STEVENS
INSTITUTE of TECHNOLOGY
THE INNOVATION UNIVERSITY®

Flight Delay Prediction

BIA 678 Big Data Technologies Fall 2022

Prof. Seyed Mohammad Nikouei

Final Project Report

Submitted By

Aishwarya Pawar

Iswariya Natarajan

Poorna Vishwanath

Salil Maharjan

Index

Introduction and Motivation

As people increasingly choose to travel by air, the number of flights that fail to depart on time also increases. The airline industry experiences financial challenges because of this growth, which also worsens the overcrowding at airports. Delays in air travel are a sign of the aviation system's inefficiency. Both airline firms and their customers pay a hefty price for it. The Total Delay Impact Study estimated that the total cost of air transportation delays to passengers and the airline sector in the US was $22 billion, resulting in a $4 billion decline in GDP. As a result, anticipating delays can enhance airline operations and passenger pleasure, which will benefit the economy. The major objective of this study is to assess how well machine learning classification methods using PySpark perform when forecasting flight delays. For a three-year period from 2020 to 2022, data on flights departing from various airports was reviewed. The study used several algorithms, and its predictions were assessed using a variety of metrics**.**

Dataset

The dataset was gathered from http://www.kaggle.com/datasets. It contains 15,412,586 rows and 62 columns. The size of this dataset is 5.38 GB and consists of flight information, including airline-specific cancellations and delays, origin, destination, airport ID, and delay minutes, etc., There is five years' worth of flight data, in our project, we used data from the years 2020 to 2022.

Exploratory Data Analysis

Data Understanding

The initial step of EDA is to understand the data and choose what features to keep and drop for our final dataset, we run a high-level EDA (Exploratory Data Analysis) on the dataset. We plot some graphs to visualize feature distributions and identify relevant features. The following figure shows the most prominent features that we picked for predicting the delay. There are 23 columns that are informative and useful for this project.

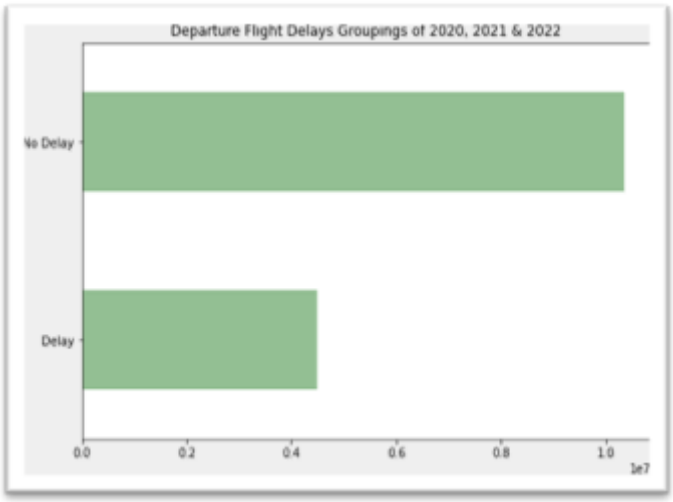
There was a total of 6852 columns in the dataset. However, most of them were identification numbers associated with airports with no association with flight delays. We manually filtered out the columns that would be helpful for our analysis.

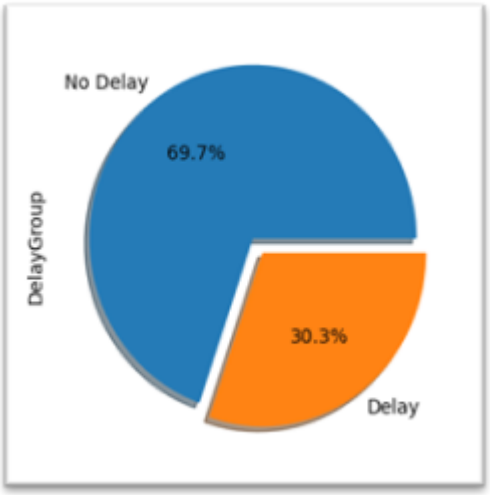
Data Processing & Cleaning

Most of the columns contained null values, we dropped the entire rows containing null values. The number of rows after dropping the null values is 14,843,716. The 'Flight Date' column is converted to DateTime datatype for visualization purposes. We created a new column called DelayStatus or DelayGroup where we group it into Delay or 1 and No delay or 0 when the delay minutes is not 0 and any other value apart from 0 respectively.

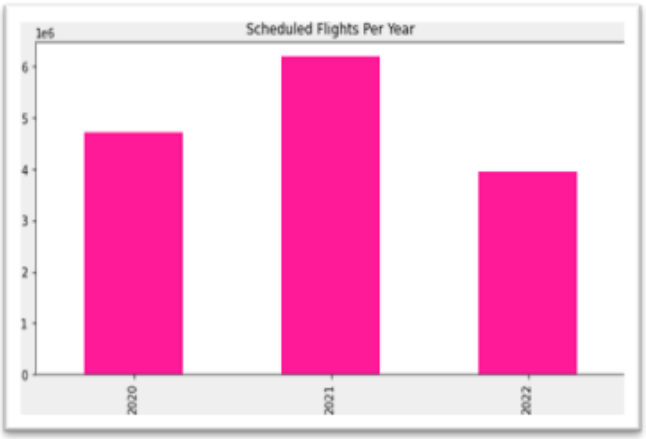
```
Data columns (total 23 columns):
 #   Column                          Dtype
---  ------                          -----
 0   FlightDate                      datetime64[ns]
 1   Airline                         object
 2   Tail_Number                     object
 3   Flight_Number_Marketing_Airline int64
 4   Origin                          object
 5   Dest                            object
 6   DayofMonth                      int64
 7   DayOfWeek                       int64
 8   CRSDepTime                      int64
 9   DepTime                         float64
 10  DepDelayMinutes                 float64
 11  OriginAirportID                 int64
 12  OriginCityName                  object
 13  OriginStateName                 object
 14  DestAirportID                   int64
 15  DestCityName                    object
 16  DestStateName                   object
 17  TaxiOut                         float64
 18  TaxiIn                          float64
 19  CRSArrTime                      int64
 20  DistanceGroup                   int64
 21  ArrTime                         float64
 22  ArrDelayMinutes                 float64
dtypes: datetime64[ns](1), float64(6), int64(8), object(8)
```

Data Visualization





Bar plot of Delay and No Delay, 69.7% of the data falls into No Delay category and 30.3% of data is under Delay group.

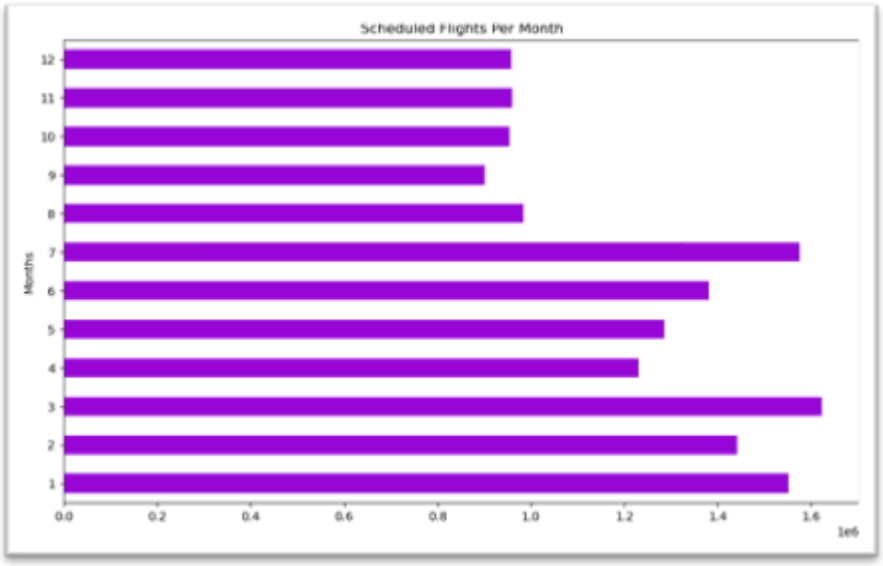


DelayGroup	Delay	No Delay
Year		
2020	19.072878	80.927122
2021	33.083856	66.916144
2022	39.328543	60.671457

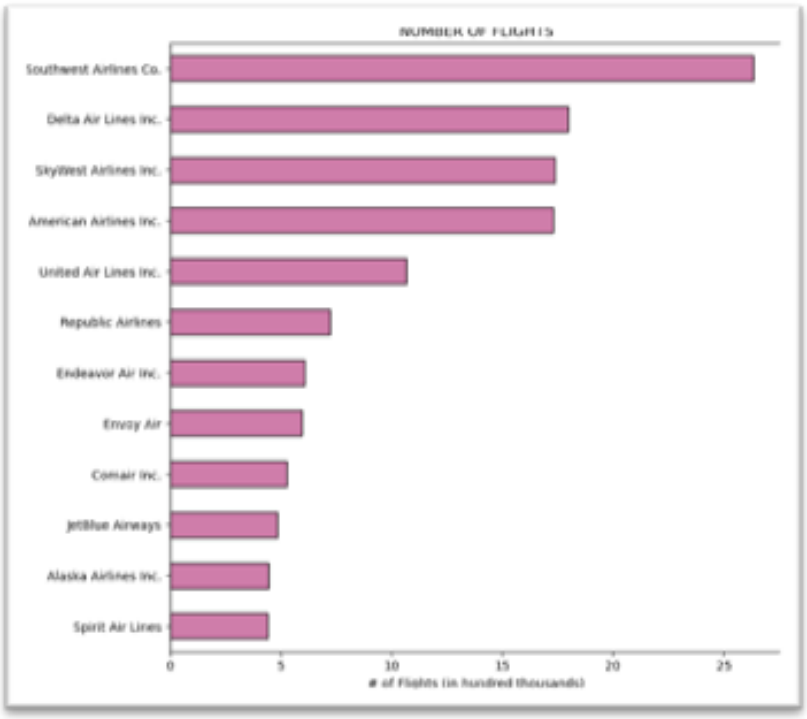
The Bar plot above shows the number of flights scheduled in each year, 2021 contains the maximum number of flights scheduled. The Data Frame shows the percentage of flights that falls into the Delay and No Delay Group each year. For instance, 19% of flights were delayed in 2020 and 80.92% of flights were on-time.

Bar chart below of flights scheduled in each month, March has the highest number of flights scheduled and September has the least.

The data frame shows the percentage of flights falling into delay and no delay in each month.



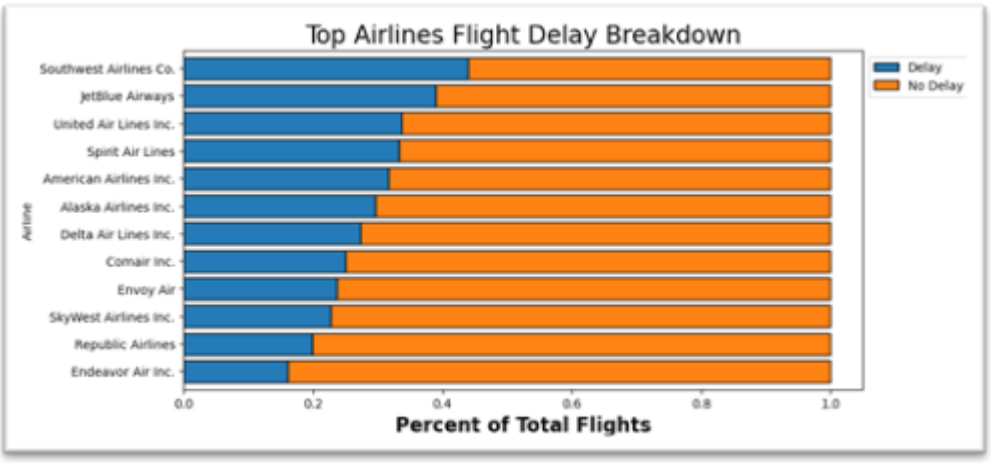
DelayGroup Month	Delay	No Delay
1	27.656822	72.343178
2	30.005483	69.994517
3	26.959488	73.040512
4	28.845255	71.154745
5	31.415105	68.584895
6	37.987099	62.012901
7	36.234604	63.765396
8	30.025918	69.974082
9	23.538436	76.461564
10	28.303284	71.696716
11	25.919694	74.080306
12	33.156424	66.843576



The count of values of the top 12 airlines in the dataset, Southwest Airlines has the maximum count.

Airline	mean	count	max	min
Spirit Air Lines	13.548411	441939.000000	1503.000000	0.000000
Alaska Airlines Inc.	8.123012	443191.000000	938.000000	0.000000
JetBlue Airways	19.475625	482978.000000	2052.000000	0.000000
Comair Inc.	11.861419	528072.000000	1919.000000	0.000000
Envoy Air	8.877991	595782.000000	5327.000000	0.000000
Endeavor Air Inc.	7.717754	606234.000000	2579.000000	0.000000
Republic Airlines	9.533838	721305.000000	7223.000000	0.000000
United Air Lines Inc.	12.089327	1067746.000000	1581.000000	0.000000
American Airlines Inc.	13.210587	1728311.000000	3890.000000	0.000000
SkyWest Airlines Inc.	12.405606	1738874.000000	2366.000000	0.000000
Delta Air Lines Inc.	9.173577	1795275.000000	1287.000000	0.000000
Southwest Airlines Co.	11.508755	2633420.000000	747.000000	0.000000

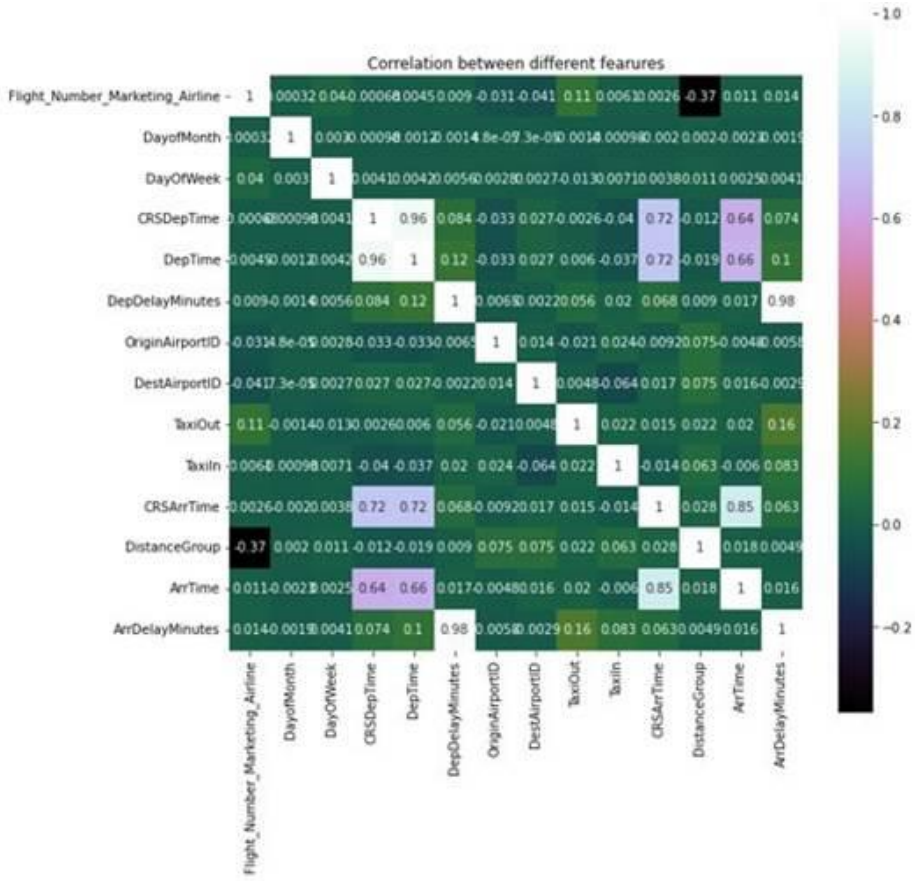
Statistics of top Airlines in terms of mean, count, max and min



Stacked bar chart of top airlines with respect to the percentage of total flights falling under delay or no delay in descending order of delay.

Data Correlation

We calculated correlation between each feature and dropped highly correlated variables such as ‘DepDelayMinutes’ and ‘ArrDelayMinutes’. We then had the following selected columns for future analysis.



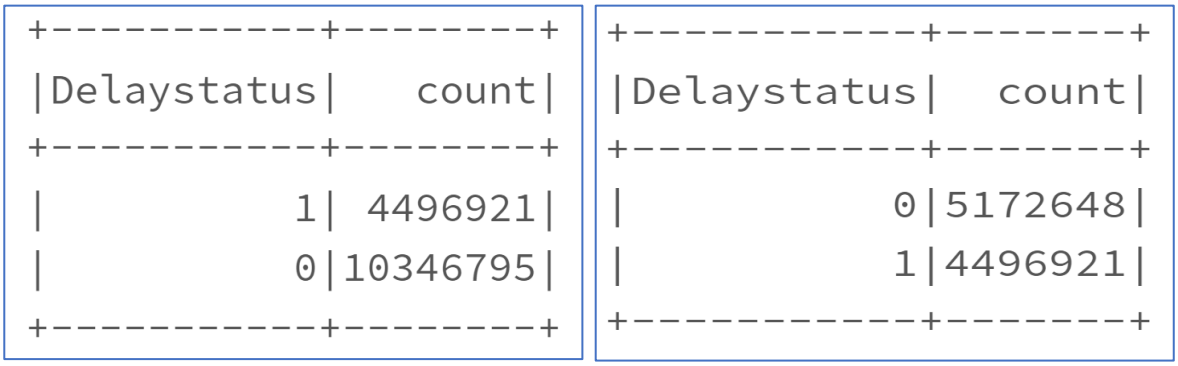

```
root
 |-- label: double (nullable = false)
 |-- features: vector (nullable = true)
 |-- Airline: string (nullable = true)
 |-- Origin: string (nullable = true)
 |-- Dest: string (nullable = true)
 |-- DayofMonth: long (nullable = true)
 |-- DayOfWeek: long (nullable = true)
 |-- CRSDepTime: long (nullable = true)
 |-- OriginCityName: string (nullable = true)
 |-- OriginStateName: string (nullable = true)
 |-- DestCityName: string (nullable = true)
 |-- DestStateName: string (nullable = true)
 |-- DistanceGroup: long (nullable = true)
 |-- DepTime: double (nullable = true)
 |-- ArrTime: double (nullable = true)
 |-- ArrDelayMinutes: double (nullable = true)
 |-- Delaystatus: integer (nullable = false)
```

We performed OneHotEncoding for the categorical features in our dataset. One hot encoding is a technique used in machine learning to convert categorical data into a numerical format that can be processed by a model. It involves creating a new binary column for each category in the original data, with a value of 1 indicating that the sample belongs to that category and a value of 0 indicating that it does not. One hot encoding is often used to handle categorical data in machine learning to improve the performance of the models.

Methodology

Tackling Data Imbalance

The original dataset was highly imbalanced as it had 10346796 data entries for aircraft without delay and 4496921 data entries for aircraft with delay. If we train a binary classification model without fixing this problem, the model will be completely biased. Hence, we resample the dataset using the under-sampling technique, after under-sampling the dataset was balanced with 4496921 data entries for – 'delayed' and 5172648 data entries for – 'not delayed'. Here since we already have sufficient data and to improve the run time and avoid memory issues we go with the under-sampled dataset.



Delaystatus	count
1	4496921
0	10346795

Delaystatus	count
0	5172648
1	4496921

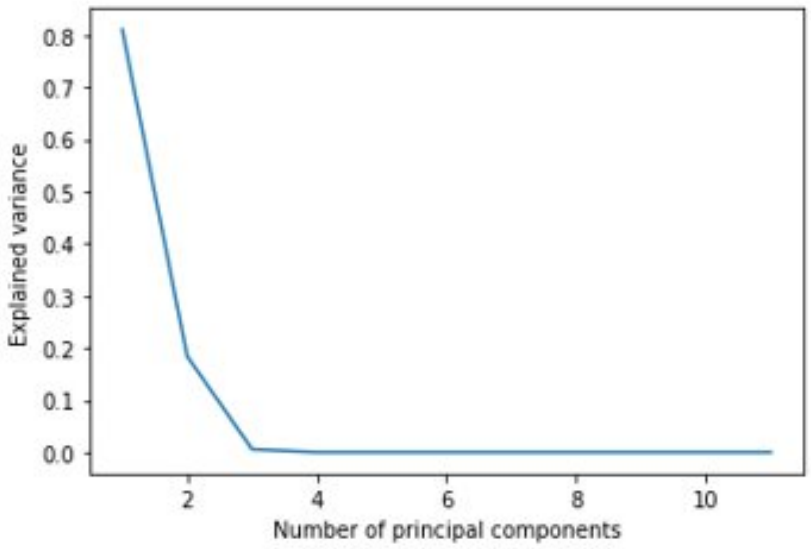
PCA

Principal component analysis (PCA) is a technique for reducing the dimensionality of such datasets, increasing interpretability but at the same time minimizing information loss. It does so by creating new uncorrelated variables that successively maximize variance. It helps in the reduction of noise in the data, further helps in feature selection, and the ability to produce independent, uncorrelated features of the data.

This value of K is often chosen based on the explained variance of the principal components. Before using PCA, we scaled the numeric data using the StandardScalar class to reduce the variance in the data to select the proper principal components. We used explained Variance attribute of the PCA object, which returns the explained variance of each principal component. We then plot the explained variance or cumulative explained variance as a function of the number of principal components to identify the point where the explained

variance starts to plateau or saturate. This point can be used to determine the optimal value of k. It is usually known as the elbow method.

Using this method, we picked out three principal components to use with our models.



Vector Assembler

A vector Assembler is a transformer that turns a list of columns into a single vector column. It is used to train ML models such as logistic regression and decision trees by combining raw features and features generated by different feature transformers into a single feature vector. The values of the input columns will be concatenated into a vector in the specified order in each row. Since our data has both string and numerical values we also use StringIndexer and OneHotEncoder. StringIndexer encodes a string column of labels to a column of label indices. By default, this is ordered by label frequencies, so the most frequent label gets index 0. A one-hot encoder that maps a column of category indices to a column of binary vectors, with at most a single one-value per row that indicates the input category index.

	0	1	2	3	4
label	0.0	0.0	0.0	0.0	0.0
features	(0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, ...	(0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, ...	(0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, ...	(0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, ...	(0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, ...
Airline	Comair Inc.	Comair Inc.	Comair Inc.	Comair Inc.	Comair Inc.
Origin	PHL	PHL	PHL	PHL	PHL
Dest	DAY	DAY	DAY	DAY	DAY
DayofMonth	1	4	6	11	13
DayOfWeek	2	5	7	5	7
CRSDepTime	1905	1905	1905	1800	1800
OriginCityName	Philadelphia, PA	Philadelphia, PA	Philadelphia, PA	Philadelphia, PA	Philadelphia, PA
OriginStateName	Pennsylvania	Pennsylvania	Pennsylvania	Pennsylvania	Pennsylvania
DestCityName	Dayton, OH	Dayton, OH	Dayton, OH	Dayton, OH	Dayton, OH
DestStateName	Ohio	Ohio	Ohio	Ohio	Ohio
DistanceGroup	2	2	2	2	2
DepTime	1858.0	1857.0	1853.0	1751.0	1755.0
ArrTime	2034.0	2027.0	2023.0	1930.0	1926.0
ArrDelayMinutes	0.0	0.0	0.0	0.0	0.0
Delaystatus	0	0	0	0	0

Model Implementation

Logistic regression:

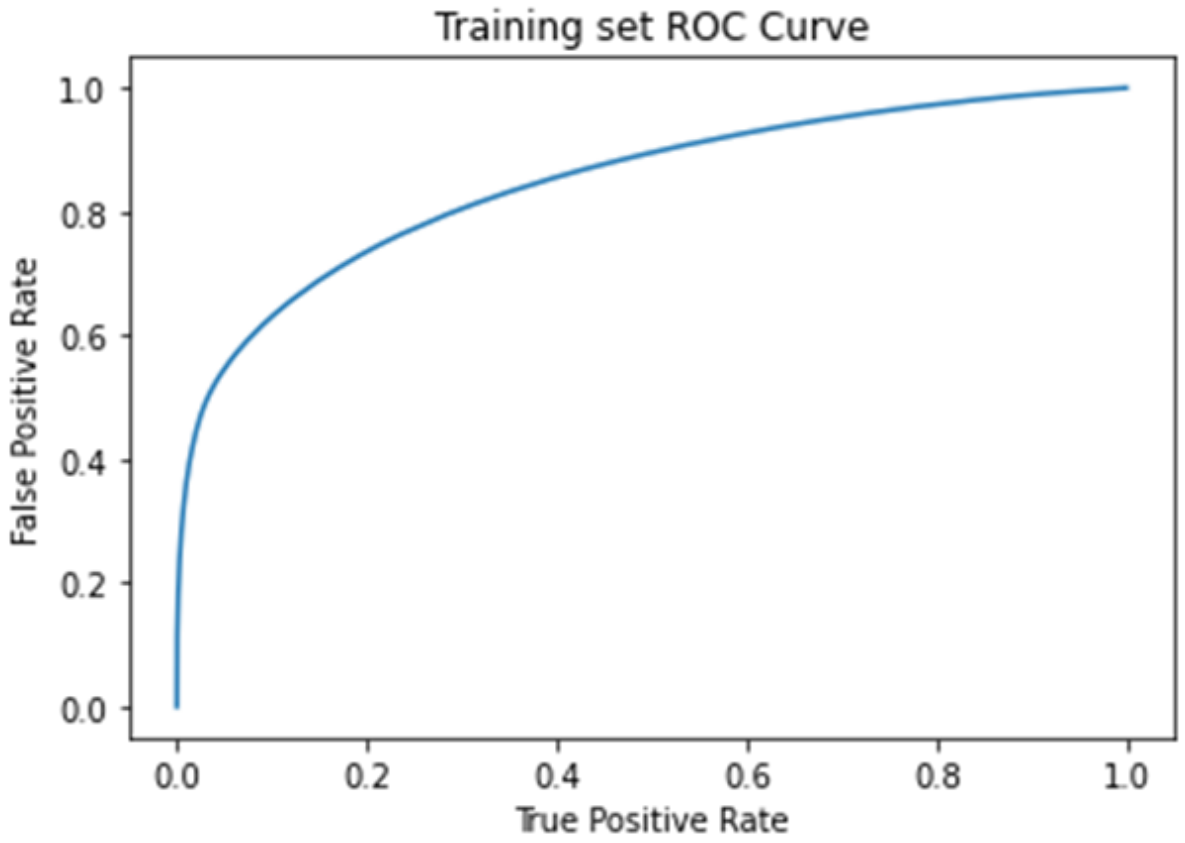
We utilize the method of logistic regression which is a supervised learning method to predict a delay in the departure of aircraft. This technique is extremely useful for predicting the likelihood of an event occurring and, more importantly, for determining probabilities between two distinct classes. The most common logistic regression models have a binary outcome, which can take two values like true/false, yes/no, and so on. Also, Logistic regression is a useful analysis method for classification problems, where we are trying to determine if a new sample fits best into a category. The logistic model (or logit model) is a statistical model that models the probability of an event taking place by having the log odds for the event be a linear combination of one or more independent variables. For our project, the Logistic regression method was able to achieve about 77 percent accuracy in predicting whether a given aircraft would be delayed or not based on the training using past data.

```
from pyspark.ml.classification import LogisticRegression
lr = LogisticRegression(featuresCol = 'features', labelCol = 'label', maxIter=10)
lrModel = lr.fit(train)
```

```
# Predictions:
predictions = lrModel.transform(test)
```

```
from pyspark.ml.evaluation import BinaryClassificationEvaluator
evaluator = BinaryClassificationEvaluator()
print('Test Area Under ROC', evaluator.evaluate(predictions))
```

```
Test Area Under ROC 0.8470263170797889
```



Random Forests:

Random forests or random decision forests are an ensemble learning method for classification, regression, and other tasks that operates by constructing a multitude of decision trees at training time. For classification tasks, the output of the random forest is the class selected by most trees. Random decision forests correct for decision trees' habit of

overfitting to their training set. Random forests outperform decision trees, but their accuracy is lower than gradient-boosted trees. However, data characteristics can affect their performance.

```
from pyspark.ml.classification import RandomForestClassifier

rf = RandomForestClassifier(featuresCol = 'features', labelCol = 'label')
rfModel = rf.fit(train)
predictions = rfModel.transform(test)
```

▸ (8) Spark Jobs

Command took 1.38 hours -- by apawar3@stevens.edu at 12/11/2022, 6:10:39 PM on project

Cmd 36

```
from pyspark.ml.evaluation import MulticlassClassificationEvaluator

evaluator = MulticlassClassificationEvaluator(labelCol="label", predictionCol="prediction")
accuracy = evaluator.evaluate(predictions)
print("Accuracy = %s" % (accuracy))
print("Test Error = %s" % (1.0 - accuracy))

```

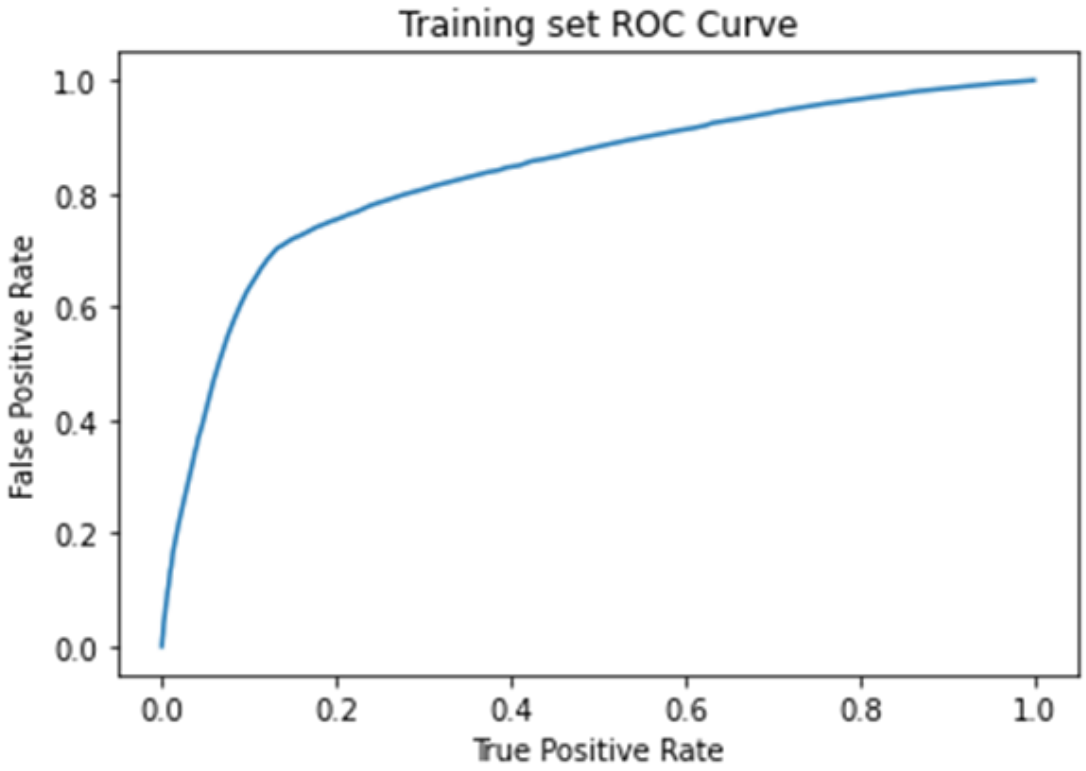
▸ (1) Spark Jobs

```
Accuracy = 0.7653496033239762
Test Error = 0.23465039667602383
```

```
from pyspark.ml.evaluation import BinaryClassificationEvaluator
evaluator = BinaryClassificationEvaluator()
print('Test Area Under ROC', evaluator.evaluate(predictions))
```

▸ (4) Spark Jobs

```
Test Area Under ROC 0.8329663383971353
```



Gradient Boosted Trees:

Gradient Boosted Trees and Random Forests are both ensembling methods that perform regression or classification by combining the outputs from individual trees. They both combine many decision trees to reduce the risk of overfitting that each individual tree faces. However, they differ in the way the individual trees are built, and the way the results are combined. Random forests use bagging to build independent decision trees and combine them in parallel. On the other hand, gradient-boosted trees use a method called **boosting**. Boosting combines weak learners (usually decision trees with only one split, called decision stumps) sequentially so that each new tree corrects the errors of the previous one.

We obtained the best score using Gradient Boosted Trees with an AUC score of 85% and an accuracy of 81%.

Cmd 50

```
from pyspark.ml.classification import GBTClassifier

# Create a GBTClassifier instance
gbt = GBTClassifier(maxIter=5, maxDepth=3)

# Train the model
model = gbt.fit(train)

# Make predictions on the test set

```

▸ (18) Spark Jobs

Command took 1.01 hours -- by smaharj3@ramapo.edu at 12/16/2022, 10:21:03 AM on gbt_final

Cmd 51

```
# Make predictions on the test set
predictions = model.transform(test)

from pyspark.ml.evaluation import BinaryClassificationEvaluator
evaluator = BinaryClassificationEvaluator()
print('Test Area Under ROC', evaluator.evaluate(predictions))
```

▸ (4) Spark Jobs

```
Test Area Under ROC 0.8587576200775682
```

Python

```
#select only prediction and label columns
preds_and_labels = predictions.select(['prediction','label'])
from pyspark.mllib.evaluation import MulticlassMetrics
metrics = MulticlassMetrics(preds_and_labels.rdd.map(tuple))

accuracy = predictions.filter(predictions.Delaystatus == predictions.prediction).count() /
float(predictions.count())
print("Accuracy : ",accuracy)
```

▸ (5) Spark Jobs

```
Accuracy :  0.8140681574975528
```

Precision Recall and F1 Score

Cmd 33

```
precision = metrics.precision(label=1)
recall = metrics.recall(label=1)
FScore = metrics.fMeasure(1.0,1.0)
print("Precision = %s" % precision)
print("Recall = %s" % recall)
print("F score = %s" % FScore)
```

```
Precision = 0.8353950205558803
Recall = 0.6430534819693131
F score = 0.7267127341544053
```

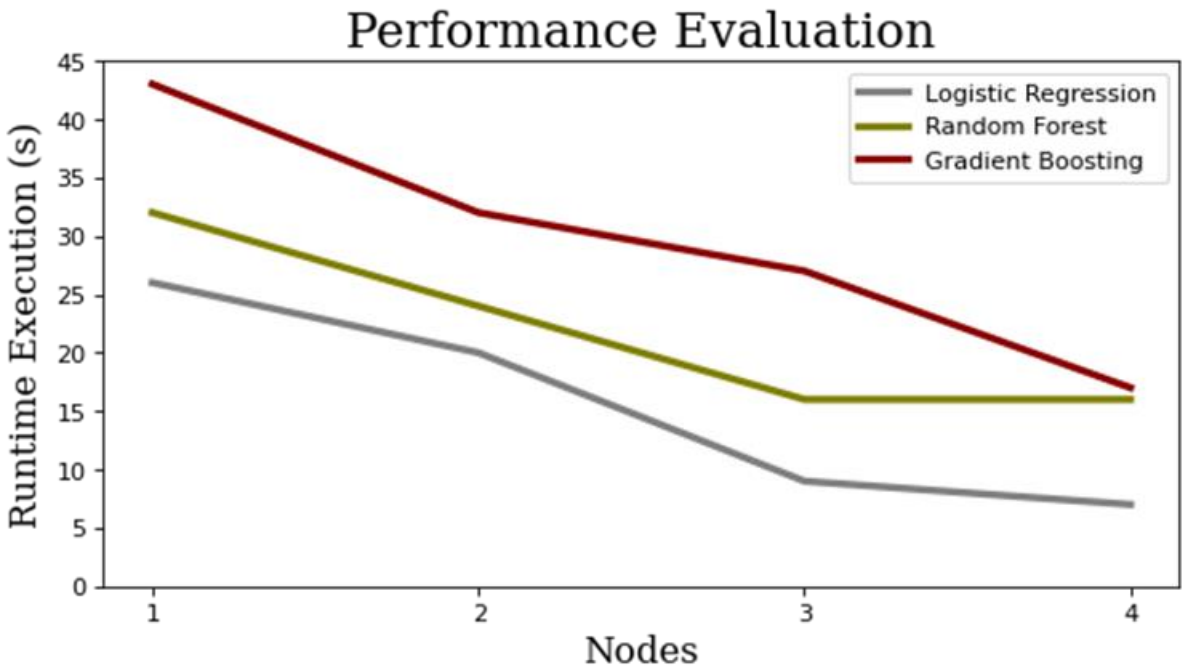
Command took 0.03 seconds -- by iramanat@stevens.edu at 16/12/2022, 3:14:52 PM on cluster 5

AWS

Amazon Simple Storage Service (Amazon S3) is an object storage service offering industry-leading scalability, data availability, security, and performance. Customers of all sizes and industries can store and protect any amount of data for any use case, such as data lakes, cloud-native applications, and mobile apps. With cost-effective storage classes and easy-to-use management features, you can optimize costs, organize data, and configure fine-tuned access controls to meet specific business, organizational, and compliance requirements.

Performance Evaluation

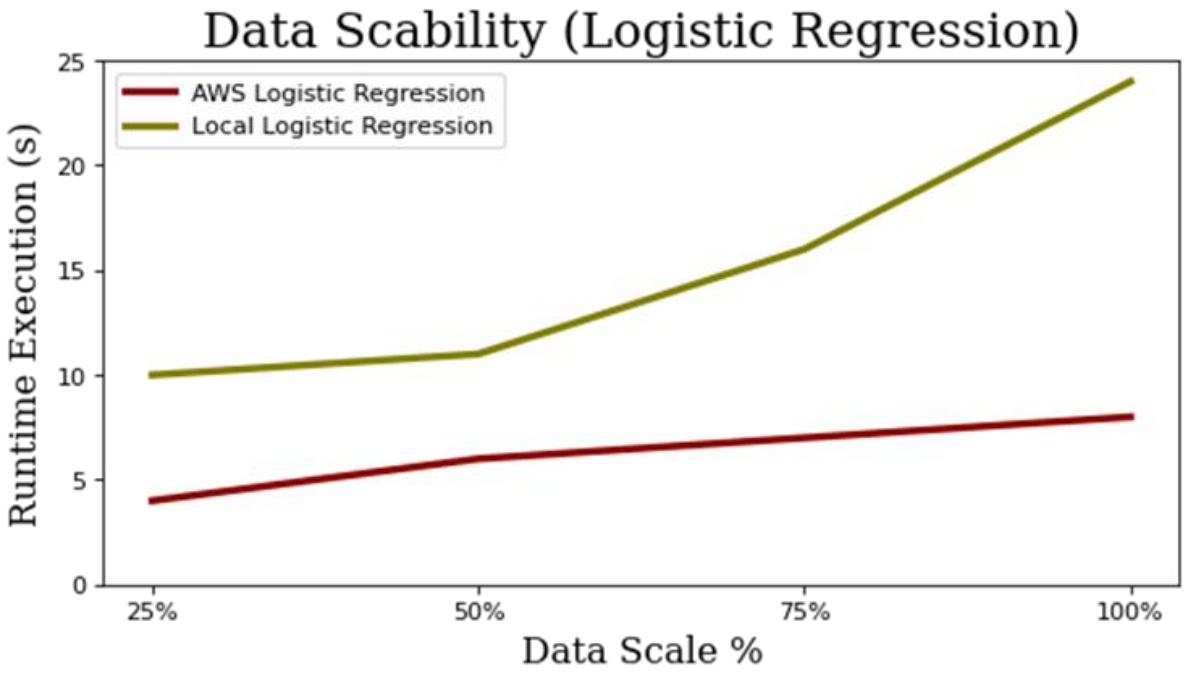
We utilized Amazon Web Services (AWS) and Amazon EMR & S3 to run these models on clusters to evaluate the performance of all three methods on a balanced dataset. Here, we were able to gauge the scalability and runtime of these three models. Our initial test was comparing the runtime to the quantity of nodes and if the models were being executed locally. According to our findings, the Logistic regression model performed the three algorithms' operations on a local computer and in clusters with 2, 3, and 4 nodes the quickest.

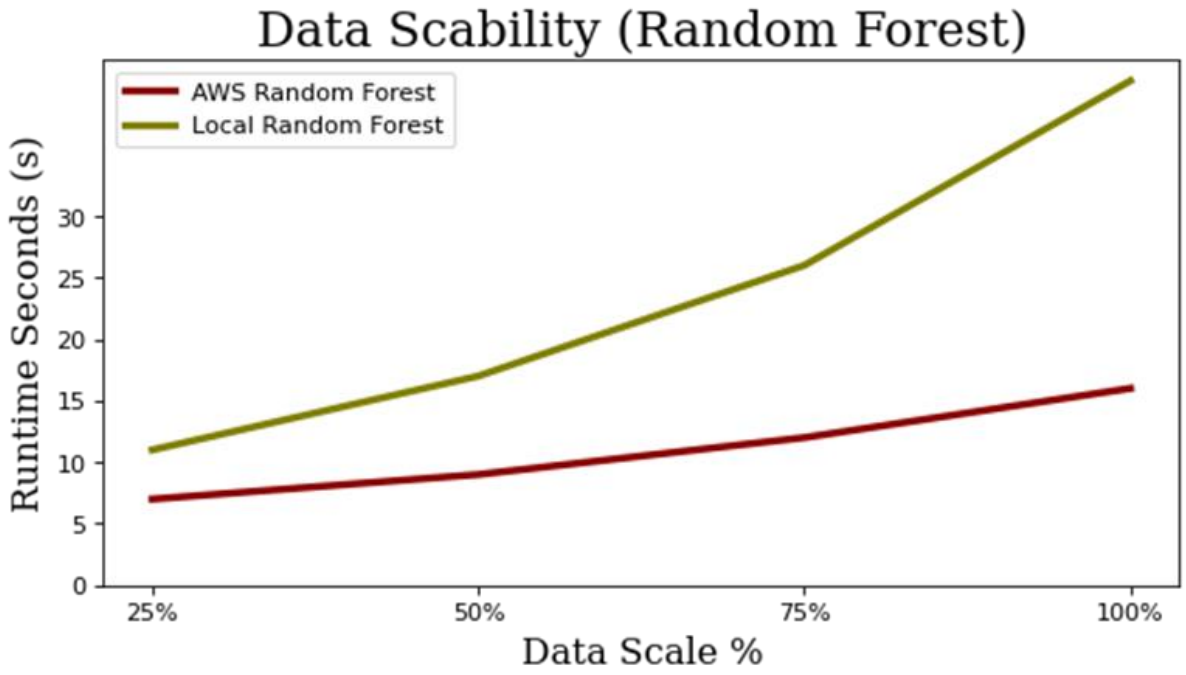


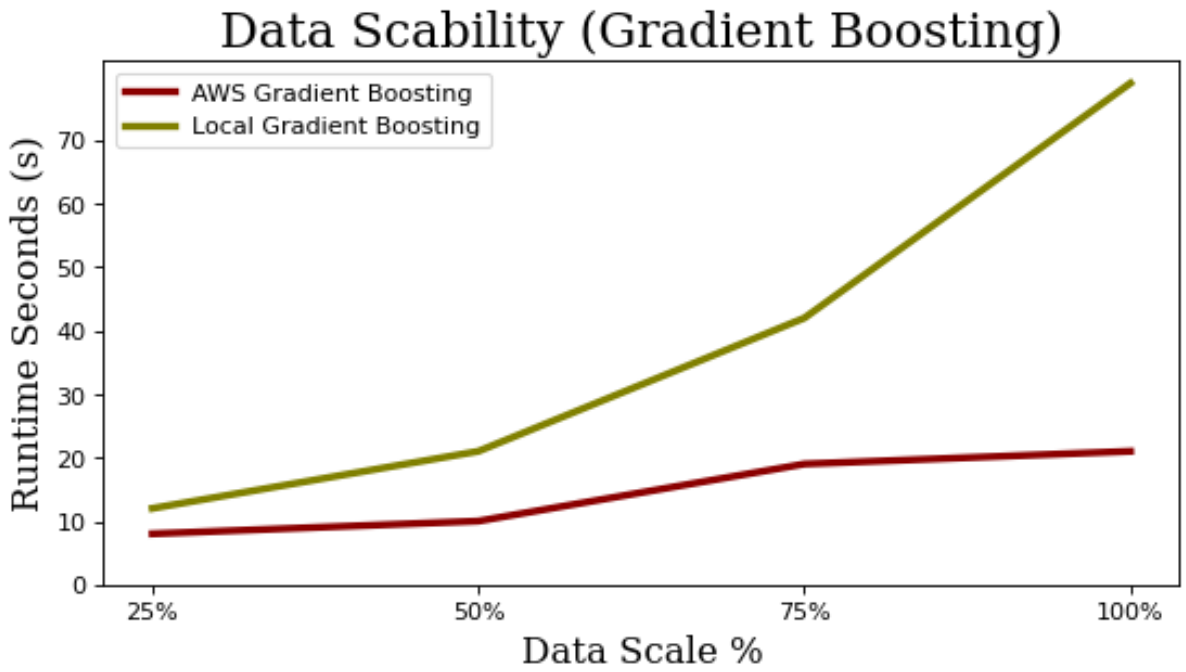
This can be seen from the results in the figure above. The Gradient Boosting Tree requires the most time to run locally and on clusters with nodes ranging from 2 to 4, and runtime execution slows down as the number of nodes increases. The Random Forest Classifier, for example, consists of several decision trees that take their time to compute and predict an outcome, after which the majority voting is computed. A similar justification is given for the Gradient Boosting Tree algorithm, as it adds weak learners to make predictions. These ensemble models allow the Random Forest and Gradient Boosting Tree algorithms to perform slower than the Logistic Regression algorithm.

The models' execution at runtime in comparison to the dataset's scalability is the following evaluation factor. The time it took for the models to run on 25%, 50%, 75%, and 100% of the dataset was evaluated on an AWS EMR cluster with four nodes.

The examination of the scalability for all three models revealed that as the data scale rose, it took longer to run these models, with intervals of 25% in the rise of data.


Data Scability (Logistic Regression)
AWS Logistic Regression
Local Logistic Regression
Runtime Execution (s)
0
5
10
15
20
25
25%
50%
75%
100%
Data Scale %


Data Scability (Random Forest)
AWS Random Forest
Local Random Forest
Runtime Seconds (s)
0
5
10
15
20
25
30
25%
50%
75%
100%
Data Scale %



All three of these models required more time to execute locally than they did on AWS. Thus, this demonstrates an increase in value of running these three machine learning algorithms on AWS since the runtime for all three models drop dramatically. If this model is implemented, this can help a business reduce expenses in the future.

Conclusion

The data was successfully analysed and visualized, which can be seen in the Explanatory Data Analysis section. Using careful feature selection, PCA, and properly handling the various features we were able to successfully train three different models on the data.

From our project, we can conclude that we have learned the following:

- Successfully able to use PySpark to analyze the Flights Delay dataset.
- Implemented SMOTE function to under-sample the dataset which lead to better results. As we got higher precision score using this balanced dataset.
- Implemented Logistic Regression, Random Forest Classifier, and Gradient Boosting Tree with an accuracy of around 76%-81 on the three models.
- It can be concluded that it is better to run this PySpark script on AWS as the runtime execution is faster as the number of nodes increase.

- As the scale of the dataset increases the runtime execution of the model is faster than the script being ran locally.

Future Scope

- There is scope for future work to be conducted as the models can be evaluated on multiple platforms such as Microsoft Azure or Google Platform.
- Due to the large number of stochastic variables involved, accurate flight departure delay prediction is particularly challenging (e.g., weather conditions, air traffic control).
- Adding more time series data such as time of the year and weather data will help improve the accuracy. Unfortunately, these features are not a part of the current dataset and hence are out of scope for this current project.
- Furthermore, while we tried hyperparameter tuning, it was beyond the scope of this project. Therefore, we can also consider hyperparameter tuning to improve the performance of the described models as well.

References

[1] A. B. Guy, "Flight delays cost $32.9 billion, passengers foot half the bill". [Online] Available: https://news.berkeley.edu/2010/10/18/flight_delays/3/. [Accessed in June 2017].

[2] M. Abdel-Aty, C. Lee, Y. Bai, X. Li and M. Michalak, "Detecting periodic patterns of arrival delay", Journal of Air Transport Management, Volume 13(6), pp. 355– 361, November 2007.

[3] S. AhmadBeygi, A. Cohn and M. Lapp, "Decreasing Airline Delay Propagation by Re-Allocating Scheduled Slack", Annual Conference, Boston, 2008.

[4] A. A. Simmons, "Flight Delay Forecast due to Weather Using Data Mining", M.S. Disseration, University of the Basque Country, Department of Computer Science, 2015.

[5] S. Choi, Y. J. Kim, S. Briceno and D. Mavris, "Prediction of weather-induced airline delays based on machine learning algorithms", Digital Avionics Systems Conference (DASC), 2016 IEEE/AIAA 35th, Sacramento, CA, USA, 2016.

CODE

```
# Databricks notebook source
import numpy as np
import pandas as pd
import os

import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns

import pyspark
from pyspark.sql import functions as f
from pyspark.sql.functions import when
from pyspark.sql import SQLContext
from pyspark.sql import SparkSession
from pyspark.sql.types import StructType, StructField, IntegerType, StringType

# COMMAND ----------

#These are the 3 parquet files of the years 2020 to 2021 that we are using for the project.
#dbfs:/FileStore/shared_uploads/iramanat@stevens.edu/Combined_Flights_2020.parquet
#dbfs:/FileStore/shared_uploads/iramanat@stevens.edu/Combined_Flights_2021.parquet
#dbfs:/FileStore/shared_uploads/iramanat@stevens.edu/Combined_Flights_2022.parquet

dbfs:/FileStore/shared_uploads/apawar3@stevens.edu/Combined_Flights_2020.parquet
dbfs:/FileStore/shared_uploads/apawar3@stevens.edu/Combined_Flights_2021.parquet
dbfs:/FileStore/shared_uploads/apawar3@stevens.edu/Combined_Flights_2022.parquet

# COMMAND ----------

data = spark.read.option("header",
"true").parquet(f"dbfs:/FileStore/shared_uploads/apawar3@stevens.edu/*.parquet")

# COMMAND ----------

#Creating a table to perform sql queries
data.createOrReplaceTempView("parquetTable")
data.printSchema()
data.show(truncate=False)

# COMMAND ----------

#Extracting features as per our requirements
df = sqlContext.sql("SELECT FlightDate, Airline, Origin, Dest, Diverted,Cancelled,
DayofMonth,DayOfWeek,CRSDepTime, DepTime, DepDelayMinutes, OriginCityName, OriginStateName,
DestCityName, DestStateName,DistanceGroup, ArrTime, ArrDelayMinutes FROM parquetTable")
```

```
# COMMAND ----------

#Dropping Null values from the rows and columns
flights_df = df.na.drop()

# COMMAND ----------

flights_df = flights_df.withColumn("DelayGroup",\
      when((flights_df.DepDelayMinutes == 0), "OnTime_Early")\
      .when((flights_df.DepDelayMinutes > 0) & (flights_df.DepDelayMinutes <= 15), "Small_Delay")\
      .when((flights_df.DepDelayMinutes > 15) & (flights_df.DepDelayMinutes <= 45), "Medium_Delay")\
      .otherwise("Large_Delay"))

# COMMAND ----------

flights_df.select(flights_df["DelayGroup"],flights_df["DepDelayMinutes"]).show()

# COMMAND ----------

flights_df_new = flights_df.withColumn("Delaystatus",\
      when((flights_df.DepDelayMinutes == 0), 0)\
      .otherwise(1))

# COMMAND ----------

spark.conf.set("spark.sql.execution.arrow.enabled", "true")

# Enable Arrow-based columnar data transfers
spark.conf.set("spark.sql.execution.arrow.pyspark.enabled", "true")


# COMMAND ----------

# Balanced/Imbalanced dataset:
flights_df_new.groupBy("Delaystatus").count().show()

# COMMAND ----------

#undersampling
from pyspark.sql.functions import col, explode, array, lit
major_df = flights_df_new.filter(col("Delaystatus") == 0)
minor_df = flights_df_new.filter(col("Delaystatus") == 1)
ratio = int(major_df.count()/minor_df.count())
print("ratio: {}".format(ratio))
sampled_majority_df = major_df.sample(False, 1/ratio)
combined_df_2 = sampled_majority_df.unionAll(minor_df)
combined_df_2.show()
combined_df_2.count()
```

```
# COMMAND ----------

from pyspark.sql.functions import col, explode, array, lit
df = spark.createDataFrame([['a',1],['b',1],['c',1],['d',1], ['e',1], ['f',1], ['x', 0], ['y', 0]], ['feature', 'label'])
df.show()
major_df = df.filter(col("label") == 1)
minor_df = df.filter(col("label") == 0)
ratio = int(major_df.count()/minor_df.count())
print("ratio: {}".format(ratio))combined_df_2.groupBy("Delaystatus").count().show()

# COMMAND ----------

#Oversampling
from pyspark.sql.functions import col, explode, array, lit
df = spark.createDataFrame([['a',1],['b',1],['c',1],['d',1], ['e',1], ['f',1], ['x', 0], ['y', 0]], ['feature', 'label'])
df.show()
major_df = df.filter(col("label") == 1)
minor_df = df.filter(col("label") == 0)
ratio = int(major_df.count()/minor_df.count())
print("ratio: {}".format(ratio))combined_df.groupBy("Delaystatus").count().show()

# COMMAND ----------

combined_df.printSchema()
# Feature extraction and transformation

numeric_features = [t[0] for t in combined_df.dtypes if t[1] in ['double','int'] ]

combined_df.select(numeric_features).describe().toPandas().transpose()

# COMMAND ----------

# dropping Delayminutes because delay status is highly corelated to delayminutes
combined_df_2 = combined_df.select('Airline', 'Origin', 'Dest', 'DayofMonth', 'DayOfWeek', 'CRSDepTime',
'OriginCityName', 'OriginStateName','DestCityName','DestStateName','DistanceGroup','DepTime', 'ArrTime',
'ArrDelayMinutes','Delaystatus')
cols = combined_df_2.columns

# COMMAND ----------

from pyspark.ml.feature import OneHotEncoder, StringIndexer, VectorAssembler
categoricalColumns = ['Airline', 'Origin', 'Dest', 'DayofMonth', 'DayOfWeek', 'CRSDepTime', 'OriginCityName',
'OriginStateName','DestCityName','DestStateName','DistanceGroup']
stages = []
for categoricalCol in categoricalColumns:
  stringIndexer = StringIndexer(inputCol = categoricalCol, outputCol = categoricalCol + 'Index')
  encoder = OneHotEncoder(inputCols=[stringIndexer.getOutputCol()], outputCols=[categoricalCol +
"classVec"])
  stages += [stringIndexer, encoder]
label_stringIdx = StringIndexer(inputCol = 'Delaystatus', outputCol = 'label')
stages += [label_stringIdx]
numericCols = ['DepTime', 'ArrTime', 'ArrDelayMinutes']
assemblerInputs = [c + "classVec" for c in categoricalColumns] + numericCols
assembler = VectorAssembler(inputCols=assemblerInputs, outputCol="features")
stages += [assembler]
```

```
# COMMAND ----------

# Scaling
from pyspark.ml.feature import StandardScaler

# Create an instance of the StandardScaler
scaler = StandardScaler(inputCols=numeric_features, outputCols=numeric_features_scaled)
# Fit the scaler to the data
scaler_model = scaler.fit(combined_df_2)

# Transform the data
combined_df_2 = scaler_model.transform(combined_df_2)

# COMMAND ----------

#PCA
from pyspark.ml.feature import PCA

# Perform PCA with k = 1 to 11 to decide on best k
pca = PCA(k=11, inputCol="features", outputCol="pca_features")
model = pca.fit(combined_df_2)

# Plot the explained variance as a function of the number of principal components
plt.plot(range(1, 12), model.explainedVariance)
plt.xlabel('Number of principal components')
plt.ylabel('Explained variance')
plt.show()

print(model.explainedVariance)

from pyspark.ml.feature import PCA

# Create an instance of the PCA class
pca = PCA(k=3, inputCol="features", outputCol="pca_features")


# Fit the PCA model on the data
model = pca.fit(combined_df_2)

# Transform the data using the PCA model
transformed_data = model.transform(combined_df_2)

# COMMAND ----------

from pyspark.ml import Pipeline
pipeline = Pipeline(stages = stages)
pipelineModel = pipeline.fit(combined_df_2)
combined_df_2 = pipelineModel.transform(combined_df_2)
selectedCols = ['label', 'features'] + cols
combined_df_2 = combined_df_2.select(selectedCols)
combined_df_2.printSchema()


# COMMAND ----------

pd.DataFrame(combined_df_2.take(5), columns=combined_df_2.columns).transpose()
```

```
# COMMAND ----------

#splitting train test data
train, test = combined_df_2.randomSplit([0.7, 0.3], seed = 2018)
print("Training Dataset Count: " + str(train.count()))
print("Test Dataset Count: " + str(test.count()))

# COMMAND ----------

#LOGISTIC REGRESSION

# COMMAND ----------

from pyspark.ml.classification import LogisticRegression
lr = LogisticRegression(featuresCol = 'features', labelCol = 'label', maxIter=10)
lrModel = lr.fit(train)

# COMMAND ----------

# Predictions:
predictions = lrModel.transform(test)

# COMMAND ----------

from pyspark.ml.evaluation import BinaryClassificationEvaluator
evaluator = BinaryClassificationEvaluator()
print('Test Area Under ROC', evaluator.evaluate(predictions))

# COMMAND ----------

#select only prediction and label columns
preds_and_labels = predictions.select(['prediction','label'])
from pyspark.mllib.evaluation import MulticlassMetrics
metrics = MulticlassMetrics(preds_and_labels.rdd.map(tuple))

print(metrics.confusionMatrix().toArray())

# COMMAND ----------

#select only prediction and label columns
preds_and_labels = predictions.select(['prediction','label'])
from pyspark.mllib.evaluation import MulticlassMetrics
metrics = MulticlassMetrics(preds_and_labels.rdd.map(tuple))

print(metrics.confusionMatrix().toArray())

# COMMAND ----------

accuracy = predictions.filter(predictions.Delaystatus == predictions.prediction).count() /
float(predictions.count())
print("Accuracy : ",accuracy)
```

```
# COMMAND ----------

# Hyperparameter tuning using crossfold validation

import pyspark.ml.tuning as tune

# Create the parameter grid
grid = tune.ParamGridBuilder()

# Add the hyperparameter
grid = grid.addGrid(lr.regParam, np.arange(0, .1, .01))
grid = grid.addGrid(lr.elasticNetParam, [0, 1])

# Build the grid
grid = grid.build()

# COMMAND ----------

# cross validation
cv = tune.CrossValidator(estimator=lr, estimatorParamMaps=grid, evaluator=evaluator)

# COMMAND ----------

cv_model= cv.fit(train)

# COMMAND ----------

prediction = cv_model.transform(test)
output= prediction.select("features", "probability", "prediction")

# COMMAND ----------

# RANDOM FOREST

# COMMAND ----------

from pyspark.ml.classification import RandomForestClassifier

rf = RandomForestClassifier(featuresCol = 'features', labelCol = 'label')
rfModel = rf.fit(train)
predictions = rfModel.transform(test)

# COMMAND ----------

from pyspark.ml.evaluation import MulticlassClassificationEvaluator

evaluator = MulticlassClassificationEvaluator(labelCol="label", predictionCol="prediction")
accuracy = evaluator.evaluate(predictions)
print("Accuracy = %s" % (accuracy))
print("Test Error = %s" % (1.0 - accuracy))
```

```
# COMMAND ----------

from pyspark.mllib.evaluation import MulticlassMetrics
from pyspark.sql.types import FloatType
import pyspark.sql.functions as F

preds_and_labels = predictions.select(['prediction','label']).withColumn('label',
F.col('label').cast(FloatType())).orderBy('prediction')
preds_and_labels = preds_and_labels.select(['prediction','label'])
metrics = MulticlassMetrics(preds_and_labels.rdd.map(tuple))
print(metrics.confusionMatrix().toArray())

# COMMAND ----------

from pyspark.ml.evaluation import BinaryClassificationEvaluator
evaluator = BinaryClassificationEvaluator()
print('Test Area Under ROC', evaluator.evaluate(predictions))

# COMMAND ----------

accuracy = predictions.filter(predictions.Delaystatus == predictions.prediction).count() /
float(predictions.count())
print("Accuracy : ",accuracy)

# COMMAND ----------

#plot ROC curve
train_summary = rfModel.summary
roc = train_summary.roc.toPandas()
plt.plot(roc['FPR'],roc['TPR'])
plt.ylabel('False Positive Rate')
plt.xlabel('True Positive Rate')
plt.title('Training set ROC Curve')
plt.show()

# COMMAND ----------

# GRADIENT BOOSTING TREES

# COMMAND ----------

from pyspark.ml.classification import GBTClassifier

# Create a GBTClassifier instance
gbt = GBTClassifier(maxIter=5, maxDepth=3)

# Train the model
model = gbt.fit(train)

# Make predictions on the test set
predictions = model.transform(test)

from pyspark.ml.evaluation import BinaryClassificationEvaluator
evaluator = BinaryClassificationEvaluator()
print('Test Area Under ROC', evaluator.evaluate(predictions))
```

```
#select only prediction and label columns
preds_and_labels = predictions.select(['prediction','label'])
from pyspark.mllib.evaluation import MulticlassMetrics
metrics = MulticlassMetrics(preds_and_labels.rdd.map(tuple))

accuracy = predictions.filter(predictions.Delaystatus == predictions.prediction).count() /
float(predictions.count())
print("Accuracy : ",accuracy)

# COMMAND ----------

# PRECISION RECALL AND F1 SCORE

# COMMAND ----------

evaluator_cv = BinaryClassificationEvaluator()
acc = evaluator_cv.evaluate(output, {evaluator_lr.metricName: "accuracy"})
f1 = evaluator_cv.evaluate(output, {evaluator_lr.metricName: "f1"})
weightedPrecision = evaluator_cv.evaluate(output, {evaluator_lr.metricName: "weightedPrecision"})
weightedRecall = evaluator_cv.evaluate(output, {evaluator_lr.metricName: "weightedRecall"})
auc = evaluator_cv.evaluate(output)

print(acc)
print(f1)
print(weightedPrecision)
print(weightedRecall)
print(auc)
```