Flight Delay Prediction

MIDS W261 | Machine Learning at Scale | Fall 2021

Team 8 - Ferdous Alam, Yao Chen, Toby Petty, Zixi Wang

Agenda

1 | Business Case

2 | Datasets & Joins

3 | EDA

4 | Feature Engineering

5 | Algorithm & Model Pipeline

6 | Evaluation metrics

7 | Novel Findings

8 | Conclusion

Business Case

Impact

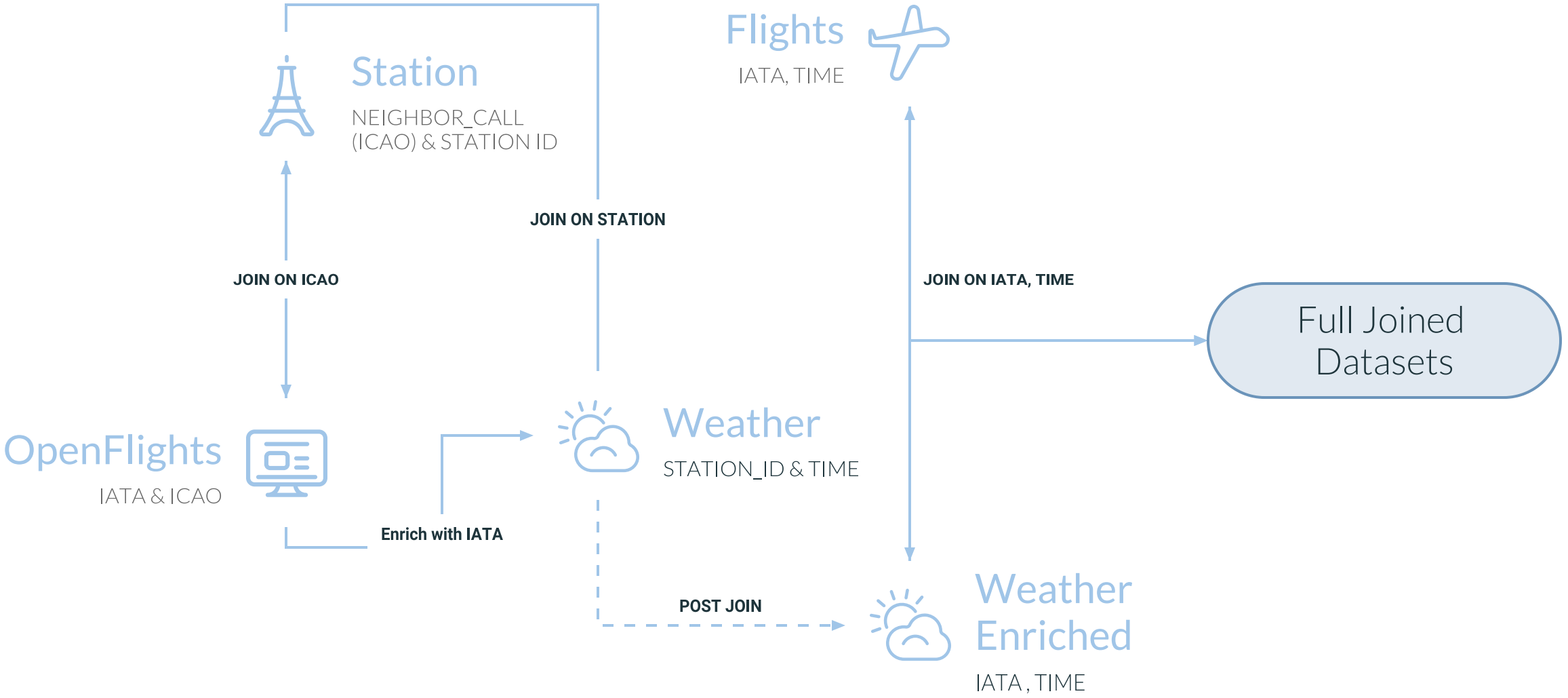
Complexity

Objective

$$F_\beta = \frac{(1 + \beta^2)(precision \cdot recall)}{(\beta^2 \cdot precision) + recall}$$

$$\beta = 2$$

Datasets


Station
NEIGHBOR_CALL
(ICAO) & STATION ID
Flights
IATA, TIME
JOIN ON STATION
JOIN ON ICAO
JOIN ON IATA, TIME
Full Joined
Datasets
OpenFlights
IATA & ICAO
Weather
STATION_ID & TIME
Enrich with IATA
POST JOIN
Weather
Enriched
IATA , TIME

Preprocessing Data

Filter weather data to only relevant airports

```
df_weather = spark.read.parquet("/mnt/mids-w261/datasets_final_project/weather_data/*")
original_weather_count = df_weather.count()
print(f"Original weather n = {original_weather_count}")
df_weather_filtered = df_weather.filter(df_weather.STATION.isin(station_ids))
filtered_weather_count = df_weather_filtered.count()
print(f"Filtered weather n = {filtered_weather_count}")
print(f"Weather data size reduced by {(1-(filtered_weather_count/original_weather_count))*100:.0f}%")
```

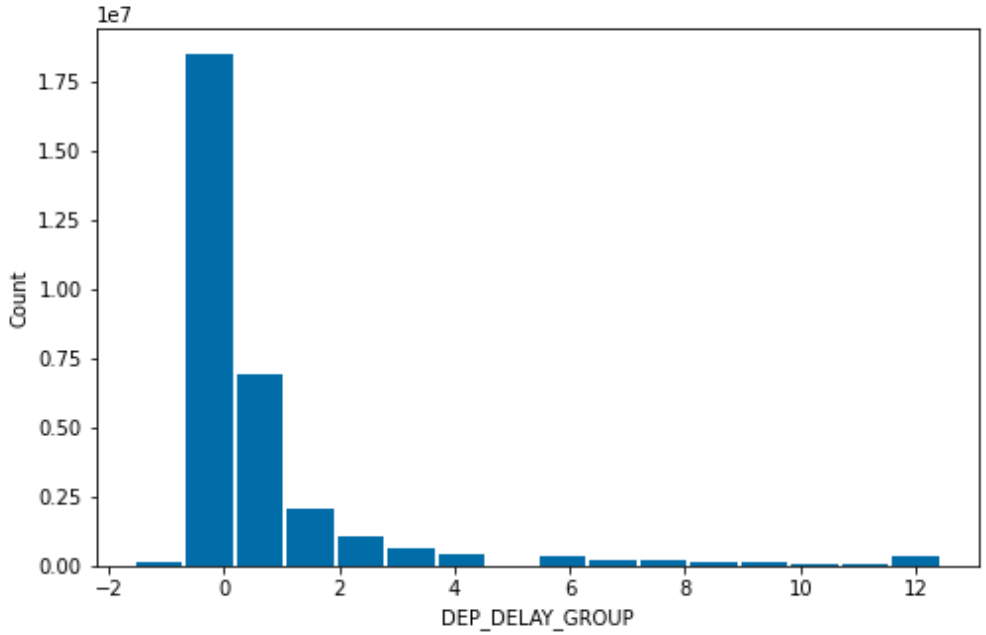
```
Filtered weather n = 24056947
Weather data size reduced by 96%
```

Convert flight timestamp to UTC

```
# Convert departure time integers to zero-padded strings, e.g. 607 -> 0000607:
flights = flights.withColumn("PADDED_DEP_TIME", format_string("0000%d", "CRS_DEP_TIME"))
# Shorten the strings to the final 4 chars, e.g. 0000607 -> 0607:
flights = flights.withColumn("FORMATTED_DEP_TIME", substring("PADDED_DEP_TIME", -4,4))
# Concatenate string columns for departure date and time:
flights = flights.withColumn("DEPT_DT_STR", concat_ws(" ", flights.FL_DATE, flights.FORMATTED_DEP_TIME))
# Convert string datetime to timestamp:
flights = flights.withColumn("DEPT_DT", to_timestamp(flights.DEPT_DT_STR, "yyyy-MM-dd HHmm"))
# Use datetime and timezone to convert dates to UTC:
flights = flights.withColumn("DEPT_UTC", to_utc_timestamp(flights.DEPT_DT, flights.ORIGIN_TZ))
# Remove minutes and round datetimes *down* to nearest hour. It is necessary to round
# down so that we don't join with weather data from less than 2 hours before:
flights = flights.withColumn("DEPT_UTC_HOUR", date_trunc("HOUR", flights.DEPT_UTC))

# Calculate arrival time in UTC using UTC departure time and scheduled flight duration in minutes:
flights = flights.withColumn("ARR_UTC", col("DEPT_UTC") + (col("CRS_ELAPSED_TIME") * expr("Interval 1 Minutes")))
```

EDA - Airlines Dataset

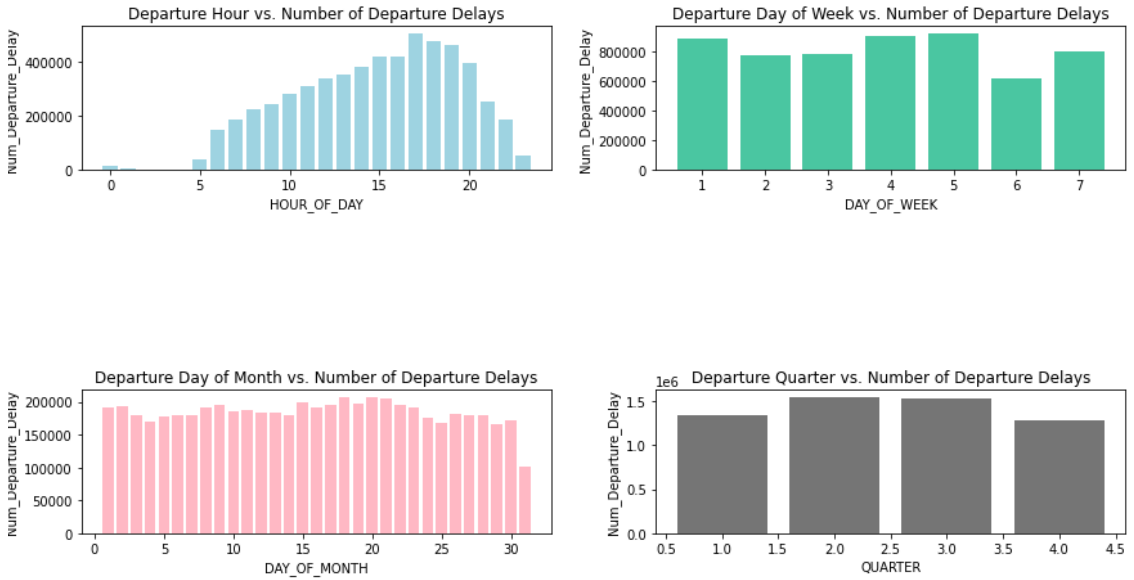
Date Range: 2015-01-01 to 2019-12-31

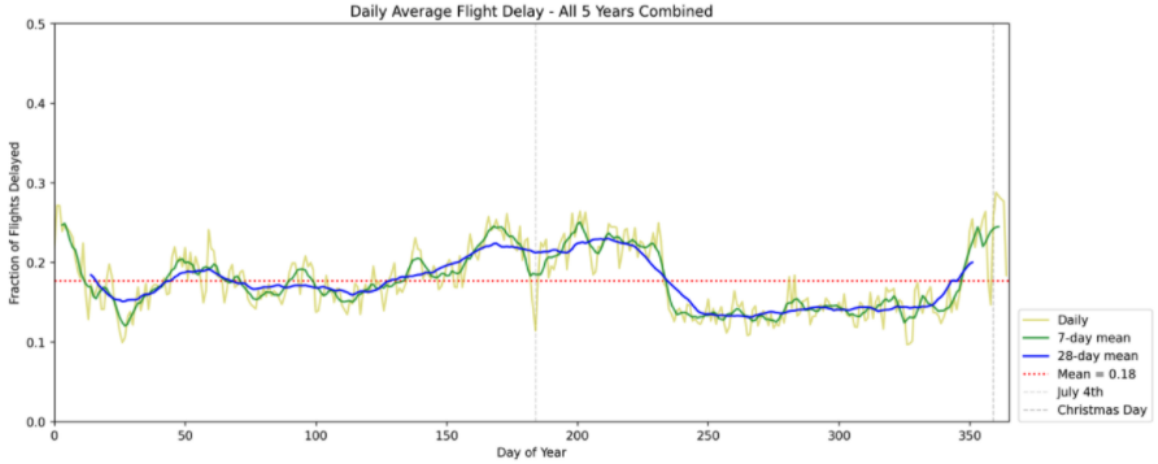


Total flight count	31,746,841
Flight count delayed by more than 15 minutes	5,693,541
Total columns/features count	107
% flights delayed (Class Imbalance)	**17.93%**
% flights on-time	80.56%
% flights cancelled	1.50%
Number of Unique Carriers	19
Number of Unique Airports	371

EDA - Airlines Dataset

Time-related Factors: Hour of Day, Day of Week, Day of Month, Quarter, Month of Year, Daily Average Delay





Hour of Day: Delay peaks between 4 - 7 pm

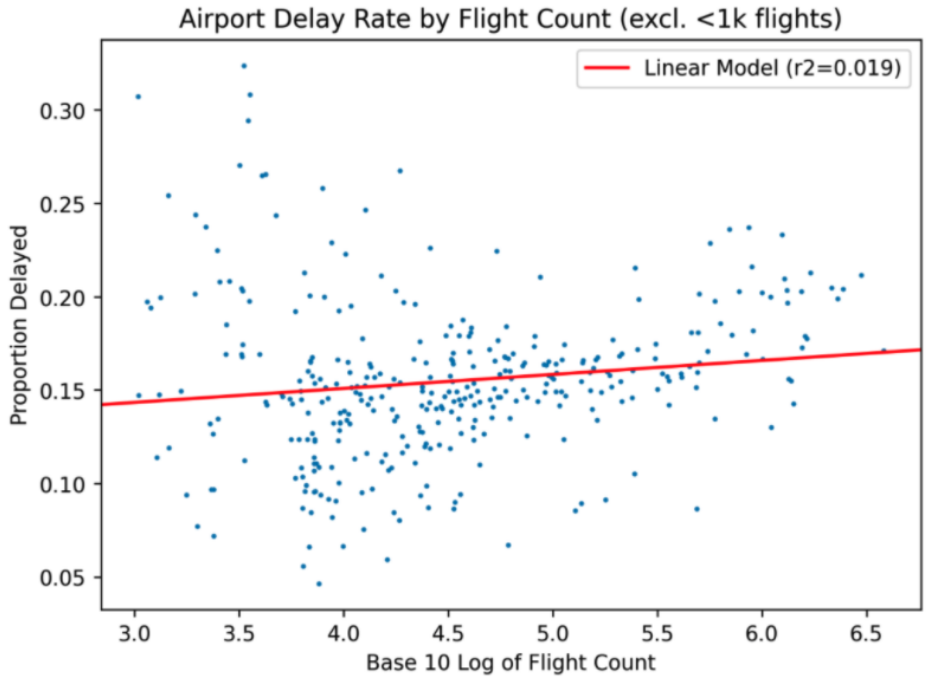
Close to uniform distribution for Day of Week, Day of Month, Quarter.

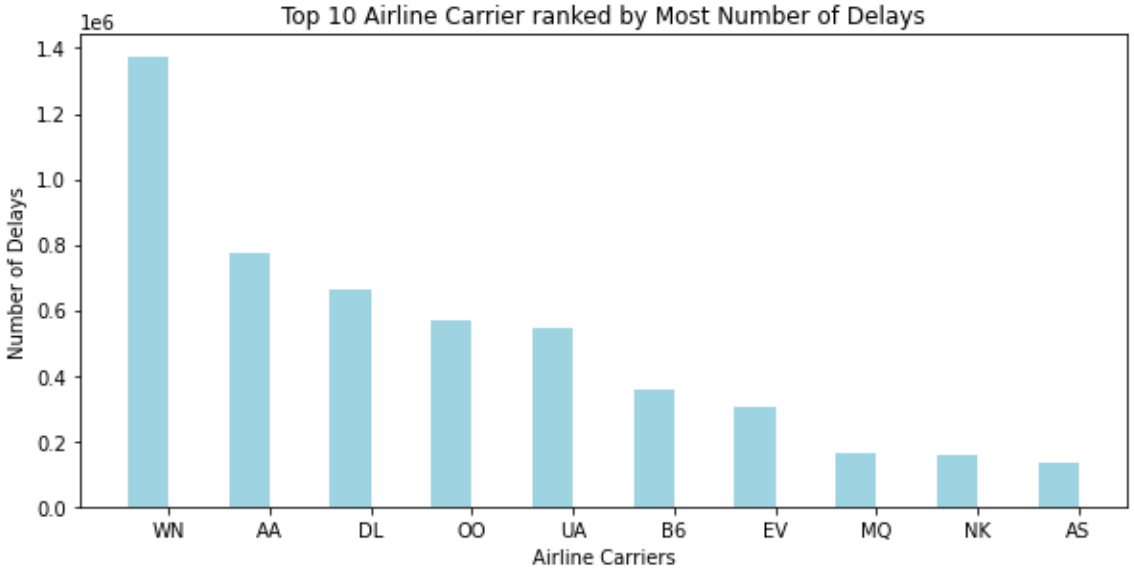
Strong seasonal impact: Summer & Christmas

July 4 & December 25 marked by vertical dotted lines showing two dips since more people travel before or after holidays.

EDA - Airlines Dataset

Airline Carrier & Origin Airport Factors



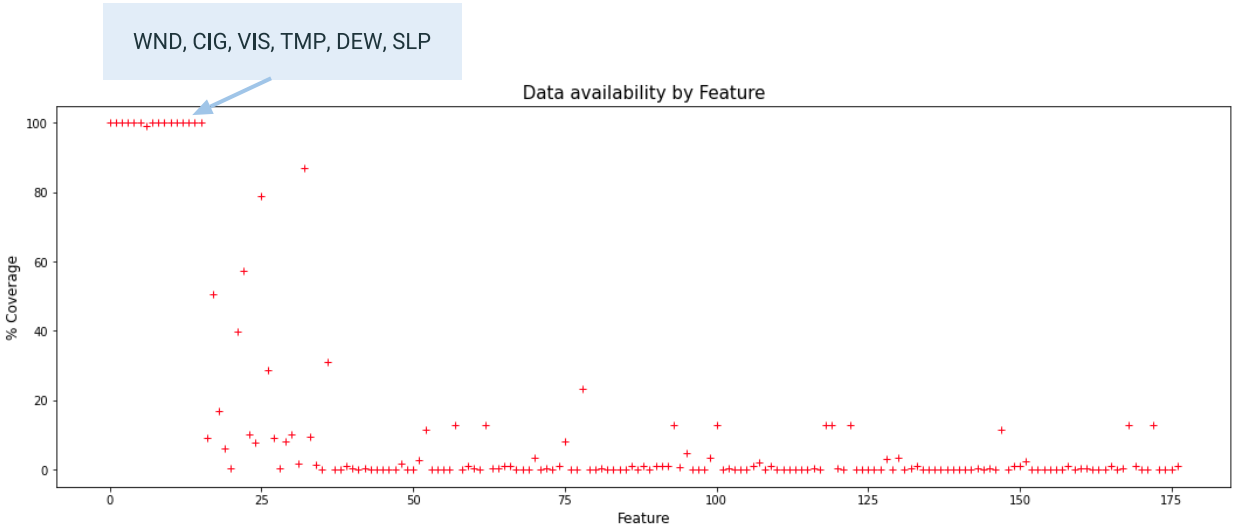


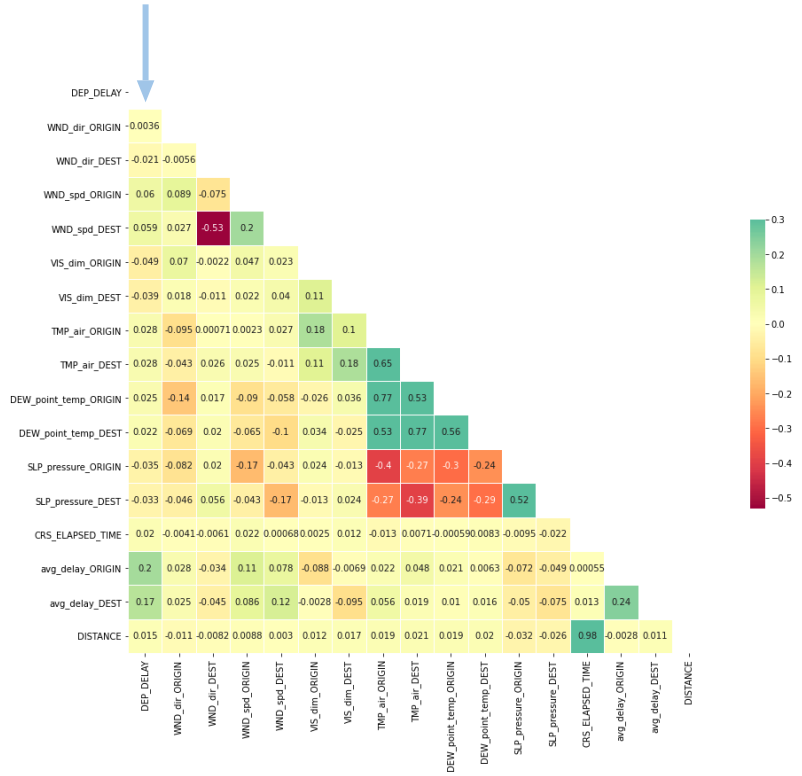
Slightly increasing linear relationship

Busyness of an airport and the proportion of its flights which are delayed. Explore average delay by airport prior to flight departure.

Number of delays is associated with airline carriers

Southwest Airlines (WN) > American Airlines (AA) > Delta Airlines (DL) > SkyWest Airlines (OO) > United Airlines (UA)





EDA - Weather Dataset

Date Range: 2015-01-01 to 2019-12-31

1. Total row count: 630,904,436

   Contains global weather data outside of U.S., which are dropped when joining on IATA code and time stamp for U.S. flights only

2. Total columns count: 175

   After the ~15th feature significant number of features have extremely sparse coverage (empty data value)

   X-axis is the index of the features vs. Y-axis is the percent of data coverage.

3. Number of Unique Weather Stations: 15,195

   Join on Station and enrich with IATA

4. Correlation matrix on Joined datasets

   Numeric Weather Features vs. `DEP_DELAY` in minutes

Feature Engineering

Average Delay (2 - 6 hours before departure)

```
def get_avg_delay(flight_data):
  '''
  Flight_data is assumed to have departure time in utc and truncated down to nearest hour
  Output is a spark dataframe with schema: ORIGIN, 6_hour_before_departure, 2_hour_before_departure, avg_delay

  Join the original flight data with output spark data frame by ORIGIN, 6_hour_before_departure, 2_hour_before_departure

  '''

  transformed_flight_data = flight_data.withColumn('6_hour_before_departure', flight_data['DEPT_UTC_HOUR'] - expr('INTERVAL 6 hours'))\
                                       .withColumn('2_hour_before_departure', flight_data['DEPT_UTC_HOUR'] - expr('INTERVAL 2 hours'))
  transformed_flight_data.createOrReplaceTempView('flight_temp')

  delay_df = spark.sql('''
  select f2.ORIGIN, f2.6_hour_before_departure, f2.2_hour_before_departure, avg(f1.DEP_DELAY) as avg_delay
  from flight_temp as f1
  inner join flight_temp as f2 on (f1.DEPT_UTC_HOUR between f2.6_hour_before_departure and f2.2_hour_before_departure) and (f1.ORIGIN = f2.ORIGIN)
  group by 1,2,3
  order by 1,2,3
  '''
  )
```

Flight delays can have an accumulative effect, meaning if there are any flight delayed in an airport, later flights depart from the same airport can be impacted as well.

Calculate the average delay (2 - 6 hours before departure) at an airport (with feature name "avg_delay").

Output produces a Spark DataFrame with schema: 6_hour_before_departure, 2_hour_before_departure, avg_delay_ORIGIN, avg_delay_DEST

Feature Engineering

Prior Flight Delay & Holiday Indicator

1. Previous Flight Delay: Tracked airplanes by tail numbers and arrival times to determine if the previous flight was delayed.
   - Set to "1" if the previous flight was delayed and "0" if it was not delayed. Null values represent the flight without prior itinerary.
2. Likelihood for Potential Delay: More likely for a flight to get delayed if the airplane has not arrived 2 hours before departure time.
   - Create an indicator where flights with more than 2 hours gap in between flights were indicated with a 0 and less than 2 hours were indicated with a 1.
3. Holiday Indicator: Denote whether the flight date is a public holiday or close to the holiday (e.g. before and after July 4th).

```
# First filter for rows where actual arrival date is greater than departure date
data = data.withColumn("ARR_UTC", f.when((data.ARR_UTC < data.DEPT_UTC),(f.from_unixtime(f.unix_timestamp('DEPT_UTC') + (data.ACTUAL_ELAPSED_TIME*60)))).otherwise(data.ARR_UTC))

# Create tail number group by actual arrival time
tail_group = Window.partitionBy('tail_num').orderBy('ARR_UTC')

data = data.withColumn('prev_actual_arr_utc', f.lag('ARR_UTC',1, None).over(tail_group))\ # prior actual arrival time of each flight
            .withColumn('prev_fl_del', f.lag('DEP_DEL15',1, None).over(tail_group)) # flag for 1 if previous flight is delayed for the same airplane (identified by tail number)

data = data.withColumn("planned_departure_utc", col("DEPT_UTC") - (col("DEP_DELAY") * expr("Interval 1 Minutes")))\
            .withColumn('inbtwn_fl_hrs', (f.unix_timestamp('planned_departure_utc') - f.unix_timestamp('prev_actual_arr_utc'))/60/60)\ # Calculate the hours in between prior actual arrival time
and planned departure time
            .withColumn('poten_for_del', expr("CASE WHEN inbtwn_fl_hrs > 2 THEN '0'" + "ELSE '1' END")) # Categorize flight gap (>2 hours = 0, < 2 hours = 1) Has the airplane arrived 2 hours
before departure? Simplify to 1 if airplane is in the airport less than 2 hours before departure, otherwise 0 if not or null.
```

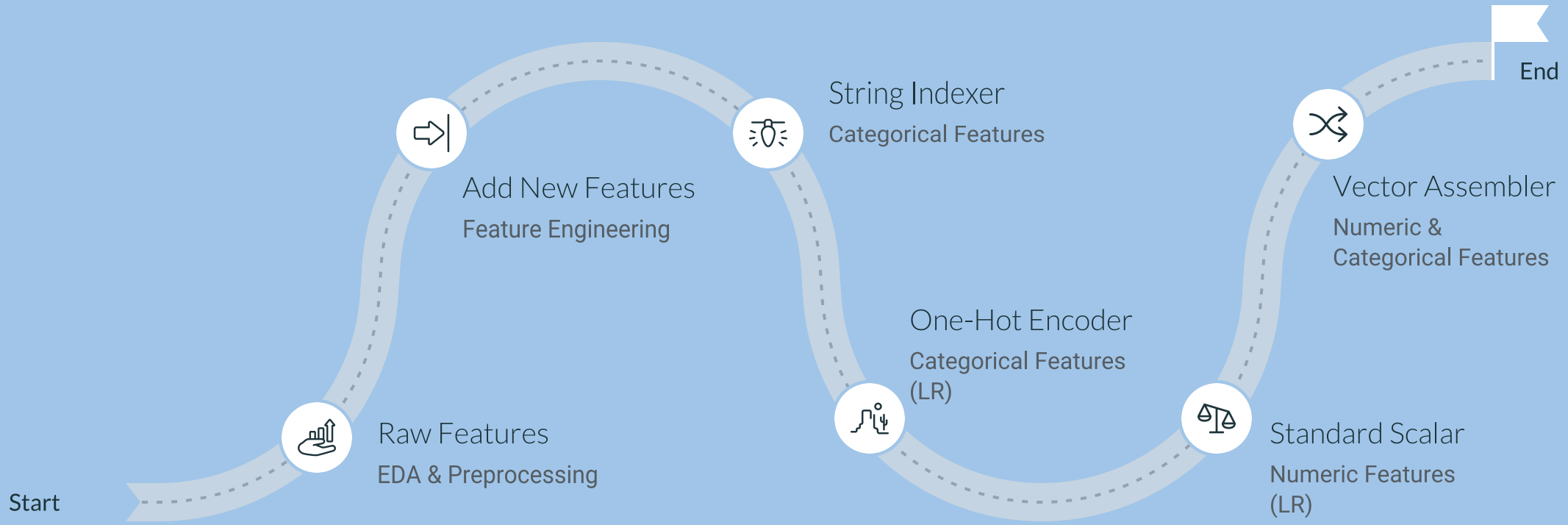
Feature Selection

Used to train our models

Numeric Features	Categorical Features
WND_dir: Wind Direction in angular degrees	WND_type: Wind type characteristic (Calm = C, Normal = N)
WND_spd: Wind speed in meters per second	VIS_var: Visibility variability (No = N, Variable = V, Missing = 9)
VIS_dim: Horizontal distance visibility in meters	OP_CARRIER: Airplane carrier
TMP_air: Air temperature in degree Celsius	'QUARTER', 'MONTH', 'DAY_OF_MONTH', 'DAY_OF_WEEK': Time related factors
DEW_point: Air temperature in degrees Celsius	STATE_ABR: State abbreviation
SLP_pressure: Sea-level pressure	prev_fl_del: Prior flight delay [New feature]
CRS_ELAPSED_TIME: Scheduled Air Time	poten_for_del: Potential for delay [New feature]
avg_delay: Average delay (minutes) 2-6 hours [New feature]	holiday: Holiday indicator [New feature]
DISTANCE: Distance between origin and destination	local_departure_hour: Local departure hour

Feature Pipeline

Feature Transformation



Algorithm & Model Pipeline

Time-Series, Cross-Validated GridSearch

```python
from pyspark.ml.classification import GBTClassifier

# Parameters to gridsearch:
grid_params = {
    "maxIter": [25, 75],  # Number of gradient boosting iterations.
    "minInstancesPerNode": [1, 5],  # Minimum number of instances each leaf node must have.
    "maxDepth": [1, 2, 4],  # Maximum depth of each tree (max allowed is 30).
}

# Parameters to use in every model version:
static_params = {
    "maxBins": 100,  # Must be more than 52 - number of categories in the state features.
    "stepSize": 0.1,  # Learning rate.
}

gscv = SparkGridSearchTimeSeriesCV(
    model_class = GBTClassifier,  # The estimator class.
    train = train,  # The full training dataset.
    n_cv = 4,  # The number of cross-validation splits (will train n_cv-1 models).
    numeric_features = numeric_features,  # Column names of numeric features to use.
    categorical_features = categorical_features,  # Column names of categorical features to use.
    one_hot_encode_features = one_hot_encode_features,  # Column names of categorical features to also one-hot-encode.
    grid_params = grid_params,  # Dict of parameter space to be searched.
    static_params = static_params,  # Additional static parameters to use in every version of the model.
    target = target,  # Name of column being predicted.
    class_weights = True,  # Whether to reweight class imbalance.
    scale_features = False,  # Whether to scale numeric features to mu=0, sigma=1.
    handle_invalid = "keep",  # What to do with Spark feature class errors.
)

# Train the models:
gscv.run()

# Save the full results to CSV:
gscv.save_results()
```

CV Iteration	2015	2016	2017	2018
1	Train	Test		
2	Train	Train	Test	
3	Train	Train	Train	Test

```
CV iteration 1:
  Train: Jan 01 2015 - Dec 31 2015
  Test:  Jan 01 2016 - Dec 30 2016
    Model 1: {maxIter': 50, 'minInstancesPerNode': 1}
    Model 2: {'maxIter': 50, 'minInstancesPerNode': 5}
    Model 3: {'maxIter': 75, 'minInstancesPerNode': 1}
    Model 4: {'maxIter': 75, 'minInstancesPerNode': 5}
CV iteration 2:
  Train: Jan 01 2015 - Dec 30 2016
  Test:  Dec 31 2016 - Dec 30 2017
    Model 1: {maxIter': 50, 'minInstancesPerNode': 1}
    Model 2: {'maxIter': 50, 'minInstancesPerNode': 5}
    Model 3: {'maxIter': 75, 'minInstancesPerNode': 1}
    Model 4: {'maxIter': 75, 'minInstancesPerNode': 5}
CV iteration 3:
  Train: Jan 01 2015 - Dec 30 2017
  Test:  Dec 31 2017 - Dec 30 2018
    Model 1: {maxIter': 50, 'minInstancesPerNode': 1}
    Model 2: {'maxIter': 50, 'minInstancesPerNode': 5}
    Model 3: {'maxIter': 75, 'minInstancesPerNode': 1}
    Model 4: {'maxIter': 75, 'minInstancesPerNode': 5}
```

Algorithm & Model Pipeline

GridSearch Results

Single GridSeach Results

model	n_cv	cv_iter	model_num	training_time	parameters	static_params	tp	fp	tn	fn
class 'pyspark.ml.classification.GBTClassifier'	4	1	0	59.009451	{'maxDepth': 2, 'maxIter': 5}	{'maxBins': 100, 'stepSize': 0.1}	391364	182355	4348553	543494
class 'pyspark.ml.classification.GBTClassifier'	4	1	1	52.663468	{'maxDepth': 2, 'maxIter': 10}	{'maxBins': 100, 'stepSize': 0.1}	391364	182355	4348553	543494
class 'pyspark.ml.classification.GBTClassifier'	4	1	2	50.094818	{'maxDepth': 4, 'maxIter': 5}	{'maxBins': 100, 'stepSize': 0.1}	389248	177262	4353646	545610
class 'pyspark.ml.classification.GBTClassifier'	4	1	3	66.446158	{'maxDepth': 4, 'maxIter': 10}	{'maxBins': 100, 'stepSize': 0.1}	376061	161677	4369231	558797

Aggregated Results

model	parameters	static_params	class_weights	scale_features	precision	recall	accuracy	fbeta	training_time
RandomForest	{'maxDepth': 7, 'numTrees': 40}	{'weightCol': 'weight', 'maxBins': 100}	True	False	0.45526	0.6087	0.755094	0.570186	317.981763
GBTClassifier	{'maxDepth': 2, 'maxIter': 75, 'minInstancesPerNode': 1, 'stepSize': 0.1}	{'maxBins': 100, 'weightCol': 'weight'}	True	False	0.436875	0.632833	0.7307	0.58064	278.528441
LogisticRegression	{'elasticNetParam': 0.1, 'regParam': 0}	{'weightCol': 'weight', 'maxIter': 10}	True	True	0.410032	0.631106	0.704896	0.569655	168.922514

Algorithm & Model Pipeline

Training on full data & Ensemble

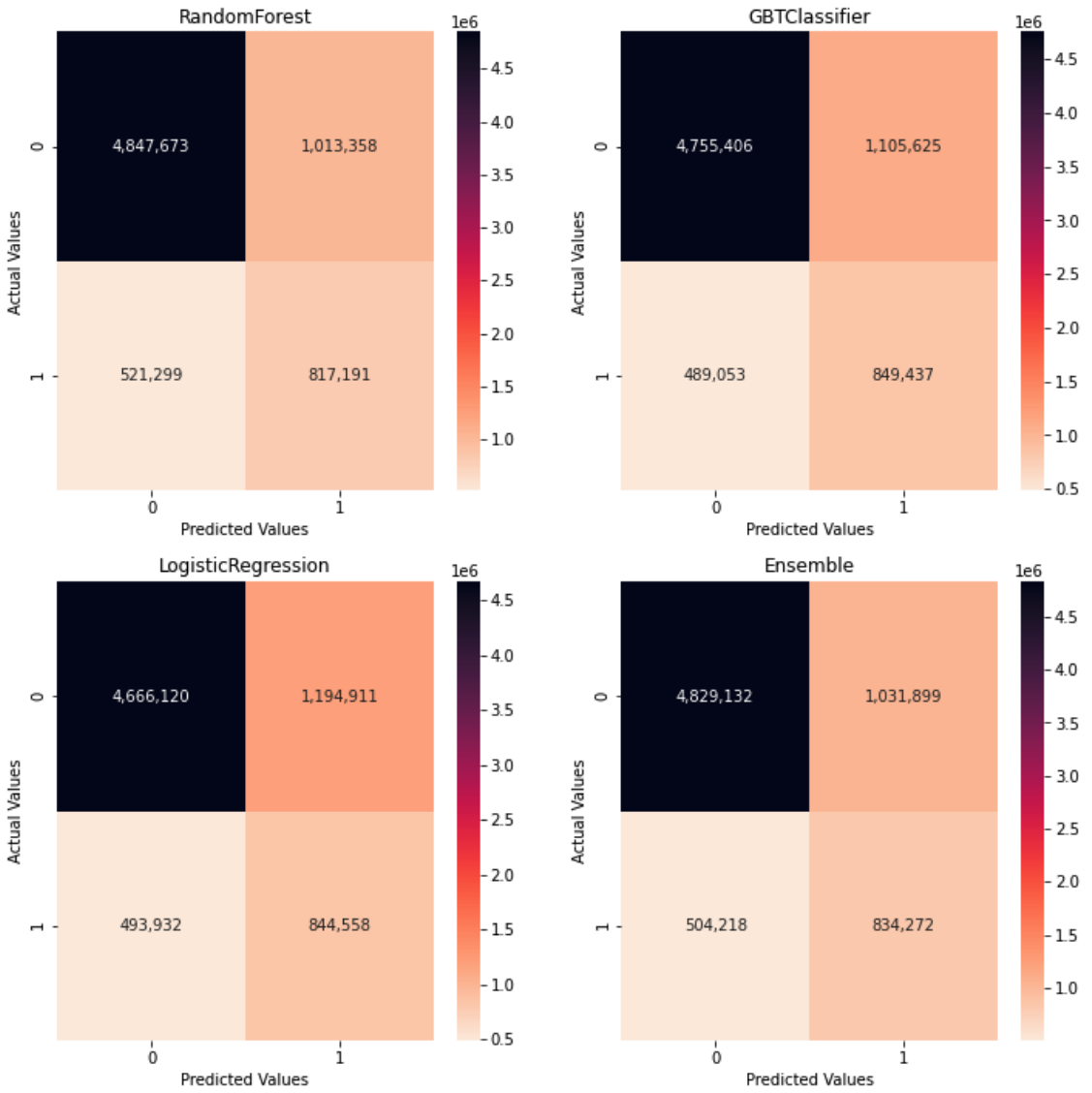
```
if __name__ == "__main__":

    # Merge and deduplicate all the intermediate results files saved in the grid-searches:
    deduplicate_gridsearch_results()

    # List all the merged grid-search results files which will be compared to identify the best performing models:
    gridsearch_result_csvs = [f for f in os.listdir(LOCAL_CSV_DIR) if f.endswith(".csv")]

    # Choose the best performing model for each classifier based on F-Beta score:
    models_ranked: pd.DataFrame = compare_models(*gridsearch_result_csvs)

    # Create the Ensemble class. Trains an instance of each of the best performing models on the full training data and
    # uses it to make predictions on the held out test data. Predictions are summed to get the majority vote which is
    # the ensemble model's prediction (1 if sum is 2 or greater, else 0).
    # Model instances are accessible in the `pipelines` attribute of the Ensemble class.
    ensemble = Ensemble(models_ranked, train_name="ML_train_filled", test_name="ML_test_filled")

    # Summarize the final test set results for all 3 models and the ensemble:
    final_results = ensemble.final_table(beta=2.0)
```

Evaluation Metrics

F-Beta and Confusion Matrix

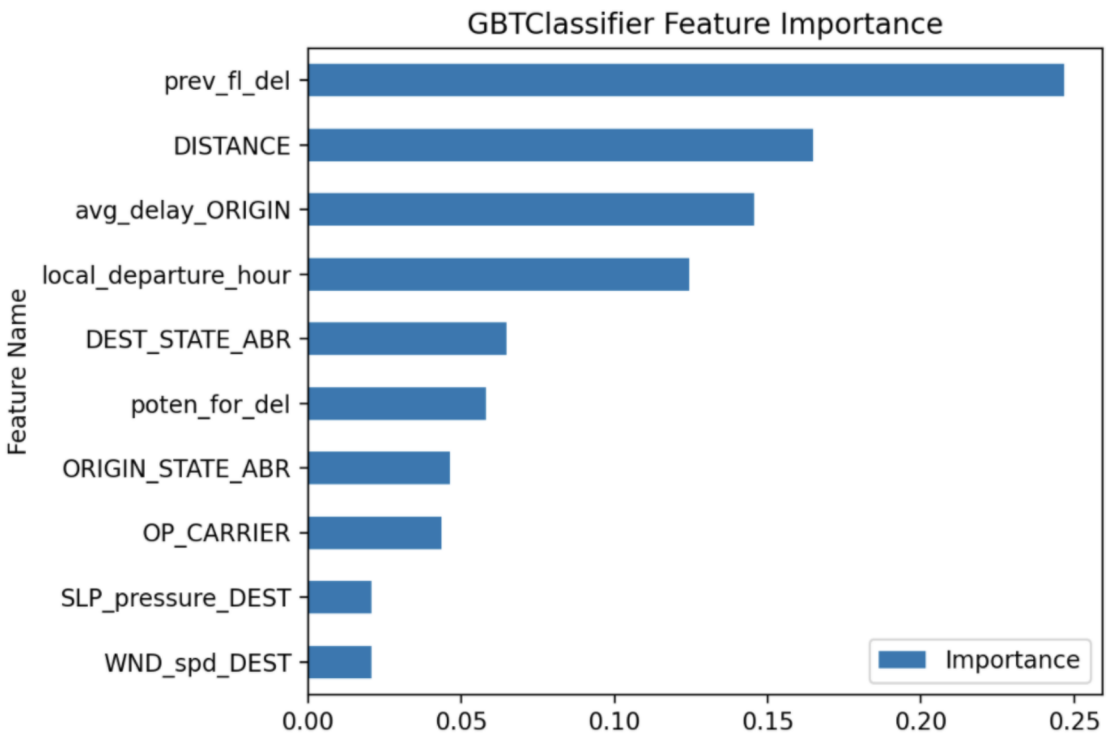
Cross Validation vs. Test Set

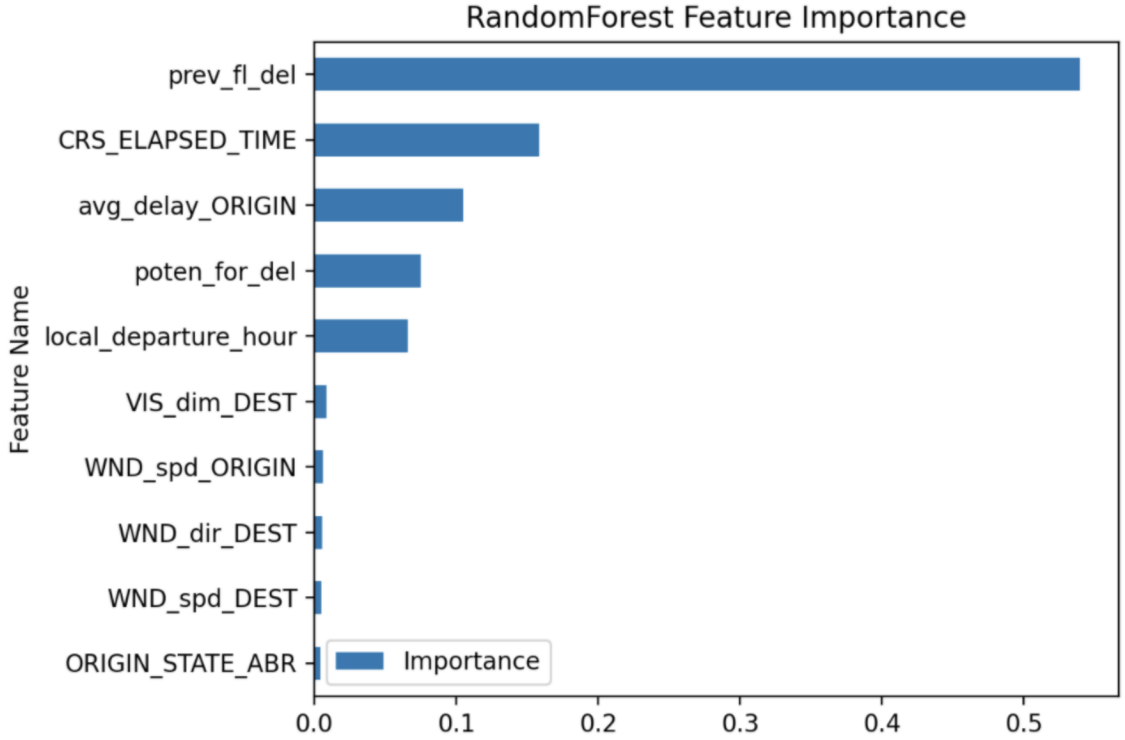
Model	Average CV fbeta	Test fbeta
RandomForest	0.570	0.569
GBTClassifier	0.581	0.581
LogisticRegression	0.570	0.571



Novel Findings

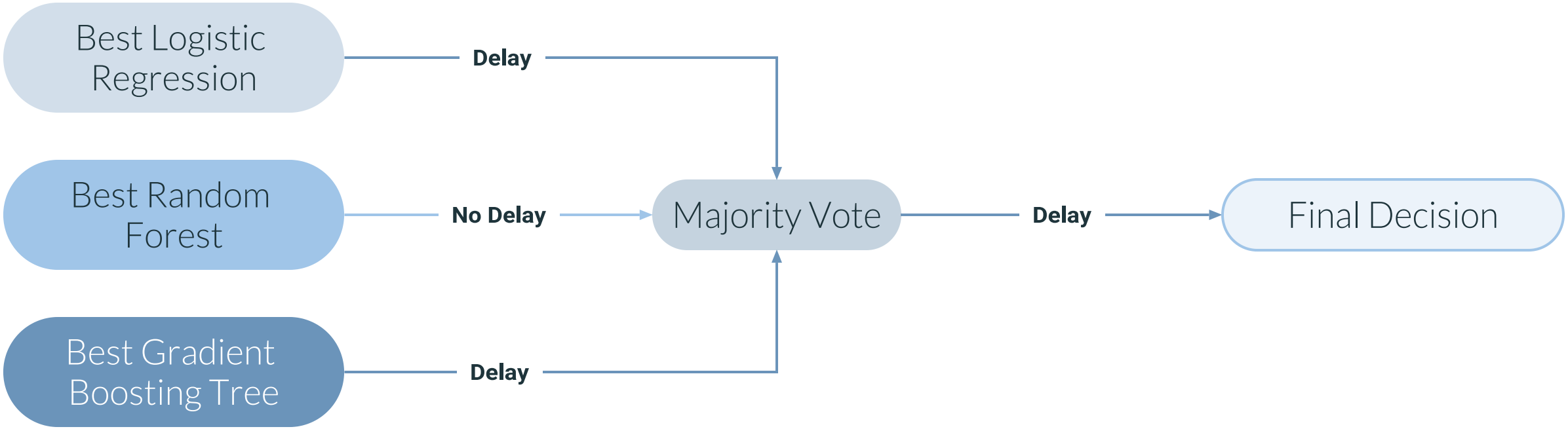
- **Feature Importance (Top 10)**





Novel Findings

- **Ensemble Method**



Conclusions

	tp	tn	fp	fn	Accuracy	Precision	Recall	F-beta (beta=2.00)	Rank
RandomForest	817,191	4,847,673	1,013,358	521,299	0.736502	0.446419	0.610532	0.568717	4
GBTClassifier	849,437	4,755,406	1,105,625	489,053	0.71709	0.434481	0.634623	0.581088	1
LogisticRegression	844,558	4,666,120	1,194,911	493,932	0.69751	0.414107	0.630978	0.571154	3
Ensemble	834,272	4,829,132	1,031,899	504,218	0.732918	0.44705	0.623293	0.57774	2

- GBT performs the best
- Ensemble achieves the second best fbeta and highest precisio
- Airline companies can use the model to inform passengers of any possible delay in advance so they can plan ahead and avoid losses.
- Limitations
  - Performance and scalability concerns

```
Training RandomForest
Start training on entire training dataset: ML_train_filled
Best params = {'maxDepth': 7, 'numTrees': 40, 'weightCol': 'weight', 'maxBins': 100}
--- 341.40 seconds ---
Training GBTClassifier
Start training on entire training dataset: ML_train_filled
Best params = {'maxDepth': 2, 'maxIter': 75, 'minInstancesPerNode': 1, 'stepSize': 0.1, 'maxBins': 100, 'weightCol': 'weight'}
--- 460.47 seconds ---
Training LogisticRegression
Start training on entire training dataset: ML_train_filled
Best params = {'elasticNetParam': 0.1, 'regParam': 0, 'weightCol': 'weight', 'maxIter': 10}
--- 158.34 seconds ---
RandomForest: Start predicting on test dataset: ML_test_filled
--- 15.21 seconds ---
GBTClassifier: Start predicting on test dataset: ML_test_filled
--- 12.76 seconds ---
LogisticRegression: Start predicting on test dataset: ML_test_filled
--- 19.29 seconds ---
```

Thank you!

Thoughts? Questions?

Open up to general feedback and discussion.

Appendix

2019 Test Set Results

	tp	tn	fp	fn	Accuracy	Precision	Recall	F-beta (beta=2.00)	Rank
RandomForest	817,191	4,847,673	1,013,358	521,299	0.736502	0.446419	0.610532	0.568717	4
GBTClassifier	849,437	4,755,406	1,105,625	489,053	0.71709	0.434481	0.634623	0.581088	1
LogisticRegression	844,558	4,666,120	1,194,911	493,932	0.69751	0.414107	0.630978	0.571154	3
Ensemble	**834,272**	**4,829,132**	**1,031,899**	**504,218**	**0.732918**	**0.44705**	**0.623293**	**0.57774**	2