Converting Scikit-Learn to PMML

Villu Ruusmann
Openscoring OÜ

"Train once, deploy anywhere"

Scikit-Learn challenge

```
pipeline = Pipeline([...])
pipeline.fit(X_train, y_train)

y_test = pipeline.predict(X_test)
```

- Serializing and deserializing the fitted pipeline:
  - (C)Pickle
  - Joblib
- Creating X_{test} that is functionally equivalent to X_{train}:
  - ???

(J)PMML solution

```
Evaluator evaluator = ...;
List<InputField> argumentFields = evaluator.getInputFields();
List<ResultField> resultFields =
    Lists.union(evaluator.getTargetFields(), evaluator.getOutputFields());
Map<FieldName, ?> arguments = readRecord(argumentFields);
Map<FieldName, ?> result = evaluator.evaluate(arguments);
writeRecord(result, resultFields);
```

- The fitted pipeline (data pre-and post-processing, model) is represented using standardized PMML data structures
- Well-defined data entry and exit interfaces

Workflow

The PMML pipeline

A smart pipeline that collects supporting information about "passed through" features and label(s):

- Name
- Data type (eg. string, float, integer, boolean)
- Operational type (eg. continuous, categorical, ordinal)
- Domain of valid values
- Missing and invalid value treatments

Making a PMML pipeline (1/2)

```
from sklearn2pmml import PMMLPipeline
from sklearn2pmml import sklearn2pmml

pipeline = PMMLPipeline([...])
pipeline.fit(X, y)

sklearn2pmml(pipeline, "pipeline.pmml")
```

Making a PMML pipeline (2/2)

```
from sklearn2pmml import make_pmml_pipeline, sklearn2pmml

pipeline = Pipeline([...])
pipeline.fit(X, y)

pipeline = make_pmml_pipeline(
    pipeline,
    active_fields = ["x1", "x2", ...],
    target_fields = ["y"]
)

sklearn2pmml(pipeline, "pipeline.pmml")
```

Pipeline setup (1/3)

1. Feature and label definition

   Specific to (J)PMML
2. Feature engineering
3. Feature selection

   Scikit-Learn persists all features, (J)PMML persists "surviving" features
4. Estimator fitting
5. Decision engineering

   Specific to (J)PMML

Pipeline setup (2/3)

Workflow:

1. Column- and column set-oriented feature definition, engineering and selection
2. Table-oriented feature engineering and selection
3. Estimator fitting

Full support for pipeline nesting, branching.

Pipeline setup (3/3)

```python
from sklearn2pmml import PMMLPipeline
from sklearn2pmml.decoration import CategoricalDomain, ContinuousDomain
from sklearn_pandas import DataFrameMapper

pipeline = PMMLPipeline([
    ("stage1", DataFrameMapper([
        (["x1"], [CategoricalDomain(), ...]),
        (["x2", "x3"], [ContinuousDomain(), ...]),
    ])),
    ("stage2", SelectKBest(10)),
    ("stage3", LogisticRegression())
])
```

Feature definition

Continuous features (1/2)

Without missing values:

```
("Income", [ContinuousDomain(invalid_value_treatment = "return_invalid",
missing_value_treatment = "as_is", with_data = True,
with_statistics = True)])
```

With missing values (encoded as None/NaN):

```
("Income", [ContinuousDomain(), Imputer()])
```

With missing values (encoded using dummy values):

```
("Income", [ContinuousDomain(missing_values = -999),
Imputer(missing_values = -999)])
```

Continuous features (2/2)

```xml
<DataDictionary>
  <DataField name="Income" optype="continuous" dataType="double">
    <Interval closure="closedClosed" leftMargin="1598.95" rightMargin="481259.5"/>
  </DataField>
</DataDictionary>
<RegressionModel>
  <MiningSchema>
    <MiningField name="Income" invalidValueTreatment="returnInvalid"
      missingValueTreatment="asMean" missingValueReplacement="84807.39297450424"/>
  </MiningSchema>
  <ModelStats>
    <UnivariateStats field="Income">
      <Counts totalFreq="1899.0" missingFreq="487.0" invalidFreq="0.0"/>
      <NumericInfo minimum="1598.95" maximum="481259.5" mean="84807.39297450424" median="60596.35"
        standardDeviation="69696.87637064351" interQuartileRange="81611.1225"/>
    </UnivariateStats>
  </ModelStats>
</RegressionModel>
```

Categorical features (1/3)

Without missing values:

```
("Education", [CategoricalDomain(invalid_value_treatment =
"return_invalid", missing_value_treatment = "as_is", with_data = True,
with_statistics = True), LabelBinarizer()])
```

With missing values, missing value-aware estimator:

```
("Education", [CategoricalDomain(), PMMLLabelBinarizer()])
```

With missing values, missing value-*<u>un</u>*aware estimator:

```
("Education", [CategoricalDomain(), CategoricalImputer(),
LabelBinarizer()])
```

Categorical features (2/3)

```xml
<DataDictionary>
  <DataField name="Education" optype="categorical" dataType="string">
    <Value value="Associate"/>
    <Value value="Bachelor"/>
    <Value value="College"/>
    <Value value="Doctorate"/>
    <Value value="HSgrad"/>
    <Value value="Master"/>
    <Value value="Preschool"/>
    <Value value="Professional"/>
    <Value value="Vocational"/>
    <Value value="Yr10t12"/>
    <Value value="Yr1t4"/>
    <Value value="Yr5t9"/>
  </DataField>
</DataDictionary>
```

Categorical features (3/3)

```xml
<RegressionModel>
  <MiningSchema>
    <MiningField name="Education" invalidValueTreatment="asIs", x-invalidValueReplacement="HSgrad"
      missingValueTreatment="asMode" missingValueReplacement="HSgrad"/>
  </MiningSchema>
  <UnivariateStats field="Education">
    <Counts totalFreq="1899.0" missingFreq="497.0" invalidFreq="0.0"/>
    <DiscrStats>
      <Array type="string">Associate Bachelor College Doctorate HSgrad Master Preschool
        Professional Vocational Yr10t12 Yr1t4 Yr5t9</Array>
      <Array type="int">55 241 309 22 462 78 6 15 55 95 4 60</Array>
    </DiscrStats>
  </UnivariateStats>
</RegressionModel>
```

Feature engineering

Continuous features (1/2)

```
from sklearn.preprocessing import Binarizer, FunctionTransformer
from sklearn2pmml.preprocessing import ExpressionTransformer

features = FeatureUnion([
    ("identity", DataFrameMapper([
        (["Income", "Hours"], ContinuousDomain())
    ])),
    ("transformation", DataFrameMapper([
        (["Income"], FunctionTransformer(numpy.log10)),
        (["Hours"], Binarizer(threshold = 40)),
        (["Income", "Hours"], ExpressionTransformer("X[:,0]/(X[:,1]*52)"))
    ]))
])
```

Continuous features (2/2)

```xml
<TransformationDictionary>
  <DerivedField name="log10(Income)" optype="continuous" dataType="double">
    <Apply function="log10"><FieldRef field="Income"/></Apply>
  </DerivedField>
  <DerivedField name="binarizer(Hours)" optype="continuous" dataType="double">
    <Apply function="threshold"><FieldRef field="Hours"/><Constant>40</Constant></Apply>
  </DerivedField>
  <DerivedField name="eval(X[:,0]/(X[:,1]*52))" optype="continuous" dataType="double">
    <Apply function="/">
      <FieldRef field="Income"/>
      <Apply function="*">
        <FieldRef field="Hours"/>
        <Constant dataType="integer">52</Constant>
      </Apply>
    </Apply>
  </DerivedField>
</TransformationDictionary>
```

Categorical features

```
from sklearn preprocessing import LabelBinarizer, PolynomialFeatures

features = Pipeline([
    ("identity", DataFrameMapper([
        ("Education", [CategoricalDomain(), LabelBinarizer()]),
        ("Occupation", [CategoricalDomain(), LabelBinarizer()])
    ])),
    ("interaction", PolynomialFeatures())
])
```

Feature selection

Scikit-Learn challenge

```
class SelectPercentile(BaseTransformer, SelectorMixin):

    def _get_support_mask(self):
        scores = self.scores
        threshold = stats.scoreatpercentile(scores, 100 - self.percentile)
        return (scores > threshold)
```

Python methods don't have a persistent state:

```
Exception in thread "main": java.lang.IllegalArgumentException: The
selector object does not have a persistent '_get_support_mask' attribute
```

(J)PMML solution

"Hiding" a state*less* selector behind a state*ful* meta-selector:

```
from sklearn2pmml import SelectorProxy, PMMLPipeline

pipeline = PMMLPipeline([
    ("stage1", DataFrameMapper([
        (["x1"], [CategoricalDomain(), LabelBinarizer(),
            SelectorProxy(SelectFromModel(DecisionTreeClassifier()))])
    ])),
    ("stage2", SelectorProxy(SelectPercentile(percentile = 50)))
])
```

Estimator fitting

Hyper-parameter tuning (1/2)

```
from sklearn.model_selection import GridSearchCV
from sklearn2pmml import PMMLPipeline
from sklearn2pmml import sklearn2pmml

pipeline = PMMLPipeline([...])

tuner = GridSearchCV(pipeline, param_grid = {...})
tuner.fit(X, y)

# GridSearchCV.best_estimator_ is of type PMMLPipeline
sklearn2pmml(tuner.best_estimator_, "pipeline.pmml")
```

Hyper-parameter tuning (2/2)

```
from sklearn2pmml import make_pmml_pipeline, sklearn2pmml

pipeline = Pipeline([...])

tuner = GridSearchCV(pipeline, param_grid = {...})
tuner.fit(X, y)

# GridSearchCV.best_estimator_ is of type Pipeline
sklearn2pmml(make_pmml_pipeline(tuner.best_estimator_, active_fields = [...], target_fields = [...]), "pipeline.pmml")
```

Algorithm tuning

```
from sklearn2pmml import make_pmml_pipeline, sklearn2pmml
from tpot import TPOTClassifier

# See https://github.com/rhiever/tpot
tpot = TPOTClassifier()
tpot.fit(X, y)

sklearn2pmml(make_pmml_pipeline(tpot.fitted_pipeline_, active_fields =
[...], target_fields = [...]), "pipeline.pmml")
```

Model customization (1/2)

```
from sklearn2pmml import PMMLPipeline
from sklearn2pmml import sklearn2pmml

# Keep reference to the estimator
classifier = DecisionTreeClassifier()

pipeline = PMMLPipeline([..., ("classifier", classifier)])
pipeline.fit(X, y)

# Apply customization(s) to the fitted estimator
classifier.compact = True

sklearn2pmml(pipeline, "pipeline.pmml")
```

Model customization (2/2)

- `org.jpmml.sklearn.HasOptions`
  - `lightgbm.sklearn.HasLightGBMOptions`
    - `compact`
    - `num_iteration`
  - `sklearn.tree.HasTreeOptions`
    - `compact`
  - `xgboost.sklearn.HasXGBoostOptions`
    - `compact`
    - `ntree_limit`

Q&A

villu@openscoring.io

https://github.com/jpmml
https://github.com/openscoring
https://groups.google.com/forum/#!forum/jpmml

Software (Nov 2017)

- The Python side:
  - `sklearn 0.19(.0)`
  - `sklearn2pmml 0.26(.0)`
  - `sklearn_pandas 1.5(.0)`
  - `TPOT 0.9(.0)`
- The Java side:
  - `JPMML-SkLearn 1.4(.0)`
  - `JPMML-Evaluator 1.3(.10)`