolute Deviation</td>
<td>This method applies a stricter interpretation of outliers than standard deviation because the measurement centers on the median and uses Median Absolute Deviation (MAD) instead of standard deviation.</td>
</tr>
<tr>
<td>Interquartile Range</td>
<td>This method is appropriate when your data exhibits an asymmetric distribution. Instead of centering the measurement on a mean or median, it uses quartiles to determine whether a value is an outlier.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Result</th>
<th>Application</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data and Outliers</td>
<td>Outliers, represented by yellow dots, are data point that fall outside of the light blue envelope. A chart to the right of the graph reports the total number of outliers. Hover over a yellow</td>
</tr>
</tbody>
</table>
Deploy numeric outlier detection

1. Click **Open in Search** to generate a New Search tab for this same dataset. This new search will open in a new browser tab, away from the Classic Assistant. This search query that uses all data, not just the training set. You can adjust the SPL directly and see results immediately. You can also save the query as a Report, Dashboard Panel or Alert.

2. Click **Show SPL** to generate a new window showing the search query that was used to calculate the outliers. Copy the SPL here for use in other aspects of your Splunk instance.

Once you navigate away from the Classic Assistant page, you cannot return to it through the Classic or Models tabs. Classic Assistants are great for generating SPL, but may not be ideal for longer-term projects.
For more information about alerts, see Getting started with alerts in the Splunk Enterprise Alerting Manual.

**Detect Categorical Outliers Classic Assistant**

Classic assistants enable machine learning through a guided user interface. The Detect Categorical Outliers Classic Assistant identifies data that indicate interesting or unusual events. This assistant works with non-numeric and multi-dimensional data, such as string identifiers and IP addresses. To detect categorical outliers, input data and select the fields for which to look for unusual combinations or a coincidence of rare values. When multiple fields have rare values, the result is an outlier.

The following image illustrates results from the **showcase example** in the Splunk Machine Learning Toolkit with Bitcoin data.

![Outlier Detection](image)

**Algorithm**

The Detect Categorical Outliers assistant uses the probabilistic measures algorithm.

**Detect Categorical Outliers**

To detect categorical outliers, input data and select the fields to analyze.

**Workflow**

Follow these steps for the Detect Categorical Outliers Classic Assistant.
1. From the MLTK navigation bar select Classic > Assistants > Detect Categorical Outliers.
2. Run a search, and be sure to select a date range.
3. Select the fields you want to analyze. The list populates every time you run a search.
4. Click Detect Outliers.

Interpret and validate

After you detect outliers, review your results and the corresponding tables. Results often have a few outliers.

<table>
<thead>
<tr>
<th>Result</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Outliers</td>
<td>This result shows the number of events flagged as outliers.</td>
</tr>
<tr>
<td>Total Events</td>
<td>This result shows the total number of events that were evaluated.</td>
</tr>
<tr>
<td>Data and Outliers</td>
<td>This table lists the events that marked as outliers, and the corresponding reason that the event is marked as an outlier.</td>
</tr>
</tbody>
</table>

Deploy categorical outlier detection

1. Click Open in Search to generate a New Search tab for this same dataset. This new search will open in a new browser tab, away from the Classic Assistant. 
   This search query uses all data, not just the training set. You can adjust the SPL directly and see results immediately. You can also save the query as a Report, Dashboard Panel or Alert.
2. Click Show SPL to generate a new window showing the search query that was used to calculate the outliers. Copy the SPL here for use in other aspects of your Splunk instance.

Once you navigate away from the Classic Assistant page, you cannot return to it through the Classic or Models tabs. Classic Assistants are great for generating SPL, but may not be ideal for longer-term projects.

For more information about alerts, see Getting started with alerts in the Splunk Enterprise Alerting Manual.
Forecast Time Series Classic Assistant

Classic Assistants enable machine learning through a guided user interface. The Forecast Time Series Classic Assistant predicts the next value in a sequence of time series data. The result includes both the predicted value and a measure of the uncertainty of that prediction. Forecasting makes predictions based on time series data from the past.

The following visualization shows a time series and the split between the training and testing data.

Algorithms

The Forecast Time Series assistant can use the following algorithms to make predictions:

- State-space method using Kalman filter
- AutoRegressive Integrated Moving Average (ARIMA)

Forecast Time Series

Input the data and select the field you want to forecast.

Workflow

Follow these steps for the Forecast Time Series Classic Assistant.

1. From the MLTK navigation bar select Classic > Assistants > Forecast Time Series.
2. Run a search, and be sure to select a date range.
3. Select an algorithm from the Algorithm drop-down menu.
   If you are not sure which algorithm to choose, start with the default Kalman filter algorithm.
4. Select the field from which you want to base the forecast from the Field to Forecast.
   The Field to Forecast list is populated with fields from the search.
5. Select the parameters. Refer to the following table as a guide.

<table>
<thead>
<tr>
<th>If</th>
<th>Then</th>
</tr>
</thead>
<tbody>
<tr>
<td>You chose the Kalman filter algorithm.</td>
<td>Select a forecasting method. These methods consider subsets of features such as local level (an average of recent values), trend (a slope of line that fits through recent values), and seasonality (repeating patterns).</td>
</tr>
<tr>
<td>You chose the ARIMA algorithm</td>
<td>Specify the values for AR (autoregressive) - $p$, I (integrated) - $d$, MA (moving average) - $q$ parameters. For example, AR(1) means you would forecast future values by looking at 1 past value. I(1) means it took 1 difference, where each data point was subtracted from the one that follows it, to make the time series stationary. MA(1) means you would forecast future values using 1 previous prediction error.</td>
</tr>
</tbody>
</table>

6. Specify the Future Timespan, which indicates how far beyond the data you want to forecast.
   The size of the confidence interval is used to gauge how confident the algorithm is in its forecast.
7. Specify the number of values to withhold in the Holdback field. Decide how many search results to use for validating the quality of the forecast.
   The larger the withholding, the fewer values available to train your model.
8. Specify the percentage of the future data you expect to fall inside of the confidence envelope with the Confidence Interval field.

<table>
<thead>
<tr>
<th>If</th>
<th>Then</th>
</tr>
</thead>
<tbody>
<tr>
<td>You chose the Kalman filter algorithm.</td>
<td>Select the Period, which indicates the period of any known repeating patterns in the data to assist the algorithm. For example, if your data includes monthly sales data that follows annual patterns,</td>
</tr>
</tbody>
</table>
### If

<table>
<thead>
<tr>
<th>If</th>
<th>Then</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>specify 12 for the period.</td>
</tr>
</tbody>
</table>

9. Click **Forecast**.

## Interpret and validate

After you forecast a time series, review your results in the following tables and visualizations.

<table>
<thead>
<tr>
<th>Result</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Raw Data Preview</td>
<td>This displays the raw data from the search.</td>
</tr>
<tr>
<td>Forecast</td>
<td>This graphs displays the actual value as a solid line and the predicted value as a dotted line, surrounded by a confidence envelope. Values that fall outside the confidence envelope are outliers. A vertical line indicates where training data stops and test data begins. When the real data ends, forecasted values are displayed in shades of green. The larger the envelope, the less confidence we have about forecasts around that time. The size of the envelope is directly related to the specified confidence interval percentage.</td>
</tr>
<tr>
<td>R2 Statistic</td>
<td>This statistic explains how well the model explains the variability of the result. 100% (a value of 1) means the model fits perfectly. The closer the value is to 1 (100%), the better the result.</td>
</tr>
<tr>
<td>Root Mean Squared Error</td>
<td>This statistic explains the variability of the result, essentially the standard deviation of the residual. The formula takes the difference between actual and predicted values, squares this value, takes an average, and then takes a square root. The result is an absolute measure of fit, the smaller the number the better the fit. These values only apply to one dataset and are not comparable to values outside of it.</td>
</tr>
<tr>
<td>Prediction Outliers</td>
<td>This result shows the total number of outliers detected.</td>
</tr>
</tbody>
</table>

### Predicting with the ARIMA algorithm

When predicting using the ARIMA algorithm, additional autocorrelation panels are present. Autocorrelation charts can be used to estimate and identify the three main parameters for the model:
• the autoregressive component $p$
• the integrated component or order of differencing $d$
• the moving average component $q$

**ACF: Autocorrelation function chart**

The autocorrelation function chart shows the predicted field’s autocorrelations at various lags, surrounded by confidence interval lines. For example, the column at lag 1 shows the amount of correlation between the time series and a lagged version of itself.

![ACF Chart](chart.png)

**PACF: Partial autocorrelation function chart**

The partial autocorrelation function chart shows the predicted field’s autocorrelations at various lags while controlling for the amount of correlation contributed by earlier lag points. This chart is also surrounded by confidence interval lines. For example, the column at lag 2 shows the amount of correlation between the time series and a lagged version of itself, while removing the correlation contributed by the lag 1 data points.
ACF Residual: Autocorrelation function residual chart

The autocorrelation function residual chart shows prediction errors. The errors are the difference between the series and the predictions. The ACF of the residuals should be close to zero. If the errors are highly correlated, the model might be poorly parameterized, or the series might not be stationary.
**PACF Residual: partial autocorrelation function residual chart**

The partial autocorrelation function chart shows prediction errors. The errors are the difference between the series and the predictions. The PACF of the residuals should be close to zero. If the errors are highly correlated, the model might be poorly parameterized or the series might not be stationary.

![PACF Residual Chart](image)

**Refine the forecast**

After you create a forecast, you can select a different algorithm to see if a different choice yields better results. The quality of the forecast primarily depends upon how predictable the data is.

**Deploy the forecast**

After you validate and refine the forecast, deploy the forecast.

**Within the Classic Assistant framework**

1. At the bottom of the visualization of the forecast, click **Schedule Alert** to create an alert for when a prediction meets criteria you set.

This opens a new modal/ window overlay with fields to fill out.

**Outside the Classic Assistant framework**

1. Click **Open in Search** to generate a New Search tab populated with the search query used for the visualization. This new search will open in a new browser tab, away from the Classic Assistant. You can adjust the
SPL directly and see results immediately. You can also save the query as a Report, Dashboard Panel or Alert.

2. Click **Show SPL** to generate a new modal/ window overlay populated with the search query used for the visualization. You can use this same query on a different data set.

Once you navigate away from the Classic Assistant page, you cannot return to it through the Classic or Models tabs. Classic Assistants are great for generating SPL, but may not be ideal for longer-term projects.

For more information about alerts, see Getting started with alerts in the Splunk Enterprise Alerting Manual.

**Cluster Numeric Events Classic Assistant**

Classic Assistants enable the creation of a machine learning model through a guided user interface. The Cluster Numeric Events Classic Assistant partitions events with multiple numeric fields into groups of events based on the values of those fields. The groupings aren’t known in advance, therefore, the learning is unsupervised.

The following visualization illustrates a clustering of humidity data results. This visualization is from the *showcase example* for Power Plant Operating Regimes.

![Cluster Numeric Events Visualization](image)

**Algorithms**

The Cluster Numeric Events Classic Assistant uses the following algorithms:

- K-means
- DBSCAN
- Birch
- Spectral Clustering
Cluster numeric events

To cluster numeric events, input data, optionally perform preprocessing, then select the algorithm to use for clustering and other parameters as necessary.

Before you begin

- The Predict Numeric Fields Assistant offers the option to preprocess your data. Read up on the preprocessing algorithms available here: Preprocessing machine data.
- The toolkit default selects the K-means algorithm. Use this default if you aren’t sure which one is best for you. Read up on the other algorithm options here: Algorithms.

Workflow

Follow these steps for the Cluster Numeric Events Classic Assistant.

1. From the MLTK navigation bar select Classic > Assistants > Cluster Numeric Events.
2. Run a search, including the selection of a date range.
3. (Optional) Click + Add a step to add preprocessing steps.
4. Select an algorithm from the Algorithm drop-down menu.
5. Specify the Fields to use for clustering.
   - If your data has been preprocessed, choose from the preprocessed fields.
6. For K-means, Birch, and Spectral Clustering, specify the number of clusters to use.
   - For DBSCAN, specify a value between 0 and 1 for eps (the size of the neighborhood).
   - Smaller numbers result in more clusters.
7. Type the name the model in Save the model as field.
   - You must specify a name for the model in order to fit a model on a schedule or schedule an alert.
   - You cannot save a model if you use the DBSCAN or Spectral Clustering algorithm.
8. Click Cluster.

Interpret and validate

After the numeric events are clustered, review the cluster visualization. The fields included in the visualization are listed on screen. You can add and remove fields, and click Visualize to change the visualization.
You can drag a selection rectangle around some of the points in a plot to see the corresponding points on the other plots.

The visualization displays a maximum of 1000 points, 20 series and 6 fields (1 label and 5 variables).

**Deploy clustering**

After you interpret and validate the clustering, deploy it.

*Within the Classic Assistant framework*

1. Click the **Schedule Training** button to the right of Cluster to run the clustering on a schedule. You can set up a regular interval to deploy clustering, for example, once a week.

You cannot schedule clustering if you use the DBSCAN or Spectral Clustering algorithms.

*Outside the Classic Assistant framework*

1. Click **Open in Search** to generate a New Search tab filled out with the search query that was used for the clustering. This new search will open in a new browser tab, away from the Classic Assistant. You can adjust the SPL directly and see results immediately. You can also save the query as a Report, Dashboard Panel or Alert.
2. Click **Show SPL** to generate a new window showing the search query that was used for the clustering. Copy the SPL here to use this same query on a different data set.
3. Click **Schedule Alert** to set up an alert that is triggered when the number of events in a cluster exceeds a threshold you specify.

Alerts cannot be scheduled if you use the DBSCAN or Spectral Clustering algorithms.
Once you navigate away from the Classic Assistant page, you cannot return to it through the Classic or Models tabs. Classic Assistants are great for generating SPL, but may not be ideal for longer-term projects.

For more information about alerts, see Getting started with alerts in the Splunk Enterprise Alerting Manual.
Preprocessing

Preparing your data for machine learning

Machine learning is a process of generalizing from examples. These generalizations, typically called models, are used to perform a variety of tasks, such as predicting the value of a field, forecasting future values, identifying patterns in data, and detecting anomalies from never-before-seen data. Machine learning operates best when you provide a clean, numeric matrix of data as the foundation for building your machine learning models.

About the example data, SPL, and images used in this document

The examples and images in this document are based on a fictional shop and use a synthetic dataset from various source types, including web logs and point-of-sale records. This dataset does not ship with the Machine Learning Toolkit.

The sample dataset lacks the kinds of organic issues addressed in this document. As a result, you might encounter these these errors with the Search Processing Language (SPL).

1. In Compute missing values the following line of SPL drops 25% of the data, creating a scenario of missing data to compute: | eval method_modified = if(random() % 4 = 0, null, method)

2. In Remedy unit disagreement the following line of SPL creates unit disagreement in the response_bytes_new field by converting some data to kilobytes (by multiplying by 10 and then dividing by 1024) and some to bytes (by multiplying by 10): | eval response_bytes_new = if (random()%5 = 0, response_bytes * 10.0 / 1024, response_bytes * 10)

3. In Cross-validation the following line of SPL uses the eval command to manufacture valid and fraud labels into the data: | eval label = if (n=-81, "valid", "fraud")

If you are new to adding data and searching data in the Splunk platform, consider taking the Splunk Enterprise Search Tutorial prior to working with your data in the Machine Learning Toolkit.
Machine data preparation stages

The Splunk platform accepts any type of machine data, including event logs, weblogs, live application logs, network feeds, system metrics, change monitoring, message queues, archive files, data from indexes, and third-party data sources. Ingested data typically goes through three stages in order to be ready for machine learning.

Stage 1

Data is ingested into the Splunk platform during the first stage. The data is typically semi-structured. You can see some commonality between the events, such as URLs and https calls in the following example:

Stage 2

After you perform some field extraction, the data is partially organized within a table. However, this table could include problems, such as missing data, non-numerical data, values in a widely-ranging scale, and words representing values. Data in the following example is not yet suitable for machine learning because the scales for the request_size and response_time are very different, which can negatively affect your chosen algorithm:
Following your completion of further field analysis and data cleaning, the data is in a clean matrix and is amenable to machine learning algorithms. This data has no missing values, is strictly numeric, and the values are scaled correctly. This data is ready for machine learning:

Identify the right data

The machine learning process starts with a question. Identify the problem you want to solve with machine learning, and use the subsequent answer as the basis to identify the data that will help you solve that problem. Ensure that the data you have is both relevant to the problem you want to solve, and that you have enough data to get meaningful results. Consider asking questions such as these:

- What data should be used to build the machine learning model?
- Do you have the right data for the type of machine learning you want to perform?
- Do you have a domain expert at your organization who you can ask about the best data to build the machine learning model?
- Is the data complete? Do you need or have access to other datasets that can support building the machine learning model?
- How should the data be weighted? Is recent data more relevant than historical data? Is data from users within an urban center more relevant?
Data integration

If you need content from more than one dataset in order to build your machine learning model, you must integrate the datasets. When you integrate the data, use an implicit join with the \texttt{OR} command rather than an SQL-style explicit join.

The Splunk platform is optimized to support implicit joins. Queries using implicit joins are more efficient than those using an SQL-style join.

The following examples join user login and logout data with the session ID data. The second example uses the recommended implicit join with the \texttt{OR} command.

\begin{verbatim}
SQL-style explicit join:
sourcetype=login | join sessionID ( search sourcetype=logout)
Implicit join:
sourcetype=login \texttt{OR} sourcetype=logout
\end{verbatim}

Clean your data

You need to clean most machine data before you can use it in machine learning. You need to clean your data in the following situations:

- The data contains errors, such as typos.
- The data has versioning issues. For example, a firmware update to a router affected how logging data is collected and how it reports on package amounts.
- The data is impacted by the upstream process, where scripts or searches add faulty fields or drop values.
- The data comes from an unreliable or out of date source that drops values.
- The data is incomplete.

Depending on the data issue, you have various methods to clean it.

Field extraction

Use field extraction techniques to integrate your data into a table. A number of fields are automatically available as a result of ingesting data into the Splunk platform, including \texttt{_time} and \texttt{sourcetype}. If the automatic field extraction options do not suffice, you can choose from technical add-ons through Splunkbase. Add-on options enable you to extract fields for certain types of data. You can also
perform manual field extraction using the built-in Interactive Field Extraction (IFX) graphical workflow and iterate with regexes and delimiters to extract fields.

- To access technical add-ons, see Splunkbase.
- For more information on IFX, see Build field extractions with the field extractor.

**Address missing values with fillnull command**

Before you can build a machine learning model, you must address missing values. The most straightforward option is to replace all the missing values with a reasonable default value, like zero. Do this by using the `fillnull` command.

In the following example, the `fillnull` command replaces empty fields with zero:

```
| search bytes_in=* 
| table JSESSIONID, bytes_in, bytes_in_modified 
| fillnull value=0
```

**Compute missing values**

An alternative to replacing missing values with zero is to compute the missing values. For example, if your data includes the contents in an online shopping basket and you know the prices of these items, you can combine this information to calculate the missing sale amount total.

The following image shows a data set that includes values you could use to compute missing values.
Another option uses the `eval` command to preserve some statistical properties of the data while replacing missing values. Using the `eval` command is a suitable method for scenarios in which you do not want to replace missing values with zero, but you lack a deterministic way to calculate the missing values.

The following example uses payment method data from a website. The gift card payment method has a 54% probability of use, and the credit card has a 45% probability of use:

```
| eval method_modified = if(random() % 4 = 0, null, method)  
| table JSESSIONID, method, method_modified  
| eval method_modified_filled = if(isnull (method_modified), if(random() % 100 < 54, "_GIFTCARD", "_CREDITCARD"), method_modified)
```

Use the `eval` command to replace missing values based on probability. The `eval` command acts like flipping a weighted coin, where the sides of the coin are weighted by the probability of each outcome. In this example the weights are the probability of the gift card or credit card payment method. By using this method to compute values, you preserve the distribution of the data. Missing fields are replaced according to the calculated probabilities, with 54% of the missing values replaced with gift card, and 45% of the missing values replaced with credit card.
Address missing values with the imputer algorithm

Another option to address missing values uses the imputer algorithm. The imputer algorithm replaces missing values with substitute values. Substitute values can be estimates or be based on other values in your dataset. Imputation strategies include mean, median, and most frequent. The default strategy is mean.

Pass the names of the fields to the imputer algorithm, along with arguments specifying the imputation strategy and the values representing missing data. The following example uses event information from `base_action` with the imputation strategy of most frequent to replace the missing values for `request_bytes_new`:

```
| eval request_bytes_new=if(random() % 3 = 0, null, base_action)
| fields - base_action
| fit Imputer request_bytes_new strategy=most_frequent
| eval imputed=if(isnull(request_bytes_new), 1, 0)
| eval request_bytes_new_imputed=round(Imputed_request_bytes_new, 1)
| fields - request_bytes_new, Imputed_request_bytes_new
```

After calculating the missing field, the imputer replaces the missing value with value computed by the chosen imputation strategy. To learn more about this algorithm, see [Imputer algorithm](#).

Remedy unit disagreement

Data with very high values and very low values in a column might be the nature of the data, or it could be a case of unit disagreement. If you suspect a change occurred in the way units of data are measured, experiment with rescaling the data to see how that affects your prediction accuracy. Consider unit disagreement as a potential red-flag that's suitable for further exploration.

You can detect unit disagreement by using box plot charting, or by time charting the field itself, to determine if there is a change in the order of magnitude over time. The data in the following example exhibits unit disagreement in the `response_bytes_new` column, where some values are in the thousands and others are single digits with a decimal. In this case, the unit disagreement stems from the smaller values being measured in kilobytes rather than bytes:
When you detect unit disagreement, address the problem by converting the data to comparable units. The following example SPL shows how to convert units using test data. The second line of the search introduces a data adjustment that forces a unit disagreement issue by making some values report in kilobytes. The third line of the search rescales values that are less than 10 by multiplying those values by 1,024, the number of bytes in a kilobyte. The original column for `response_bytes_new` is retained in case you observe better machine learning results without rescaling.

```
index=demo sourcetype="apache:access" saleamount="*"
| eval response_bytes_new = if(random()%5 = 0, response_bytes * 10.0 / 1024, response_bytes * 10)  
| eval rescaled=if(response_bytes_new < 10, response_bytes_new * 1024, response_bytes_new) 
| table JSESSIONID, response_bytes_new, rescaled
```

If you explicitly convert your data consider retaining your original data. By retaining your data, you will preserve the option to investigate and validate the hypothesis that the reporting value changed.

**Address differing feature scales**

Ingested data can have very different values and scales of values between columns. Value and value scale differences are normal given the nature of machine data, the different ways in which data is measured, and the amount of data available for analysis. Differing feature scales between columns does not prevent you from successfully using machine learning, but it may cause performance issues with your chosen algorithm. Look across your data for cases that might benefit from rescaling as a means to improve the accuracy of your prediction results.

In the following example, the values in the `request_bytes_new` column are less than 1, whereas the values in the `response_time_microseconds_new` column are...
100,000. You can rescale these values to improve your algorithm performance:

Rescale values between columns by using the StandardScaler algorithm. The StandardScaler algorithm is an option in the MLTK Assistants that include a Preprocessing Steps section. The specific Assistants that include built-in preprocessing are:

- Predict Numeric Fields
- Predict Categorical Fields
- Forecast Time Series
- Cluster Numeric Events

Select the fields to preprocess with respect to mean and standard deviation. When you are finished, the algorithm generates new columns of data in the data table. You can then examine the performance metrics for both the original values and the rescaled values.

**Convert categorical and numeric data**

Machine learning algorithms work best when you provide numeric data rather than explicit words or text. The Splunk platform can convert categorical and numeric data in different ways.

**Categorical to numeric**

In cases of categorical data, you can let the MLTK perform the conversion. The `fit` and `apply` commands both perform one-hot encoding on your data. After search results are pulled into memory and null events handled, `fit` and `apply` convert fields that contain strings or characters into binary indicator variables (1 or 0). This change is performed on the search results copy stored in memory.
To learn about the specific ML-SPL commands, see **ML-SPL commands**.

To learn more about the processes of the `fit` and `apply` commands, see **Understanding the fit and apply commands**.

Data conversion through one-hot encoding is not always the best option for your data. The following example includes data from ticketing system logs where tickets are marked as one of four categories: info, debug, warn, or critical. Converting these categorical values using one-hot encoding through the `fit` and `apply` commands results in binary numeric conversion, as seen in the following table. This manner of conversion is more useful when you want to know if values are equal or not equal.

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>0</th>
<th>0</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>INFO</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DEBUG</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>WARN</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>CRITICAL</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

In cases like categorical ticket logs, the numeric conversion goes beyond telling you that the values are different, but also how the values are different. You can use the `eval` command to convert categorical values in combination with a case. The case operates like a switch statement, converting the example ticket categories to a corresponding number:

- If INFO, then 1.
- If DEBUG, then 2.
- If WARN, then 3.
- If CRITICAL, then 4.

```
| eval severity_numerical = case(severity == "INFO", 1, severity == "DEBUG", 2, severity == "WARN", 3, severity == "CRITICAL", 4)
| table JSESSIONID, severity, severity_numerical
```

The output preserves the original `severity` column and creates a new column for `severity_numerical`. The `severity_numerical` column shows the numeric value for the corresponding ticket category:
Numeric to categorical

You might have data where a field is numeric, but the information from those numbers is not especially valuable. In the following example, the data includes several error code values, such as 91512, 81535, and 81715. Although the error code numbers 81535 and 81715 are closer numerically than 91512, there are no insights into what the numbers mean. You can gather more insight knowing if these values are equal or not equal by converting these numeric fields into categorical fields.

To coerce the ErrorCode column from numeric to categorical, use the eval command to add a string suffix. In this example, the suffix of _str is used.

| eval ErrorCodeCategorical = ErrorCode."_str" |
| table JSESSIONID, ErrorCode, ErrorCodeCategorical |
The new suffix results in the content being treated as categorical. Use the fields sidebar to identify fields with a pound sign (#) as numeric and a cursive letter "a" as categorical.

**Apply feature engineering**

Features are informative values and are fields in your data that you want to analyze using machine learning. Feature engineering is the process of gleaning informative, relevant values from your data so that you can create features to train your machine learning model. Feature engineering can make your data more predictive for your machine learning outcomes and goes beyond performing data cleaning.

To successfully perform feature engineering, you must have domain knowledge.

**Aggregates**

Aggregates let you combine data in different ways to create more valuable data for your model training. For example, rather than clustering user data based on initial events that correspond to a single page visit, you can use aggregates to compute data by days active or visits per day to represent the users. Alternatively, rather than clustering data based on visits, you can use aggregates to produce a user type, such as power user or new user, based on data in lookup files.

The following example uses lookup files for the computed aggregates of `user_activity`, `user_id`, `days_active`, and `visits_per_day`. The `fit` command finds clusters based on the quantities from these lookup files:

```bash
| stats count by action user_id
| xyseries user_id action count
| fillnull
| lookup user_activity user_id OUTPUT days_active visits_per_day
| fit KMeans k=10 days_active visits_per_day
```

**Feature interactions**

Another opportunity for feature engineering is feature interactions. Do you have features that might be related and do you want to explicitly capture that relationship in your data? In the Splunk platform, you can create a new feature based on the values of original features in your data.

For example, you could get insight from user role data and user industry data as independent values towards your machine learning goals. However, given that
these values are related, you might get better insight from combining these values. The same is true for values capturing escalations marked by importance or urgency. You can use the plus sign character (+) to perform string concatenation on two values.

The following SPL shows the + on example data:

```
| eval categorical_factor = role + industry
| eval numeric_factor = importance + urgency
```

**Non-linearity**

Feature engineering is suitable for data that's not easily separated by a linear model. You can use the `eval` command to introduce non-linearity and manipulate the data to use a linear model.

In the following visual, the example data is clustered into an inner circle and outer circle, both of which contain blue (0) and yellow (1) data points. With the current clustering, you have no way to draw a line to accurately separate these clusters. Because different algorithms work better on different data, you could consider selecting an algorithm such as DBscan to separate the clusters. A different algorithm might correct the division of clustered data, but it could also result in you losing control over the number of clusters. You can use non-linearity to compute a new field \( r \) as a function of the existing fields.
You can consider \( x \) and \( y \) as the 2 sides adjacent to the right angle of a triangle, and the new field of \( r \) as an operator of the hypotenuse of the triangle. This type of feature engineering computes the radius of the data point from a central point. When calculating \( r \) and fitting Kmeans clustering on the \( r \) field, rather than as originally calculated on \( x \) and \( y \), feature engineering separates the 2 circles of data:

\[
\text{radius} = \sqrt{x^2 + y^2}
\]

**Text analytics**

You can use feature engineering to perform machine learning on the basis of text data. Rather than explicitly converting text data to numeric, the term frequency-inverse document frequency (TFIDF) algorithm gives you the option to perform analytics and gain insights on text data as-is. See TFIDF algorithm.

TFIDF does not provide semantic values, such as defining which words have more meaning with respect to another metric. Semantic values are a separate metric to TFIDF. You can use lookups to bring in semantic values from another source.

TFIDF finds words and n-grams that are common in a given event, but that are rare in your set of events. A high TFIDF score indicates that the word or n-gram is potentially important. Create TFIDF features using the TFIDF preprocessing step in the MLTK, and then use the KMeans algorithm to cluster the raw data patterns.
Visualizing clustered data can be informative, but you cannot visualize high-dimensional data, like the output from TFIDF. By using the PCA algorithm, you can find a low-dimensional representation of your data that also captures the most interesting variation from your original data. See PCA algorithm.

The PCA algorithm finds sets of dimensions based on your original dimensions, and can create a two-dimensional representation of your data.
You could also use a three-dimensional scatterplot add-on from to move and twist the plotted data to see the clusters more clearly. See Splunkbase.

You can choose visualization options in the toolkit to observe if clusters become more or less common over time. Data projected into a two-dimensional space with colors representing source-type clusters lets you visualize cluster changes. Add visualizations to your dashboard to see changes as they arise:

![Visualization Example](image)

**Leading indicators**

Use leading indicators to increase the accuracy of your machine learning prediction or forecast. Leading indicators are sourced from data in the past, and you can use them to make future predictions based on current event data.

The following example predicts failures (`FAILS`) based on changes (`CHANGES`). Changes could stem from code commits, requested changes, or another proximal cause. You can use the `streamstats` command to pull information into the current event from past events. In combination with the `reverse` command, you can forecast future failures. By combining the `streamstats` and `reverse` commands, you can put all of your data into the same feature vector, which is a requirement for passing data to the `fit` command to make a prediction.

```
index-application_log OR index=tickets
| timechart span=1d count(failure) as FAILS count("Change Request") as CHANGES
| reverse
| streamstats window=3 sum(FAILS) current=f as FAILS_NEXT_3DAYS
| reverse
| fit LinearRegression FAILS_NEXT_3DAYS from CHANGES into FAILS_PREDICTION_MODEL
```
The first use of the `reverse` command flips time to examine future events before the past events. The `streamstats` command pulls data to the current event. The second use of the `reverse` command puts time back into the correct order, and is good practice for both SPL composition and if you need to put the results into a time-chart.

**Quantization**

Quantization allows you to translate multi-digit timestamps into discrete buckets. Quantized data is more readily comparable than raw timestamp data, which does not lend itself to comparison.

The following image includes timestamp data.

![Timestamp Data](image)

The following timestamp data looks like events that are far apart because the numbers are far apart. If these timestamp values were inputted into a machine learning algorithm the model would also interpret these values as being very different:

1553260380000
1552657320000
1552080679000

Quantization appropriately buckets timestamp data together. The timestamps that appeared to be different are in fact all from the same day of the week and at approximately the same time of day. Quantization allows you to group timestamp data as needed for your machine learning goals.

Friday March 22, 2019 at 1:13 pm
Friday March 15, 2019 at 1:42 pm
Friday March 8, 2019 at 1:31 pm

**Data splitting**

In machine learning, you don't want to train and test your model on the same data. If you test on the same data that you used to train the model, you cannot assess the quality of the generalization being produced by the model. As such,
part of the process of machine learning is splitting your data so that one portion is used for training, and a separate portion is used for testing. A common data split is 70% of data for training, and 30% of data for testing.

In cases where your data lacks well-balanced examples, or you lack sufficient amounts of data, is it important to perform data splitting using more specialized methods.

**Stratified sampling**

Class imbalance occurs when two classes in your data are present in different quantities. Suppose you want to use machine learning to detect instances of fraudulent transactions. Your data has limited examples of fraudulent transactions, but many examples of valid transactions. You must manage imbalanced data carefully in order to get more accurate results from your machine learning models.

The following image shows the class imbalance between the valid and fraud transaction data.

One method you can use to handle class imbalance is stratified sampling. The stratified sampling technique forces proportional representation of the minority class data.

Perform the following steps to implement stratified sampling:

1. In an example with a majority class for valid data and minority class for fraud data, begin by splitting the data into valid and fraud buckets.
2. Use the `sample` command to produce the partition number.
3. Include the partition number as well as a seed number in the search string. The seed number ensures that each call to the `sample` command
produces the same partition numbers.

4. Use one call to get the training set and one call to get the test set so that each event gets the same partition number in each of the two calls. If you failed to use the same seed, you would potentially assign an event different partition numbers in each of the two calls, which could cause some events to end up in both the training and test datasets. The following SPL sets the training sets for the valid and fraud examples. A data split of 70/30 is assumed. If less than or equal to 70, the examples go into the training set.

| search label="valid" |
| sample partitions=100 seed=1001 |
| where partition_number <=70 |
| outputlookup training_valid.csv |

| search label="fraud" |
| sample partitions=100 seed=1001 |
| where partition_number <=70 |
| outputlookup training_fraud.csv |

The following SPL shows sets up the testing set for the valid and fraud examples. A data split of 70/30 is assumed. If greater than 70, the examples go into the testing set.

| search label="valid" |
| sample partitions=100 seed=1001 |
| where partition_number >70 |
| outputlookup testing_valid.csv |

| search label="fraud" |
| sample partitions=100 seed=1001 |
| where partition_number >70 |
| outputlookup testing_fraud.csv |

These actions result in four lookup files with 70% of the fraud examples in the training set, and the remaining 30% of the fraud examples in the testing set.

5. You can use the `append` command to complete building your training and testing datasets. The following example uses the `append` command to merge the training samples from the majority and minority classes into one training dataset.

| inputlookup training_valid.csv | append | inputlookup training_fraud.csv |

The following SPL example uses the `append` command to merge the testing samples from the majority and minority classes into one testing dataset.
By using stratified sampling, you can create a balanced dataset with equal representation of both your classes, giving you more power to predict the minority class of fraud.

**Downsampling**

Another method to address class imbalance is downsampling. Downsampling is suitable for cases with a large number of minority class examples. You can downsample the majority class to get the same number of examples.

The following image shows the disparity in classes between valid and fraud data.

![Disparity in classes between valid and fraud data](image.png)

Use the `sample` command to downsample to the smallest class. The following SPL shows the `sample` command on the test data:

```spl
| sample 22958 by label
```

**Cross-validation**

Class imbalance also occurs when you don’t have large amounts of data. Cross-validation lets you use as much data as you can to train your model while allowing you to be confident that your model is generalizing well on unseen data. Cross-validation alternates holding out one piece of your data, fitting the model on the remaining data, and testing on the piece that was held out. Cross-validation repeats this pattern until all of your data is trained and tested.

Cross-validation is not well suited for time-series chart data.

Use the partition number field to leave out one partition at a time, and fit the model on the other partitions. The following SPL example shows two-fold cross-validation:

```spl
| eval label = if (n=-81, "valid", "fraud") [<- line 1 manufactures issue into data, do not copy/ paste]
| sample partitions=2 seed=43
| appendpipe [ where partition_number=0 ] fit LogisticRegression label
```
from event type date_hours state into validOrFraud ]
| where partition_number=1
| apply validOrFraud

You can also use built-in k-fold cross-validation in the MLTK. Add the kfold_cv field to the fit command to run cross-validation. To learn more about built-in k-fold cross-validation in the MLTK, see K-fold scoring.

The following diagram shows the pattern used in k-fold cross-validation of alternating holding out data for training and testing iterations:

![Diagram showing the pattern of k-fold cross-validation]

The following SPL example shows k-fold cross-validation on a test set:

```
| inputlookup iris.csv
| fit LogisticRegression species from * kfold_cv=3
```

**Learn more**

Machine data must be transformed and organized to be ready for machine learning. Tools in the Splunk platform let you identify the right data, integrate that data, clean the data, convert categorical and numeric data, apply feature engineering, and split the data for training and testing machine learning models.

Navigate to the following documents and resources to learn more about data preparation and machine learning:

- To learn how to preprocess data when using the Machine Learning Toolkit Assistants, see Preprocessing machine data using Assistants.
To learn how to use Experiments to bring all aspects of a monitored machine learning pipeline into one interface, see the Experiments overview.

To review the list of supported algorithms in the Machine Learning Toolkit, see Supported algorithms.

To find a cheat sheet of ML-SPL commands and machine learning algorithms used in the Machine Learning Toolkit, download the Machine Learning Toolkit Quick Reference Guide. This document is also available in Japanese.

To learn about implementing analytics and data science projects using Splunk’s statistics, machine learning, built-in and custom visualization capabilities, see the Splunk for Analytics and Data Science course.

Preprocessing machine data using Assistants

Preprocessing steps transform your machine data into fields ready for modeling or visualization. Preprocessing steps include algorithms that reduce the number of fields, produce numeric fields from unstructured text, or re-scale numeric fields.

This document covers preprocessing in the Experiments and Classic Assistant context. For information on all the available preprocessing algorithm options to use in the MLTK, please see the preprocessing section of the algorithms document.

When you use the Predict Numeric Fields, Predict Categorical Fields, or Cluster Numeric Events Assistants in the MLTK, you have the option to apply one or more preprocessing algorithms. Choose the algorithm to best suit the preprocessing needs of your data. There are five preprocessing algorithm options:

- FieldSelector
- KernelPCA
- PCA
- StandardScaler
- TFIDF

Preprocessing steps are included in the Smart Forecasting Assistant but not covered in this document. For details, see Smart Forecasting Assistant.
Apply preprocessing to your data

For both Experiment and Classic Assistant workflows, you'll always perform these steps:

1. Under the Preprocessing Steps section, click **+Add a step**.
2. From the Preprocess method drop-down menu, choose FieldSelector, KernelPCA, PCA, StandardScaler or TFIDF. Fill in the fields for the selected method.
3. Click **Apply** to perform the specified preprocessing.
4. Click **Preview Results** to see a table with the preprocessing results, including any newly created fields.

<table>
<thead>
<tr>
<th>If</th>
<th>Then</th>
</tr>
</thead>
<tbody>
<tr>
<td>You wish to add more than one preprocessing step:</td>
<td>Click <strong>+Add a step</strong> again in the Preprocessing Steps section. The preprocessing step includes new fields or settings generated by previous preprocessing steps. Additional preprocessing applies sequential transformations. After you apply each preprocessing step, review the results against your data. You can modify a preprocessing step as long as no other steps have been added. Once you add another preprocessing step, previous steps cannot be modified. You can be remove a step, but doing so also removes all subsequent preprocessing steps as well as any fields selected in later sections of the Assistant for model training and fitting.</td>
</tr>
<tr>
<td>You are satisfied with the preprocessing results:</td>
<td>Continue through the remaining Assistant sections to complete training and testing the model. When you click the <strong>fit</strong> button, it will both fit that model as well as any preprocessing steps. When you save the main model (available in the experiments workflows) preprocessing models are saved as well.</td>
</tr>
<tr>
<td>You are not satisfied with the results of your preprocessing:</td>
<td>Try to remove the preprocessing step, modify the settings in the preprocessing step, or add more preprocessing steps to apply more data transformations.</td>
</tr>
</tbody>
</table>
About the preprocessing algorithms

Choose the algorithm to best suit the preprocessing needs of your data. Refer to this list to guide your decision:

**FieldSelector**
The FieldSelector algorithm uses the scikit-learn GenericUnivariateSelect to select the best predictor fields based on univariate statistical tests.

**KernelPCA**
The KernelPCA algorithm uses the scikit-learn KernelPCA to reduce the number of fields by extracting uncorrelated new features out of data. It is strongly recommended to standardize fields using StandardScaler before using the KernelPCA method. To reduce the number of dimensions, use the KernelPCA or PCA algorithms to increase performance. KernelPCA and PCA can also be used to reduce the number of dimensions for visualization purposes, for example, to project into 2D in order to display a scatterplot chart.

**PCA**
The PCA algorithm uses the scikit-learn PCA algorithm to reduce the number of fields by extracting new uncorrelated features out of the data. To reduce the number of dimensions, use the PCA or KernelPCA algorithms to increase performance. PCA and KernelPCA can also be used to reduce the number of dimensions for visualization purposes, for example, to project into 2D in order to display a scatterplot chart.

**StandardScaler**
The StandardScaler algorithm uses the scikit-learn StandardScaler algorithm to standardize the data fields by scaling their mean and standard deviation to 0 and 1, respectively. This standardization helps to avoid dominance of one or more fields over others in subsequent machine learning algorithms. StandardScaler is useful when the fields have very different scales. StandardScaler standardizes numeric fields by centering about the mean, rescaling to have a standard deviation of one, or both.

**TFIDF**
The TFIDF algorithm converts raw text into numeric fields, making it possible to use that data with other machine learning algorithms. The TFIDF algorithm selects N-grams, which are groups of N consecutive string (or term), from fields containing free-form text and converts them into numeric fields amenable to machine learning. For example, running TFIDF on a field containing email Subjects might select the bi-gram 'project proposal' and create a field indicating
the weighted frequency of that bi-gram in each Subject.

**Fields specific to each preprocessing algorithm**

Each of the preprocessing algorithms has its own unique fields and field order. Within the toolkit, hover over any field to see more information, or review content below.

**FieldSelector**

1. Select the field to predict: Make a single selection from the drop-down menu.
2. Select the predictor fields: Click to choose one at a time, or choose to select all.
3. Type: Select categorical or numeric as the field to predict.
4. Mode: Select the mode for field selection.
   - Percentile: Select a percentage of fields with the highest scores.
   - K-best: Select the K fields with the highest scores.
   - False positive rate: Select fields with p-values below alpha based on a false positive rate (FPR) test.
   - False discovery rate: Select fields with p-values below alpha for an estimated false discovery rate (FDR) test.
   - Family-wise error rate: Select fields with p-values below alpha based on family-wise error rate (FWER).
5. Percent: Input a percent value of features to return.

**KernelPCA**
If you select 'KernelPCA as your preprocess method, the processed fields will be renamed PC_, for example, PC_1, PC_2.

1. Select the fields to preprocess: Click to choose one at a time, or choose to select all.
2. K (# of Components): Specify the number of principal components.
3. Gamma: Enter the kernel coefficient for the rbf kernel.
4. Tolerance: Enter the convergence tolerance. If 0, an optimal value is chosen using arpack.
5. Max iteration: Enter the maximum number of iterations. If not specified, an optimal value is chosen using arpack.

**PCA**

If you select PCA as your preprocess method, the processed fields are renamed PC_, for example: PC_1, PC_2.

1. Select the fields to preprocess: Click to choose one at a time, or choose to select all.
2. K (# of Components): Specify the number of principal components.

**StandardScaler**

Fields processed using StandardScaler are prefixed with ss_. For example, if you select StandardScaler as the preprocessing method and the crime_rate field for preprocessing, the standardized field is named SS_crime_rate

1. Select the fields to preprocess: Click to choose one at a time, or choose to select all.
2. Standardize Fields: Select whether to center values with respect to the mean, scale them with respect to the standard deviation, or both.
1. Select the field to preprocess: Make a single selection from the drop-down menu.

2. Max features: Build a vocabulary that only considers the top K features ordered by term frequency.

3. Max document frequency: Ignore terms that have a document frequency strictly higher than the given threshold.
   This field supports one of the following value types:
   - Integer: absolute count
   - Float: a frequency of documents (between 0 and 1)

4. Min document frequency (cut-off): Ignore terms that have a document frequency strictly lower than the given threshold.
   This field supports one of the following value types:
   - Integer: absolute count
   - Float: a frequency of documents (between 0 and 1)

5. N-gram range: The lower and upper boundary of the range of N-values for different N-grams to be extracted.

6. Analyzer: Select whether the feature is made of word or character N-grams. This field defaults as set to word. Choose "char" to treat each letter like a word, resulting in sub-strings of N consecutive characters, including spaces.

7. Norm: Norm used to normalize term vectors.

8. Token pattern: Regular expression denoting what constitutes a "token".

9. Stop words: Enter any words you want to omit from the analysis. Stop words typically include common words such as "the" or "an".
Troubleshooting

Frequently Asked Questions

Please see the following list of the most frequently asked questions about the Machine Learning Toolkit. Don't see the information you need? Ask your question and get answers through community support at Splunk Answers.

Getting started

How do I get started with machine learning in Splunk?

If you have not done so already, we recommend reviewing these introductory documents:

- New to Splunk
- Welcome to the Machine Learning Toolkit

We are active bloggers, and encourage you to read our wonderful tutorials on machine learning and other related topics:

- Interested in Statistical Forecasts and Anomalies? Check out Part 1, Part 2 and Part 3 of this blog
- If you have ITSI or are interested in predictive analytics, check out this blog on ITSI and Sophisticated Machine Learning
- If you have ITSI or are interested in custom anomaly detection check out Part 1 and Part 2 of this blog
- If you are hungry for ice cream or anomalies in three flavors, check out Part 1 and Part 2 of this blog

Visit our YouTube channel to see the MLTK in action.

We also have an active GitHub Community where you can not only connect with other MLTK users but also share and reuse custom algorithms.

Splunk also offers a course we encourage you to take: Splunk for Data Science and Analytics
**Do I have to use .csv files to load data into the MLTK?**

The Splunk platform accepts any type of data. In particular, it works with all IT streaming and historical data. The source of the data can be event logs, web logs, live application logs, network feeds, system metrics, change monitoring, message queues, archive files, data from indexes, third-party data sources, and so on. Basically any data that can be retrieved by a Splunk search can be used by the toolkit.

In general, data sources are grouped into the following categories.

<table>
<thead>
<tr>
<th>Data source</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Files and directories</td>
<td>Most data that you might be interested in comes directly from files and directories.</td>
</tr>
<tr>
<td>Network events</td>
<td>The Splunk software can index remote data from any network port and SNMP events from remote devices.</td>
</tr>
<tr>
<td>Other sources</td>
<td>Other input sources are supported, such as FIFO queues and scripted inputs for getting data from APIs, and other remote data interfaces.</td>
</tr>
</tbody>
</table>

For many types of data, you can add the data directly to your Splunk deployment. If the data that you want to use is not automatically recognized by the Splunk software, you need to provide information about the data before you can add it.

**What are the most common use cases for machine learning in Splunk?**

The most common use cases are as follows:

- Anomaly detection
- Prediction Analytics (Forecasting)
- Clustering
What is a Machine Learning Model and how is it different from a Splunk Data Model?

A machine learning model is an encoded lookup file created by from a `fit` search command using the `into` clause, persisting the learned behaviors to a file on disk for use in later searches on net new data using the `apply` command.

Splunk Data Models are knowledge objects for organizing and accelerating your data in the Splunk platform.

Why do I need a dedicated search head for MLTK app?

You need a dedicated search head for the MLTK if you are freely experimenting and creating large numbers of machine learning models of substantial size. The search load and the machine learning workload can get large and impact your production search environment. For applying machine learning models in production (generally extremely light on resource use), or periodically retraining production models, you should be able to use your normal Splunk infrastructure. Please work with your Splunk admin for your specific Splunk deployment.

Why am I seeing the error of `Error in `SearchOperator: loadjob`? 

You need to configure sticky sessions on your load balancer. For further information, see Use a load balancer with search head clustering.

MLTK know-how

Can I use the MLTK in other apps - how do I do that?

Yes you can use ML-SPL commands in other apps. You need to make the MLTK global if you want to use the ML-SPL commands across all the apps. Remember that the model files follow all the same rules as Splunk lookupfiles- permissions, access control, and replication.
Please follow these steps:

1. From the top navigation bar choose Apps ⇒ Manage Apps

![Screenshot of Splunk interface](image1.png)

2. Find the Splunk Machine Learning Toolkit in the list, and click on the Permissions link in the Sharing column.

![Apps list](image2.png)
3. Change the Sharing setting to **All apps** and (optionally) change any role-based permissions as well. Click **Save** when done.
**What are the performance costs of the MLTK searches?**

Machine learning requires compute resources and disk space. Each algorithm has a different cost, complicated by the number of input fields you select and the total number of events processed. Model files are lookups and will increase bundle replication costs.

For each algorithm implemented in ML-SPL, we measure run time, CPU utilization, memory utilization, and disk activity when fitting models on up to 1,000,000 search results, and applying models on up to 10,000,000 search results, each with up to 50 fields.

Ensure you know the impact of making changes to the algorithm settings by adding the ML-SPL Performance App for the Machine Learning Toolkit to your setup via Splunkbase.

**Do I have options outside of the 30 native algorithms in the toolkit?**

Yes! On-prem customers looking for solutions that fall outside of the 30 native algorithms can use GitHub to add more algorithms. Solve custom uses cases through sharing and reusing algorithms in the Splunk Community for MLTK on GitHub. Here you can also learn about new machine learning algorithms from the Splunk open source community, and connect with fellow users of the toolkit.

Cloud customers can also use GitHub to add more algorithms via an app. The Splunk GitHub for Machine learning app provides access to custom algorithms and is based on the Machine Learning Toolkit open source repo. Cloud customers need to create a support ticket to have this app installed.

To access the Machine Learning Toolkit open source repo, see the MLTK GitHub repo.

The Machine Learning Toolkit and Python for Scientific computing add-on must be installed in order for GitHub to work in your Splunk environment.

**What are the side effects of fit and apply on my data?**

Machine learning commands from the MLTK are very powerful and have a number automation steps built into them. Fit and Apply, detailed here, have a number of caveats and features to accelerate your success with machine learning in Splunk.
At the highest level:

- The `fit` command produces a learned model based on the behavior of a set of events.
- The `fit` command then applies that model to the current set of search results in the search pipeline.
- The `apply` command repeats the field selection of the `fit` command steps.

What the `fit` command does

1. Search results pulled into memory.
2. The `fit` command transforms the search results in memory through these data preparation actions:
   1. Discard fields that are null throughout all the events.
   2. Discard non-numeric fields with more than (>) 100 distinct values.
   3. Discard events with any null fields.
   4. Convert non-numeric fields into "dummy variables" by using one-hot encoding.
   5. Convert the prepared data into a numeric matrix representation and run the specified machine learning algorithm to create a model.
3. Apply the model to the prepared data and produce new (predicted) columns.
4. Learned model is encoded and saved as a knowledge object.

What the `apply` command does

1. Load the learned model.
2. The `apply` command transforms the search results in memory through these data preparation actions:
   1. Discard fields that are null throughout all the events.
   2. Discard non-numeric fields with more than (>) 100 distinct values.
   3. Convert non-numeric fields into "dummy variables" by using one-hot encoding.
   4. Discard dummy variables that are not present in the learned model.
   5. Fill missing dummy variables with zeros.
   6. Convert the prepared data into a numeric matrix representation.
3. Apply the model to the prepared data and produce new (predicted) columns.
What is `partial_fit` and how does it work?

If an algorithm supports `partial_fit`, you can incrementally learn on net new data without loading the entire training history in a single search.

We recommend watching this brief video for details on the ways you can have your machine learning workflows update and learn through time.: How Does the Splunk Machine Learning Toolkit Learn?

As with the `fit` command, you want a lightweight search. Please refer back to this question for more information.

Several of the MLTK algorithms offer the `partial_fit` option. For a detailed list of available algorithms, please see this document: Algorithms

What is automatic sampling/performance Settings for the MLTK and why should I change these?

By default, reservoir sampling is enabled and will start sampling once the maximum number of events crosses 100,000 in your search events prior to the `fit` command.

If you do not wish to enable reservoir sampling and have resources available on your Splunk machine, then you can disable it and change the number of maximum input to a preferred number. In an environment set aside for machine learning workloads, and to avoid impact with production searches, it is not uncommon to increase the `max_inputs` setting into the millions.

Please ensure you have enough compute and memory resources available before making these changes. You will likely need to change `max_memory_usage_mb` and other options in Settings as you increase the number of events you want to process.

How do you nest multiple uses of the `score` command?

For the time being you will need to nest your `score` commands. Follow a pattern such as in this example with your own data.

```
| inputlookup track_day.csv
| sample partitions=100 seed=1234
| search partition_number > 70
| apply example_vehicle_type as DT_prediction probabilities=true
| multireport
```
Model management

How often should I run \texttt{fit} to retrain models?

In general, you are unlikely to need to run a \texttt{fit} search to update a specific set of models in production more often than once a day. When you are exploring and experimenting, you may be running \texttt{fit} more frequently to iteratively create your production machine learning solutions.

You should consider following factors:

- How often is your data significantly changing it's overall behavior?
- How resource expensive is your base search before the machine learning commands (for example are you loading 3 billion events with your search over a 30 day window, and the search takes 45 minutes to load before the first \texttt{fit} command is called?)?
- How computationally intensive are your selected algorithms? Remember to check out the ML SPL performance app!

Consider accelerating your base search, perhaps using Data Models or Summary Indexes, to speed up the base search!

How do I manage version control for my model files?

We do not have model history as part of the MLTK today, but we do have Experiment history stored to reload any saved change made to your Experiment.

If you want to version your models, remember they are just Splunk lookup objects and follow all the rules of Splunk knowledge objects. You can rename your models just like any other lookup object.

Renaming of models in this instance refers to those outside of the Experiment Management Framework. Not models created within an Experiment.

How do I move my model files from one Splunk instance to another?

Model files are lookups in Splunk and follow all the rules for lookups - so you can find the files on disk and with command line access you can move those lookups to another Splunk instance. Click to learn more about \texttt{namespacing and permissions} of lookups.
**How do you feed data from an existing Splunk data model into the Machine Learning Toolkit?**

This is done in the same way you search for data in a Data Model anywhere else in Splunk.

For example: `| datamodel network_traffic search | search tag=destination`

Remember that any data that can be retrieved by a Splunk search can be used with the Machine Learning Toolkit, including data from indexes or third-party data sources. You simply append that search with the applicable `| fit ...` or `| apply ...` command.
Release notes

What's new

Here's what's new in each version of the Splunk Machine Learning Toolkit:

Version 4.3.0

Features and improvements

- Introduction of the new Smart Forecasting Assistant. This Assistant offers enhanced time-series analysis for users with little to no SPL knowledge, and leverages the StateSpaceForecasting algorithm.
- Gain familiarity with the new Smart Forecasting Assistant with three new Showcase examples. Use these Showcases to click through the updated user interface and view forecast parameter options prior to working with your own data.
- Introduction of the NPR algorithm for feature extraction.
- A new document covering methods for preparing your data for machine learning is now available.
- The sample parameter is now available for use with the DensityFunction algorithm.
- Time-saving MLTK macros are now documented for your review and use.

In order to save models, users need the upload_lookup_files capability included in their role.

Version 4.2.0

Features and improvements

- Pairwise distances scoring now fully supports the wildcard (*) character.
- Pairwise distances scoring now supports the Kolmogorov-Smirnov (2 samples) and Wasserstein distance metrics.
- Wasserstein distance is now available as a statstest scoring method.
- The following score commands now support the wildcard (*) character in 1-to-n cases: Silhouette score, Accuracy score, F1-score, Precision score, Recall score, R2 score, Explained variance score, Mean absolute error score, Mean squared error score, T-test (2 independent samples) score, T-test (2 related samples) score.
• Introduction of a 3D Scatter Plot visualization. Use this visualization to see clusters of similar data points, or to drill down to see singular data points.
• Introduction of the StateSpaceForecast algorithm for time series analysis.
• Introduction of the ICA algorithm for preprocessing.
• Introduction of the DensityFunction algorithm for anomaly detection.
• Models created using the PCA algorithm can be inspected with the summary command.
• Models created using the FieldSelector algorithm can be inspected with the summary command.
• Time-saving MLTK macros are now documented for your review and use.
• Experiment Assistant alerts have been updated. When creating an alert, you now go directly to the Save As Alert window. Use this space to select from standard Trigger Conditions, as well as new Machine Learning Conditions.

Version 4.1.0

Features and improvements

• The wildcard character (*) is now enabled for single array scoring methods. For more information, see Using the score command.
• Introduction of the Hashing vectorizer algorithm.
• Introduction of the Pairwise distances scoring method.
• Introduction of the Imputer preprocessing algorithm.
• When using the BernoulliNB algorithm, GaussianNB algorithm, and MLPClassifier algorithm you can now inspect trained models using the summary command.
• The variance parameter is now available when using the PCA algorithm.
• The anomaly_score parameter is now available when using the LocalOutlierFactor algorithm.
• The fit_intercept and normalize parameters are now available when using the Lasso algorithm.
• The Box plot visualization option has been updated and improved.
• The Machine Learning Toolkit as deployed on Splunk Enterprise now sends anonymized data to Splunk Inc. Learn more here.
• We've also published a new document to capture the most Frequently Asked Questions regarding the Machine Learning Toolkit.
**Version 4.0.0**

*Features and improvements*

- A number of new demos have been added to the Showcase tab based on customer requests.
- The Experiment History tab now captures the model history from scheduled retraining.
- Introduction of the LocalOutlierFactor algorithm. Accessing this algorithm requires an upgrade to PSC version 1.3.
- The MLPClassifier algorithm now supports `partial_fit` for incremental learning.
- Introduction of the score command to validate models and statistical tests for any use case.
- Introduction of k-fold cross validation as a method to help test for model overfitting.
- A new tab called Settings is now part of the MLTK nav bar. Users with admin access can work within this interface to configure the toolkit settings of the `fit` and `apply` commands. Ensure you know the impact of setting changes by adding the ML-SPL Performance App for the Machine Learning Toolkit to your setup.
- Customers can now share and reuse custom algorithms in the Splunk Community for MLTK on GitHub.
- The Splunk MLTK Connector for Apache Spark™ allows users to leverage their own Spark clusters for large data sets. This is a limited availability release. Reach out to Splunk’s ML Spark team for LAR application access.
- The Splunk MLTK Container for TensorFlow™ is an add-on docker container, allowing multiple local GPU/ CPU acceleration. This is available via Splunk’s Professional Services department.
- For bug fixes, see Fixed issues.

**Version 3.4.0**

*Features and improvements*

- Version 1.3 of the Python for Scientific Computing add-on is now available in Splunkbase. Upgrading to version 3.4 of the MLTK requires upgrading to PSC version 1.3.
- Introduction of the MLPClassifier algorithm. Accessing this algorithm requires an upgrade to PSC version 1.3.
- Introduction of Boxplot Chart to the search visualization options.
- Models created within the Experiments framework can now be published and more easily used outside of the MLTK environment.
• For bug fixes, see Fixed issues.

Upgrading to version 3.4.0 of the MLTK requires upgrading to version 1.3 of the Python for Scientific Computing add-on. Two previous versions of the MLTK (3.2.0 and 3.3.0) will successfully operate on version 1.2 or 1.3 of the PSC add-on. However, users cannot access new features in the 3.4.0 MLTK without updating to that version.

Version 3.3.0

Features and improvements

• Introduction of another preprocessing method - TFIDF (term frequency-inverse document frequency).
• Addition of the RobustScaler algorithm.
• For bug fixes, see Fixed issues.

Version 3.4.0 of the MLTK will require an update to Version 1.3 of Python for Scientific Computing. The release for the MLTK 3.4.0 will coincide with the availability of PSC 1.3 in Splunkbase.

Version 3.2.0

Features and improvements

• Introduction of the Experiment Management Framework. This framework ties the experiment, along with any alerts or scheduled trainings, together. Users can now see which alerts or scheduled trainings are assigned to any experiment, and which experiment has or has not undergone pre-processing steps. See Using the Splunk Machine Learning Toolkit and Experiments.
• Relocation within the Machine Learning Toolkit of the previously free-standing Assistant module. This version of Assistants now lives under the Legacy tab of the MLTK bar. It is recommended that you do not create Models via this version of Assistants, and instead create Models via the Experiments Management Framework. Doing so will ensure that you can both:
  ♦ Create Alerts and Scheduled Trainings on the saved Experiment
  ♦ See Alerts and Scheduled Trainings organized by Experiment
• The Splunk Machine Learning Toolkit version 3.2 does not support Splunk Enterprise version 6.4 or earlier.
• For bug fixes, see Fixed issues.
Version 3.1.0

Features and improvements

- The FieldSelector algorithm can now be used in the preprocessing panel. See FieldSelector in the Machine Learning Toolkit User Guide.
- The maximum number of distinct values supported for categorical fields, formerly 100, can now be configured for both features fields and target fields.
  - Use `max_distinct_cat_values` to change the setting for feature fields, the input to your ML algorithm.
  - Use `max_distinct_cat_values_for_classifiers` to change the setting for a target field, the field you are trying to predict, in classifier algorithms.
- The Splunk Machine Learning Toolkit has a new clustering algorithm:
  - X-Means
- For bug fixes, see Fixed issues.

Version 3.0.0

Features and improvements

Introduced a new interface for managing models. You can now easily see what types of models you have, inspect the settings of each model (such as which variables were used to train it), and view or update each model's sharing settings. For more information, see Manage models.

Version 2.4.0

Features and improvements

- You must update any custom algorithms from earlier releases (before 2.4.0) to use the algos.conf file. See Register an algorithm in the ML-SPL API Guide.
- You can now package and distribute custom algorithms as apps. See Package an algorithm for Splunkbase in the ML-SPL API Guide.
- The Splunk Machine Learning Toolkit has two new algorithms.
  - Gradient Boosting Classifier
  - Gradient Boosting Regressor
- For bug fixes, see Fixed issues.
Version 2.3.0

Features and improvements

- The Splunk Machine Learning Toolkit's custom search commands are now fully integrated with Splunk's knowledge object permissions. For more details, see Models.

- Entries in the "Load Existing Settings" tab are now unique per-user instead of being shared with all users. Entries created prior to version 2.3 will continue to be accessible by all users.

- Two new algorithms have been added:
  - ACF (autocorrelation function)
  - PACF (partial autocorrelation function)
- The Forecast Time Series assistant now allows for the selection of the ARIMA forecasting algorithm. Additional panels have been added for inspecting properties unique to ARIMA models.

Version 2.2.1

This version contains bug fixes. See Fixed issues for details.

Version 2.2.0

Features and improvements

- The preprocessing feature has been redesigned and is offered in the Predict Numeric Fields, Predict Categorical Fields, and Clustering Numeric Events assistants. See Preprocessing for information.

- The ML-SPL API has been updated to make it easier for developers and partners to import custom algorithms in order to extend the capabilities of the Splunk Machine Learning Toolkit. See ML-SPL API Guide for information.

- A new video overview of what's new in versions 2.1.0 and 2.2.0 of the Splunk Machine Learning Toolkit.
Version 2.1.0

Features and improvements

Enhancements to the Detect Numeric Outliers assistant:

- You can now specify one or more fields to split by (up to 5). Specifying
  one or more split by fields enables you to see the values of the field you
  are analyzing grouped by the values of the split by fields in visualizations.
- Enhanced visualizations including a new Data Distribution histogram that
  shows the number of data points within the threshold and the number of
  data points outside the threshold.

For more information, see Detect Numeric Outliers.

Version 2.0.1

The Downsampld Line Chart custom visualization now supports the same
drilldown actions as the built-in Line Chart visualization.

Version 2.0.0

Features and improvements

- The app has been renamed to "Machine Learning Toolkit."
- New Cluster Numeric Events assistant that steps you through how to
  perform clustering on your own data. This assistant includes the ability to
  preprocess data by applying StandardScaler, PCA, or KernelPCA
  methods. See Cluster Numeric Events.
- Updated examples for the Cluster Numeric Events showcase.
- A streaming_apply setting has been added to the mlspl.conf file, which
  allows you to run the apply command on your indexers. For details, see
  Use your indexers to apply models.
- The Predict Numeric Fields and Predict Categorical Fields assistants now
  support multiple algorithms.
- A new visualization type has been added: Scatterplot matrix. This
  visualization is available in the Cluster Numeric Events assistant.
- The Machine Learning Toolkit app has a walk-through tour and each
  assistant has its own walk-through tour.
- A link to machine learning video tutorials has been added to the top menu
  bar and the Showcase page.
- Tooltips have been added for the fields in each of the assistants.
Algorithms

• The SGDClassifier algorithm is now supported. For details, see Algorithms.
• The SGDRegressor algorithm is now supported. For details, see Algorithms.
• The ARIMA algorithm is now supported. For details, see Algorithms.
• The LogisticRegression algorithm supports a new parameter probabilities=<true|false>. For details, see Algorithms.
• Summary support has been added to the RandomForestClassifier and RandomForestRegressor algorithms. For details, see Algorithms.
• The BernoulliNB, GaussianNB, Birch, and StandardScaler algorithms support a new parameter partial_fit=<true|false>. For details, see Algorithms.

Version 1.3.0

Features and improvements

• You can now create alerts within the Machine Learning Toolkit from some of the panels in the assistants. Alerts can be viewed under Scheduled Jobs > Alerts.
• You can now schedule model training in the Predict Numeric Fields and Predict Categorical Fields assistants by clicking the icon on the right side of the Fit Model button.

Schedules can be viewed under Scheduled Jobs > Scheduled Training.

• The Training/Test split can now be set to a 100/0 split (no split).

Version 1.2.0

Features and improvements

• The DecisionTreeClassifier and DecisionTreeRegressor algorithms are now supported. For details, see Algorithms.
• The Detect Numeric Outliers assistant now includes an Include current point checkbox to support the "current" parameter of the streamstats command.
- The Predict Numeric Fields assistant has an improved Actual vs. Predicted Line Chart, which replaces the Actual vs. Predicted Overlay.
- Two macros in the Forecast Time Series assistant have been merged into one macro.
- The `max_features` parameter of the RandomForestClassifier and RandomForestRegressor algorithms now accepts values with the `float` data type.
- The **Remove from history** confirmation dialog box has been improved.
- A basic framework has been implemented for displaying Bootstrap’s modal dialog boxes in the Machine Learning Toolkit and Showcase UI.

**Version 1.1.0**

**Features and improvements**

- The visualizations in the Cluster Events showcase have been updated.
- The Predict Numeric Fields and Predict Categorical Fields assistants now allow you to enter wildcards in **Fields to use for predicting**. For example, to specify both the Packets Received and Packets Sent fields, enter "Packets*". Wildcards are case sensitive.
- The **Select All** and **Select None** buttons on the Predict Numeric Fields and Predict Categorical Fields assistants have been moved inside the dropdown list.

**Algorithms**

- The KernelRidge regression algorithm is now supported. For details, see Algorithms.

**Bug fixes**

The following bugs were fixed:

- Changing the time range or search mode on assistant search bars will now re-run the search in the search bar, the same as the default Search page in Splunk Enterprise.
- Custom visualizations will now display time stamps correctly when the event time differs from browser time.
- Caching issues have been fixed, and the app no longer loads old versions of resources after an update.
- Exit points in assistants now correctly have the same time range as that assistant’s search bar.
Version 1.0.0

This is the first release of the Machine Learning Toolkit and Showcase app.

Known issues

The following are known issues in the Splunk Machine Learning Toolkit version 4.3.0.

<table>
<thead>
<tr>
<th>Date filed</th>
<th>Issue number</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>2019-06-17</td>
<td>MLA-3204</td>
<td>Error while fitting &quot;DensityFunction&quot; model: Domain error in arguments.</td>
</tr>
<tr>
<td>2019-05-14</td>
<td>MLA-3100</td>
<td>&quot;Publish&quot; link on the Experiments list page is enabled for Experiments that have never fit a model</td>
</tr>
<tr>
<td>2019-05-14</td>
<td>MLA-3099</td>
<td>Experiments list page will sometimes only show a single Experiment</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Workaround: Refreshing the page will load all Experiments.</td>
</tr>
<tr>
<td>2019-03-28</td>
<td>MLA-2965</td>
<td>&quot;Edit Job Settings&quot; does not work in Assistants</td>
</tr>
<tr>
<td>2018-09-24</td>
<td>MLA-2547</td>
<td>Experiments can't be renamed until they have been filled and saved</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Workaround: Load the Experiment, fill all necessary fields, and save it. You'll be able to rename the Experiment in the Save dialog.</td>
</tr>
<tr>
<td>2018-06-12</td>
<td>MLA-2258</td>
<td>TFIDF returns a vague error message when the &quot;token_pattern&quot; parameter is invalid</td>
</tr>
<tr>
<td>2018-04-18</td>
<td>MLA-2147</td>
<td>search.log does not show tracebacks on Windows on Splunk 7.1</td>
</tr>
</tbody>
</table>
|            |               | Workaround: The user can still look at the logs on windows in mlspl.log which captures the traceback. However, many users may not have access to mlspl.log (as it is indexed in the _internal index which normal users}
don't have by default). If this is the case, they will need to ask their Splunk administrators for access to this index.

<table>
<thead>
<tr>
<th>Date filed</th>
<th>Issue number</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>2017-08-25</td>
<td>MLA-1916</td>
<td>Uncaught StatsModels error with ARIMA order=0-2-0</td>
</tr>
<tr>
<td>2017-02-28</td>
<td>MLA-1670</td>
<td>&quot;Outliers Over Time&quot; plot doesn't display any data if the _time field does not contain a Unix timestamp.</td>
</tr>
<tr>
<td>2016-12-13</td>
<td>MLA-1485</td>
<td>In the Forecast Time Series assistant, forecasting a very large number of points may cause the browser to crash.</td>
</tr>
</tbody>
</table>

### Fixed issues

The following are fixed issues in the Splunk Machine Learning Toolkit version 4.3.0.

<table>
<thead>
<tr>
<th>Date resolved</th>
<th>Issue number</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>2019-04-12</td>
<td>MLA-2945</td>
<td>DensityFunction does not support the &quot;as&quot; clause</td>
</tr>
<tr>
<td>2019-03-21</td>
<td>MLA-2920</td>
<td>DensityFunction - IsOutlier and BoundaryRanges are blank if split by fields are numeric</td>
</tr>
</tbody>
</table>

### Third-party software

The Splunk Machine Learning Toolkit contains some components or libraries that were written by others and are being redistributed as part of the app under their respective open source licenses. We want to thank the contributors of these projects.

View the licenses by selecting a library name on the right.

**Anaconda/Miniconda**

https://docs.continuum.io/anaconda/eula

Anaconda/Miniconda version 3.19.0

Copyright 2016, Continuum Analytics, Inc.
All rights reserved under the 3-clause BSD License:

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- Neither the name of Continuum Analytics, Inc. nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL CONTINUUM ANALYTICS, INC. BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Notice of Third Party Software Licenses

Anaconda contains open source software packages from third parties. These are available on an "as is" basis and subject to their individual license agreements. These licenses are available in Anaconda or at http://docs.continuum.io/anaconda/pkg-docs. Any binary packages of these third party tools you obtain via Anaconda are subject to their individual licenses as well as the Anaconda license. Continuum reserves the right to change which third party tools are provided in Anaconda.

Cryptography Notice

This distribution includes cryptographic software. The country in which you currently reside may have restrictions on the import, possession, use, and/or re-export to another country, of encryption software. BEFORE using any
encryption software, please check your country's laws, regulations and policies concerning the import, possession, or use, and re-export of encryption software, to see if this is permitted. See the Wassenaar Arrangement <http://www.wassenaar.org/> for more information.

Continuum Analytics has self-classified this software as Export Commodity Control Number (ECCN) 5D002.C.1, which includes information security software using or performing cryptographic functions with asymmetric algorithms. The form and manner of this distribution makes it eligible for export under the License Exception ENC Technology Software Unrestricted (TSU) exception (see the BIS Export Administration Regulations, Section 740.13) for both object code and source code.

The following packages are included in this distribution that relate to cryptography:

openssl: The OpenSSL Project is a collaborative effort to develop a robust, commercial-grade, full-featured, and Open Source toolkit implementing the Transport Layer Security (TLS) and Secure Sockets Layer (SSL) protocols as well as a full-strength general purpose cryptography library.

pycrypto: A collection of both secure hash functions (such as SHA256 and RIPEMD160), and various encryption algorithms (AES, DES, RSA, ElGamal, etc.).

pyopenssl: A thin Python wrapper around (a subset of) the OpenSSL library.

kerberos (krb5, non-Windows platforms): A network authentication protocol designed to provide strong authentication for client/server applications by using secret-key cryptography.

cryptography: A Python library which exposes cryptographic recipes and primitives.

conda

http://conda.pydata.org/docs/license.html

conda version 3.19.0

(c) 2015 Continuum Analytics, Inc. http://continuum.io All Rights Reserved
Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- Neither the name of Continuum Analytics, Inc. nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL CONTINUUM ANALYTICS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

conda-env

https://github.com/conda/conda-env/blob/develop/LICENSE.txt

conda-env version 2.4.5

(c) 2014 Continuum Analytics, Inc. http://continuum.io All Rights Reserved

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
functools32

PSF LICENSE AGREEMENT FOR PYTHON 2.7.14

1. This LICENSE AGREEMENT is between the Python Software Foundation ("PSF"), and the Individual or Organization ("Licensee") accessing and otherwise using Python 2.7.14 software in source or binary form and its associated documentation.

2. Subject to the terms and conditions of this License Agreement, PSF hereby grants Licensee a nonexclusive, royalty-free, world-wide license to reproduce, analyze, test, perform and/or display publicly, prepare derivative works, distribute, and otherwise use Python 2.7.14 alone or in any derivative version, provided, however, that PSF's License Agreement and PSF's notice of copyright, i.e., "Copyright © 2001-2017 Python Software Foundation; All Rights Reserved" are retained in Python 2.7.14 alone or in any derivative version prepared by Licensee.

3. In the event Licensee prepares a derivative work that is based on or incorporates Python 2.7.14 or any part thereof, and wants to make the derivative work available to others as provided herein, then Licensee hereby agrees to include in any such work a brief summary of the changes made to Python 2.7.14.

4. PSF is making Python 2.7.14 available to Licensee on an "AS IS" basis. PSF MAKES NO REPRESENTATIONS OR WARRANTIES, EXPRESS OR IMPLIED.
BY WAY OF EXAMPLE, BUT NOT LIMITATION, PSF MAKES NO AND
DISCLAIMS ANY REPRESENTATION OR WARRANTY OF
MERCHANTABILITY OR FITNESS FOR ANY PARTICULAR PURPOSE OR
THAT THE USE OF PYTHON 2.7.14 WILL NOT INFRINGE ANY THIRD PARTY
RIGHTS.

5. PSF SHALL NOT BE LIABLE TO LICENSEE OR ANY OTHER USERS OF
PYTHON 2.7.14 FOR ANY INCIDENTAL, SPECIAL, OR CONSEQUENTIAL
DAMAGES OR LOSS AS A RESULT OF MODIFYING, DISTRIBUTING, OR
OTHERWISE USING PYTHON 2.7.14, OR ANY DERIVATIVE THEREOF, EVEN
IF ADVISED OF THE POSSIBILITY THEREOF.

6. This License Agreement will automatically terminate upon a material breach of
its terms and conditions.

7. Nothing in this License Agreement shall be deemed to create any relationship
of agency, partnership, or joint venture between PSF and Licensee. This License
Agreement does not grant permission to use PSF trademarks or trade name in a
trademark sense to endorse or promote products or services of Licensee, or any
third party.

8. By copying, installing or otherwise using Python 2.7.14, Licensee agrees to be
bound by the terms and conditions of this License Agreement.

**highcharts-downsample**

https://github.com/sveinn-steinarsson/highcharts-downsample

highcharts-downsample version 0.1

MIT License

Permission is hereby granted, free of charge, to any person obtaining a copy of
this software and associated documentation files (the "Software"), to deal in the
Software without restriction, including without limitation the rights to use, copy,
modify, merge, publish, distribute, sublicense, and/or sell copies of the Software,
and to permit persons to whom the Software is furnished to do so, subject to the
following conditions:

The above copyright notice and this permission notice shall be included in all
copies or substantial portions of the Software.
THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NON INFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

jsonschema

MIT License

Copyright (c) 2016

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

libgfortran

libgfortran version 3.0.0

http://www.gnu.org/licenses/gpl.html
http://www.gnu.org/licenses/gcc-exception-3.1-faq.html
http://www.gnu.org/licenses/gcc-exception.html
**MKL**

https://anaconda.org/anaconda/mkl https://docs.continuum.io/anaconda/eula

MKL version 11.3.1

Anaconda/Miniconda contains re-distributable, run-time, shared-library files from the Intel(TM) Math Kernel Library (MKL binaries).

**no-data-to-display**

http://code.highcharts.com/

no-data-to-display 5.0.7

(c) 2010-2016 Highsoft AS

**msvc_runtime**

https://anaconda.org/anaconda/msvc_runtime

msvc_runtime version 1.0.1

Copyright © 2018 Anaconda, Inc. All Rights Reserved.

Anaconda Accelerate ("the Software Product" or "Accelerate") and accompanying documentation is licensed and not sold. The Software Product is protected by copyright laws and treaties, as well as laws and treaties related to other forms of intellectual property. Continuum Analytics, Inc. or its subsidiaries, affiliates, and suppliers (collectively "Continuum") own intellectual property rights in the Software Product. The Licensee's ("you" or "your") license to download, use, copy, or change the Software Product is subject to these rights and to all the terms and conditions of this End User License Agreement ("Agreement").

**nomkl**

https://anaconda.org/anaconda/nomkl

nomkl version 1.0.0

Copyright (C) 2006, Jonathan E. Taylor All rights reserved.
This library is distributed in the hope that it will be useful, but WITHOUT ANY
WARRANTY; without even the implied warranty of MERCHANTABILITY or
FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public
License for more details.

You should have received a copy of the GNU Lesser General Public License
along with this library; if not, write to the Free Software Foundation, Inc., 51
Franklin St, Fifth Floor, Boston, MA 02110-1301 USA

NumPy

http://www.numpy.org/license.html

NumPy version 1.13.1, 1.13.3

Copyright © 2005-2013, NumPy Developers. All rights reserved.

Redistribution and use in source and binary forms, with or without modification,
are permitted provided that the following conditions are met:

• Redistributions of source code must retain the above copyright notice, this
  list of conditions and the following disclaimer.

• Redistributions in binary form must reproduce the above copyright notice,
  this list of conditions and the following disclaimer in the documentation
  and/or other materials provided with the distribution.

• Neither the name of the NumPy Developers nor the names of any
  contributors may be used to endorse or promote products derived from
  this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND
CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES,
INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR
CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT
NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;
LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR
OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

OpenBLAS

https://github.com/xianyi/OpenBLAS/blob/develop/LICENSE

OpenBLAS version 0.2.19

Copyright (c) 2011-2014, The OpenBLAS Project All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
3. Neither the name of the OpenBLAS project nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

pandas

https://github.com/pydata/pandas/blob/master/LICENSE

pandas version 0.22.0

Copyright (c) 2011-2012, Lambda Foundry, Inc. and PyData Development Team All rights reserved.
Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

- Neither the name of the copyright holder nor the names of any contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDER AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

patsy

https://github.com/pydata/patsy

patsy version 0.4.0

Copyright (C) 2011-2012, Patsy Developers. All rights reserved.
Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS AND ANY EXPRESS OR IMPLIED WARRANTIES, Including, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

pip

https://github.com/pypa/pip/blob/develop/LICENSE.txt

pip version 7.1.2

Copyright (c) 2008-2016 The pip developers (see AUTHORS.txt file)

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

plotly.js

https://github.com/plotly/plotly.js

plotly.js version 1.42.5

The MIT License (MIT)

Copyright (c) 2019 Plotly, Inc

Permission is hereby granted, free of charge, to any person obtaining a copy of
this software and associated documentation files (the "Software"), to deal in the
Software without restriction, including without limitation the rights to use, copy,
modify, merge, publish, distribute, sublicense, and/or sell copies of the Software,
and to permit persons to whom the Software is furnished to do so, subject to the
following conditions:

The above copyright notice and this permission notice shall be included in all
copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY
KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE
WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR
PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS
OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR
OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR
OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

psutil

https://github.com/giampaolo/psutil/blob/master/LICENSE

psutil version 5.4.3

Copyright (c) 2009, Jay Loden, Dave Daeschler, Giampaolo Rodola' All rights
reserved.

Redistribution and use in source and binary forms, with or without modification,
are permitted provided that the following conditions are met:
• Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
• Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
• Neither the name of the psutil authors nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

pycosat

https://github.com/ContinuumIO/pycosat/blob/master/LICENSE

pycosat version 0.6.1

Copyright (c) 2013, Ilan Schnell, Continuum Analytics, Inc.

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE
WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

Python

https://docs.python.org/2/license.html

Python version 2.7.11

This LICENSE AGREEMENT is between the Python Software Foundation ("PSF"), and the Individual or Organization ("Licensee") accessing and otherwise using Python 2.7.10 software in source or binary form and its associated documentation.

Subject to the terms and conditions of this License Agreement, PSF hereby grants Licensee a nonexclusive, royalty-free, world-wide license to reproduce, analyze, test, perform and/or display publicly, prepare derivative works, distribute, and otherwise use Python 2.7.10 alone or in any derivative version, provided, however, that PSF's License Agreement and PSF's notice of copyright, i.e., "Copyright © 2001-2015 Python Software Foundation; All Rights Reserved" are retained in Python 2.7.10 alone or in any derivative version prepared by Licensee.

In the event Licensee prepares a derivative work that is based on or incorporates Python 2.7.10 or any part thereof, and wants to make the derivative work available to others as provided herein, then Licensee hereby agrees to include in any such work a brief summary of the changes made to Python 2.7.10.

PSF is making Python 2.7.10 available to Licensee on an "AS IS" basis. PSF MAKES NO REPRESENTATIONS OR WARRANTIES, EXPRESS OR IMPLIED. BY WAY OF EXAMPLE, BUT NOT LIMITATION, PSF MAKES NO AND DISCLAIMS ANY REPRESENTATION OR WARRANTY OF MERCHANTABILITY OR FITNESS FOR ANY PARTICULAR PURPOSE OR THAT THE USE OF PYTHON 2.7.10 WILL NOT INFRINGE ANY THIRD PARTY RIGHTS.

PSF SHALL NOT BE LIABLE TO LICENSEE OR ANY OTHER USERS OF PYTHON 2.7.10 FOR ANY INCIDENTAL, SPECIAL, OR CONSEQUENTIAL DAMAGES OR LOSS AS A RESULT OF MODIFYING, DISTRIBUTING, OR
OTHERWISE USING PYTHON 2.7.10, OR ANY DERIVATIVE THEREOF, EVEN IF ADVISED OF THE POSSIBILITY THEREOF.

This License Agreement will automatically terminate upon a material breach of its terms and conditions.

Nothing in this License Agreement shall be deemed to create any relationship of agency, partnership, or joint venture between PSF and Licensee. This License Agreement does not grant permission to use PSF trademarks or trade name in a trademark sense to endorse or promote products or services of Licensee, or any third party.

By copying, installing or otherwise using Python 2.7.10, Licensee agrees to be bound by the terms and conditions of this License Agreement.

**python-dateutil**

https://github.com/dateutil/dateutil/blob/master/LICENSE

python-dateutil version 2.6.1

dateutil - Extensions to the standard Python datetime module.

Copyright (c) 2003-2011 - Gustavo Niemeyer <gustavo@niemeyer.net>

Copyright (c) 2012-2014 - Tomi Pieviläinen <tomi.pievilainen@iki.fi>

Copyright (c) 2014 - Yaron de Leeuw <me@jarondl.net>

All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- Neither the name of the copyright holder nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.
THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND
CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES,
INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR
CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT
NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;
LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR
OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

pytz

http://pytz.sourceforge.net/

pytz version 2017.3

Copyright (c) 2003-2009 Stuart Bishop <stuart@stuartbishop.net>

Permission is hereby granted, free of charge, to any person obtaining a copy of
this software and associated documentation files (the "Software"), to deal in the
Software without restriction, including without limitation the rights to use, copy,
modify, merge, publish, distribute, sublicense, and/or sell copies of the Software,
and to permit persons to whom the Software is furnished to do so, subject to the
following conditions:

The above copyright notice and this permission notice shall be included in all
copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY
KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE
WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR
PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS
OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR
OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR
OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
PyYAML

http://svn.pyyaml.org/pyyaml/trunk/LICENSE

PyYAML version 3.11

Copyright (c) 2006 Kirill Simonov

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

react-style-prototype

https://github.com/brigand/react-style-proptype

react-style-prototype version 3.2.1

MIT License

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:
The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NON INFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

Requests

http://docs.python-requests.org/en/master/user/intro/

Requests version 2.9.0

Copyright 2014 Kenneth Reitz

Licensed under the Apache License, Version 2.0 (the "License"); you may not use this file except in compliance with the License. You may obtain a copy of the License at

http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.

scikit-learn

https://github.com/scikit-learn/scikit-learn

scikit-learn version 0.19.0

[Scikit-learn: Machine Learning in Python]

All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:
1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

3. Neither the name of the copyright holder nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

SciPy

http://www.scipy.org/scipylib/license.html

SciPy version 0.19.1

Copyright © 2001, 2002 Enthought, Inc. All rights reserved.

Copyright © 2003-2013 SciPy Developers. All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation.
and/or other materials provided with the distribution.

- Neither the name of Enthought nor the names of the SciPy Developers may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE REGENTS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

six

https://bitbucket.org/gutworth/six/src/3deee854df8a5f1cc04dd721c18dee2128584f8c/LICENSE?fileviewer=file-view-default

six version 1.11.0

Copyright (c) 2010-2015 Benjamin Peterson

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR
OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

spin.js

http://spin.js.org

spin.js version 2.3.2

Copyright (c) 2011-2015 Felix Gnass [fgnass at gmail dot com]

The MIT License

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

statsmodels

https://github.com/statsmodels/statsmodels/blob/master/LICENSE.txt

statsmodels version 0.8.0

Copyright (C) 2006, Jonathan E. Taylor All rights reserved.

Copyright (c) 2006-2008 Scipy Developers. All rights reserved.
Copyright (c) 2009-2012 Statsmodels Developers. All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

a. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
b. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
c. Neither the name of Statsmodels nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL STATSMODELS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

wheel

https://bitbucket.org/pypa/wheel/src/4da780b849affb98a0defb21ad1f30a94d6f57/LICENSE.txt?fileviewer=file-view-default

wheel version 0.26

copyright (c) 2012-2014 Daniel Holth <dholth@fastmail.fm> and contributors.

The MIT License

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the
following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

YAML

http://pyyaml.org/browser/libyaml/trunk/LICENSE

YAML version 0.1.6

Copyright (c) 2006 Kirill Simonov

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
zlib

http://zlib.net/zlib_license.html

zlib version 1.2.8

Copyright (C) 1995-2013 Jean-loup Gailly and Mark Adler

This software is provided 'as-is', without any express or implied warranty. In no event will the authors be held liable for any damages arising from the use of this software.

Permission is granted to anyone to use this software for any purpose, including commercial applications, and to alter it and redistribute it freely, subject to the following restrictions:

1. The origin of this software must not be misrepresented; you must not claim that you wrote the original software. If you use this software in a product, an acknowledgment in the product documentation would be appreciated but is not required.

2. Altered source versions must be plainly marked as such, and must not be misrepresented as being the original software.

3. This notice may not be removed or altered from any source distribution.

Jean-loup Gailly Mark Adler jloup@gzip.org madler@alumni.caltech.edu

Dataset credits

The Machine Learning Toolkit contains datasets that were provided by others. We want to thank and acknowledge the contributors for them, and provide the licenses for their use.

Disclaimer

This application may contain certain sample files and datasets, which are provided for your convenience only. Such files and datasets contain information and data compiled by third parties, and Splunk makes no representation or warranty that the data contained in such files and datasets are true, accurate, complete or sanitized. In using the datasets, you understand and agree that the
data contained therein are subject to error and cannot be relied upon to perform the task you intend. You understand and agree that your use of the data is at your sole risk. The datasets are made available on an "as is" and "as available" basis without any warranties of any kind, whether express or implied, including without limitation implied warranties of merchantability, fitness for a particular purpose, and non-infringement. In no event will Splunk assume any legal liability or responsibility for loss or damages arising from the sample datasets.

**App logons count**

**Dataset:** applogonscount.txt

**Used in example:**

- Forecast App Logons with Special Days

**License terms:** Free to use, collected by Splunk.

**App statistics**

**Dataset:** app_usage.csv

**Used in examples:**

- Predict VPN Usage
- Cluster Behavior by App Usage

**License terms:** Free to use, collected by Splunk.

**Authorization log**

**Dataset:** authorization.csv

**Used in example:**

- Cluster failed ssh login attempts.

**License terms:** Free to use with citation. Security Repo (http://www.secrepo.com/) by Mike Sconzo is licensed under a Creative Commons Attribution 4.0 International License
Bitcoin transactions

Dataset: bitcoin_transactions.csv

Used in example:

• Detect Outliers in Bitcoin Transactions

License terms: Free to use with citation: http://compbio.cs.uic.edu/data/bitcoin/

Bluetooth devices

Dataset: bluetooth.csv

Used in example:

• Forecast the Number of Bluetooth Devices

License terms: CRAWDAD Data License

Dear Licensee:

Thank you for your interest in obtaining and using data from the CRAWDAD archive, hereinafter referred to as "Data". CRAWDAD is the Community Resource for Archiving Wireless Data At Dartmouth, and is operated by Dartmouth College under a grant from the National Science Foundation. Data Licensing Information:

Dartmouth College hereby grants a nonexclusive, nontransferable license to use the Data for commercial, educational, and research purposes only. The Data shall not be redistributed without the express written prior approval of Dartmouth College.

Licensee agrees to respect the privacy of those human subjects whose wireless-network activity is captured by the Data. Do not attempt to reverse the anonymization process to identify specific MAC addresses, IP address, telephone number, or other identifiers, or to identify their actual location. Use only the header information in packet traces; do not attempt to extract further information. (Header information specifies the type of information that is being transferred over the network, and specifically excludes the contents of the data, such as usernames, passwords, filenames, files, or URLs.)
Licensee agrees to acknowledge the source of the Data in any publications reporting on Licensee’s use of it. For example, “We gratefully acknowledge the use of wireless data from the CRAWDAD archive at Dartmouth College.”

Dartmouth expressly reserves the right to use the Data by its faculty, staff and researchers, for educational and research purposes. Dartmouth further reserves the right to provide Data Providers with statistical information regarding licensee's access to and use of the Provider’s Data and with the Licensee's name and address.

Dartmouth College provides the Data "AS IS," without any warranty or promise of technical support, and disclaims any liability of any kind for any damages whatsoever resulting from use of Data.

DARTMOUTH MAKES NO WARRANTIES, EXPRESS OR IMPLIED WITH RESPECT TO THE DATA, INCLUDING ANY IMPLIED WARRANTY OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE, WHICH ARE HEREBY EXPRESSLY DISCLAIMED.

Your acceptance and use of the Data binds you to the terms and conditions of this License as stated herein.

Trustees of Dartmouth College David F. Kotz, Ph.D. Professor of Computer Science 6211 Sudikoff Lab Hanover, NH 03755 USA

E-mail: kotz@cs.dartmouth.edu http://crawdad.org/nus/bluetooth/20070903/sql/

**Call center data**

Dataset: callcenter.csv

Used in example:

- Detect Cyclical Outliers in Call Center Data

License terms: Free to use, collected by Splunk.

**Churn**

Dataset: churn.csv

Used in example:
• Predict Telecom Customer Churn

License terms: Free to use, with citation request: http://www.sgi.com/tech/mlc/db/churn.all


Cyclical business process

Dataset: cyclical_business_process.csv

Used in examples:
  • Detect Cyclical Outliers in Logins
  • Predict Future Logins
  • Cluster Business Anomalies to Reduce Noise

License terms: Free to use, collected by Splunk.

Cyclical business process with anomalies

Dataset: cyclical_business_process_with_external_anomalies.csv

Used in example:
  • Predict External Anomalies

License terms: Free to use, collected by Splunk.

Diabetes

Dataset: diabetes.csv

Used in example:
  • Predict Incidence of Diabetes from Health Metrics

Database originally owned by National Institute of Diabetes and Digestive and Kidney Diseases; Database Donor: Sigillito Vincent, Research Center RMI Group Leader, The Johns Hopkins University. 9 May 1990.


**Diabetic data**

**Dataset:** diabetic.csv

**Used in example:**

- Detect Outliers in Diabetes Patient Records

**License terms:** Free to use with citation:


**Disk failures**

**Dataset:** disk_failures.csv

**Used in examples:**

- Detect Outliers in Disk Failure Events
- Predict the Failure of Hard Drives using SMART Metrics
- Cluster Hard Drives by SMART Metrics

**License terms:** Free to use with the following constraints:

A. We are encouraged to cite Backblaze as the source (not a mandatory requirement).
B. We accept that we are solely responsible for how we use the data.

C. We do not sell this data to anyone, it is free.


**Employee logins**

**Dataset:** logins.csv

**Used in examples:**

- Forecast the Number of Employee Logins
- Detect Outliers in Number of Logins (vs. Predicted Value)

**License terms:** Free to use, collected by Splunk.

**Exchange rate TWI**

**Dataset:** exchange.csv

**Used in example:**

- Forecast Exchange Rate TWI using ARIMA

**License terms:** DataMarket, Default Open License

Source:

This data release is licensed as follows:

- You may copy and redistribute the data.
- You may make derivative works from the data.
- You may use the data for commercial purposes.
- You may not sublicense the data when redistributing it.
- You may not redistribute the data under a different license.
- Source attribution on any use of this data: Must refer source.

**Firewall traffic**

**Dataset:** firewall_traffic.csv
Used in example:

- Predict the Presence of Malware from Firewall Traffic

License terms: Free to use, collected by Splunk.

Housing

Dataset: housing.csv

Used in examples:

- Predict Median House Value
- Cluster Neighborhoods by Properties

License terms: Free to use with citation:
https://archive.ics.uci.edu/ml/datasets/Housing


Internet traffic

Dataset: internet_traffic.csv

Used in example:

- Forecast Internet Traffic


This data release is licensed as follows:

- You may copy and redistribute the data.
- You may make derivative works from the data.
- You may use the data for commercial purposes.
- You may not sublicense the data when redistributing it.
- You may not redistribute the data under a different license.
- Source attribution on any use of this data: Must refer source.
Milk 2 dataset

**Dataset:** milk2.csv

**Used in examples:**

- Documentation for StateSpaceForecast algorithm
- Integration tests
- conftest.py

**License terms:** Derived from milk.csv, for which has Splunk legal approval.

Mortgage loans

**Dataset:** mortgage_loan_ny.csv

**Used in example:**

- Detect Outliers in Mortgage Contract Data

**License terms:** http://www.fhfa.gov/AboutUs/Policies/Pages/API.aspx

This product uses FHFA Data but is neither endorsed nor certified by FHFA.

PDF demo data

**Dataset:** pdfdemo.csv

**Used in example:**

- Detect Outliers on the fitted density functions onto measurements by two different cities in PDF Demo Data

**License terms:** Free to use, synthesized by Splunk.

Phone usage

**Dataset:** phone_usage.csv

**Used in example:**

- Detect Outliers in Mobile Phone Activity
License terms: CRAWDAD Data License Dear Licensee:

Thank you for your interest in obtaining and using data from the CRAWDAD archive, hereinafter referred to as "Data". CRAWDAD is the Community Resource for Archiving Wireless Data At Dartmouth, and is operated by Dartmouth College under a grant from the National Science Foundation. Data Licensing Information:

Dartmouth College hereby grants a nonexclusive, nontransferable license to use the Data for commercial, educational, and research purposes only. The Data shall not be redistributed without the express written prior approval of Dartmouth College.

Licensee agrees to respect the privacy of those human subjects whose wireless-network activity is captured by the Data. Do not attempt to reverse the anonymization process to identify specific MAC addresses, IP address, telephone number, or other identifiers, or to identify their actual location. Use only the header information in packet traces; do not attempt to extract further information. (Header information specifies the type of information that is being transferred over the network, and specifically excludes the contents of the data, such as usernames, passwords, filenames, files, or URLs.)

Licensee agrees to acknowledge the source of the Data in any publications reporting on Licensee's use of it. For example, "We gratefully acknowledge the use of wireless data from the CRAWDAD archive at Dartmouth College."

Dartmouth expressly reserves the right to use the Data by its faculty, staff and researchers, for educational and research purposes. Dartmouth further reserves the right to provide Data Providers with statistical information regarding licensee's access to and use of the Provider's Data and with the Licensee's name and address.

Dartmouth College provides the Data "AS IS," without any warranty or promise of technical support, and disclaims any liability of any kind for any damages whatsoever resulting from use of Data.

DARTMOUTH MAKES NO WARRANTIES, EXPRESS OR IMPLIED WITH RESPECT TO THE DATA, INCLUDING ANY IMPLIED WARRANTY OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE, WHICH ARE HEREBY EXPRESSLY DISCLAIMED.

Your acceptance and use of the Data binds you to the terms and conditions of this License as stated herein.
Power plant humidity

Dataset: power_plant.csv

Used in examples:

- Predict Power Plant Energy Output
- Detect Outliers in Power Plant Humidity
- Cluster Power Plant Operating Regimes


Server power

Dataset: server_power.csv

Used in example:

- Predict Server Power Consumption

License terms: Free to use, with citation request: https://www.usenix.org/legacy/event/hotpower08/tech/full_papers/rivoire/rivoire.pdf
Server response time

**Dataset**: hostperf.csv

**Used in example**:

- Detect Outliers in Server Response Time

**License terms**: Free to use, collected by Splunk.

Souvenir sales

**Dataset**: souvenir_sales.csv

**Used in example**:

- Forecast Monthly Sales for a Souvenir Shop

**License terms**: Default open license. This data release is licensed as follows:

- You may copy and redistribute the data.
- You may make derivative works from the data.
- You may use the data for commercial purposes.
- You may not sublicense the data when redistributing it.
- You may not redistribute the data under a different license.
- Source attribution on any use of this data: Must refer to source:


Special days

**Dataset**: specialdays.txt

**Used in example**:

- Forecast App Logons with Special Days

**License terms**: Free to use, collected by Splunk.
Supermarket purchases

Dataset: supermarket.csv

Used in example:

• Detect Outliers in Supermarket Purchases


Track day

Dataset: track_day.csv

Used in examples:

• Predict Vehicle Type from Onboard Metrics
  • Cluster Vehicles by Onboard Metrics

License terms: Free to use, collected by Splunk.