CIS5560 Term Project Tutorial

Authors: Prathibha Gubbi Prakash; Anusha Manjappa; Srihitha Reddy Sivannagari

Instructor: Jongwook Woo

Date: 05/19/2017

Predictive analysis of Newyork motor cycle collisions

Predictive analysis of various elements associated with motorcycle accidents in New York city like borough, kind of vehicle, contributing factor of the accident, count of accidents in a particular area and factors contributing to those accidents based on available features in the data set using Machine Learning Algorithms with tools like AzureML and SparkML.

Creating a Classification Model using Random forest classifier, Decision tree classifier, Logistic regression

In this notebook, we will implement three types of classification model using *Random forest classifier*, *Decision tree classifier*, *LogisticRegression* that uses features of a Newyork motorcycle collision data to predict the number of accidents that took place in each borough in newyork state.

Steps to build, train and test the model from the dataset:

1. Import the libraries you will need and prepare the training and test data
2. Load the source data into table
3. Prepare the data with the features (input columns, output column as label)
4. Split the data using data.randomSplit(): Training and Testing
5. Transform the columns to a vector using VectorAssembler

6. set features and label from the vector
7. Define a pipeline that creates a feature vector
8. Build a Model with the label and features
9. Train the model
10. Prepare the testing Data Frame with features and label from the vector; Rename label to trueLabel
11. Predict and test the testing Data Frame using the model trained at the step 8
12. Compare the predicted result and trueLabel

Import Spark SQL and Spark ML Libraries

First, import the libraries you will need and prepare the training and test data:

```
# Import Spark SQL and Spark ML libraries
from pyspark.sql.types import *
from pyspark.sql.functions import *

from pyspark.ml import Pipeline
from pyspark.ml.classification import LogisticRegression , RandomForestClassifier
, DecisionTreeClassifier , MultilayerPerceptronClassifier
from pyspark.ml.feature import VectorAssembler, StringIndexer
from pyspark.ml.tuning import ParamGridBuilder, TrainValidationSplit,
CrossValidator
from pyspark.ml.evaluation import BinaryClassificationEvaluator
```

Load Source Data

1. The data for this exercise is provided as a CSV file containing details of Newyork motorcycle collisions. The data includes specific characteristics (or *features*) for each Accidents, as well as a column indicating how many Accidents occured in each borough along with date time factors. It also records the vehicle type invloved in accidents, contributing factors, SeverityOfInjury and so on. we will load this data into a DataFrame and display it.

2. We load this data into tables on databricks and make necessary changes to the columns data type.

3. we then use sqlContext to select and view sample data on databricks

```
csv= sqlContext.sql("Select * from nymc_csv");
csv.show(2);
```

Selecting labels and features

For this experiment we choose borough as our feature, this column was a categorical column of type string. So we had to use***StringIndexer*** to provide indices to these feature column. Secondly, we introduced a derived column in our dataset which is the sum of all type of accidents corresponding to each row. (Incidents = NUMBEROFPERSONSINJURED + NUMBEROFPERSONSKILLED + NUMBEROFPEDESTRIANSINJURED + NUMBEROFPEDESTRIANSKILLED +NUMBEROFCYCLISTINJURED + NUMBEROFMOTORISTINJURED + NUMBEROFMOTORISTKILLED). We used this derived column as our label and further trained this using classification model.

```
data = sqlContext.sql("select Borough, NUMBEROFPERSONSINJURED +
NUMBEROFPERSONSKILLED + NUMBEROFPEDESTRIANSINJURED + NUMBEROFPEDESTRIANSKILLED
+NUMBEROFCYCLISTINJURED + NUMBEROFMOTORISTINJURED + NUMBEROFMOTORISTKILLED as
Incidents from ppp");

indexer= StringIndexer(inputCol="Borough", outputCol="indx_borough")
indexed= indexer.fit(data).transform(data)

# rename the Incidents column to label
indx_feat=indexed.select("indx_borough", col("Incidents").alias("label"))
```

Split data

It is common practice when building supervised machine learning models to split the source data, using some of it to train the model and reserving some to test the trained model. In this exercise, we will use 70% of the data for training, and reserve 30% for testing. In the testing data, the **label** column is renamed to **trueLabel** so we can use it later to compare predicted labels with known actual values.

```
# Split the data
splits = indx_feat.randomSplit([0.7, 0.3])
train = splits[0]
test = splits[1].withColumnRenamed("label", "trueLabel")
train = train.count()
test = test.count()
print "Training data:", train
print "Testing data:", test
train.show(10)
test.show(10)
```

Transform the feature columns into a vector

To train the classification model, you need a training data set that includes a vector of numeric features, and a label column. In this exercise, you will use the **VectorAssembler** class to transform the feature columns into a vector.

```
vectorAssembler = VectorAssembler(inputCols=["indx_borough"],
outputCol="features")
```

Train a Classification Model : RandomForestClassifier

Next, you need to train a classification model using the training data. To do this, create an instance of the classification algorithm you want to use and use its **fit** method to train a model based on the training DataFrame. In this exercise, you will use a *Logistic Regression* classification algorithm - though you can use the same technique for any of the classification algorithms supported in the spark.ml API.

```
# build RandomForestClassifier model with features and label

dt_1 = RandomForestClassifier(labelCol="label", featuresCol= "features")
pipeline_1 = Pipeline(stages=[vectorAssembler, dt_1])

# train a model based on the training DataFrame
model_1 = pipeline_1.fit(train)
print "First Model trained!"
```

Test the Model :RandomForestClassifier

Now you're ready to use the **transform** method of the model to generate some predictions. You can use this approach to predict delay status for flights where the label is unknown; but in this case you are using the test data which includes a known true label value, so you can compare the predicted status to the actual status.

```
predictions_1 = model_1.transform(test)
predictions_1.select("prediction", "trueLabel")
```

Metrics Evaluation : RandomForestClassifier

As we have used Multiclass classification model, we calculated the accurracy and test error of our model using **MulticlassClassificationEvaluator**

```
evaluator_1= MulticlassClassificationEvaluator()
.setLabelCol("trueLabel")
.setPredictionCol("prediction")
.setMetricName("accuracy")
treeModel_1 = model_1.stages[1]

print "Learned classification tree model:" , treeModel
accuracy_1 = evaluator.evaluate(predictions_1)
print "Average Accuracy =", accuracy_1
print "Test Error = " , (1.0 - accuracy_1)
```

Result : RandomForestClassifier

Average Accuracy = 0.859718490594

Test Error = 0.140281509406

Train a Classification Model : LogisticRegression

Next, you need to train a classification model using the training data. To do this, create an instance of the classification algorithm you want to use and use its **fit** method to train a model based on the training DataFrame. In this exercise, you will use a *Logistic Regression* classification algorithm - though you can use the same technique for any of the classification algorithms supported in the spark.ml API.

```
# build LogisticRegression model with features and label

dt_2 = LogisticRegression(labelCol="label", featuresCol= "features")
pipeline_2 = Pipeline(stages=[vectorAssembler, dt_2])

# train a model based on the training DataFrame
model_2 = pipeline_2.fit(train)
print "Second Model trained!"
```

Test the Model : LogisticRegression

Now you're ready to use the **transform** method of the model to generate some predictions. You can use this approach to predict delay status for flights where the label is unknown; but in this case you are using the test data which includes a known true label value, so you can compare the predicted status to the actual status.

```
predictions_2 = model_2.transform(test)
predictions_2.select("prediction", "trueLabel")
```

Metrics Evaluation : LogisticRegression

As we have used Multiclass classification model, we calculated the accurracy and test error of our model using **MulticlassClassificationEvaluator**

```
evaluator_2= MulticlassClassificationEvaluator()
.setLabelCol("trueLabel")
.setPredictionCol("prediction")
.setMetricName("accuracy")
treeModel_2 = model_2.stages[1]

print "Learned classification tree model:" , treeModel
accuracy_2 = evaluator.evaluate(predictions_2)
print "Average Accuracy =", accuracy_2
print "Test Error = " , (1.0 - accuracy_2)
```

Result : LogisticRegression

Average Accuracy = 0.851718490594

Test Error = 0.1418201509406

Train a Classification Model : DecisionTreeClassifier

Next, you need to train a classification model using the training data. To do this, create an instance of the classification algorithm you want to use and use its **fit** method to train a model based on the training DataFrame. In this exercise, you will use a *Logistic Regression* classification algorithm - though you can use the same technique for any of the classification algorithms supported in the spark.ml API.

```
# build DecisionTreeClassifier model with features and label

dt_3 = DecisionTreeClassifier(labelCol="label", featuresCol= "features")
pipeline_3 = Pipeline(stages=[vectorAssembler, dt_3])

# train a model based on the training DataFrame
model_3 = pipeline_3.fit(train)
print "Third Model trained!"
```

Test the Model : DecisionTreeClassifier

Now you're ready to use the **transform** method of the model to generate some predictions. You can use this approach to predict delay status for flights where the label is unknown; but in this case you are using the test data which includes a known true label value, so you can compare the predicted status to the actual status.

```
predictions_3 = model_3.transform(test)
predictions_3.select("prediction", "trueLabel")
```

Metrics Evaluation : DecisionTreeClassifier

As we have used Multiclass classification model, we calculated the accurracy and test error of our model using **MulticlassClassificationEvaluator**

```
evaluator_3 = MulticlassClassificationEvaluator()
.setLabelCol("trueLabel")
.setPredictionCol("prediction")
.setMetricName("accuracy")
treeModel_3 = model_3.stages[1]

print "Learned classification tree model:" , treeModel
accuracy_3 = evaluator.evaluate(predictions_3)
print "Average Accuracy =", accuracy_3
print "Test Error = ", (1 - accuracy_3)
```

Result : DecisionTreeClassifier

Average Accuracy = 0.858718490594

Test Error = 0.141281509406

Conclusion: Best model

All three models gave almost same accuracyy value with slight variation in the precision value. Random forest classifier is slightly better than other two models, because of higher accuracy rate of **85.9%**.

(Since it is a multiclass classification we don't get TP,TN,FP,FN values hence we cannot calculate precission and recall value)

References:

1. https://data.cityofnewyork.us/Public-Safety/NYPD-Motor-Vehicle-Collisions/h9gi-nx95
2. https://docs.microsoft.com/en-us/azure/hdinsight/hdinsight-apache-spark-jupyter-spark-sql
3. https://spark.apache.org/docs/1.6.1/ml-classification-regression.html