On the representation and reuse of machine learning models

Villu Ruusmann
Openscoring OÜ

https://github.com/jpmml

Def: "Model"

Output = func(Input)

Def: "Representation"

The problem

"Train once, deploy anywhere"

A solution

Matching model representation (MR) with the task at hand:

1. Storing a generic and stable MR
2. Generating a wide variety of more specific and volatile MRs upon request

The Predictive Model Markup Language (PMML)

- XML dialect for marking up models and associated data transformations
- Version 1.0 in 1999, version 4.3 in 2016
- "Conventions over configuration"
- 17 top-level model types + ensembling

http://dmg.org/
http://dmg.org/pmml/pmml-v4-3.html
http://dmg.org/pmml/products.html

A continuum from black to white boxes

Introducing transparency in the form of rich, easy to use, well-documented APIs:

1. Unmarshalling and marshalling
2. Static analyses. Ex: schema querying
3. Dynamic analyses. Ex: scoring
4. Tracing and explaining individual predictions

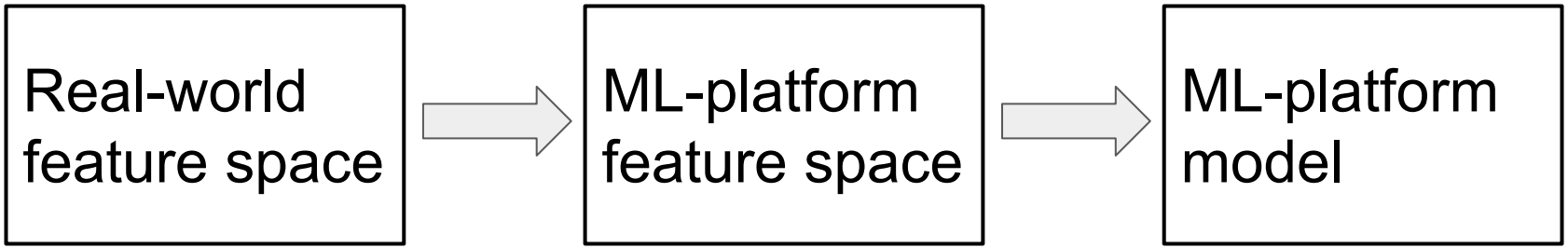
The Zen of Machine Learning

"Making the model requires large data and many cpus.
Using it does not"

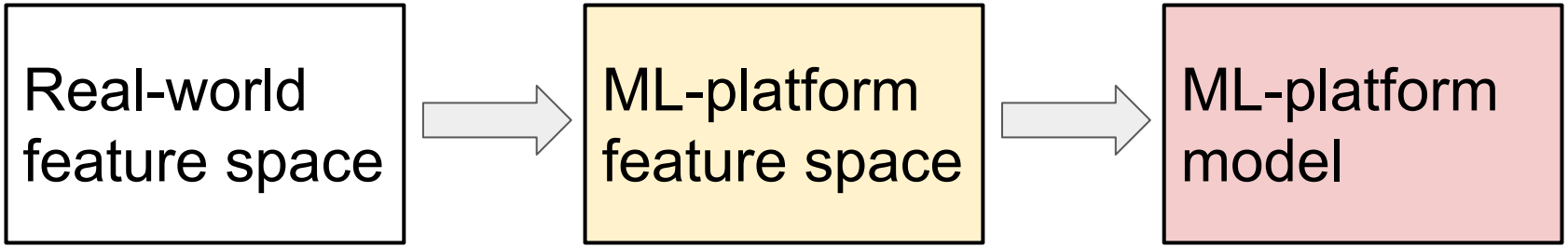
--darren

https://www.mail-archive.com/user@spark.apache.org/msg40636.html

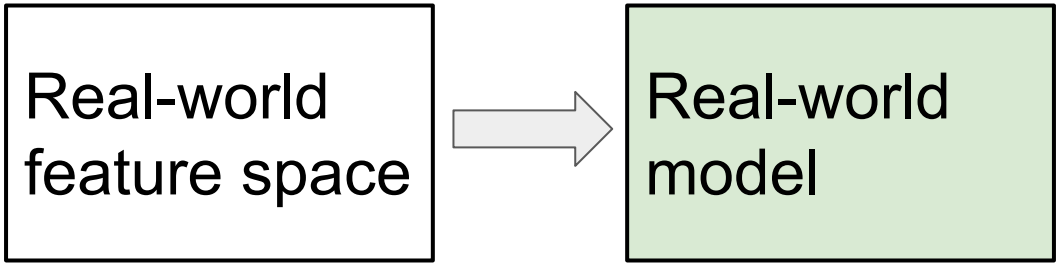
Model training workflow



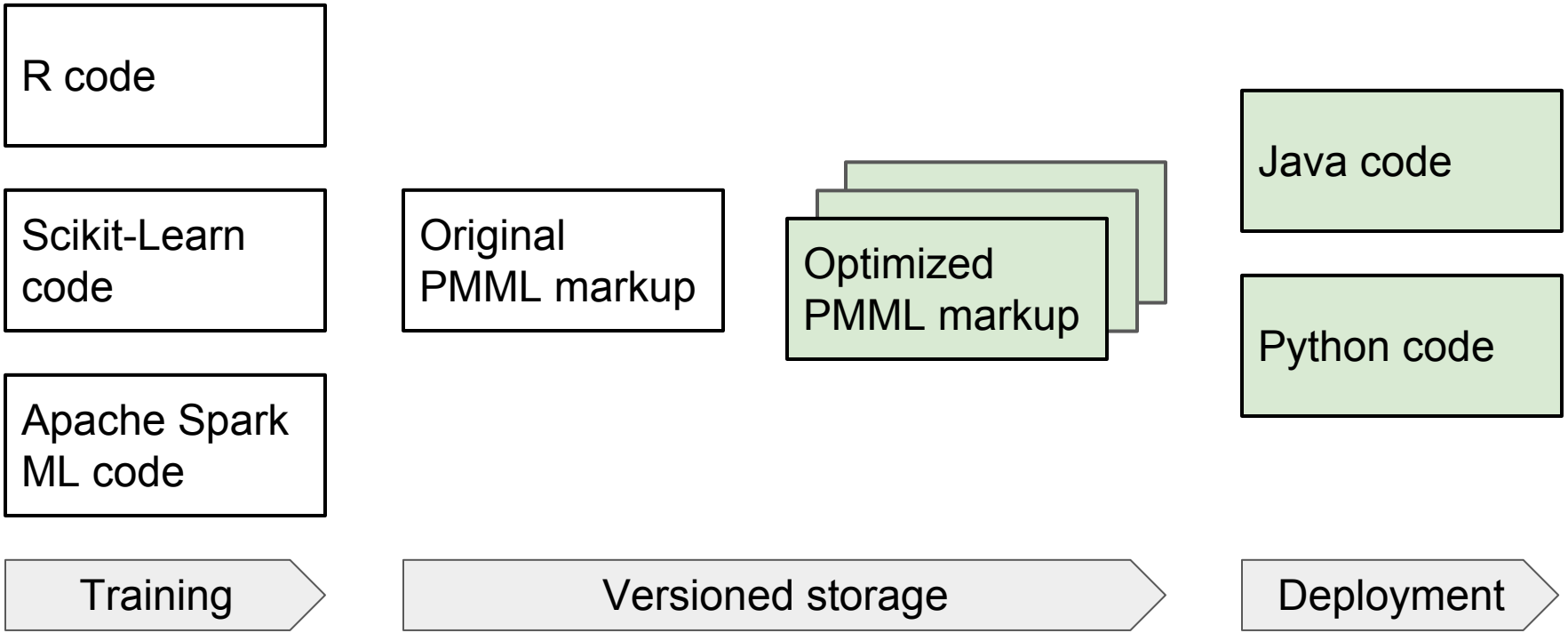
Model deployment workflow



vs.



Model resources



Comparison of model persistence options

	R	Scikit-Learn	Apache Spark ML
Model data structure stability	Fair to excellent	Fair	Poor
Native serialization data format	RDS (binary)	Pickle (binary)	SER (binary) and JSON (text)
Export to PMML	Few external	N/A	Built-in trait `PMMLWritable`
Import from PMML	Few external	N/A	
JPMML projects	JPMML-R and r2pmml	JPMML-SkLearn and sklearn2pmml	JPMML-SparkML (-Package)

PMML production: R

```
library("r2pmml")

auto <- read.csv("Auto.csv")
auto$origin <- as.factor(auto$origin)

auto.formula <- formula(mpg ~
      (.) ^ 2 + # simple features and their two way interactions
      I(displacement / cylinders) + I(log(weight))) # derived features

auto.lm <- lm(auto.formula, data = auto)
r2pmml(auto.lm, "auto_lm.pmml", dataset = auto)

auto.glm <- glm(auto.formula, data = auto, family = "gaussian")
r2pmml(auto.glm, "auto_glm.pmml", dataset = auto)
```

R quirks

- No pipeline concept. Some workflow standardization efforts by third parties. Ex: `caret` package
- Many (equally right-) ways of doing the same thing. Ex: "formula interface" vs. "matrix interface"
- High variance in the design and quality of packages. Ex: academia vs. industry
- Model objects may enclose the training data set

PMML production: Scikit-Learn

```
from sklearn2pmml import sklearn2pmml
from sklearn2pmml.decoration import CategoricalDomain, ContinuousDomain

audit_df = pandas.read_csv("Audit.csv")
audit_mapper = DataFrameMapper([
      (["Age", "Income", "Hours"], ContinuousDomain()),
      (["Employment", "Education", "Marital", "Occupation"], [CategoricalDomain(), LabelBinarizer()]),
      (["Gender", "Deductions"], [CategoricalDomain(), LabelEncoder()]),
      ("Adjusted", None)])
audit = audit_mapper.fit_transform(audit_df)

audit_classifier = DecisionTreeClassifier(min_samples_split = 10)
audit_classifier.fit(audit[:, 0:48], audit[:, 48].astype(int))

sklearn2pmml(audit_classifier, audit_mapper, "audit_tree.pmml")
```

Scikit-Learn quirks

- Completely schema-less at algorithm level. Ex: no identification of columns, no tracking of column groups
- Very limited, simple data structures. Mix of Python and C
- No built-in persistence mechanism. Serialization in generic `pickle` data format. Upon de-serialization, hope that class definitions haven't changed in the meantime.

PMML production: Apache Spark ML

```
// $ spark-shell --packages org.jpmml:jpmml-sparkml-package:1.0-SNAPSHOT ..
import org.jpmml.sparkml.ConverterUtil

val df = spark.read.option("header", "true").option("inferSchema", "true").csv("Wine.csv")

val formula = new RFormula().setFormula("quality ~ .")
val regressor = new DecisionTreeRegressor()
val pipeline = new Pipeline().setStages(Array(formula, regressor))
val pipelineModel = pipeline.fit(df)

val pmmlBytes = ConverterUtil.toPMMLByteArray(df.schema, pipelineModel)

Files.write(Paths.get("wine_tree.pmml"), pmmlBytes)
```

Apache Spark ML quirks

- Split schema. Static def via `Dataset#schema()`, dynamic def via Dataset column metadata
- Models make predictions in transformed output space
- High internal complexity, overhead. Ex: temporary Dataset columns for feature transformation
- Built-in PMML export capabilities leak the JPMML-Model library to application classpath

PMML consumption: Apache Spark ML

```
// $ spark-submit --packages org.jpmml:jpmml-spark:1.0-SNAPSHOT ..
import org.jpmml.spark.EvaluatorUtil;
import org.jpmml.spark.TransformerBuilder;

Evaluator evaluator = EvaluatorUtil.createEvaluator(new File("audit_tree.pmml"));

TransformerBuilder pmmlTransformerBuilder = new TransformerBuilder(evaluator)
      .withLabelCol("Adjusted") // String column
      .withProbabilityCol("Adjusted_probability", Arrays.asList("0", "1")) // Vector column
      .exploded(true);
Transformer pmmlTransformer = pmmlTransformerBuilder.build();

Dataset<Row> input = ...;
Dataset<Row> output = pmmlTransformer.transform(input);
```

Comparison of feature spaces

	R	Scikit-Learn	Apache Spark ML
Feature identification	Named	Positional	Pseudo-named
Feature data type	Any	Float, Double	Double
Feature operational type	Continuous, Categorical, Ordinal	Continuous	Continuous, pseudo-categorical
Dataset abstraction	List<Map<String,?>>	float[][] or double[][]	List<double[]>
Effect of transformations on dataset size	Low	Medium (sparse) to high (dense)	

Feature declaration

```
<DataField name="Age" dataType="float" optype="continuous">
      <Interval closure="closedClosed" leftMargin="17.0" rightMargin="83.0"/>
</DataField>
<DataField name="Gender" dataType="string" optype="categorical">
      <Value value="Male"/>
      <Value value="Female"/>
      <Value value="N/A" property="missing"/>
</DataField>
```

http://dmg.org/pmml/v4-3/DataDictionary.html

```
<MiningField name="Age" outliers="asExtremeValues" lowValue="18.0" highValue="75.0"/>
<MiningField name="Gender"/>
```

http://dmg.org/pmml/v4-3/MiningSchema.html

Feature statistics

```
<UnivariateStats field="Age">
      <NumericInfo mean="38.30279" standardDeviation="13.01375" median="3.70"/>
      <ContStats>
            <Interval closure="openClosed" leftMargin="17.0" rightMargin="23.6"/>
            <!-- Intervals 2 through 9 omitted for clarity -->
            <Interval closure="openClosed" leftMargin="76.4" rightMargin="83.0"/>
            <Array type="int">261 360 297 340 280 156 135 51 13 6</Array>
      </ContStats>
</UnivariateStats>
<UnivariateStats field="Gender">
      <Counts totalFreq="1899" missingFreq="0" invalidFreq="0"/>
      <DiscrStats>
            <Array type="string">Male Female</Array>
            <Array type="int">1307 592</Array>
      </DiscrStats>
</UnivariateStats>
```


http://dmg.org/pmml/v4-3/Statistics.html

Comparison of tree models

	R	Scikit-Learn	Apache Spark ML
Algorithms	No built-in, many external	Few built-in	Single built-in
Split type(s)	Binary or multi-way; simple and derived features	Binary; simple features	
Continuous features	Rel. op. (<, <=)	Rel. op. (<=)	
Categorical features	Set op. (%in%)	Pseudo-rel. op. (==)	Pseudo-set op.
Reuse	Hard	Easy to medium	

Tree model declaration

```
<TreeModel functionName="classification" splitCharacteristic="binarySplit">
      <Node id="1" recordCount="165">
            <True/>
            <Node id="2" score="1" recordCount="35">
                  <SimplePredicate field="Education" operator="equal" value="Master"/>
                  <ScoreDistribution value="1" recordCount="25"/>
                  <ScoreDistribution value="0" recordCount="10"/>
            </Node>
            <Node id="3" score="0" recordCount="130">
                  <SimplePredicate field="Education" operator="notEqual" value="Master"/>
                  <ScoreDistribution value="1" recordCount="20"/>
                  <ScoreDistribution value="0" recordCount="110"/>
            </Node>
      </Node>
</TreeModel>
```

http://dmg.org/pmml/v4-3/TreeModel.html

Optimization (1/3)

Replacing "deep" binary splits with "shallow" multi-splits:

```
<Node>
      SimplePredicate: Gender != "Male"
      <Node>
            SimplePredicate: Age <= 34.5
      </Node>
      <Node>
            SimplePredicate: Age > 34.5
      </Node>
</Node>
<Node>
      SimplePredicate: Gender == "Male"
</Node>
```

```
<Node>
      SimplePredicate: Gender == "Male"
</Node>
<Node>
      SimplePredicate: Age <= 34.5
</Node>
<Node>
      SimplePredicate: Age > 34.5
</Node>
```

Optimization (2/3)

Cutting the number of terminal nodes in half:

```
<TreeModel
      noTrueChildStrategy="returnNullPrediction"
>
      <Node>
            <True/>
            <Node score="4.333333333333333">
                  SimplePredicate: pH <= 2.93
            </Node>
            <Node score="5.483870967741935">
                  SimplePredicate: pH > 2.93
            </Node>
      </Node>
</TreeModel>
```

```
<TreeModel
      noTrueChildStrategy="returnLastPrediction"
>
      <Node score="5.483870967741935">
            <True/>
            <Node score="4.333333333333333">
                  SimplePredicate: pH <= 2.93
            </Node>
      </Node>
</TreeModel>
```

Optimization (3/3)

Removing split levels that don't affect the prediction:

```
<Node>
      SimplePredicate: Age > 35.5
      <Node score="0" recordCount="40">
            SimplePredicate: Income <= 194386
            ScoreDistribution:
                  "0" = 40, "1" = 0
      </Node>
      <Node score="0" recordCount="8">
            SimplePredicate: Income > 194386
            ScoreDistribution:
                  "0" = 7, "1" = 1
      </Node>
</Node>
```

```
<Node score="0" recordCount="48">
      SimplePredicate: Age > 35.5
      ScoreDistribution:
            "0" = 47, "1" = 1
</Node>
```

Code generation

```
<TreeModel
      missingValueStrategy="defaultChild"
>
      <Node id="1" defaultChild="3">
            <True/>
            <Node id="2" score="0">
                  SimplePredicate: Age <= 52
                  ScoreDistribution:
                        "0" = 7, "1" = 0
            </Node>
            <Node id="3" score="1">
                  SimplePredicate: Age > 52
                  ScoreDistribution:
                        "0" = 1, "1" = 2
            </Node>
      </Node>
</TreeModel>
```

```
Object[] node_1(FieldValue age, ...){
      if(age != null && age.asFloat() <= 52f){
            return node_2(...);
      } else {
            return node_3(...);
      }
}

Object[] node_2(...){
      return new Object[]{"0", 7d, 0d};
}

Object[] node_3(...){
      return new Object[]{"1", 1d, 2d};
}
```

Bad Idea!

Common pitfalls

Not treating splits on continuous features with the required precision (tolerance < 0.5 ULP):

- Truncating values. Ex: "1.5000(..)1" → "1.50"
- Changing data type. Ex: float ↔ double
- Changing arithmetic expressions. Ex: $(x_1 / x_2) \leftrightarrow (1 / x_2) * x_1$

Comparison of regression models

	R	Scikit-Learn	Apache Spark ML
Algorithms	Few built-in, many external	Many built-in	Few built-in
Term type(s)	Simple and derived features; interactions	Simple features	
Reuse	Hard	Easy	

Regression model declaration

```
<RegressionModel functionName="regression" normalizationMethod="none">
     <RegressionTable intercept="15.50143741004145">
          <!-- Simple continuous feature -->
          <NumericPredictor name="cylinders" coefficient="2.1609496766686194"/>
          <!-- Simple categorical feature -->
          <CategoricalPredictor name="origin" value="2" coefficient="-35.87525051244351"/>
          <CategoricalPredictor name="origin" value="3" coefficient="-38.206750156693424"/>
          <!-- Interaction -->
          <PredictorTerm coefficient="-0.007734946028064237">
               <FieldRef field="cylinders"/>
               <FieldRef field="displacement"/>
          </PredictorTerm>
          <!-- Derived feature; I(log(weight)) is backed by a DerivedField element -->
          <NumericPredictor name="I(log(weight))" coefficient="4.874500863508498"/>
     </RegressionTable>
</RegressionModel>
```


http://dmg.org/pmml/v4-3/Regression.html

Generalized regression model declaration

```
<GeneralRegressionModel functionName="regression" linkFunction="identity">
      <ParameterList>
            <Parameter name="p0" label="(intercept)"/>
            <Parameter name="p11" label="cylinders:displacement"/>
      </ParameterList>
      <PPMatrix>
            <PPCell value="1" predictorName="cylinders" parameterName="p11"/>
            <PPCell value="1" predictorName="displacement" parameterName="p11"/>
      </PPMatrix>
      <ParamMatrix>
            <PCell parameterName="p0" beta="15.50143741004145"/>
            <PCell parameterName="p11" beta="-0.007734946028064237"/>
      </ParamMatrix>
</GeneralRegressionModel>
```

http://dmg.org/pmml/v4-3/GeneralRegression.html

Code generation

```
double mpg(FieldValue cylinders, FieldValue displacement, FieldValue weight, FieldValue origin){
	return 15.50143741004145d + // intercept
		2.1609496766686194d * cylinders.asDouble() + // simple continuous feature
		origin(origin.asString()) + // simple categorical feature
		-0.007734946028064237d  * (cylinders.asDouble() * displacement.asDouble()) + // interaction
		4.874500863508498d * Math.ln(weight.asDouble()) // derived feature
}

double origin(String origin){
	switch(origin){
		case "1": return 0d; // baseline
		case "2": return -35.87525051244351d;
		case "3": return -38.206750156693424d;
	}
	throw new IllegalArgumentException(origin);
}
```

Grand summary

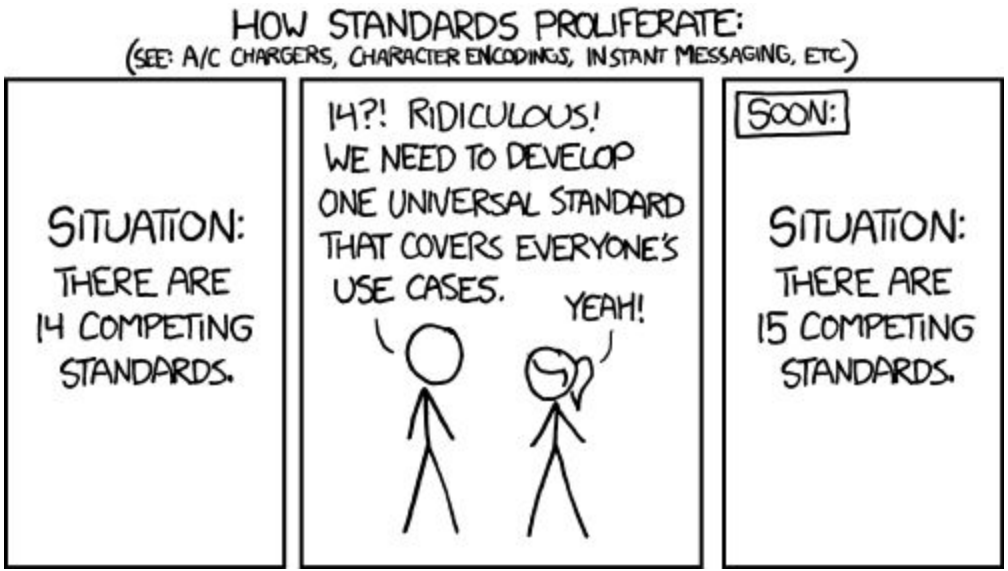
	R	Scikit-Learn	Apache Spark ML
ML-platform model information density	Medium to high	Low	Low to medium
ML-platform model interpretability	Good	Bad	Bad
PMML markup optimizability	Low	High	Medium
ML-platform to PMML learning and migration effort	Medium	Low to medium	Low

Q&A

villu@openscoring.io

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

PMML vs. PFA



https://xkcd.com/927/