Assignment 7

Abhijeet Sharma, Deepend Mehta, Akshay Raje, Afan Ahmad Khan

Saturday, March 19, 2016

INTRODUCTION

The program builds a Flight Prediction model which trains on history data and predicts on test dataset. The purpose of the program is to propose two-hop routes that minimize the chance of missed connections for a future date. There is only 1 Model which is trained on all of the History Data and predicted on the test data. The model is built using Apache Spark framework. The Classifier used is Random Forest. The Classifier's output predictions is found in Output/TestPredictions/test-predictions in the format: "year, month, day, originAirport ,destinationAirport, originFlightNum, destinationFlightNum, duration and predicted missed connections".

Fine print: 1. We Provide code that can run pseudo-distributed hadoop/spark as well as on AWS EMR. 2. We have provided a Confusion Matrix evaluating the prediction Results. 3. We also calculated error rate as a measure of accuracy of our model where error rate = % of on-time flights misclassified as delayed + % of delayed flights misclassified as on-time. 4. We have included a MakeFile that executes the piepline and produces this Report. 5. This one page report documents our implementation and describes our results. The report is automatically constructed as part of running the pipeline. 6. We have submitted a tar.gz file which unpacks into a directory name "Sharma_Mehta_Raje_Khan_A7". The directory contains a README file that explains how to build and run our code.

IMPLEMENTATION

1. The Whole Program Pipeline is divided into four parts, a MR Job for cleaning Train Data, a MR Job for cleaning Test Data, a Spark job for building a Prediction Model for Missed Connections a R program for running the Validation data on the Predicted Dataset We built a Random Forest Model and trained in with approx **25 million Records** and tested it with approx **181000** Records. The Model is trained on the History Data and predicted on the test data. This test data is the data filtered out after reading records from the Requested data. We have implemented a specific Seed(=42) for our Random Forest Model for reproducing results.
2. In the Map Reduce Jobs, The Mapper calls the map method with "Carrier Code + Origin/Destination" as Key. The record is checked for sanity tests and Feature Columns are sent to the Reducer as Values. Note: The choice of Carrier Code and Origin/Destination as key is done for computing missed connections.It has no impact on the Prediction Model.
3. The reducer does an equi-join on intermmediate airports for a two-hop route.It also calculates the number of missed connections for training data and extracts the required features for the predication model. Rows from the request file are used to filter on the test data.
4. The output Training and Test files are consolidated and sent to the AWS Spark program as inputs.
5. The Spark program builds a random forest model on the training set and predicts on the test data.It outputs a file containing records in the format: [year, month, day, originAirport ,destinationAirport, originFlightNum, destinationFlightNum, duration and predicted missed connections].
6. The R script reads the Spark output file and computes the flight with the minimum duration. It also calculates the accuracy of the prediction model with the help of the validation file.
7. Make file is created to automate the pipeline in both local machine and AWS EMR clusters.

DIRECTORY STRUCTURE

```
* has to be set by user manually before run
+ is created by the Program during execution

|Sharma_Mehta_Raje_Khan_A7
|README.txt                         (Description of the Submission)
|MakeFile                           (Makefile for the project)
|Assignment7_Report.pdf             (PDF Report of the project results)
|Assignment7_Report.Rmd             (Rmd file for the Report)
|PreProcessFlight.java              (Hadoop MR Driver Java file)
|PreProcessMapper.java              (Hadoop MR Mapper Java file)
|PreProcessMapper.java              (Hadoop MR Driver Java file)
|CSVParser.java                     (Open Source Implementation For parsing of input records)
|CSVReaderNullFieldIndicator.java   (Helper File for CSVParser.java)
|commons-lang3-3.4.jar              (Helper File for CSVParser.java)
|clusterWaitingCheck.sh             (Shell script which waits for cluster to complete a step)
|sparkConfig.json                   (Configuration File for Spark in JSON)
|SparkModel
|SparkModel.scala               (Spark Machine Learning Model Creation Scala file)
|build.sbt                      (sbt File for building scala JAR package)
|input*
|a7history|(36 csv.gz files)*
|a7test|04redacted.csv.gz*
|a7validate|04missed.csv.gz*
|a7request|04req10k.csv.gz*
|output+
|a7history+
|a7test+
|TestPredictions
|FeatureImportance
```

Install the required R Packages

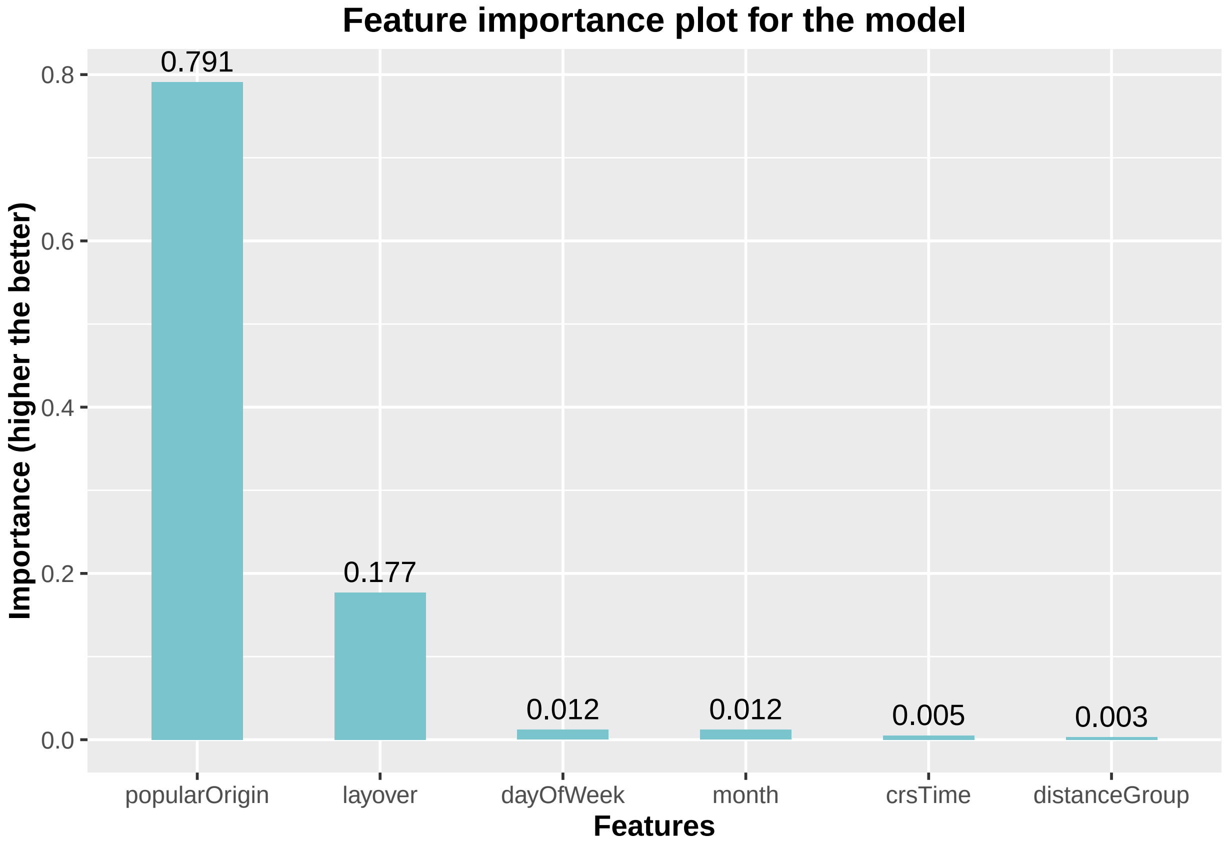
```r
install.packages("rmarkdown")
install.packages("ggplot2")
install.packages("e1071")
install.packages("caret")
install.packages("plyr")
install.packages("R.utils")
```

Confusion Matrix

```
## Confusion Matrix and Statistics
##
##              Actual
## Predicted    Not-Missed Missed
##   Not-Missed      95898  25305
##   Missed          36006  24603
##
##                Accuracy : 0.6628
##                  95% CI : (0.6606, 0.665)
##     No Information Rate : 0.7255
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.2063
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.7270
##             Specificity : 0.4930
##          Pos Pred Value : 0.7912
##          Neg Pred Value : 0.4059
##              Prevalence : 0.7255
##          Detection Rate : 0.5275
##    Detection Prevalence : 0.6666
##       Balanced Accuracy : 0.6100
##
##        'Positive' Class : Not-Missed
##
```

Feature Importances (in decreasing order of importance)

```
Layover (Minutes)                          -> 0.7910760348376943
CRS Time (Hour)                            -> 0.17667292405685459
Popular Origin (1:Popular,0:Not Popular)   -> 0.01225633896699857
Day of Week (in number)                    -> 0.011876846115361201
Month                                      -> 0.005054899900618705
DistanceGroup                              -> 0.003062956122472584
```



Timings (All times are in minutes)

```
1. Local Machine
  1.1 Preprocessing Train and Test Data     18 minutes
  1.2 Spark ML                              20 minutes

2. AWS Cluster 3 CORE m3.xlarge machines
  2.1 Preprocessing Train and Test Data     10 minutes
  2.2 Spark ML                              7 minutes
```

- AWS timings might vary depending on load. ** AWS timings include only step execution times.

SYSTEM SPECIFICATION & REQUIREMENTS:

```
1. Ubuntu 14.04 64-bit, 8GB RAM
2. Java 1.7.0_79
3. Apache Hadoop v.2.6.3 (http://www-us.apache.org/dist/hadoop/common/hadoop-2.6.3/hadoop-2.6.3.tar.gz)
4. Scala v.2.10.6 (http://www.scala-lang.org/download/2.10.6.html)
4. sbt
Linux: http://www.scala-sbt.org/release/docs/Installing-sbt-on-Linux.html
Mac: http://www.scala-sbt.org/release/docs/Installing-sbt-on-Mac.html
5. Apache Spark v.1.6.0 (http://www-eu.apache.org/dist/spark/spark-1.6.0/spark-1.6.0-bin-hadoop2.6.tgz)
6. Pandoc (https://github.com/jgm/pandoc/releases/tag/1.16.0.2)
7. R Packages:
  7.1 ggplot2
  7.2 rmarkdown
  7.3 e1071
  7.4 caret
  7.5 dplyr
  7.6 R.utils
```

Design Justifications:

1. Implement the sanity function and feature extraction logic in Spark itself instead of Hadoop jobs. -> "We weren't able to do this due to unfamiliarity with Scala/Spark and lack of time.Hopefully in the future, we will be incorporating it."
2. Implement Cross-Validation Techniques -> "This assignment simulates a production scenario where cross-validation and data exploration would have already been done with small samples of data."
3. Use Classifier -> "Due to lack of time, we only tested our pipeline with Gradient Boosted Trees and Random Forest.RF had higher accuracy than GBT although the difference in accuracy was minimal."
4. Implement Weka, R… instead of Spark -> "We didn't opt for Weka or R since they weren't built for handling huge datasets.To make it work, we would have had to either a) use workaounds like calling different processes, b) limit the scope of our model per year or per quarter.We believed implementing either of these choices would be right when Apache Spark can handle huge datasets and provides DataaFrames and Machine Learning libraries.We hence opted to build our Prediction model in Apache Spark.Instead of building multiple models, we can build one single effective model.Also, building in spark and deploying in AWS makes our program scalable in the future"
5. Sampling of training data -> The history data was highly skewed in favour of not missed connections. Only 15% of training data is missed. Due to this skewness our model would be highly susceptible to overfit on not-missed data. To resolve this we have done exploratory data analysis in R and confirmed that taking 20% of non-missed data from the training dataset would create a more balanced model. Hence we have trained the model on 20% of non-missed data and 100% of missed data which helped create a more balanced model.