HOW TO USE APACHE SPARK TO MAKE PREDICTIONS FOR PREVENTIVE MAINTENANCE



In part one we explained how to create a training model. In this part we show how to make predictions to show which machines in our dataset should be taken out of service for maintenance.

First, here is how to submit the job to Spark with spark-submit:

- jar file that contains com.bmc.lr.makePrediction
- what file to read, i.e., the one you just generated in Part I (put link to previous article)
- which model to use i.e., the one you just generated in Part I (put link to previous article)

```
spark-submit
--class com.bmc.lr.makePrediction
--master local
hdfs://localhost:9000/maintenance/lr-assembly-1.0.jar
hdfs://localhost:9000/maintenance/2018.04.20.15.48.54.csv
hdfs://localhost:9000/maintenance/maintenance_model
```

(This tutorial is part of our Apache Spark Guide. Use the right-hand menu to navigate.)

Code

Here are the usual imports and package name.

```
package com.bmc.lr
```

```
import org.apache.spark.sql.SQLContext
import org.apache.spark.ml.classification.{BinaryLogisticRegressionSummary,
LogisticRegression}
import org.apache.spark.mllib.linalg.{Vector, Vectors}
import org.apache.spark.ml.evaluation.BinaryClassificationEvaluator
import org.apache.spark.ml.feature.{VectorAssembler, StringIndexer}
import org.apache.spark.ml.classification.LogisticRegressionModel
import org.apache.spark.SparkConf
import org.apache.spark.sql.SQLContext
import org.apache.spark.SparkContext
```

Read the command line arguments and set up the Spark and SQL context. Create the object makePrediction so that we can instantiate class **com.bmc.lr.makePrediction** from **spark-submit**.

```
object makePrediction {

def main(args: Array): Unit = {
 val conf = new SparkConf().setAppName("lr")
 val sc = new SparkContext(conf)
var modelFile = args(1);
var file = args(0);

val sqlContext = new SQLContext(sc)
```

Use **databricks** to read the input file and create a **org.apache.spark.sql.DataFrame**. Later databricks will be useful to save the output to Hadoop as it requires only line of code.

```
val df = sqlContext.read.format("com.databricks.spark.csv").option("header",
"true").option("inferSchema", "true").option("delimiter",";").load(file)
```

At this point **df** looks like this:

```
lifetime|broken| pressureInd|       moistureInd|    temperatureInd|team|  provider|
+-----------------+------+-----------------+-----------------+-------------
|17.792044609594086| 0|104.25543343273085| 88.67373165099184|122.75111030066334|  47|Ford
F-750|
| 69.39805687060903| 0| 93.34561942201603| 86.62996015487022|  84.9796059428202|  34|Ford
F-750|
| 84.53664924532875| 0|110.64579687466193|125.89351825805036| 58.34915688191312|   8|Ford
F-750|
```

org.apache.spark.ml.featureVectorAssembler transforms the features in **featureCols** into a vector column. We need it in this format to plug into **LogisticRegressionModel.transform()** which takes the features**(lifetime, pressureInd, moistureInd, temperatureInd)** and labels (**broken** which we rename to **label** for clarity) as a step to predict which machines will fail.

Then we take the **broken** (1 or 0) label and put that into **org.apache.spark.ml.feature.StringIndexer** to contain a single value, the label. org.apache.spark.ml.feature.StringIndexer.fit() fits the data to the model, meaning make the predictions.

You can see what the features vectors look like after we create **df3** (see

below).

```
val featureCols =  Array("lifetime", "pressureInd", "moistureInd",
"temperatureInd")

val assembler = new
VectorAssembler().setInputCols(featureCols).setOutputCol("features")

val labelIndexer = new
StringIndexer().setInputCol("broken").setOutputCol("label")

val df2 = assembler.transform(df)

val df3 = labelIndexer.fit(df2).transform(df2)
```

Here we show the features column (albeit it is chopped off in the display). If you recall this is like the dense vectors we have been using to plug into to training models.

```
df3.select("features").show()
+--------------------+
|            features|
+--------------------+
|[17.7920446095940...|
|[69.3980568706090...|
|[84.5366492453287...|
|[71.6002965259175...|
|[46.8176995523900...|
```

Load saved **org.apache.spark.ml.classification.LogisticRegressionModel** model from Hadoop file system. We trained this model in the first program.

```
val model = LogisticRegressionModel.load(modelFile)
```

Here we select just the columns that we want (getting rid of features, since it is hard to print and view and redundant too). Then we show only those machines that require maintenance, i.e., those whose predicted value of broken is 1 (true).

```
val predictions = model.transform(df3)

var df4 = predictions.select ("team", "provider", "pressureInd",
"moistureInd", "temperatureInd", "label", "prediction")

val df5 = df4.filter("prediction=1")

df5.show()
```

With df5.show() we see that 4 machines need maintenance:

```
+----+----------+-----------------+-----------------+----------------+-----+----------
+
```

```
|team|  provider|       pressureInd|       moistureInd|
temperatureInd|label|prediction|
+----+----------+------------------+------------------+----------------+-----+----------
+
|  34|Ford F-750| 93.34561942201603| 86.62996015487022| 84.9796059428202|  0.0|
1.0|
|   8|Ford F-750|110.64579687466193|125.89351825805036|58.34915688191312|  0.0|
1.0|
|  83|Ford F-750| 55.77009802003853|   66.832777175712|125.2982705340028|  0.0|
1.0|
|   2|Ford F-750|  84.1763960666348|  82.1342684415311|57.73202884026434|  0.0|
1.0|
+----+----------+------------------+------------------+----------------+-----+----------
+
```

Now we see another example of how useful databricks is by legging us write the predictions in a .csv to Hadoop. We save the output in a file whose name is the date. Note also that we import java.util.Date since it is the easiest way to create different date formats.

```
import java.util.Date
import java.text.SimpleDateFormat
val date = new Date()
var dformat:SimpleDateFormat = new SimpleDateFormat("yyyy.MM.dd.HH.mm.ss");
val csvFile = "hdfs://localhost:9000/maintenance/" + dformat.format(date) +
".csv"
df5.write.format("com.databricks.spark.csv").option("header",
"true").save(csvFile)

}
}
```

Now we show the data:

```
hadoop fs -cat
hdfs://localhost:9000/maintenance/2018.05.16.09.07.33.csv/part-00000-7a9a4de6
-ab90-4cd4-b9f5-d0d30c881b2c-c000.csv
team,provider,pressureInd,moistureInd,temperatureInd,label,prediction
34,Ford F-750,93.34561942201603,86.62996015487022,84.9796059428202,0.0,1.0
8,Ford F-750,110.64579687466193,125.89351825805036,58.34915688191312,0.0,1.0
83,Ford F-750,55.77009802003853,66.832777175712,125.2982705340028,0.0,1.0
2,Ford F-750,84.1763960666348,82.1342684415311,57.73202884026434,0.0,1.0
```