Data 603 Project Presentation: How Weather Conditions affect Airline Delay

Amal Khan
Kam Hin Ho

Background

- Weather conditions are considered one of the number 1 reasons as to why airlines are delayed.
- “weather caused 69% of system impacting delays (> 15 minutes) over the six years from 2008 to 2013, as recorded in the OPSNET Standard "Delay by Cause" Reports.”
- Weather conditions are generally unpredictable, but geographical locations play a key role in affecting airline statuses.
- Historical datasets containing weather information prove to be useful in helping airline traffic planners to control and plan ahead.
- “As the weather impacts become longer lived and/or affect larger regions of the country, management of the demand must be planned strategically.”

Goal

- Using the weather data, we will train the weather dataset with the airline delays dataset to analyze **to and predict the airline delays with weather conditions.**

Dataset - Flight Data

- Retrieved the data set from: https://www.kaggle.com/usdot/flight-delays
- Referenced from United States Department of Transportation: https://www.transtats.bts.gov/Tables.asp?DB_ID=120&DB_Name=Airline%20On-Time%20Performance%20Data&DB_Short_Name=On-Time
- 2009-2018 datasets contain Airline On-Time Performance Data such as airline status, carrier information, destination, origin, and reasons of the delay etc.
- Specifically focused on the “Weather Delay” categorization, but the dataset didn’t contain information on the specific weather conditions causing airline delay

df_flight: pyspark.sql.dataframe.DataFrame

```
FL_DATE: string
OP_CARRIER: string
OP_CARRIER_FL_NUM: string
ORIGIN: string
DEST: string
CRS_DEP_TIME: string
DEP_TIME: string
DEP_DELAY: string
TAXI_OUT: string
WHEELS_OFF: string
WHEELS_ON: string
TAXI_IN: string
CRS_ARR_TIME: string
ARR_TIME: string
ARR_DELAY: string
CANCELLED: string
CANCELLATION_CODE: string
DIVERTED: string
CRS_ELAPSED_TIME: string
ACTUAL_ELAPSED_TIME: string
AIR_TIME: string
DISTANCE: string
CARRIER_DELAY: string
WEATHER_DELAY: string
NAS_DELAY: string
SECURITY_DELAY: string
LATE_AIRCRAFT_DELAY: string
```

CARRIER_DELAY	WEATHER_DELAY	NAS_DELAY	SECURITY_DELAY	LATE_AIRCRAFT_DEL
null	null	null	null	null
23.0	0.0	0.0	0.0	0.0
65.0	0.0	5.0	0.0	0.0
65.0	0.0	0.0	0.0	0.0
0.0	70.0	20.0	0.0	0.0
null	null	null	null	null
null	null	null	null	null

Dataset - Weather Data from NOAA

- Retrieved the data set from: https://www.ncdc.noaa.gov/cdo-web/
- National Centers for Environmental Information (NCEI) is the leading provider of weather and climate data
- Gathered weather data information from 2009-2018 from 17 different airports.
- Weather dataset contains various weather condition features such as: fog, thunder, hail, smoke haze, tornado, mist, high winds, rain, freezing rain, snow, and more

	Name
1	SEATTLE TACOMA AIRPORT, WA US
2	AUSTIN BERGSTROM INTERNATIONAL AIRPORT, TX US
3	MIAMI INTERNATIONAL AIRPORT, FL US
4	DENVER INTERNATIONAL AIRPORT, CO US
5	PHOENIX AIRPORT, AZ US
6	MADISON DANE CO REGIONAL AIRPORT, WI US
7	LOS ANGELES INTERNATIONAL AIRPORT, CA US
8	BALTIMORE WASHINGTON INTERNATIONAL AIRPORT, MD US
9	JFK INTERNATIONAL AIRPORT, NY US
10	CHICAGO OHARE INTERNATIONAL AIRPORT, IL US
11	ATLANTA HARTSFIELD JACKSON INTERNATIONAL AIRPORT, GA US
12	SAN FRANCISCO INTERNATIONAL AIRPORT, CA US
13	CHARLOTTE DOUGLAS AIRPORT, NC US
14	BOSTON, MA US
15	WASHINGTON DULLES INTERNATIONAL AIRPORT, VA US
16	DAL FTW WSCMO AIRPORT, TX US
17	NEWARK LIBERTY INTERNATIONAL AIRPORT, NJ US

Joining the Dataset

- In order to join both the weather dataset with the combined airline delays datasets:
  - we created a Airport Abbr column on the weather dataset using UDF method
  - Using inner join, we merged the datasets on Airport Abbreviations and Origin as well as the date column on both datasets

```
#Create an abbr dictionary for each airport for joinning the flight dataset later
abbr = {
    "SEATTLE TACOMA AIRPORT, WA US": "SEA",
    "MIAMI INTERNATIONAL AIRPORT, FL US": "MIA",
    "SAN FRANCISCO INTERNATIONAL AIRPORT, CA US": "SFO",
    "BALTIMORE WASHINGTON INTERNATIONAL AIRPORT, MD US":'BWI',
    'LOS ANGELES INTERNATIONAL AIRPORT, CA US':'LAX',
    'DAL FTW WSCMO AIRPORT, TX US':'DFW',
    "DENVER INTERNATIONAL AIRPORT, CO US": "DEN",
    "AUSTIN BERGSTROM INTERNATIONAL AIRPORT, TX US": "AUS",
    "PHOENIX AIRPORT, AZ US": "PHX",
    "MADISON DANE CO REGIONAL AIRPORT, WI US":'MSN',
    'BOSTON, MA US':'BOS',
    'WASHINGTON DULLES INTERNATIONAL AIRPORT, VA US':'IAD',
    'CHARLOTTE DOUGLAS AIRPORT, NC US':'CLT',
    "NEWARK LIBERTY INTERNATIONAL AIRPORT, NJ US": "EWR",
    "ATLANTA HARTSFIELD JACKSON INTERNATIONAL AIRPORT, GA US": "ATL",
    "JFK INTERNATIONAL AIRPORT, NY US": "JFK",
    "CHICAGO OHARE INTERNATIONAL AIRPORT, IL US":'ORD'
}
```

```
#Create a UDF that create a column according to the abbr dictionary
from pyspark.sql.functions import col, udf, StringType
udf_airport_to_abbr = udf(lambda airport: abbr[airport], StringType())
df_weather = df_weather.withColumn("Airport_abbr",udf_airport_to_abbr("Airport"))
display(df_weather)
```

```
# Verifying if the airport_abbr is mapped to the correct airport
display(df_weather.select('Airport_abbr','Airport').distinct())
```

▸ (2) Spark Jobs

	Airport_abbr	Airport
1	IAD	WASHINGTON DULLES INTERNATIONAL AIRPORT, VA US
2	MSN	MADISON DANE CO REGIONAL AIRPORT, WI US
3	SFO	SAN FRANCISCO INTERNATIONAL AIRPORT, CA US
4	BOS	BOSTON, MA US
5	BWI	BALTIMORE WASHINGTON INTERNATIONAL AIRPORT, MD US
6	EWR	NEWARK LIBERTY INTERNATIONAL AIRPORT, NJ US
7	CLT	CHARLOTTE DOUGLAS AIRPORT, NC US
8	DFW	DAL FTW WSCMO AIRPORT, TX US
9	PHX	PHOENIX AIRPORT, AZ US

```
#Joinning the Weather Data and the Flight Data
weather_flight_join_expr =
[df_weather['Airport_abbr']==df_flight['ORIGIN'],df_weather['Date']==df_flight['FL_DATE']]
df_wf_join = df_weather.join(df_flight, weather_flight_join_expr,'inner')
```

df_wf_join: pyspark.sql.dataframe.DataFrame

```
Airport: string
Date: string
TMAX: integer
TMIN: integer
TAVG: integer
Fog_IceFog: double
Heavyfog: double
Thunder: double
Ice_Pellets: double
Hail: double
Glaze: double
Dust_Volcanic_ash: double
Smoke_Haze: double
Blowing_Snow: double
Tornado: double
High_Winds: double
Mist: double
Drizzle: double
Freezing_Drizzle: double
Rain: double
Freezing_Rain: double
Snow: double
```

Data Preprocessing

- To successfully train the datasets using the models, we assigned 0's to all NULL values in the merged dataframe, and 1's indicate a weather feature has occurred
- We transformed the Categorical Columns "OP_CARRIER", "ORIGIN",'OP_CARRIER_FL_NUM' into vectors.
- Transform all feature values (rain, snow, fog, etc) into vectors, using VectorAssembler
- Created a new column in the dataframe classifying whether or not the airlines were delayed due to weather for that date
- Split the data into training and test set 70/30

```
### Randomly split data into training and test sets. set seed for reproducibility
(trainingData, testData) = dataset.randomSplit([0.7, 0.3], seed=100)
#print(trainingData.count())
#print(testData.count())
```

```
#categorical cols
from distutils.version import LooseVersion

categoricalColumns = ["OP_CARRIER", "ORIGIN",'OP_CARRIER_FL_NUM']
stages = [] # stages in our Pipeline
for categoricalCol in categoricalColumns:
    # Category Indexing with StringIndexer
    stringIndexer = StringIndexer(inputCol=categoricalCol, outputCol=categoricalCol + "Index")
    # Use OneHotEncoder to convert categorical variables into binary SparseVectors
    if LooseVersion(pyspark.__version__) < LooseVersion("3.0"):
        from pyspark.ml.feature import OneHotEncoderEstimator
        encoder = OneHotEncoderEstimator(inputCols=[stringIndexer.getOutputCol()], outputCols=[categoricalCol + "classVec"])
    else:
        from pyspark.ml.feature import OneHotEncoder
        encoder = OneHotEncoder(inputCols=[stringIndexer.getOutputCol()], outputCols=[categoricalCol + "classVec"])
    # Add stages.  These are not run here, but will run all at once later on.
    stages += [stringIndexer, encoder]
```

Why Logistic Regression?

- It makes no assumptions about distributions of classes in feature space.
- It also supports multiple features.
- It can interpret model coefficients as indicators of feature importance.

Why Random Forest ?

- Logistic Regression used for Binary Classification
- Utilized Decision Tree methods because they are easy to interpret, can handle continuous and categorical data effectively, helpful in large datasets, and not sensitive to outliers.
- Decision Tree methods are however prone to overfitting, Random Tree Forest proves to be useful in tackling this issue.
- Pros to Random Tree Forest:
    - Works well with non linear data,
    - Lower risk of overfitting
    - Runs effectively on large dataset (ours over 5MG)

- Con: can be biased when dealing with categorical variables.

Evaluation Result from Logistic Regression Model

- We use the `BinaryClassificationEvaluator` to evaluate our models, which uses areaUnderROC as the default metric.
- We created a Logistic Regression model with the training dataset to make predictions on the test dataset we created, resulting with the below:

```
1  from pyspark.ml.evaluation import BinaryClassificationEvaluator
2
3  # Evaluate model
4  evaluator = BinaryClassificationEvaluator(rawPredictionCol="rawPrediction")
5  evaluator.evaluate(predictions)
```

▸ (4) Spark Jobs

```
Out[35]: 0.8251220676389761
```

Evaluation Result from Random Forest Model

- We used the RandomForestClassifier to create the Random Forest model, training with the training data and making predictions using the test data, our result shown below:

Cmd 53

```
#We will evaluate our Random Forest model with BinaryClassificationEvaluator.
from pyspark.ml.evaluation import BinaryClassificationEvaluator

# Evaluate model
evaluator = BinaryClassificationEvaluator()
evaluator.evaluate(predictions)
```

▸ (4) Spark Jobs

```
Out[45]: 0.6268574076385202
```

Results/Conclusion

- According to the evaluations of both logistic regression and random forest models, we discovered that logistic regression has a better separability than the random forest model. Logistic regression AUC is 0.825, while random forest AUC is 0.626. In which, logistic regression has 82.5% chance that it will be able to distinguish between positive class and negative class, while random forest model only has 62.6%.
- One possible reason to why the Random Forest model didn’t work as effectively and strongly with our data is due to the fact that we had more categorical variables.

Thank You!