FN1540: You Only Learn Once (YOLO)

Ankit Bhagat
Forward Deployed Software Engineer | Splunk

Karthika Krishnan
Senior Forward Deployed Software Engineer | Splunk

Forward-Looking Statements

During the course of this presentation, we may make forward-looking statements regarding future events or plans of the company. We caution you that such statements reflect our current expectations and estimates based on factors currently known to us and that actual events or results may differ materially. The forward-looking statements made in the this presentation are being made as of the time and date of its live presentation. If reviewed after its live presentation, it may not contain current or accurate information. We do not assume any obligation to update any forward-looking statements made herein.

In addition, any information about our roadmap outlines our general product direction and is subject to change at any time without notice. It is for informational purposes only, and shall not be incorporated into any contract or other commitment. Splunk undertakes no obligation either to develop the features or functionalities described or to include any such feature or functionality in a future release.

Want to use ML with your Splunk'd Data?

Spoiler Alert: Use MLTK!

Agenda

What are we going to cover today?

1) What is MLTK?

2) Prerequisites and MLTK installation

3) Using built-in MLTK models

4) Creating your own custom models 😉

Let’s get the prerequisites out of the way!

A quick introduction to MLTK

splunk> .conf19

Machine Learning Toolkit Introduction

The Splunk Machine Learning Toolkit helps you apply a variety of machine-learning techniques and methods, such as classification (predicting a yay or nay), regression, anomaly detection, and outlier detection against your data.

Installing MLTK

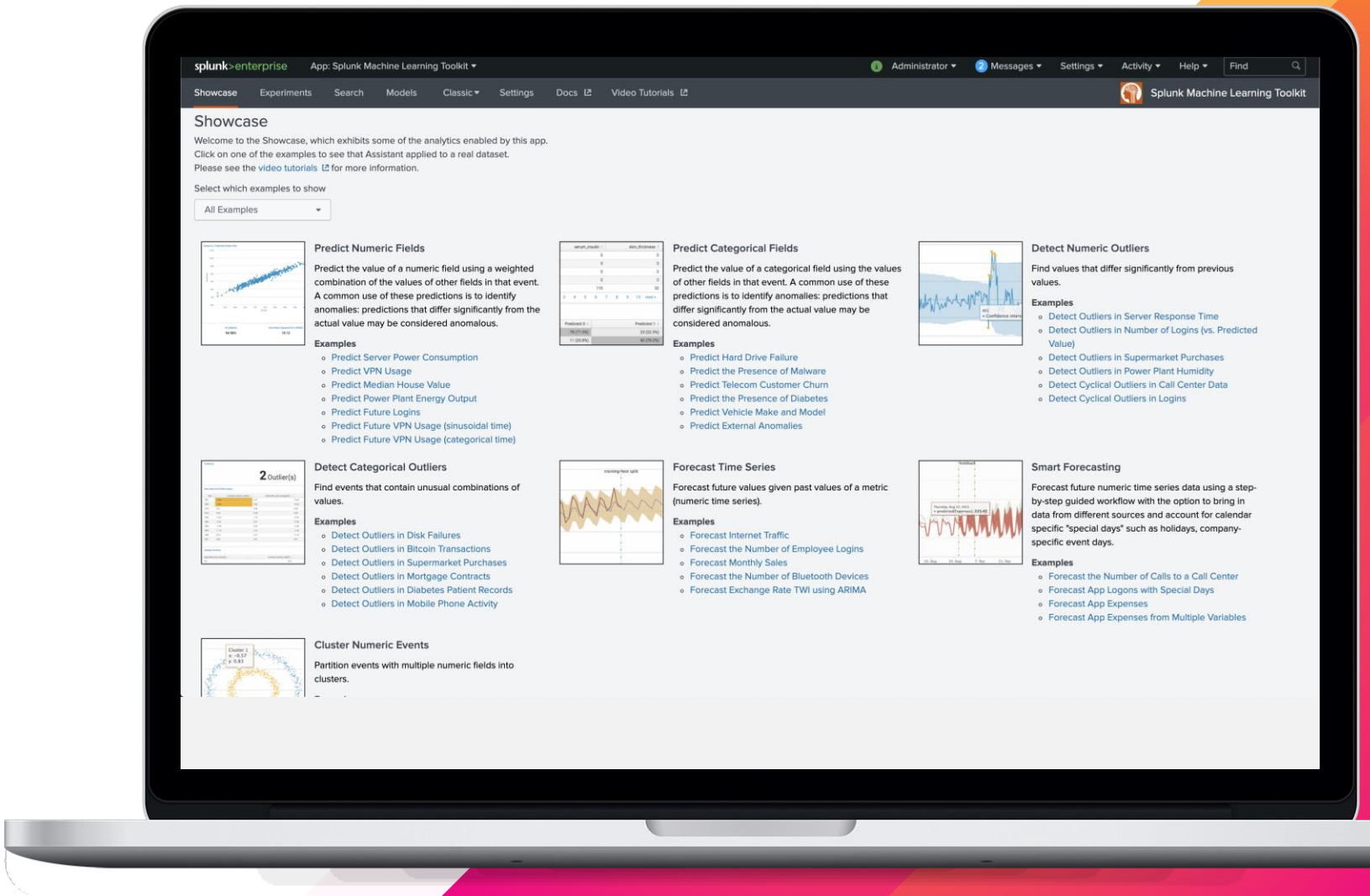
[0 ------>100%]

Install Python for Scientific Computing app from SplunkBase

Install Machine Learning Toolkit from SplunkBase

Restart the instance once applications are installed

Follow the above order for expected results



How to use MLTK?

Two common ways

Use the built-in algorithms from the MLTK library

Use other custom algorithms and add them to your MLTK library

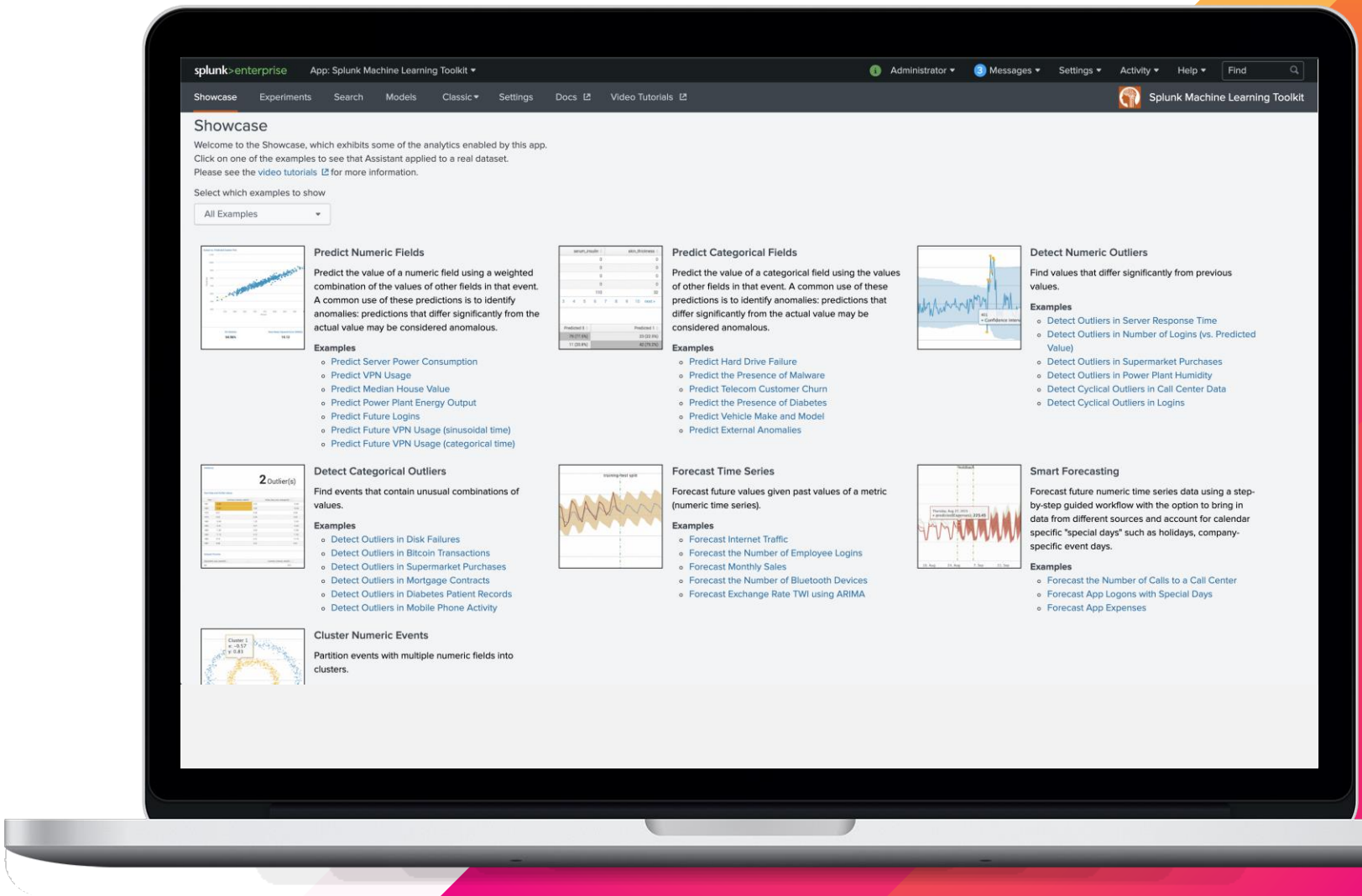
Using built-in MLTK models

The easier part 😉

splunk> .conf19

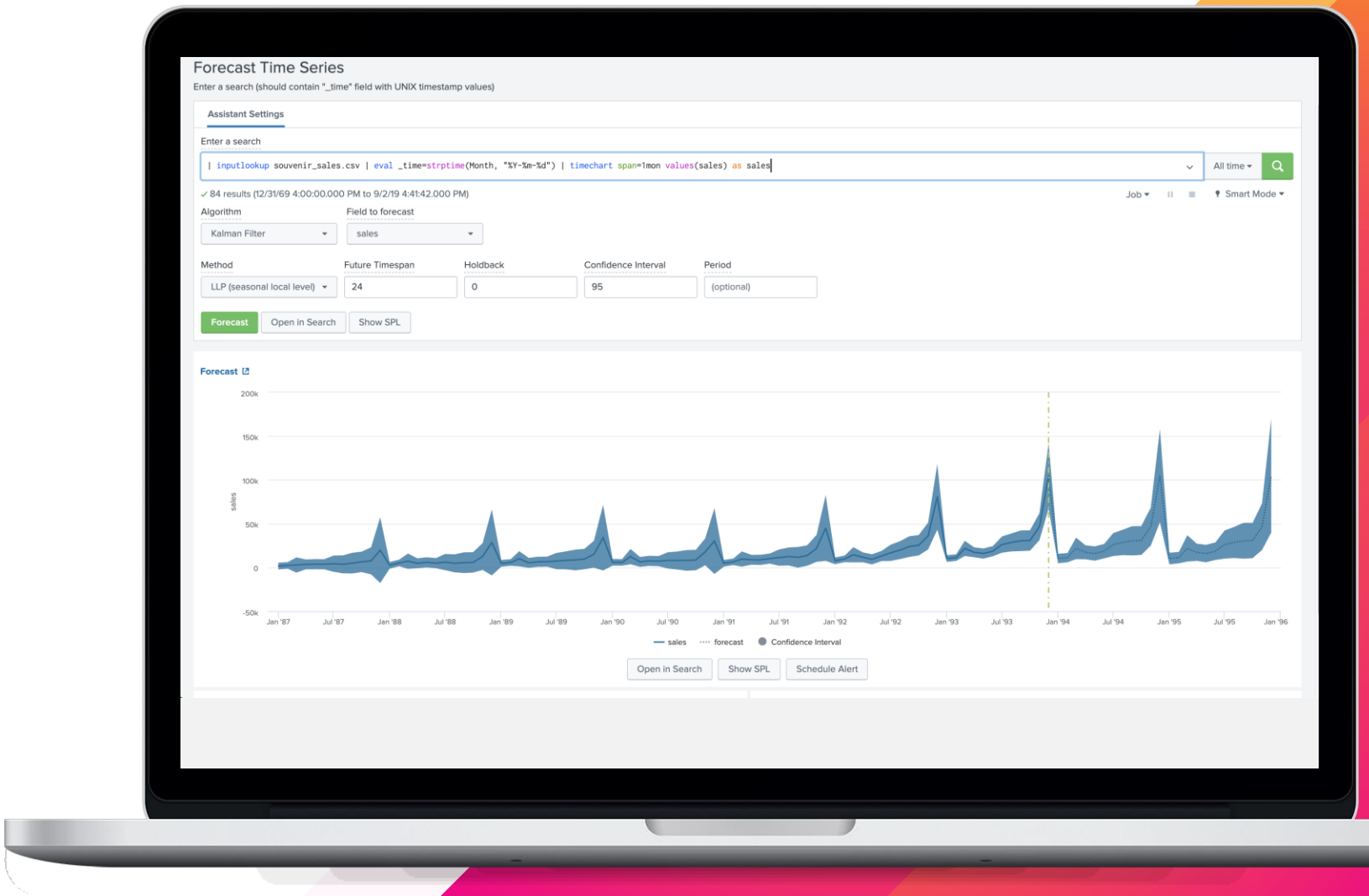
MLTK Homepage

Assistant and Examples



Experiments

Deploying ML models with a click of a few buttons



Welcome to the Jungle

Deploying custom ML algorithms from Scikit-Learn

splunk> .conf19

Adding a custom algorithm to MLTK

Steps to create an algorithm using your library.

1. Registering the algorithm in the MLTK application
2. Getting familiar with method conventions
3. Writing a Python class to implement the algorithm
4. Using codecs to save the model
5. Packaging the algorithm to add it to the GitHub repo

Register and Implement the algorithm.

The algorithms can inherit the methods from BaseAlgo, which defines the interface for ML-SPL algorithms.

Methods are entry point to a custom algorithm: __init__, fit, apply, register_codecs, partial_fit, summary

Registering an algorithm makes the algorithm visible to the Splunk Platform.

Register your algorithm using 2 different ways:

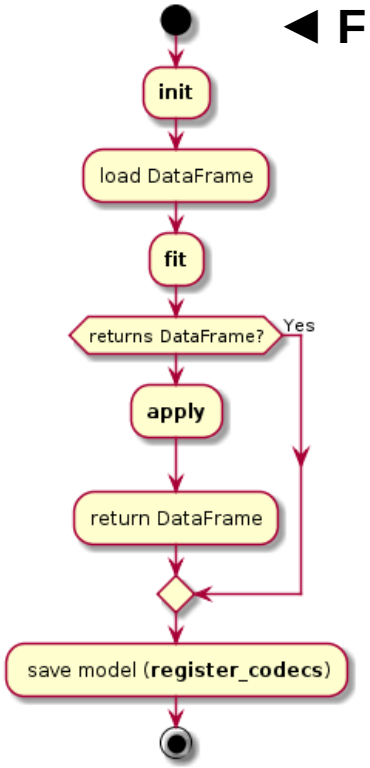
- Manual File Update
- REST API

Remember to follow best practices when implementing the algorithm:

- Assume invalid inputs
- Check for validity of a parameter passed
- Ensure required parameters are passed

Method conventions

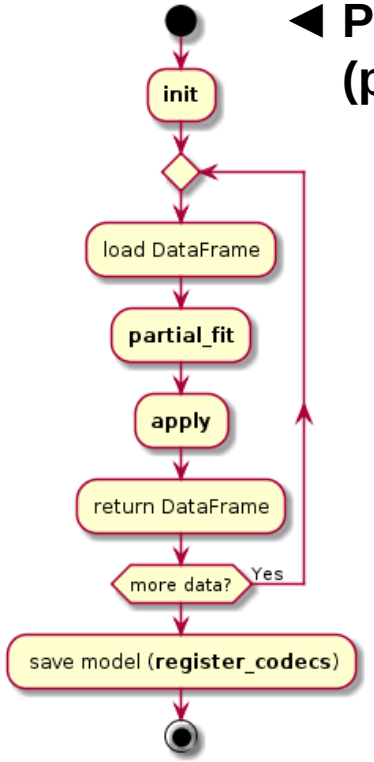
Flowcharts for working of different functions




```
def fit(self, df, options):
    # Make a copy of data, to not alter original dataframe
    X = df.copy()

    X, _, self.columns = df_util.prepare_features(
        X=X, variables=self.feature_variables, mlspl_limits=options.get('mlspl_limits')
    )
    self.estimator.fit(X.values)
```

Method conventions

Flowcharts for working of different functions

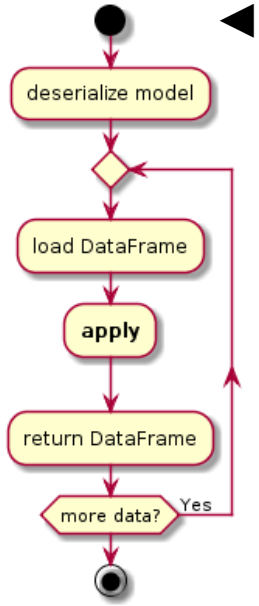


◀ **Partial Fit (partial fit = t)**

```
def partial_fit(self, df, options):
    # Make a copy of data, to not alter original dataframe
    X = df.copy()

    algo_util.assert_estimator_supports_partial_fit(self.estimator)
    X, _, columns = df_util.prepare_features(
        X=X, variables=self.feature_variables, mlspl_limits=options.get('mlspl_limits')
    )

    if getattr(self, 'columns', None):
        df_util.handle_new_categorical_values(X, None, options, self.columns)
        if X.empty:
            return
    else:
        self.columns = columns

    self.estimator.partial_fit(X)
```

Method conventions

Flowcharts for working of different functions

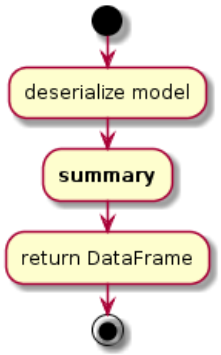


◀ **Apply Command**

```python
def apply(self, df, options):
    # Make a copy of data, to not alter original dataframe
    X = df.copy()

    X, nans, _ = df_util.prepare_features(
        X=X,
        variables=self.feature_variables,
        final_columns=self.columns,
        mlspl_limits=options.get('mlspl_limits'),
    )
    y_hat = self.estimator.predict(X.values)

    # Ensure the output has no floating points
    y_hat = y_hat.astype('str')

    # Assign output_name
    default_name = 'cluster'
    new_name = options.get('output_name', None)
    output_name = self.rename_output(default_names=default_name, new_names=new_name)

    # Create output dataframe
    output = df_util.create_output_dataframe(
        y_hat=y_hat, nans=nans, output_names=output_name
    )

    # Merge with original dataframe
    output = df_util.merge_predictions(df, output)
    return output
```

Method conventions

Flowcharts for working of different functions



◀ **Summary Command**

```
def summary(self, options):
    """The summary method defines how to summarize the model.

    The summary method is only necessary with a saved model. This method
    must return a pandas DataFrame.

    By default, the `options` dictionary only returns:

        {
            'model_name': 'some_custom_model_name',
            'mlspl_limits': { ... },
        }


    Parameters added to the search will be added to the `options`.

    An example:

        | summary my_custom_model key=value

    will return

        {
            'model_name': 'some_custom_model_name',
            'mlspl_limits': { ... },
            'params': {'key': 'value'},
        }

    as the `options`.
    """
    msg = 'The {} algorithm does not support summary.'
    msg = msg.format(self.__class__.__name__)
    raise MLSPLNotImplementedError(msg)
```

Saving Models & Packaging for GitHub!

Let's make it official.

MLTK uses codecs to serialize and deserialize algorithm models.

- Uses a string representation of __dict__ or __getstate__ and __setstate__.
- Implement register_codecs() method to save the model.
- There are a set of classes already loaded into the codec manager.
- For most of the other objects, SimpleObjectCodec can be used to represent any dictionary or list.
- If the object doesn't have a built-in codec support, a custom codec can be written.

Create a fork of the mltk-algo-contrib repository on GitHub.

Add your algorithm to the src folder and register the algorithm by adding it's stanza to algos.conf .

Don't forget to add test cases to the tests folder 😉

Demo

splunk> .conf19

Cheatsheet / Resources

ML-SPL Guide to add a custom algorithm

Command Guides and Flowcharts

Sample Example for a Correlation Matrix

MLTK Algorithms Contribution Repository

MLTK Algorithms Application on Splunkbase

.conf19

splunk>

Thank You!

Go to the .conf19 mobile app to

RATE THIS SESSION