Cascading Pattern - How to quickly migrate Predictive Models (PMML) from SAS, R, MicroStrategy onto Hadoop and deploy them at scale

V1.0
September 12, 2013

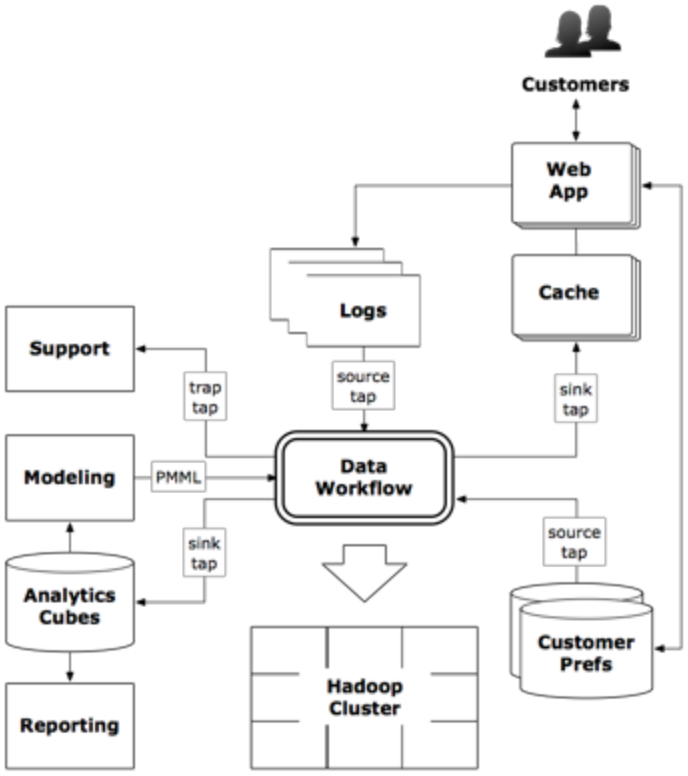
Introduction

Summary

Cascading Pattern is a machine learning project within the Cascading development framework used to build enterprise data workflows. The Cascading framework provides an abstraction layer on top of Hadoop and other computing topologies. It allows enterprises to leverage existing skills and resources to build data processing applications on Apache Hadoop, without specialized Hadoop skills. Pattern, in particular, leverages an industry standard called Predictive Model Markup Language (PMML), which allows data scientists to leverage their favorite statistical & analytics tools such as R, Oracle, etc., to export predictive models and quickly run them on data sets stored in Hadoop. Pattern's benefits include reduced development costs, time savings and reduced licensing issues at scale – all while leveraging Hadoop clusters, core competencies of analytics staff, and existing intellectual property in the predictive models.

Enterprise Use Cases

In the context of an Enterprise application, Cascading Pattern would tend to fit in the diagram shown below. Analytics work in the back office typically is where predictive modeling gets performed -- using SAS, R, SPSS, Microstrategy, Teradata, etc.



By using Cascading Pattern, predictive modeling can now be exported as PMML from a variety of analytics frameworks, then run on Apache Hadoop at scale. This saves licensing costs while allowing for applications to scale-out and allows for predictive modeling to be integrated directly within other business logic expressed as Cascading apps.

Target Audience

Data Scientists or Data Analysts with intermediate experience with statistical modeling tools like SAS, R, Microstrategies and novice Java experience.

Prerequisites:

- System requirements: Mac OS, Windows 7, Linux, Minimum 4GB RAM
- JDK 1.6
- Gradle
- Hortonworks Data Platform Sandbox
- R and R-Studio

Overview

During the course of this tutorial you'll perform the following steps:

- Environment setup
- Download source code
- Model creation
- Cascading Build
- Enterprise App
- Additional Resources

Step 1 – Environment Setup

In this section, we will go through the steps needed to setup your environment.

1. To set up Java for your environment, download Java (http://www.java.com/getjava/) and follow the installation instructions.

Notes:
- Version 1.6.x was used to create the examples used here
- Get the JDK, not the JRE
- Install according to vendor instructions
- Be sure to set the JAVA_HOME environment variable correctly

2. To set up Gradle for your environment, download Gradle (http://www.gradle.org/downloads) and follow the installation instructions.

Notes:
- Version 1.4 and later is required for some examples in this tutorial
- Install according to vendor instructions
- Be sure to set the GRADLE_HOME environment variable correctly

3. Set up the Hortonworks Data Platform Sandbox, per vendor instructions.

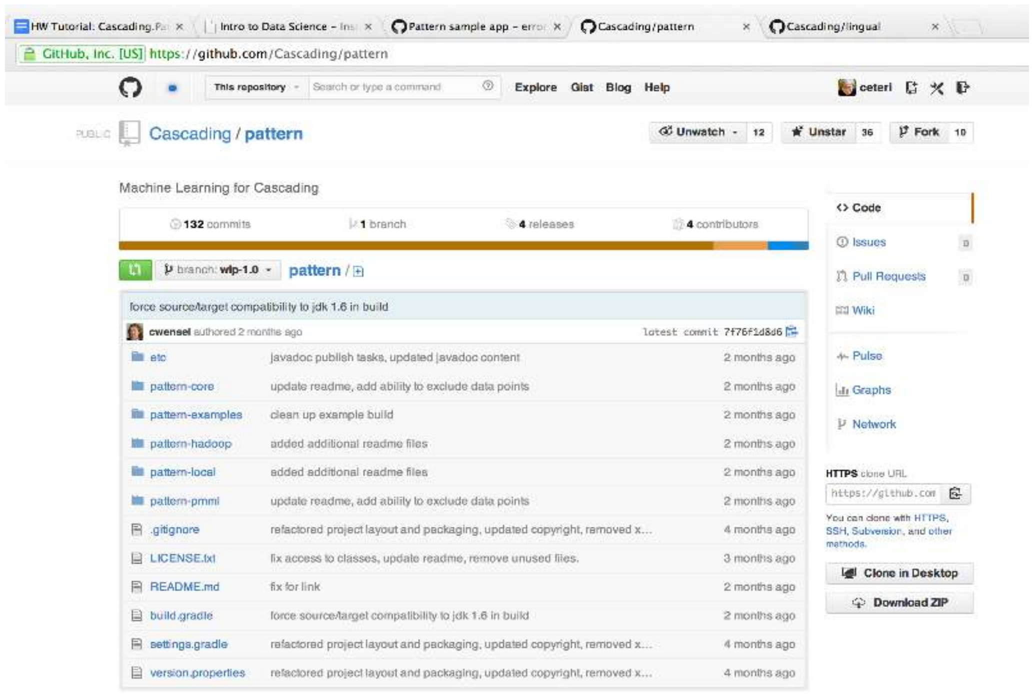
4. Set up R and RStudio for your environment by visiting:
   - http://cran.r-project.org/
   - http://www.rstudio.com/ide/download/

Step 2 – Source Code

In this section, we will…

1. Navigate to GitHub (https://github.com/Cascading/pattern), to download the source code which we will be using.

2. In the bottom right corner of the screen, click "Download ZIP" to download a ZIP compressed archive of the source code for the Cascading Pattern project.



Machine Learning for Cascading

132 commits | 1 branch | 4 releases | 4 contributors

branch: wip-1.0 | pattern / +

force source/target compatibility to jdk 1.6 in build

cwensel authored 2 months ago — latest commit 7f76f1d8d6

etc	javadoc publish tasks, updated javadoc content	2 months ago
pattern-core	update readme, add ability to exclude data points	2 months ago
pattern-examples	clean up example build	2 months ago
pattern-hadoop	added additional readme files	2 months ago
pattern-local	added additional readme files	2 months ago
pattern-pmml	update readme, add ability to exclude data points	2 months ago
.gitignore	refactored project layout and packaging, updated copyright, removed x…	4 months ago
LICENSE.txt	fix access to classes, update readme, remove unused files.	3 months ago
README.md	fix for link	2 months ago
build.gradle	force source/target compatibility to jdk 1.6 in build	2 months ago
settings.gradle	refactored project layout and packaging, updated copyright, removed x…	4 months ago
version.properties	refactored project layout and packaging, updated copyright, removed x…	4 months ago

3. Once this has completed downloading, unzip and move the directory "pattern" to a location on your file system where you have space available to work.

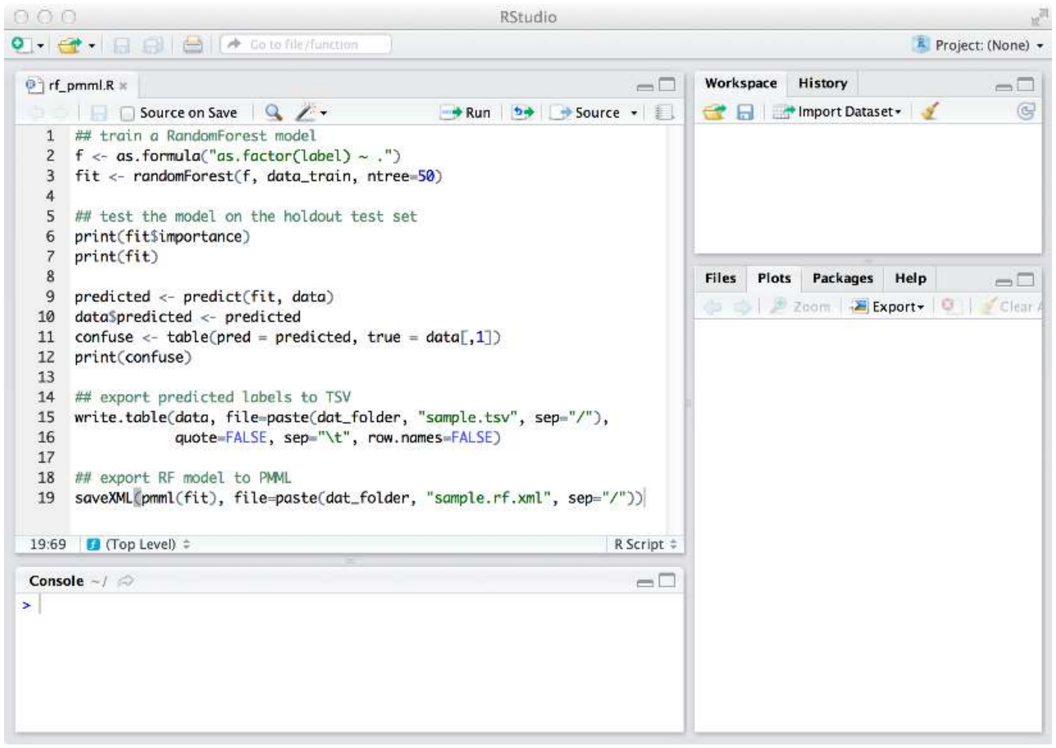
Step 3 – Model Creation

1. Connect to the "pattern" directory, and then into its "pattern-examples" subdirectory. There is an example R script in "examples/r/rf_pmml.R" that creates a Random Forest model. This is representative of a predictive model for an anti-fraud classifier used in e-commerce apps.

```
## train a RandomForest model
f <- as.formula("as.factor(label) ~ .")
fit <- randomForest(f, data_train, ntree=50)

## test the model on the holdout test set
print(fit$importance)
print(fit)

predicted <- predict(fit, data)
data$predicted <- predicted
confuse <- table(pred = predicted, true = data[,1])
print(confuse)

## export predicted labels to TSV
write.table(data, file=paste(dat_folder, "sample.tsv", sep="/"),
  quote=FALSE, sep="\t", row.names=FALSE)

## export RF model to PMML
saveXML(pmml(fit), file=paste(dat_folder, "sample.rf.xml", sep="/"))
```

2. Load the “rf_pmml.R” script into RStudio using the “File” menu and “Open File...” option.

3. Click the “Source” button in the upper middle section of the screen.



```
## train a RandomForest model
f <- as.formula("as.factor(label) ~ .")
fit <- randomForest(f, data_train, ntree=50)

## test the model on the holdout test set
print(fit$importance)
print(fit)

predicted <- predict(fit, data)
data$predicted <- predicted
confuse <- table(pred = predicted, true = data[,1])
print(confuse)

## export predicted labels to TSV
write.table(data, file=paste(dat_folder, "sample.tsv", sep="/"),
            quote=FALSE, sep="\t", row.names=FALSE)

## export RF model to PMML
saveXML(pmml(fit), file=paste(dat_folder, "sample.rf.xml", sep="/"))
```

This will execute the R script and create the predictive model. The last line saves the predictive model into a file called “sample.rf.xml” as PMML. If you take a look at that PMML, it’s XML and not optimal for humans to read, but efficient for machines to parse:

```
<?xml version="1.0"?>
<PMML version="4.0" xmlns="http://www.dmg.org/PMML-4_0"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://www.dmg.org/PMML-4_0
 http://www.dmg.org/v4-0/pmml-4-0.xsd">
 <Header copyright="Copyright (c)2012 Concurrent, Inc."
  description="Random Forest Tree Model">
  <Extension name="user" value="ceteri" extender="Rattle/PMML"/>
  <Application name="Rattle/PMML" version="1.2.30"/>
  <Timestamp>2012-10-22 19:39:28</Timestamp>
 </Header>
 <DataDictionary numberOfFields="4">
  <DataField name="label" optype="categorical" dataType="string">
   <Value value="0"/>
   <Value value="1"/>
  </DataField>
  <DataField name="var0" optype="continuous" dataType="double"/>
  <DataField name="var1" optype="continuous" dataType="double"/>
  <DataField name="var2" optype="continuous" dataType="double"/>
 </DataDictionary>
 <MiningModel modelName="randomForest_Model"
functionName="classification">
  <MiningSchema>
   <MiningField name="label" usageType="predicted"/>
   <MiningField name="var0" usageType="active"/>
   <MiningField name="var1" usageType="active"/>
   <MiningField name="var2" usageType="active"/>
  </MiningSchema>
  <Segmentation multipleModelMethod="majorityVote">
   <Segment id="1">
    <True/>
    <TreeModel modelName="randomForest_Model"
functionName="classification"
     algorithmName="randomForest" splitCharacteristic="binarySplit">
     <MiningSchema>
      <MiningField name="label" usageType="predicted"/>
      <MiningField name="var0" usageType="active"/>
      <MiningField name="var1" usageType="active"/>
      <MiningField name="var2" usageType="active"/>
     </MiningSchema>
...
```

Cascading Pattern supports additional models, as well as ensembles of the following models.

- General Regression
- Regression
- Clustering
- Tree
- Mining

Step 4 – Cascading Build

1. Now that we have a model created and exported as PMML, let's work on running it at scale atop Apache Hadoop. In the "pattern-examples" directory, execute the following Bash shell commands:

```
gradle clean jar
```

2. That line invokes Gradle to run the build script "build.gradle", and compile the Cascading Pattern example app.

3. After that compiles look for the built app as a JAR file in the "build/libs" subdirectory:

```
ls -lts build/libs/pattern-examples-*.jar
```

4. Now we're ready to run this Cascading Pattern example app on Apache Hadoop. First, we make sure to delete the output results (required by Hadoop). Then we run Hadoop: we specify the JAR file for the app, the PMML file using a "--pmml" command line option, along with sample input data "data/sample.tsv" and the location of the output results:

```
rm -rf out
hadoop jar build/libs/pattern-examples-*.jar \
 data/sample.tsv out/classify --pmml data/sample.rf.xml
```

5. After that runs, check the "out/classify" subdirectory. Look at the results of running the PMML model, which will be in the "part-*" partition files:

```
more out/classify/part-*
```

6. Let's take a look at what we just built and ran. The source code for this example is located in the "src/main/java/cascading/pattern/Main.java" file:

```
public class Main
  {
  /** @param args  */
  public static void main( String[] args ) throws RuntimeException
    {
    String inputPath = args[ 0 ];
    String classifyPath = args[ 1 ];
    // set up the config properties
    Properties properties = new Properties();
    AppProps.setApplicationJarClass( properties, Main.class );
    HadoopFlowConnector flowConnector = new
HadoopFlowConnector( properties );

    // create source and sink taps
    Tap inputTap = new Hfs( new TextDelimited( true, "\t" ), inputPath );
    Tap classifyTap = new Hfs( new TextDelimited( true, "\t" ),
classifyPath );

    // handle command line options
    OptionParser optParser = new OptionParser();
    optParser.accepts( "pmml" ).withRequiredArg();

    OptionSet options = optParser.parse( args );

    // connect the taps, pipes, etc., into a flow
    FlowDef flowDef = FlowDef.flowDef()
      .setName( "classify" )
      .addSource( "input", inputTap )
      .addSink( "classify", classifyTap );

    // build a Cascading assembly from the PMML description
    if( options.hasArgument( "pmml" ) )
      {
      String pmmlPath = (String) options.valuesOf( "pmml" ).get( 0 );

      PMMLPlanner pmmlPlanner = new PMMLPlanner()
        .setPMMLInput( new File( pmmlPath ) )
        .retainOnlyActiveIncomingFields()
        .setDefaultPredictedField( new Fields( "predict",
Double.class ) );
         // default value if missing from the model

      flowDef.addAssemblyPlanner( pmmlPlanner );
      }

    // write a DOT file and run the flow
    Flow classifyFlow = flowConnector.connect( flowDef );
    classifyFlow.writeDOT( "dot/classify.dot" );
    classifyFlow.complete();
    }
  }
```

Most of the code is the basic plumbing used for Cascading apps. The portions which are specific to Cascading Pattern and PMML are the few lines involving the “pmmlPlanner” object.

Survey

Thank you for downloading and working through this turtorial. We’re interested in your feedback. Please take a minute to answer a few questions in this survey so that we can continue to improve.

Additional Resources

- Community - http://www.cascading.org/
- Cascading Pattern Home - http://www.cascading.org/pattern/
- Cascading Lingual - http://www.cascading.org/lingual/
- O’Reilly Book Cascading - http://oreil.ly/143JST6

About Concurrent

About Hortonworks