Advanced ML Using The Extensible ML-SPL API

Alexander Johnson  |  Software Engineer
Zidong Yang  |  Software Engineer

September 2017  |  Washington, DC
Forward-Looking Statements

During the course of this presentation, we may make forward-looking statements regarding future events or the expected performance 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 could differ materially. For important factors that may cause actual results to differ from those contained in our forward-looking statements, please review our filings with the SEC.

The forward-looking statements made in this presentation are being made as of the time and date of its live presentation. If reviewed after its live presentation, this presentation may not contain current or accurate information. We do not assume any obligation to update any forward looking statements we may make. 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 functionality described or to include any such feature or functionality in a future release.

Splunk, Splunk>, Listen to Your Data, The Engine for Machine Data, Splunk Cloud, Splunk Light and SPL are trademarks and registered trademarks of Splunk Inc. in the United States and other countries. All other brand names, product names, or trademarks belong to their respective owners. © 2017 Splunk Inc. All rights reserved.
Who Are We?

Xander Johnson
- Splunker for 3 years
- Was Technical Training Instructor
- Software Engineer on ML Team
- BA in Linguistics @ USCB
- Cycling fanatic

Zidong Yang
- Splunker for 2 years
- Software Engineer on ML Team
- PhD in Computational Nanoscience @ George Washington University
Outline

- Overview of ML-SPL
  - What & Why
  - Commands & Algorithms

- ML-SPL Extensibility API
  - Motivation
  - Background
  - Examples
    - Hello World
    - Adaptive Boosting Classifiers!
ML-SPL Overview

Fit apply you some coefficients for great good!
Machine Learning Is Not Magic

... it’s a process.
**Data preparation** accounts for about 80% of the work of data scientists.
Splunk For Data Preparation

- Collect Data
- Clean/Transform
- Explore/Visualize
- Publish/Deploy
- Evaluate

Alerts, Dashboards, Reports

Add-ons from Splunkbase, etc.

props.conf, transforms.conf, Datamodels

ML Toolkit

Pivot, Table UI, SPL
ML-SPL: What Is It?

A suite of SPL search commands specifically for Machine Learning:
- Fit
- Apply
- Summary
- Listmodels
- Deletemodel
- Sample

Implemented using modules from the Python for Scientific Computing Add-on for Splunk:
- scikit-learn, numpy, pandas, statsmodels, scipy
ML-SPL Commands: A “Grammar” For ML

- Fit (i.e. train) a model from search results
  
  ... | fit \texttt{<ALGORITHM>} \texttt{<TARGET>} from \texttt{<VARIABLES ...>} \texttt{<PARAMETERS>} into \texttt{<MODEL>}

- Apply a model to obtain predictions from (new) search results
  
  ... | apply \texttt{<MODEL>}

- Inspect the model inferred by \texttt{<ALGORITHM>} (e.g. display coefficients)
  
  | summary \texttt{<MODEL>}

```
ML-SPL Commands: fit

... | fit <ALGORITHM> <TARGET> from <VARIABLES> ...

|PARAMETERS> into <MODEL>

Examples:

... | fit LinearRegression
    system_temp from cpu_load fan_rpm
    into temp_model

... | fit KMeans k=10
    downloads purchases posts days_active visits_per_day
    into user_behavior_clusters

... | fit LinearRegression
    petal_length from species
Toy Example
Titanic Survival Prediction

In [1]: import pandas as pd
In [2]: from sklearn.linear_model import LogisticRegression
In [3]: data = pd.read_csv("~/data/titanic.csv")
In [4]: target = data.Survived.values
In [5]: inputs = data[['Pclass', 'Sex', 'Age', 'Fare']].values
In [6]: model = LogisticRegression()
In [7]: model.fit(inputs, target)
Toy Example
Titantic Survival Prediction

ValueError: could not convert string to float: male
Toy Example

Titanic Survival Prediction

```python
In [8]: inputs = pd.get_dummies(data[['Pclass', 'Sex', 'Age', 'Fare']]).values

In [9]: model.fit(inputs, target)
```
Toy Example
Titanic Survival Prediction

Library/Python/2.7/site-packages/sklearn/linear_model/logistic.pyc in fit(self, X, y, sample_weight)
   271     return LinearPerceptron.fit(self, X, y, sample_weight)
   272
-> 273     return super(LogisticRegression, self).fit(X, y, sample_weight)
   274
   275 def fit(self, X, y, sample_weight=..., *args, **kwargs):

Library/Python/2.7/site-packages/sklearn/utils/validation.pyc in check_array(array, accept_sparse, dtype, order, copy, force_all_finite, ensure_2d, allow_nd, ensure_min_samples, ensure_min_features,warn_on_dtype, estimator)
   508     X = check_array(X, accept_sparse, dtype, order, copy, force_all_finite,
   509             ensure_2d, allow_nd, ensure_min_samples,
-> 510             ensure_min_features, warn_on_dtype, estimator)
   511     if multi_output:
   512         y = check_array(y, 'csr', force_all_finite=True, ensure_2d=False,

Library/Python/2.7/site-packages/sklearn/utils/validation.pyc in check_array(array, accept_sparse, dtype, order, copy, force_all_finite, ensure_2d, allow_nd, ensure_min_samples, ensure_min_features, warn_on_dtype, estimator)
   596     array = np.array(array, order=order, copy=copy, ndmin=1)
   597     if estimator is None:
-> 598         _assert_all_finite(array)
   599       else:
   600         _assert_all_finite(array, estimator_name)

Library/Python/2.7/site-packages/sklearn/utils/validation.pyc in _assert_all_finite(self, array)
   376             raise ValueError("Input contains NaN, infinity" +
   377                 " or a value too large for \%r." % X.dtype)
-> 378         " or a value too large for \%r." % X.dtype)
   379     _assert_all_finite(array.shape[0])
   380
ValueError: Input contains NaN, infinity or a value too large for dtype('float64').
Toy Example
Titanic Survival Prediction

```python
In [8]: inputs = pd.get_dummies(data[['Pclass', 'Sex', 'Age', 'Fare']]).values

In [10]: inputs = pd.get_dummies(data.dropna()[['Pclass', 'Sex', 'Age', 'Fare']]).values

In [11]: target = data.dropna().Survived.values

In [12]: model.fit(inputs, target)
Out[12]:
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, max_iter=100, multi_class='ovr', n_jobs=1,
                   penalty='l2', random_state=None, solver='liblinear', tol=0.0001,
                   verbose=0, warm_start=False)
```
Finally we have a machine learning model!

How do we…

• Collect and utilize raw incoming data
• Save, distribute, and control access to the model
• Schedule re-fitting of model
• Publish reports of predictions
• Alert on predictions

Operationalize?
Still must deploy the model!
Toy Example
Titanic Survival Prediction

```
| inputlookup   titanic.csv |
| fit LogisticRegression Survived from Pclass Sex Age Fare into model |
```
Beyond Simply Fitting Models

▶ Anticipates your pain points
  • Categorical encoding
  • Missing data
  • Sampling
  • Saving
▶ Chooses the best option
▶ Integrates with data in Splunk
  • Cleaning data
  • Creating features
We can use Splunk Enterprise to…

- Collect and utilize raw incoming data (forwarders, inputs.conf)
- Save, distribute, and control access to the model (knowledge objects, search bundle)
- Schedule re-fitting of model (scheduled searches)
- Handle unknown fields (wildcards)
- Publish reports of predictions (dashboards)
- Alert on predictions (alert actions)
30 Packaged algorithms come with the MLTK

- Regressors – predicting numeric output
- Classifiers – predicting categorical output
- Clusterers – grouping like with like
- Preprocessing
- Time series analysis – e.g. ARIMA, ACF, PACF
- Feature extraction – e.g. PCA, TFIDF
Python For Scientific Computing (PSC)
Free add-on available on Splunkbase

- Required dependency of the MLTK
- Provides needed libraries for ML
- Miniconda-based
- Most notable packages:
  - scikit-learn
  - pandas
  - NumPy
  - SciPy
  - StatsModels
Why Custom Algorithms?

What happens when the packaged algorithms aren’t the right ones?

• Fulfilling customer requests
• Operationalizing existing analyses or models
• Novel or proprietary algorithms
• Changing default behavior
  • Handling missing values
  • Arbitrary transformations
ML-SPL Extensibility API

Mixins, Methods, and Machine Learning
The ML-SPL Extensibility API allows one to add custom algorithms that can be used with the MLTK’s search commands.

ML-SPL API: Similar to…
- Python SDK for custom commands API
- Custom Visualization API (a.k.a. “modviz”)
- scikit-learn estimator API

Can be used in separate standalone apps too!
- Still must have MLTK & PSC installed
Directory Structure: MLTK

$SPLUNK_HOME/etc/apps/Splunk_ML_Toolkit

- bin
  - algos
    - LogisticRegression.py
    - ...
    - LinearRegression.py
- default
  - algos.conf
Directory Structure: MLTK
$SPLUNK_HOME/etc/apps/Splunk_ML_Toolkit

- bin
  - algos
    - LogisticRegression.py
    - HelloWorld.py ➔ algorithm source
      - LinearRegression.py
  - local
    - algos.conf ➔ register in algos.conf
    - default
      - algos.conf
Directory Structure: Custom App

$SPLUNK_HOME/etc/apps/CustomApp

```
  ├── bin
  │    └── algos
  │        └── HelloWorld.py  ← algorithm source
  └── default
      └── algos.conf  ← register in algos.conf
```
Used to add additional algorithms

Simplest .conf you’ve ever seen

- Each algorithm is only a stanza header

Allows you to package custom algorithms in custom apps, just like

- Custom commands
- Custom visualizations
- Custom alert actions
from base import BaseAlgo

class CustomAlgo(BaseAlgo):
    def __init__(self, options):
        # Option checking & initializations here
        pass

    def fit(self, df, options):
        # Fit an estimator to df, a pandas DataFrame of the search results
        pass

    def apply(self, df, options):
        # Apply a saved model
        return df

    @staticmethod
    def register_codecs():
        # Add codecs to the codec manager
        pass
from base import BaseAlgo

class HelloWorld(BaseAlgo):
    def __init__(self, options):
        pass

    def fit(self, df, options):
        df['message'] = "Hello World!"
        return df
<table>
<thead>
<tr>
<th>source</th>
<th>count</th>
<th>message</th>
</tr>
</thead>
<tbody>
<tr>
<td>/opt/splunk/var/log/splunk/conf.log</td>
<td>1</td>
<td>Hello World!</td>
</tr>
<tr>
<td>/opt/splunk/var/log/splunk/license_usage.log</td>
<td>7</td>
<td>Hello World!</td>
</tr>
<tr>
<td>/opt/splunk/var/log/splunk/metrics.log</td>
<td>4512</td>
<td>Hello World!</td>
</tr>
<tr>
<td>/opt/splunk/var/log/splunk/mlspl_watchdog.log</td>
<td>12</td>
<td>Hello World!</td>
</tr>
<tr>
<td>/opt/splunk/var/log/splunk/mongod.log</td>
<td>74</td>
<td>Hello World!</td>
</tr>
</tbody>
</table>
Fit AdaBoostClassifier
Fitting an ensemble classifier

from sklearn.ensemble import AdaBoostClassifier as _AdaBoostClassifier
from base import ClassifierMixin, BaseAlgo
from codec import codecs_manager
from util.param_util import convert_params

class AdaBoostClassifier(ClassifierMixin, BaseAlgo):
    def __init__(self, options):
        self.handle_options(options)

        params = options.get('params', {})
        converted_params = convert_params(params, ints=['n_estimators'],
                                           floats=['learning_rate'])

        self.estimator = _AdaBoostClassifier(**converted_params)
Fit AdaBoostClassifier
Fitting an ensemble classifier

@staticmethod
def register_codecs():
    from codec.codecs import SimpleObjectCodec, TreeCodec
    codecs_manager.add_codec('algos.AdaBoostClassifier',
        'AdaBoostClassifier', SimpleObjectCodec)
    codecs_manager.add_codec('sklearn.ensemble.weight_boosting',
        'AdaBoostClassifier', SimpleObjectCodec)
    codecs_manager.add_codec('sklearn.tree.tree',
        'DecisionTreeClassifier', SimpleObjectCodec)
    codecs_manager.add_codec('sklearn.tree._tree',
        'Tree', TreeCodec)
```
| inputlookup iris.csv 
| fit AdaBoostClassifier species from * into clf n_estimators=100 learning_rate=0.9 
| apply clf as predictions 
```

<table>
<thead>
<tr>
<th>petal_length</th>
<th>petal_width</th>
<th>predicted(species)</th>
<th>predictions</th>
<th>sepal_length</th>
<th>sepal_width</th>
<th>species</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.4</td>
<td>0.2</td>
<td>setosa</td>
<td>setosa</td>
<td>5.1</td>
<td>3.5</td>
<td>setosa</td>
</tr>
<tr>
<td>1.4</td>
<td>0.2</td>
<td>setosa</td>
<td>setosa</td>
<td>4.9</td>
<td>3.0</td>
<td>setosa</td>
</tr>
<tr>
<td>1.3</td>
<td>0.2</td>
<td>setosa</td>
<td>setosa</td>
<td>4.7</td>
<td>3.2</td>
<td>setosa</td>
</tr>
<tr>
<td>1.5</td>
<td>0.2</td>
<td>setosa</td>
<td>setosa</td>
<td>4.6</td>
<td>3.1</td>
<td>setosa</td>
</tr>
<tr>
<td>1.4</td>
<td>0.2</td>
<td>setosa</td>
<td>setosa</td>
<td>5.0</td>
<td>3.6</td>
<td>setosa</td>
</tr>
<tr>
<td>1.7</td>
<td>0.4</td>
<td>setosa</td>
<td>setosa</td>
<td>5.4</td>
<td>3.9</td>
<td>setosa</td>
</tr>
<tr>
<td>1.4</td>
<td>0.3</td>
<td>setosa</td>
<td>setosa</td>
<td>4.6</td>
<td>3.4</td>
<td>setosa</td>
</tr>
<tr>
<td>1.5</td>
<td>0.2</td>
<td>setosa</td>
<td>setosa</td>
<td>5.0</td>
<td>3.4</td>
<td>setosa</td>
</tr>
<tr>
<td>1.4</td>
<td>0.2</td>
<td>setosa</td>
<td>setosa</td>
<td>4.4</td>
<td>2.9</td>
<td>setosa</td>
</tr>
</tbody>
</table>
Using Built-In Utilities
Mixins are helper classes in Splunk_ML_Toolkit/bin/base.py

► MLTK Provides Mixin classes for common ML problems:
  • RegressorMixin – continuous target
  • ClassifierMixin – categorical target
  • TransformerMixin – arbitrary transformation (no target)
  • ClustererMixin – unknown target (unsupervised learning)

► Utility methods
  • df_util.prepare_features
  • df_util.create_output_dataframe

► Minimizes boilerplate
fit: How It Works

1. Discard fields that are null for all search results
2. Discard non-numeric fields with >100 distinct values
3. Discard search results with any null fields
4. Convert non-numeric fields to binary indicator variables (i.e. “dummy coding”)
5. Convert to a numeric matrix and hand over to <ALGORITHM>
6. Compute predictions for all search results
7. Save the learned model
fit: How It Works

1. Discard fields that are null for all search results.

<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>
<tr>
<td>ok</td>
<td>43</td>
</tr>
<tr>
<td></td>
<td>2</td>
</tr>
</tbody>
</table>
**fit: How It Works**

... | fit LogisticRegression field_A from field_*

2. Discard non-numeric fields with >100 distinct values.

<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>
<tr>
<td>ok</td>
<td>43</td>
</tr>
<tr>
<td></td>
<td>2</td>
</tr>
</tbody>
</table>
## fit: How It Works

... | fit LogisticRegression field_A from field_*

3. Discard search results with any null fields.

### Target | Explanatory Variables

<table>
<thead>
<tr>
<th>field_A</th>
<th>field_B</th>
<th>field_D</th>
<th>Color</th>
</tr>
</thead>
<tbody>
<tr>
<td>ok</td>
<td>41</td>
<td>red</td>
<td>ok</td>
</tr>
<tr>
<td>ok</td>
<td>32</td>
<td>green</td>
<td>ok</td>
</tr>
<tr>
<td>FRAUD</td>
<td>1</td>
<td>blue</td>
<td>ok</td>
</tr>
<tr>
<td>ok</td>
<td>43</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>blue</td>
<td></td>
</tr>
</tbody>
</table>
fit: How It Works

... | fit LogisticRegression field_A from field_*

4. Convert non-numeric fields to binary indicator variables.

<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>
fit: How It Works

... | fit LogisticRegression field_A from field_*

5. Convert to a numeric matrix and hand over to ALGORITHM.

\[
y = [1, 1, 0] \quad X = [41, 1, 0, 0], \\
\quad \quad \quad [32, 0, 1, 0], \\
\quad \quad \quad [1, 0, 0, 1]
\]

e.g. for Logistic Regression:

\[
\hat{y} = \frac{1}{1 + e^{-(\theta^T x)}} \quad \text{Find } \theta \text{ using maximum likelihood estimation.}
\]

Model inference generally delegated to scikit-learn and statsmodels.
(e.g. sklearn.linear_model.LogisticRegression)
**fit: How It Works**

... | fit LogisticRegression field_A from field_*

6. Compute predictions for all search results.

<table>
<thead>
<tr>
<th>Target</th>
<th>Explanatory Variables...</th>
<th>Prediction</th>
</tr>
</thead>
<tbody>
<tr>
<td>field_A</td>
<td>field_B</td>
<td>field_C</td>
</tr>
<tr>
<td>ok</td>
<td>41</td>
<td>red</td>
</tr>
<tr>
<td>ok</td>
<td>32</td>
<td>green</td>
</tr>
<tr>
<td>FRAUD</td>
<td>1</td>
<td>blue</td>
</tr>
<tr>
<td>ok</td>
<td>43</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>blue</td>
</tr>
</tbody>
</table>

*Note:* The table illustrates the process of predicting fraud using Logistic Regression with explanatory variables.
fit: How It Works

... | fit LogisticRegression field_A from field_* into logreg_model

7. Save the learned model.

Serialize model settings, coefficients, etc. into a Splunk lookup table.
- Replicated amongst members of Search Head Cluster
- Automatically distributed to Indexers with search bundle
- Safe! No pickles
Writing Your Own!
Check the guide!

- We have ML-SPL API documentation
  http://docs.splunk.com/Documentation/MLApp/latest/API/Introduction

- Examples include
  - CorrelationMatrix – using parameters in your search
  - AgglomerativeClustering – using df_util methods to clean data, convert categorical, etc.
  - Support Vector Regressor – using Mixins
  - Savitzky-Golay Filter – arbitrary statistical transformations with NumPy and SciPy
ML-SPL uses sampling to control size of input

Also has a “watchdog” process configured

• Memory consumption
• Max time spent fitting

```
[default]
max_inputs = 100000
use_sampling = true
max_fit_time = 600
max_memory_usage_mb = 1000
handle_new_cat = default
max_model_size_mb = 15
streaming_apply = false

[SVM]
max_inputs = 10000
```
Thank You

Don't forget to rate this session in the .conf2017 mobile app