Flight Delay Prediction

Machine Learning at Scale - Final Project Presentation

Team 11 - Amber Chen, Jeffrey Day, Sanjay Elangovan, and Menglu He

Agenda

- Business Case
- Dataset and Joining
- EDA
- Feature Engineering
- Algorithm & Model Tuning
- Evaluation & Performance
- Limitations & Future Work

Business Case

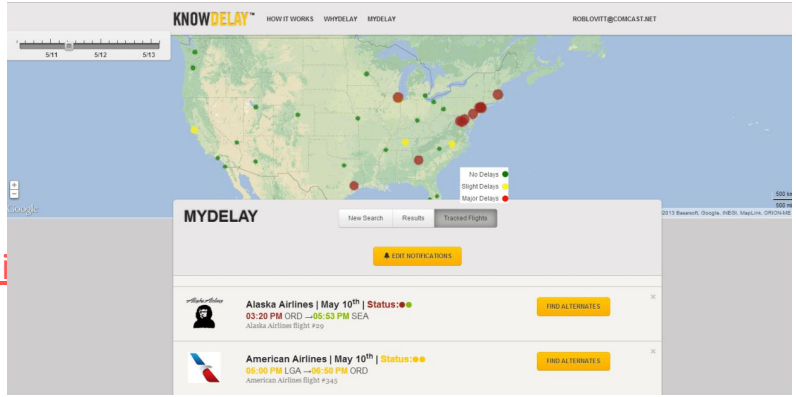
- In 2019, flight delays cost airlines approximately $8 billion and passengers approximately $18 billion (https://www.faa.gov/data_research/aviation_data_statistics/media/cost_delay_estimates.pdf)

Total Cost of Delay in the U.S. (dollars, billion)

	2016	2017	2018	2019
Airlines	5.6	6.4	7.7	8.3
Passengers	13.3	14.8	16.4	18.1
Lost Demand	1.8	2.0	2.2	2.4
Indirect	3.0	3.4	3.9	4.2
Total	**23.7**	**26.6**	**30.2**	**33.0**

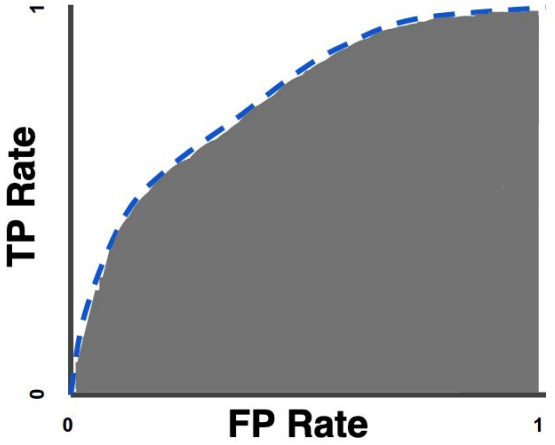
Objective

- Target Audience: Traveler
- Predict whether a given flight will be delayed by more than 15 mins given 2 hours of lead time based on provided data set.
- State of the Art:
  - KnowDelay: 90% accurate
  - FlightCaster: 85% accurate
  - DelayCast: 80-90% accurate
  - Accuracies from - Predictive Modeling: Flight Delays and Associated factors, Henriques et al. https://www.sciencedirect.com/science/article/pii



Evaluation Metrics

- Evaluation Metrics:
  - Accuracy compared against baseline of all negatives (~80%)
  - Model Tuning:
    - ROC-AUC curve will be used to evaluate the model
    - False positives are more damaging than false negatives



Dataset

Airlines 

2015 - 2019
(31.7M data points, 109 attributes)

Flight information
- Date, scheduled and actual departure/arrival time, time elapsed
- Carrier, airline ID

Airport information
- Origin and destination airport codes and locations

Flight Performance
- Departure/arrival delay, cancellation
- Causes of delay

Missing Value Treatment:
- Removed Cancelled and Diverted flights
- Kept 31.2M data points

Source: parquet_airlines_data

Weather 

2015 - 2019
(627M data points, 177 attributes)

- U.S. weather station locations
- Weather record timestamp
- Wind, air temperature, visibility, dew point, sea level, etc
- quality control codes

Missing Value Treatment:
- Replaced missing values “999*” with Nulls
- Replaced with Nulls for quality codes = suspect/ erroneous

Source: weather_data

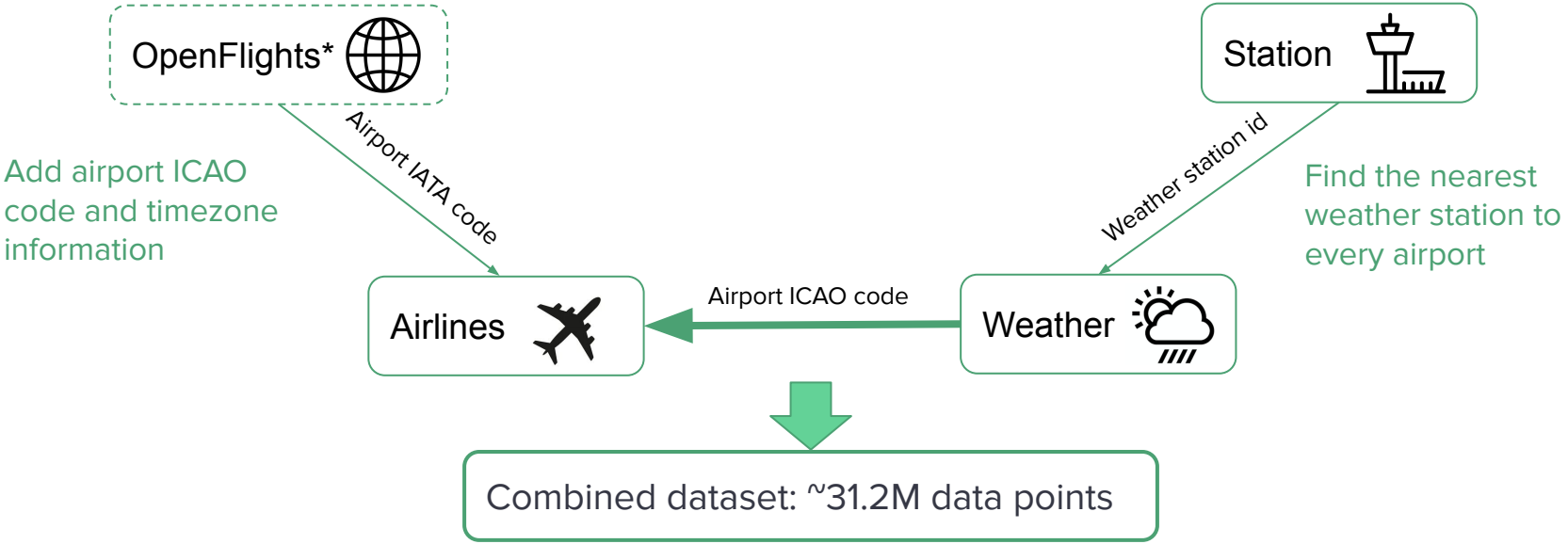
Stations 

Static (5M data points, 12 attributes)

- Worldwide weather station locations
- Neighbor airports
- Distance to neighbor airports

Source: stations_data

Joining Datasets



Find the latest weather record from the station nearest to the departure airport 2 hours prior to scheduled departure

*Source from https://openflights.org/data.html;
The data contains ~7,000 airports spanning the globe and each entry includes airport IDs, location, and timezone information

Joining Datasets (code)

Join airline dataset with external OpenFlight data

```
# Add timezone and airport ICAO code to airline data

# Join on origin airport
df_airlines_airport = airlines.join(df_airport, airlines.ORIGIN == df_airport.IATA, how='left').select(airlines['*'],
df_airport['AirportID'].alias('ORIGIN_airportID'), df_airport['ICAO'].alias('ORIGIN_ICAO'),
df_airport['Timezone'].alias('ORIGIN_Timezone'), df_airport['TZ_Timezone'].alias('ORIGIN_TZ'))

# Join on destination airport
df_airlines_airport = df_airlines_airport.join(df_airport, df_airlines_airport.DEST == df_airport.IATA,
how='left').select(df_airlines_airport['*'], df_airport['AirportID'].alias('DEST_airportID'),
df_airport['ICAO'].alias('DEST_ICAO'), df_airport['Timezone'].alias('DEST_Timezone'),
df_airport['TZ_Timezone'].alias('DEST_TZ'))
```

Weather+airline join

Cmd 56

```
# attach the nearest weather station to origin airport
df_airlines_airport_station_processed = df_airlines_airport.join(df_station_processed,
                                                                 df_airlines_airport.ORIGIN_ICAO == df_station_processed.neighbor_call, 'left') \
                                    .select(df_airlines_airport['*'], df_station_processed['station_id'].alias('ORIGIN_near_station'))

```

Joining Datasets (code)

Find the latest weather timestamp before 2hrs of leadtime for each flight

Cmd 62

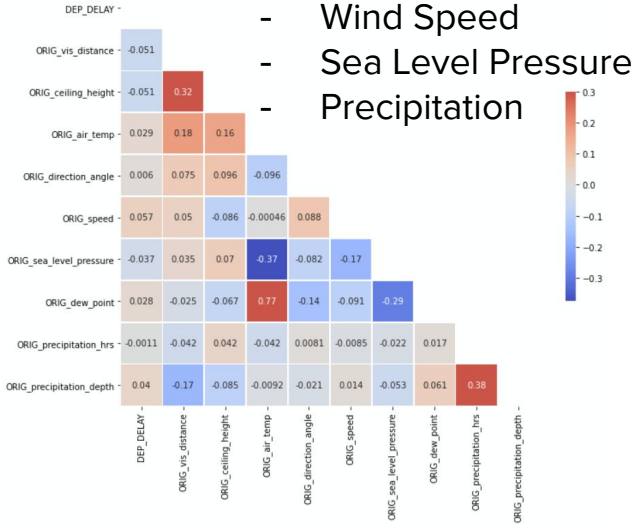
```
nearest_weather_time_ORIGIN = df_airlines_airport_station_processed \
                      .join(df_weather_select, df_airlines_airport_station_processed.ORIGIN_near_station == df_weather.STATION, how='left') \
                      .where(df_airlines_airport.EARLIER_DATETIME >= df_weather_select.DATE) \
                      .groupBy(df_airlines_airport['EARLIER_DATETIME'], df_airlines_airport_station_processed['flightID'], \
                              df_airlines_airport_station_processed['ORIGIN_near_station']) \
                      .agg(f.max(df_weather_select.DATE).alias('ORIGIN_weather_time'))
```

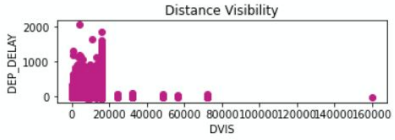
Join the latest weather record on origin airport for each flight

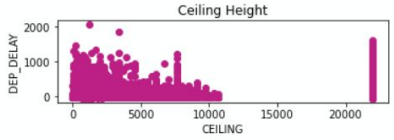
```
# Join df_weather with df_airlines_airpor_station_processed for origin airport based on 'flightID', 'nearest_weather_time_ORIGIN'
df_intermediate = df_airlines_airport_station_processed \
                  .join(nearest_weather_time_ORIGIN, \
                        df_airlines_airport_station_processed.flightID == nearest_weather_time_ORIGIN.flightID, how='left') \
                  .join(df_weather_select,
                        (df_airlines_airport_station_processed.ORIGIN_near_station == df_weather_select.STATION) & \
                        (nearest_weather_time_ORIGIN.ORIGIN_weather_time == df_weather_select.DATE) & \
                        (nearest_weather_time_ORIGIN.ORIGIN_near_station == df_weather_select.STATION), how='left') \
                  .select(df_airlines_airport_station_processed['*'],df_weather_select['*']) \
```

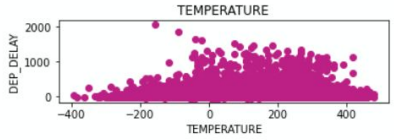
EDA - Weather Dataset

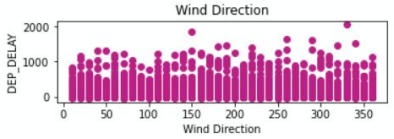
- Explored factors might impact departure delay:
  - Weather Impact
    - Distance Visibility
    - Ceiling Height
    - Temperature
    - Wind Direction
    - Wind Speed
    - Sea Level Pressure
    - Precipitation

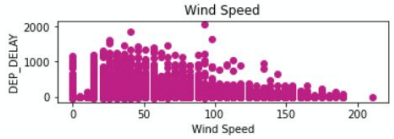


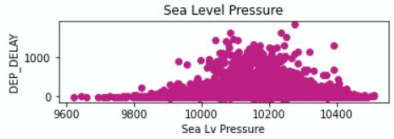


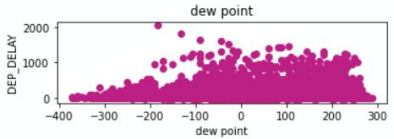


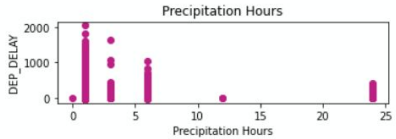


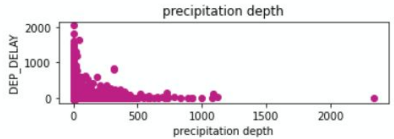






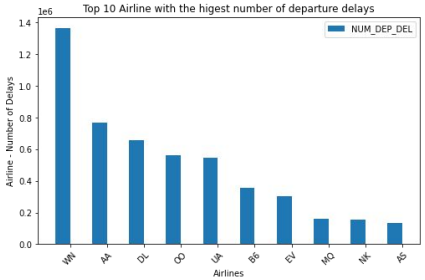


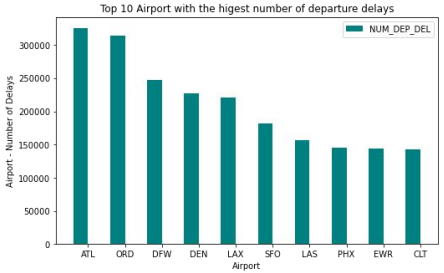


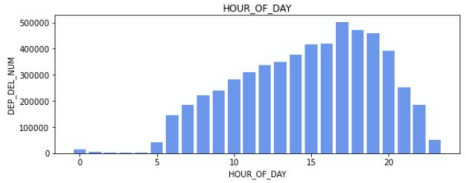


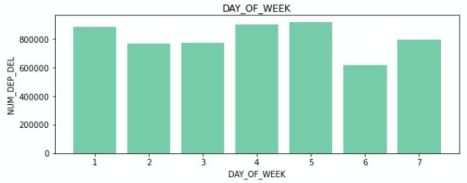
EDA - Airline Dataset

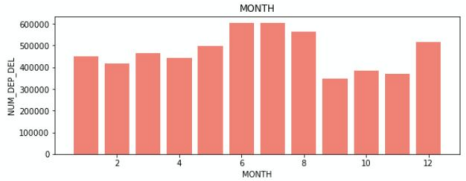
- Dataset:
  - distinct airports is: 369
  - distinct airlines is: 19
  - 2015-01-01 to 2019-12-31
- 18.1% of the flights is delayed by > 15 mins
- Explored factors might impact departure delay:
  - Temporal impact:
    - **Hour of day**
    - **Day of week**
    - **Month of year**
    - Quarter
    - Day of month
  - Airline impact
  - Airport impact

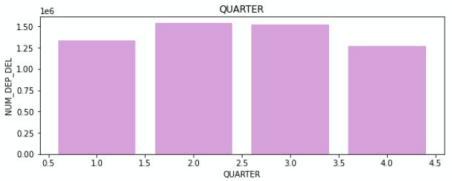


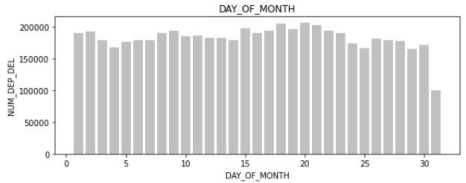


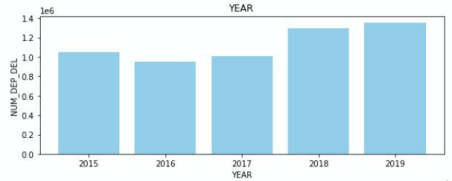




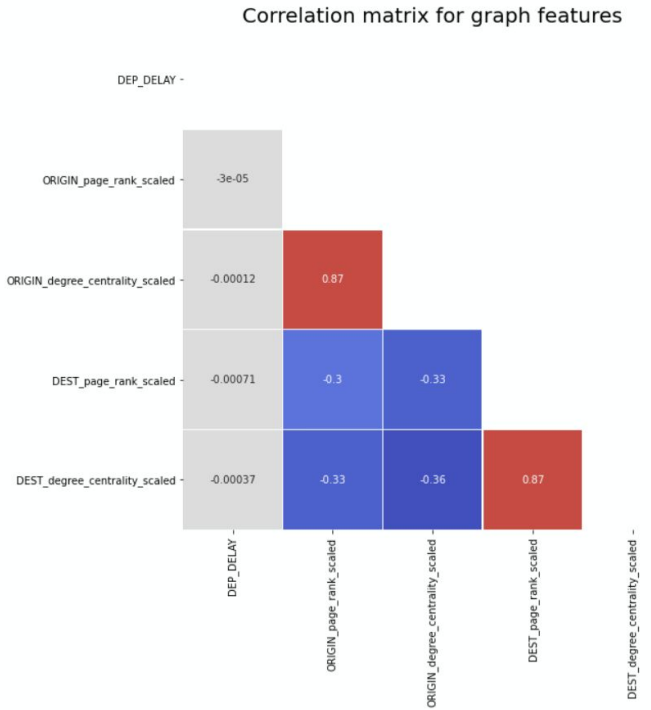








EDA - Airline Dataset (cntd.)

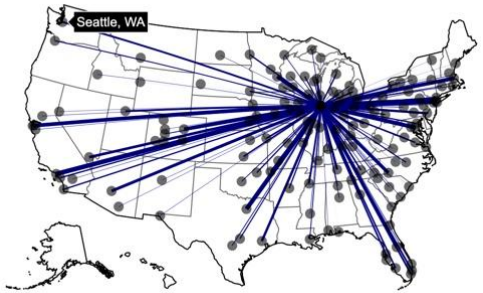


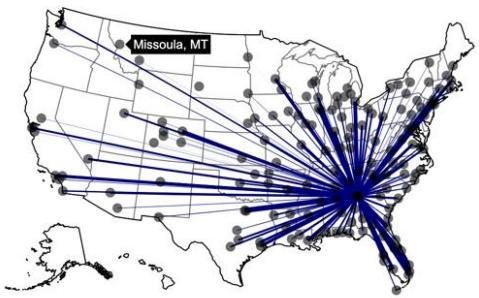
Cmd 10

```
num_var = ["ORIG_ELEVATION", "DEST_ELEVATION", "ORIG_direction_angle", "DEST_direction_angle","ORIG_speed", "DEST_speed", "ORIG_ceiling_height",
"DEST_ceiling_height", "ORIG_vis_distance", "DEST_vis_distance", "ORIG_air_temp", "ORIG_dew_point", "DEST_dew_point", "ORIG_sea_level_pressure",
"DEST_sea_level_pressure", "CRS_DEP_TIME", "CRS_ARR_TIME", "ORIG_precipitation_hrs", "ORIG_precipitation_depth", "DEST_precipitation_hrs",
"DEST_precipitation_depth"]

df_joined_5y = df_spark_5y.select("*").withColumn("id", monotonically_increasing_id()).select("id")

for i in num_var:
  df_temp_5y = df_spark_5y \
    .select(mean(i) \
    .alias("mean_"+i), stddev(i).alias("std_"+i)) \
    .crossJoin(df_spark_5y) \
    .withColumn(i+"_scaled", (col(i)-col("mean_"+i))/col("std_"+i)) \
    .select(i+"_scaled") \
    .withColumn("id", monotonically_increasing_id())
  df_joined_5y = df_temp_5y.join(df_joined_5y, on = "id", how='inner')
```

▸ df_temp_5y: pyspark.sql.dataframe.DataFrame = [DEST_precipitation_depth_scaled: double, id: long]
▸ df_joined_5y: pyspark.sql.dataframe.DataFrame = [id: long, DEST_precipitation_depth_scaled: double ... 20 more fields]

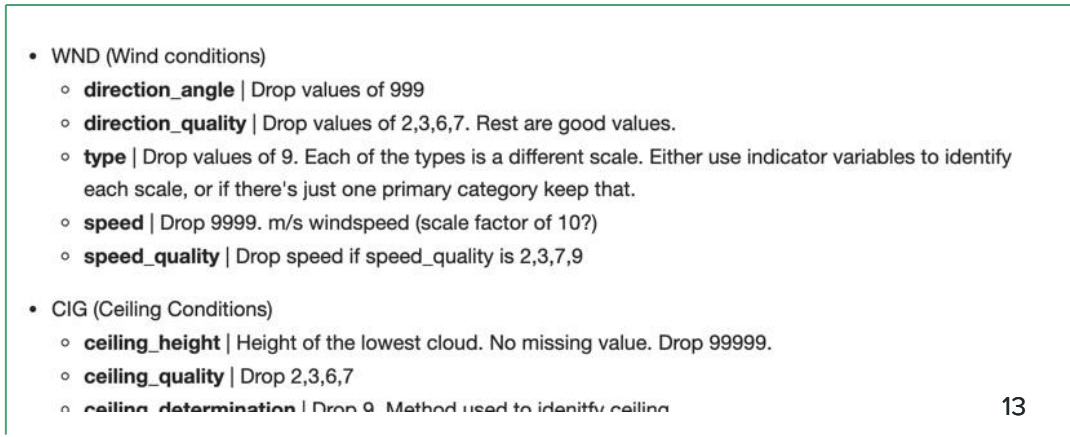
Command took 2.22 seconds -- by sanjay_elangovan@berkeley.edu at 7/28/2021, 5:08:09 PM on w261-su21-team11





Feature Engineering - Weather Dataset

- Split columns with multiple data points into single elements (ie. wind direction, angle, speed)
- Used quality codes and data dictionary to identify nullable data
  - Dictionary - https://www.ncei.noaa.gov/data/global-hourly/doc/isd-format-document.pdf
- UTC time conversion for actual flight time and scheduled departure time
- Created a datetime field for 2 hours prior to scheduled departure time
- Normalized weather variables using mean and standard deviation



- WND (Wind conditions)
  - **direction_angle** | Drop values of 999
  - **direction_quality** | Drop values of 2,3,6,7. Rest are good values.
  - **type** | Drop values of 9. Each of the types is a different scale. Either use indicator variables to identify each scale, or if there's just one primary category keep that.
  - **speed** | Drop 9999. m/s windspeed (scale factor of 10?)
  - **speed_quality** | Drop speed if speed_quality is 2,3,7,9
- CIG (Ceiling Conditions)
  - **ceiling_height** | Height of the lowest cloud. No missing value. Drop 99999.
  - **ceiling_quality** | Drop 2,3,6,7
  - **ceiling_determination** | Drop 9. Method used to identify ceiling

Code to Normalize Weather Variables

Cmd 10

```
num_var = ["ORIG_ELEVATION", "DEST_ELEVATION", "ORIG_direction_angle", "DEST_direction_angle","ORIG_speed", "DEST_speed", "ORIG_ceiling_height",
"DEST_ceiling_height", "ORIG_vis_distance", "DEST_vis_distance", "ORIG_air_temp", "ORIG_dew_point", "DEST_dew_point", "ORIG_sea_level_pressure",
"DEST_sea_level_pressure", "CRS_DEP_TIME", "CRS_ARR_TIME", "ORIG_precipitation_hrs", "ORIG_precipitation_depth", "DEST_precipitation_hrs",
"DEST_precipitation_depth"]

df_joined_5y = df_spark_5y.select("*").withColumn("id", monotonically_increasing_id()).select("id")

for i in num_var:
  df_temp_5y = df_spark_5y \
    .select(mean(i) \
    .alias("mean_"+i), stddev(i).alias("std_"+i)) \
    .crossJoin(df_spark_5y) \
    .withColumn(i+"_scaled", (col(i)-col("mean_"+i))/col("std_"+i)) \
    .select(i+"_scaled") \
    .withColumn("id", monotonically_increasing_id())
  df_joined_5y = df_temp_5y.join(df_joined_5y, on = "id", how='inner')
```

▸ df_temp_5y: pyspark.sql.dataframe.DataFrame = [DEST_precipitation_depth_scaled: double, id: long]

▸ df_joined_5y: pyspark.sql.dataframe.DataFrame = [id: long, DEST_precipitation_depth_scaled: double ... 20 more fields]

Command took 2.22 seconds -- by sanjay_elangovan@berkeley.edu at 7/28/2021, 5:08:09 PM on w261-su21-team11

DEST_sea_level_pressure_scaled	ORIG_sea_level_pressure_scaled	DEST_dew_point_scaled	ORIG_dew_point_scaled	ORIG_air_temp_scaled	DEST_vis_distance_scaled
-0.967865958244261	-1.2495922240110326	-0.1359027373991108	-0.34809091279015547	1.9037462810213683	0.33633640695763234
0.666371041928481	0.5644385387347975	-1.1544414729072754	-0.4063192513077546	0.19986060267687905	0.30698709275813835
-1.7774879583298395	0.054710886392994	-0.02919867939349355	0.6806097343540954	0.41045321460709683	0.33633640695763234
-1.882438958340933	-1.7593198763528362	-0.7276252408848064	-1.5320671293146706	0.41045321460709683	0.33633640695763234
-1.2377399582727873	0.5194625694105207	0.562523824092202	0.8941136422519588	0.5157495205722057	0.33633640695763234
-0.9228869582395066	-0.350072837525497	0.9408382115666631	0.8941136422519588	0.24772255993374673	0.33633640695763234

Graph Data Code

- Graph Features:
  - Since most nodes are not connected, closeness and betweenness metrics don’t help.
  - We rely on PageRank and Degree Centrality.

```
counts = df_graph_5y.groupby(["ORIG_LATITUDE","ORIG_LONGITUDE","DEST_LATITUDE","DEST_LONGITUDE"]).agg({"ORIG_LATITUDE":'count',
                                                    "ORIGIN_CITY_NAME":'min',"DEST_CITY_NAME":'min'})

counts.rename(columns={'ORIG_LATITUDE':'weight'},inplace=True)
counts.head()

g = nx.from_pandas_edgelist(df_graph_5y, source='ORIGIN_CITY_NAME', target='DEST_CITY_NAME', create_using=nx.DiGraph())

for index, row in counts.iterrows():
  g[row.ORIGIN_CITY_NAME][row.DEST_CITY_NAME]['weight'] = row.weight
```

```
Command took 0.78 seconds -- by sanjay_elangovan@berkeley.edu at 7/28/2021, 8:24:22 PM on w261-su21-team11
```

```
Cmd 19
```

```
pagerank = nx.pagerank(g)
pagerank_df = pd.DataFrame.from_dict(pagerank, orient='index').reset_index()
pagerank_df.rename(columns={'index':'city_name',0:'page_rank'}, inplace=True)

pagerank_df.sort_values(by='page_rank', ascending=False)
```

Feature Engineering - Snowball Delays

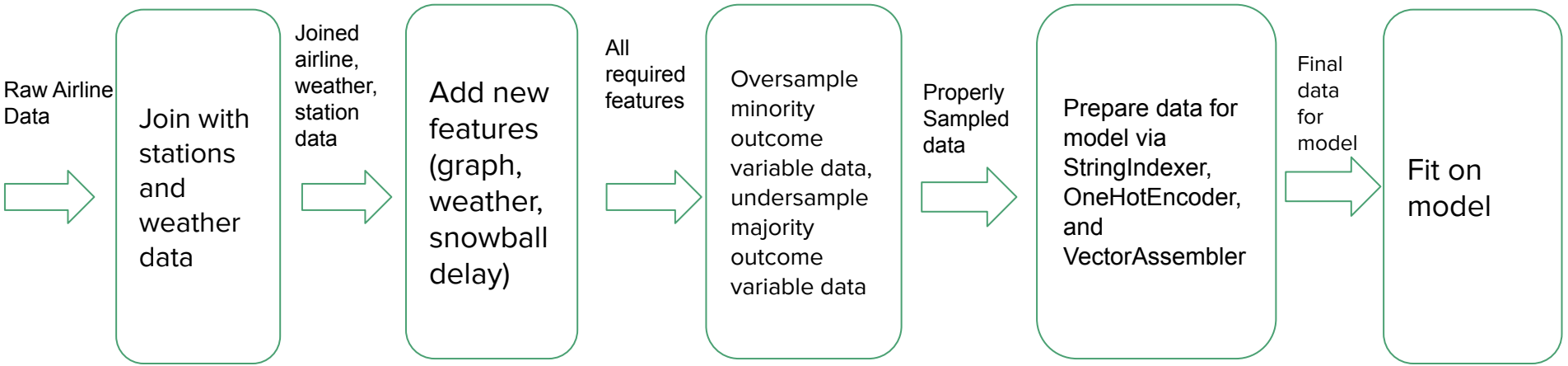
- Added two new features to encapsulate potential for previous delays to cause future delays
  - **DEP_DEL15_PREV_FIN** - Was the plane with tail number for current flight's most recent flight delayed by at least 15 minutes?
  - **NUM_PREV_DELAYS_ORIGIN** - How many previous flights at the origin airport for the flight have been delayed at least 15 minutes?
  - Both of these features can only consider flights more than 135 minutes in the past, since we are trying to predict for current flight two hours ahead.

```
def add_num_delays_origin_airport(df):
  column_list = ["FL_DATE","ORIGIN"]
  windowval = (Window.partitionBy([col(x) for x in column_list]).orderBy(col('CRS_DEP_DATETIME').cast("long")) \
                                                                 .rangeBetween(Window.unboundedPreceding, -135*60))
  df_num_origin_delays = df.withColumn('NUM_PREV_DELAYS_ORIGIN', f.sum(f.when(col("DEP_DELAY_NEW") >= 15,1) \
                                                                       .otherwise(0)).over(windowval))
  return df_num_origin_delays
```

Final Feature Selection

- **Response variable:** DEP_DEL15 (binary)
- **Explanatory variables**:
  - Airline
    - Carrier, departure airport, arrival airport, distance, scheduled departure time, scheduled arrival time, time of day, day of week
  - Weather (for origin and destination, normalized):
    - Elevation, wind direction, wind speed, ceiling height, visibility distance, air temperature, dew point temperature, sea level pressure, liquid precipitation
  - New Features:
    - ‘DEP_DEL15_PREV_FIN’, ‘NUM_PREV_DELAYS_ORIGIN’
    - PageRank, Degree Centrality (Origin and Destination)

Modeling - Pipeline (Training)

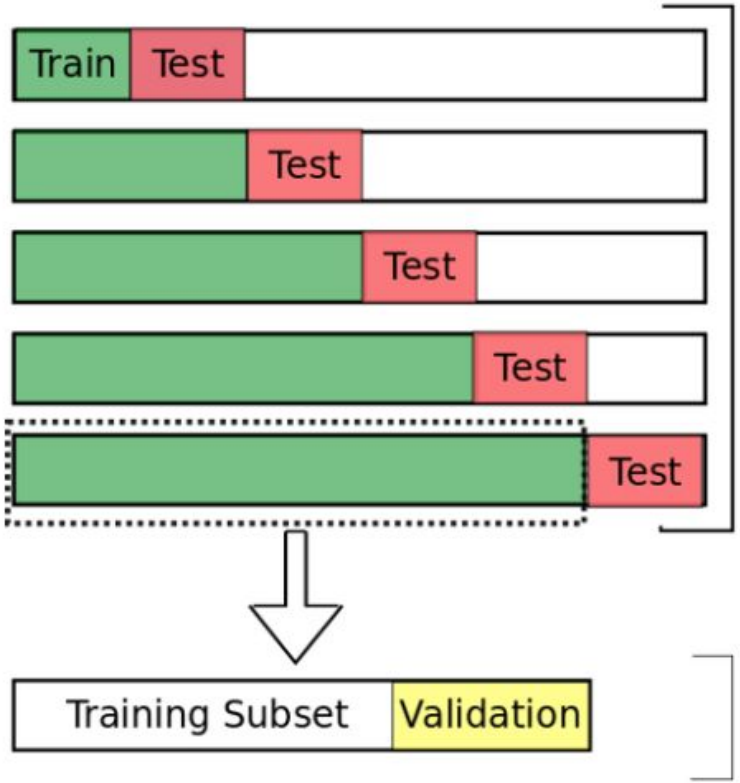


Code in Appendix

Code in Appendix

Cross Validation

- Used 2015-2018 flight data for time series nested cross validation and hyperparameter tuning
- Used three fold cross validation similar to diagram
- Code in appendix



Reference: https://towardsdatascience.com/time-series-nested-cross-validation-76adba623eb9

Algorithms Tried & Evaluation Metrics

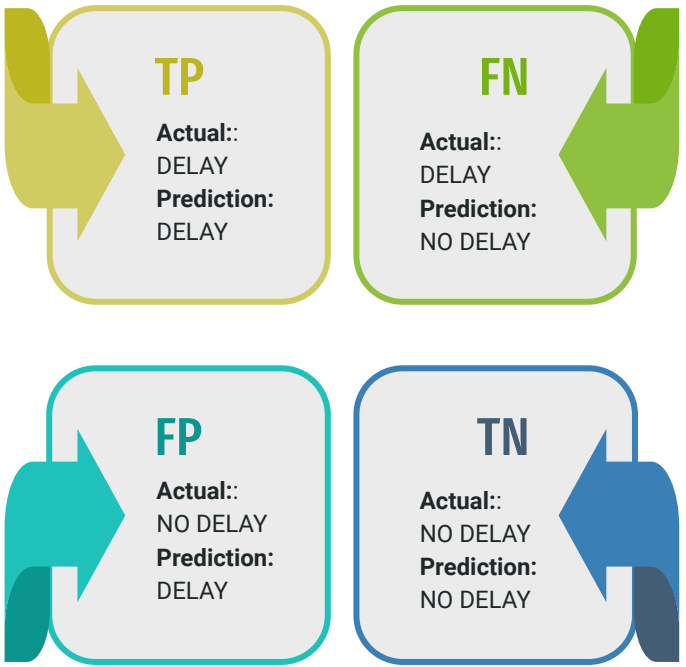
Models

- Logistic Regression
- Random Forest
- Gradient Boost Tree

Evaluation

- Accuracy : (TP +TN) / (TP + TN + FP + FN)
- Precision: TP / (TP + FP)
- Recall: TP / (TP + FN)
- F1-Score: TP / ½*(TP + FN)
- AUC ROC

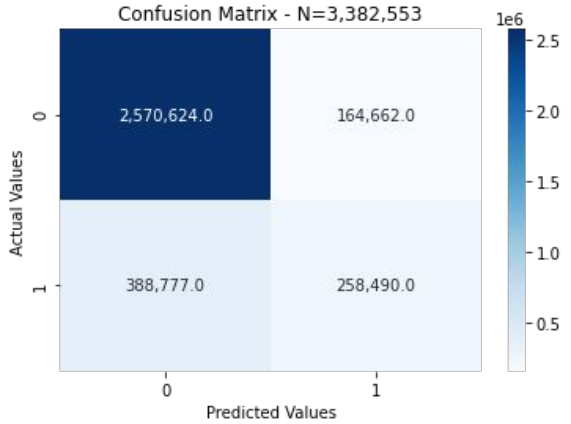
Confusion Matrix

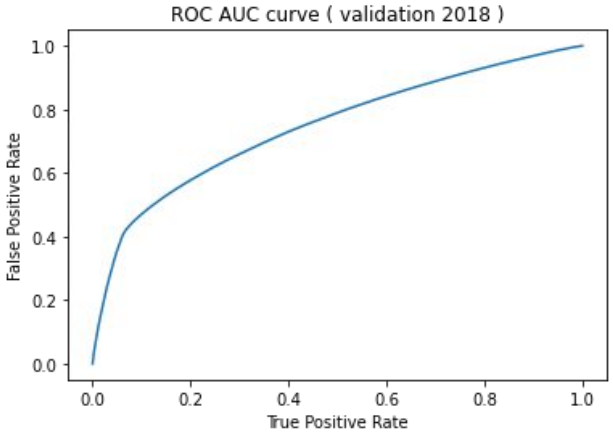


Baseline Model - Logistic Regression

- Code
```
lr = LogisticRegression(featuresCol = 'features', labelCol = 'label', maxIter = 30, regParam = 0.001, elasticNetParam = 0.25, standardization = False)
model = lr.fit(data)
preds = model.transform(data)
```

- Model Performance Evaluation Output
  - Time: 14 minutes
  - AUC ROC: 0.75

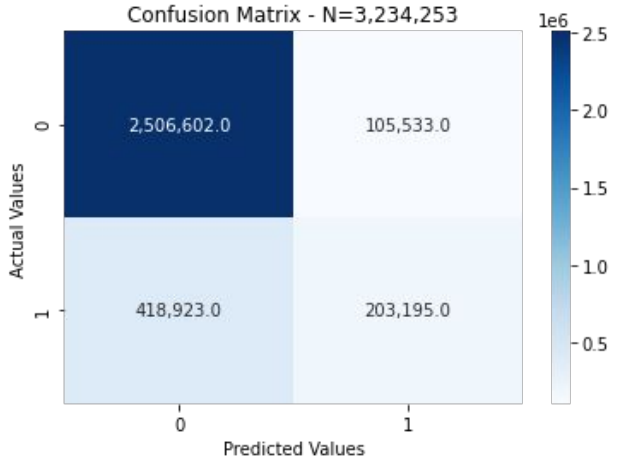


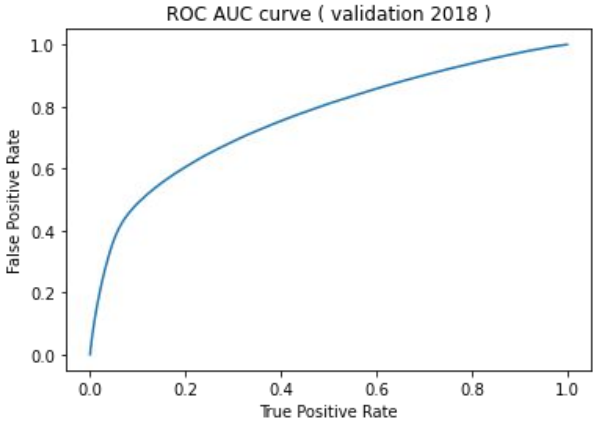


	Training	Validation
Accuracy	0.84	0.84
Precision	0.70	0.61
Recall	0.40	0.40
F-1 Score	0.50	0.48

Model 2 - Random Forest

- Code
```
rf = RandomForestClassifier(labelCol = "label", featuresCol = "features",
numTrees = 15, featureSubsetStrategy = "auto", impurity = "gini",
subsamplingRate = 0.6, maxDepth = 25, maxBins = 16)
model = rf.fit(data)
preds = model.transform(data)
```
- Model Performance Evaluation Output
  - Time: 38 minutes
  - AUC ROC: 0.76





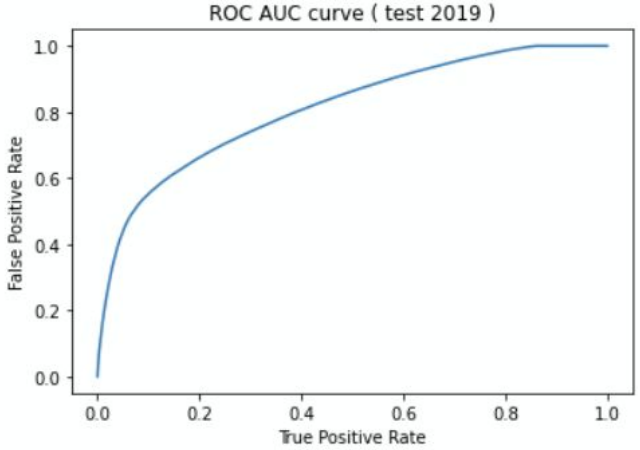
	Training	Validation
Accuracy	0.83	0.83
Precision	0.81	0.65
Recall	0.39	0.33
F-1 Score	0.52	0.44

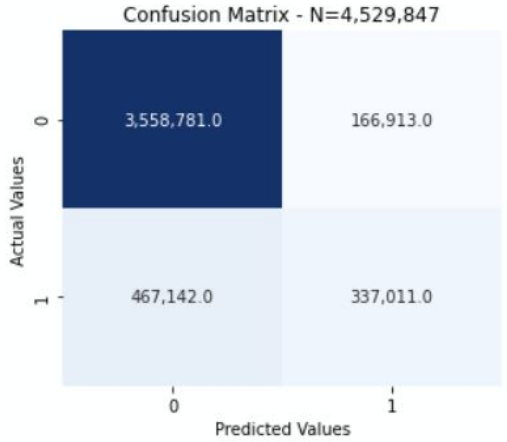
Model 3 - Gradient Boosted Trees

- Code

```
gbt = GBTClassifier(maxDepth=7, maxBins=350, maxIter=7, stepSize=0.35)
model = gbt.fit(train)
preds = model.transform(test)
```

- Model Performance Evaluation Output
  - Time: 46 minutes
  - AUC ROC: 0.81





	Training	Test
Accuracy	0.86	0.86
Precision	0.70	0.67
Recall	0.44	0.42
F-1 Score	0.54	0.52

Model Performance Summary & Final Model

Evaluation Metrics on Test 2019					
Model	AUC ROC	Accuracy	Precision	Recall	F1-score
Logistic Regression	0.75	0.84	0.61	0.40	0.48
Random Forest	0.76	0.83	0.65	0.33	0.44
Gradient Boosted Tree	0.81	0.86	0.67	0.42	0.52

The Gradient Boosted Tree model is selected based on our choice of evaluation metrics

Limitation & Future Work

- General applicability of airports not in the training dataset
- SMOTE application and HyperParameter tuning using Cross Validation challenged by resources
- Future efforts should be spent in the following areas:
  - Scope to get improved model performance if we explore more SMOTE for imbalance adjustment and hyperparameter tuning via grid search
  - Creative feature engineering
  - Deep learning models such as neural networks

Thank you! Any Questions?

Cross-Validation Split

```
def time_series_cross_validation_split(num_folds, df, sort_order_column_name, dependent_column_name):
  if sort_order_column_name not in df.columns or dependent_column_name not in df.columns:
    return
  df = df.withColumn("rank", percent_rank().over(Window.partitionBy().orderBy(sort_order_column_name)))
  fold_percentage = 1 / (num_folds + 2)
  df_train_list = []
  df_validate_list = []
  df_test_list = []
  for i in range(1, num_folds + 1):
    train_df = df.where("rank <= " + str(fold_percentage * i)).drop("rank")
    validate_df = df.where("rank <= " + str(fold_percentage * (i + 1)) + " and rank > " + str(fold_percentage * i)).drop("rank")
    test_df = df.where("rank <= " + str(fold_percentage * (i + 2)) + " and rank > " + str(fold_percentage * (i + 1))).drop("rank")
    df_train_list.append(train_df)
    df_validate_list.append(validate_df)
    df_test_list.append(test_df)
  return df_train_list, df_validate_list, df_test_list
```

String Indexer, One Hot Encoder

```
#Specify Categorical Variables to be string indexed and encoded
### Used Distance_group instead of distance
### scheduled departure time&scheduled arrival time >> need some feature engineering?
cat_var = ["OP_UNIQUE_CARRIER","ORIGIN","DEST","DAY_OF_WEEK","DISTANCE_GROUP", "DAY_OF_MONTH", "MONTH", "DEP_DEL15_PREV_FIN"]

#Empty vector to store pipeline
stages = []

#Use the OneHotEncoderEstimator to convert categorical features into one-hot vectors
#Loop through the specified categorical variables
for CatVar in cat_var:
  #Encoding
  stringIndexer = StringIndexer(inputCol = CatVar, outputCol = CatVar+'_Index', handleInvalid='skip')

  #Rename (Column Name + "ClassVec")
  encoder = OneHotEncoder(inputCols = [stringIndexer.getOutputCol()],outputCols = [CatVar + "classVec"], handleInvalid='error')

  #Temp staging the encoder
  stages +=[stringIndexer, encoder]

#Label outcome variable "DEP_DEL15" as "label"
label_index = StringIndexer(inputCol = 'DEP_DEL15', outputCol = 'label', handleInvalid='skip')

#Stage the labeling var (Dep_Delay15)
stages +=[label_index]
```

Pipeline Model

```
#Create a ML Pipeline Object and pass the stages vector
pipeline = Pipeline(stages = stages)

#Fit the pipeline onto the dataset
PipelineModel = pipeline.fit(train)
train = PipelineModel.transform(train)

#Only select the features of interest to pass into the model
selectedCols =  ['label','features'] + assemblerInputs
train = train.select(selectedCols)

#Getting rid of label == 2
#train = train.where("label != 2")

#Print the schema to check
train.printSchema()
```

Oversample & Undersample

```
#OverSampling by 20% for "lable == 1"
df_class_1_over = df_class_1.sample(True, 1.2, seed = None)

#UnderSampling by 20% for "lable == 0"
df_class_0_under = df_class_0.sample(False, 0.8, seed = None)

#Join the two  sampled sets together
train = df_class_1_over.union(df_class_0_under)

#Print the distribution post rebalancing
display(train.groupby("label").count())
```