Assignment 6

Abhijeet Sharma, Afan Ahmad Khan

Sunday, March 6, 2016

INTRODUCTION

The program builds a Flight Prediction Delay model which trains on history data and predicts on tests dataset. There is only 1 Model which is trained on all of the History Data and predicted on the test data. The Classifier used is Random Forest. The Classifier's output predictions is found in Output/TestPredictions/test-predictions in the format: ", ACTUAL_LABEL, PREDICTED_LABEL".

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_Khan_A6". The directory contains a README file that explains how to build and run our code.

IMPLEMENTATION

1. The Whole Program Pipeline is divided into three parts, a MR Job for cleaning Train Data, a MR Job for cleaning Test Data, a Spark job for building a Prediction Model for Flight Delays We built a Random Forest Model and trained in with **15,355,412 Records** and tested with **5,227,680** Records. The Model is trained on all of the History Data and predicted on the test 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" as Key and a record of a csv file as Value. The record is checked for sanity tests and Feature Columns are sent to the Reducer as Values. Note: The choice of Carrier Code as key is done for node balance.It has no impact on the Prediction Model.
3. The reducer is a default reducer which outputs the values to the output file.
4. The output Training and Test files are consolidated and sent to the AWS Spark program as inputs along with the validation file.
5. The Spark program builds a random forest model on the training set and, after joining the test data with the validation data, predicts on the test data.It outputs a file containing records in the format: [id, actual label, predicted label].
6. The R script reads the Spark output file and calculates the confusion matrix and error rate.
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_Khan_A6
    |README.txt                          (Description of the Submission)
    |MakeFile                            (Makefile for the project)
    |Assignment6_Report.pdf              (PDF Report of the project results)
    |Assignment6_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*
        |a6history|(36 csv.gz files)*
        |a6test|98redacted.csv.gz*
        |a6validate|98validate.csv.gz*
    |output+
        |a6history+
        |a6test+
```

Install the required R Packages

```
install.packages("rmarkdown")
install.packages("ggplot2")
install.packages("caret")
```

Confusion Matrix

```
## Confusion Matrix and Statistics
##
##          Actual
## Predicted On-Time   Delay
##   On-Time 1098431  809764
##   Delay   1609402 1710083
##
##                Accuracy : 0.5372
##                  95% CI : (0.5368, 0.5377)
##     No Information Rate : 0.518
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.0834
##  Mcnemar's Test P-Value : < 2.2e-16
##
##             Sensitivity : 0.4056
##             Specificity : 0.6786
##          Pos Pred Value : 0.5756
##          Neg Pred Value : 0.5152
##              Prevalence : 0.5180
##          Detection Rate : 0.2101
##    Detection Prevalence : 0.3650
##       Balanced Accuracy : 0.5421
##
##        'Positive' Class : On-Time
##
```

Total Error Rate:

```
## [1] 91.5705
```

Feature Importances (in decreasing order of importance)

```
CrsArrGroup (Hour)                        -> 0.46266543076994343
CrsDepGroup (Hour)                        -> 0.20187488438819967
DayOfWeek                                 -> 0.25076367809990746
Popular Origin (1:Popular,0:Not Popular) -> 0.03667819335689688
DistanceGroup                             -> 0.032184771345197496
Quarter                                   -> 0.008941879005327542
Week ID (Day/7)                           -> 0.0030492211372830182
Month                                     -> 0.0022793325312848492
PopularDest (1:Popular,0:Not Popular)     -> 8.047333315582839E-4
FlightNum                                 -> 7.578760344012271E-4
```

Timings (All times are in minutes)

```
1. Local Machine
  1.1 Preprocessing Train and Test Data      8-9 minutes
  1.2 Spark ML                               7 minutes
2. AWS Cluster 10 CORE c1.medium machines
  2.1 Preprocessing Train and Test Data      8 minutes
  2.2 Spark ML                               22 minutes
3. AWS Cluster 3 CORE m3.xlarge machines
  2.1 Preprocessing Train and Test Data       4 minutes
  2.2 Spark ML                                7-14 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 caret
```

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"