Big Data Analytics Techniques and Applications

Homework 4

Name: 陳以瑄 Student ID:109705001

Q1: Show the predictive framework you designed.

- Problem definition

  Predicting whether the flight will have an “arrival delay ”.

- Framework

  Data labeling → Feature extraction → Preprocessing (standardization/ one hot encoding) → Modeling (using logistic regression algorithm)

- Data label

  Code:

```
[33]  1 total_df = df3.union(df4).union(df5)
      2 total_df = total_df.filter(total_df.Cancelled==0)
      3 total_df = total_df.withColumn("IsDelay", when((total_df.ArrDelay>0) , 1).otherwise(0))
```

  Since I only predict whether it has an “arrival delay ”, I use the value in “ArrDelay” for labeling. If the value is greater than 0, it has an “arrival delay ” and is labeled 1. If it is less than or equal to 0, it doesn’t have an “arrival delay ” and is labeled 0. (line 3)

  I also find out that when the flight was canceled, it will not have a record of “arrival delay ”.

ArrDelay	DepDelay	Origin	Dest	Distance	TaxiIn	TaxiOut	Cancelled	CancellationCode
null	null	BOS	IAD	413.0	0	0	1	NA
null	null	ORD	MSP	334.0	0	0	1	NA
null	null	TPA	ORD	1012.0	0	0	1	NA

  Thus, I only select those flights that were not canceled. (line 2)

- Features I extract

```
total_df = total_df.select("Year","DayOfWeek",\
                           "DepHour","ArrHour",\
                           "TailNum",\
                           "Origin","Dest",\
                           "Distance","DepDelay",
                           "IsDelay")
```

I use 〈DayOf Week, CRSDepTime, CRSArrTime, TailNum, Origin, Dest, Distance, DepDelay〉. Below is the reason why I use these features:

1. Date info

   I only extract the "DayOfWeek" column, because I believe whether the day is weekday or weekend might impact the probability of delay.

2. Time info

   I extract "CRSDepTime" and "CRSArrTime", but I modify them from hhmm to hh, since I believe that the hour info is much more important.

```
[9]  total_df = total_df.withColumn('ArrHour', floor(col('CRSArrTime') / 100).cast("integer"))
     total_df = total_df.withColumn('DepHour', floor(col('CRSDepTime') / 100).cast("integer"))
```

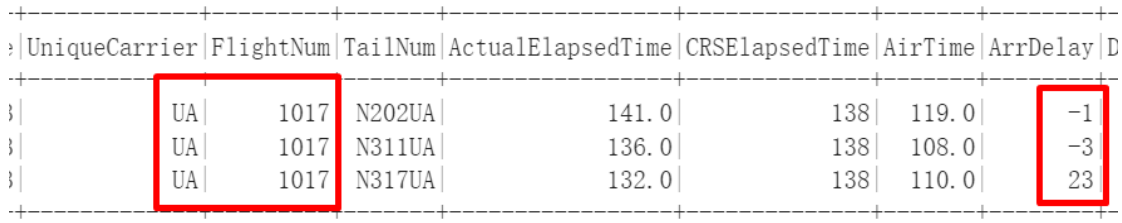
   Also, I believe that when doing prediction, it is proper to use the CRS data rather than actual data, because during the flight, we only know the CRS arrival time but don't know the actual arrival time.

   Besides, if I extract "CRSArrTime" and "ArrTime" at the same time, I will reveal the answer, since ArrDelay = ArrTime – CRSArrTime.

e	ArrTime	CRSArrTime	UniqueCarrier	FlightNum	TailNum	ActualElapsedTime	CRSElapsedTime	AirTime	ArrDelay	De
5	1912	1913	UA	1017	N202UA	141.0	138	119.0	-1	
5	1910	1913	UA	1017	N311UA	136.0	138	108.0	-3	
5	1936	1913	UA	1017	N317UA	132.0	138	110.0	23	

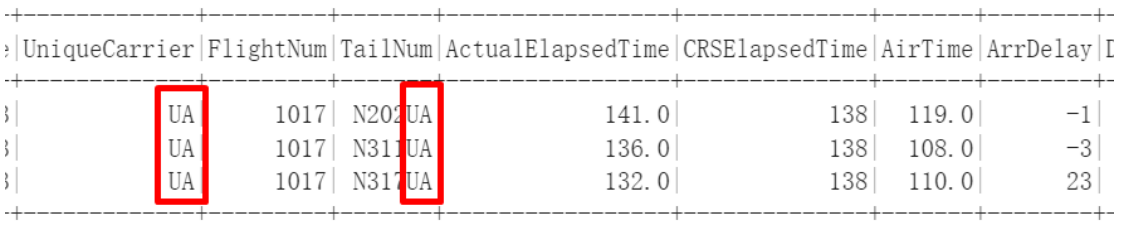
3. Flight info

   I extract the “TailNum” column only, since I believe that different airplanes might have different performances even if they are from the same company and share the same flight number.



UniqueCarrier	FlightNum	TailNum	ActualElapsedTime	CRSElapsedTime	AirTime	ArrDelay	D
UA	1017	N202UA	141.0	138	119.0	-1	
UA	1017	N311UA	136.0	138	108.0	-3	
UA	1017	N317UA	132.0	138	110.0	23	

   I also find out that the tail number was included in the carrier number.



UniqueCarrier	FlightNum	TailNum	ActualElapsedTime	CRSElapsedTime	AirTime	ArrDelay	I
UA	1017	N202UA	141.0	138	119.0	-1	
UA	1017	N311UA	136.0	138	108.0	-3	
UA	1017	N317UA	132.0	138	110.0	23	

4. Location info

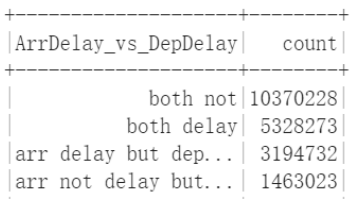
   I extract “Origin” and “Dest”, since some airports might be busier and cause flight delays.

   I also extract “Distance” because I believe that the longer the distance the higher probability to have a delay.

5. Delay info

   I extract the “DepDelay” column, since I believe that if the flight has a departure delay, its probability of arrival delay will be higher.

   Besides, the table below shows that the departure delay will not necessarily cause the arrival delay, so this feature doesn’t reveal the answer.



ArrDelay_vs_DepDelay	count
both not	10370228
both delay	5328273
arr delay but dep...	3194732
arr not delay but...	1463023

- Data preprocessing

1. Standardization

I scaled the features “Distance” and “DepDelay” into the same scale, 0~1, by using MinMaxScaler (line 6,7).

```
1 DepDelay_assembler = VectorAssembler().setInputCols(['DepDelay']).setOutputCol('DepDelay_vec')
2 Distance_assembler = VectorAssembler().setInputCols(['Distance']).setOutputCol('Distance_vec')
3 total_df = DepDelay_assembler.transform(total_df)
4 total_df = Distance_assembler.transform(total_df)
5 DepDelay_scaler = MinMaxScaler(inputCol="DepDelay_vec", outputCol="DepDelayStd")
6 Distance_scaler = MinMaxScaler(inputCol="Distance_vec", outputCol="DistanceStd")
7 total_df = DepDelay_scaler.fit(total_df).transform(total_df)
8 total_df = Distance_scaler.fit(total_df).transform(total_df)
```

2. One hot encoding

For categorical attributes, “DayOf Week”, “CRSDepTime”, “CRSArrTime”, “TailNum”, “Origin” and “Dest”, I convert them into binary vectors by using StringIndexer and OneHotEncoder.

```
1 DayOfWeek_indexer = StringIndexer(inputCol = "DayOfWeek",outputCol = "DayOfWeekIndex")
2 TailNum_indexer = StringIndexer(inputCol = "TailNum",outputCol = "TailNumIndex")
3 Dest_indexer = StringIndexer(inputCol = "Dest",outputCol = "DestIndex")
4 Origin_indexer = StringIndexer(inputCol = "Origin",outputCol = "OriginIndex")
5 DepHour_indexer = StringIndexer(inputCol = "DepHour",outputCol = "DepHourIndex")
6 ArrHour_indexer = StringIndexer(inputCol = "ArrHour",outputCol = "ArrHourIndex")
7
8 OHE = OneHotEncoder(inputCols =["DayOfWeekIndex","TailNumIndex","DestIndex","OriginIndex","DepHourIndex","ArrHourIndex"],\
9                      outputCols = ["DayOfWeekOHE","TailNumOHE","DestOHE","OriginOHE","DepHourOHE","ArrHourOHE"])
10
11 total_df = DayOfWeek_indexer.fit(total_df).transform(total_df)
12 total_df = TailNum_indexer.fit(total_df).transform(total_df)
13 total_df = Dest_indexer.fit(total_df).transform(total_df)
14 total_df = Origin_indexer.fit(total_df).transform(total_df)
15 total_df = DepHour_indexer.fit(total_df).transform(total_df)
16 total_df = ArrHour_indexer.fit(total_df).transform(total_df)
17
18 total_df = OHE.fit(total_df).transform(total_df)
```

Then, I assemble all features in a vector by using VectorAssembler.

```
1 featureAssembler = VectorAssembler(inputCols=["Year","DayOfWeekOHE",\
2                                               "DepHourOHE","ArrHourOHE",\
3                                               "TailNumOHE",\
4                                               "OriginOHE","DestOHE",\
5                                               "DistanceStd","DepDelayStd"\
6                                               ],outputCol="features")
7
8 feature_df = featureAssembler.transform(total_df)
9 final_total_df = feature_df.select(["Year","features","IsDelay"])
10 final_total_df = final_total_df.withColumnRenamed("IsDelay", "label")
```

After standardization and one hot encoding, it has 6762 features

```
+----+--------------------+-----+
|Year|            features|label|
+----+--------------------+-----+
|2003|(6762, [0, 4, 18, 32, ...|    0|
+----+--------------------+-----+
```

- Algorithm

  I use logistic regression for this classification problem.

```
lr = LogisticRegression(labelCol="label", featuresCol="features")
```

Q2: Explain the validation method you use.

I use the 5-fold method by CrossValidator.

```
lrParamGrid = (ParamGridBuilder()
               .addGrid(lr.regParam, [0.01, 0.1, 0.2])
               .addGrid(lr.elasticNetParam, [0, 0.25, 0.5, 0.75, 1.0])
               .addGrid(lr.maxIter, [5, 10, 15])
               .build())
```

```
lrEval = BinaryClassificationEvaluator(rawPredictionCol="rawPrediction")
```

```
lr_5fold = CrossValidator(estimator = lr,
                          estimatorParamMaps = lrParamGrid,
                          evaluator = lrEval,
                          numFolds = 5)
```

After training, I got the best hyperparameter:

regParam = 0.01, elasticNetParam = 0.25, maxIter = 10.

Q3: Explain the evaluation metric you use.

As the table below shows, the dataset is quite balanced.

```
1 total_df.groupBy("
```

```
+-------+--------+
|IsDelay|   count|
+-------+--------+
|      1| 8522973|
|      0|11872419|
+-------+--------+
```

```
1 test.groupBy("label")
```

```
+-----+-------+
|label|  count|
+-----+-------+
|    1|3063222|
|    0|3943644|
+-----+-------+
```

Therefore, I use accuracy as the evaluation metric. I also use confusion matrix for model evaluation.

```
1 lrPred = lrModel.transform(test)
```

```
1 y_true = lrPred.select("label")
2 y_true = y_true.toPandas()
3
4 y_pred = lrPred.select("prediction")
5 y_pred = y_pred.toPandas()
```

```
1 acc = accuracy_score(y_true, y_pred)
2 cnf_matrix = confusion_matrix(y_true, y_pred)
3 cnf_matrix
```

Q4: Show the validation results and give a summary of results.

- Accuracy

```
Accuracy: 0.7739
```

- Confusion Matrix

```
array([[3739276,  204368],
       [1379648, 1683574]])
```

	Predict not delay	Predict delay
Actual not delay	3739276	204368
Actual delay	1379648	1683574

- Summary

The accuracy is not quite well. As we can see in the confusion matrix, if the flight is actually not delayed, the model can predict well. However, as for

the flight being really delayed, the model only has a 55% chance

(= 1683574/3063222) to tell that this flight will delay.