Prediction of Flight Delays at Scale

Written By: *Mai La, Uzair Mansuri*

Business Context

Flight delays has become a very important subject for air transportation all over the world because of the associated financial loses that the aviation industry is going through. Flight delays can have an impact on airlines, airports & travelers. Flight delay is one of the most important Key Performance Indicators (KPI) for any transportation system. Any delay wastes the resources of airports, airlines, and passengers along with adverse environmental impact caused by carbon emissions from an aircraft taxiing on the runway. The goal of our analysis is to develop a model that predicts flight departure delays to help optimize airline and airport operations while reducing the impact related to time delays and resources. To achieve our goal, we will be looking at airline, weather, and station data to build a predictive model that will focus on predicting whether or not a flight will be delayed by at least 15 minutes. Machine Learning (ML) model with the ability to predict flight delays will be implemented by using provided data at scale. From a business perspective, predicting ~60% flight delays with high precision will favor our primary customers: Airlines. In our analysis, if we mark a flight as delayed that would have departed on time otherwise, is considered as False Positive (FP). If a flight is marked as "*on-time*" that would've not departed on time, it would be considered False Negative (FN). In this case, we lose the opportunity to target a flight and subsequently improve the delay performance. We believe that a False Positive (FP) will incur higher cost to an airline customer compared to a False Negative (FN), since an airline would've allocated costly resources on wrong flight and take away useful resources from other delayed flights that has the potential of being delayed in the future.

Dataset Overview

Three different large datasets are provided to us. The first one is on-time performance data taken from the TranStats data collection available from the U.S. Department of Transportation (DOT). The second one is the weather dataset which was downloaded from the National Oceanic and Atmospheric Administration repository. The last set is the Airport dataset downloaded from the US Department of Transportation. We will be using the information for flight and weather from 2015 to 2021.

Airlines Dataset – Bureau of Transportation Statistics

This dataset contains about ~31 million airline trips from an origin airport to a destination airport for various airports and carriers between 2015 and 2021. Contains attributes such as the date of the flight, the scheduled vs actual departure and arrival times, tail number of aircraft, etc.

Weather Dataset – National Oceanic and Atmospheric Administration (NOAA)

This dataset contains weather recordings at various time intervals for weather stations across many geographic locations from 2015 - 2021. As weather recordings can be taken as often as every several minutes, we can find what the weather was like around the time of a flight's actual departure time. This dataset contains various weather attributes such as temperature, precipitation rate, rain, snow. Each attribute has a 'missing' value to show that the recorded value for a given attribute is unavailable.

Stations Dataset – Weather station information

This dataset contains information on various weather stations recording weather information such as location of the station, station id. Necessary link between Airlines and Weather data (details in subsequent Feature Engineering section). We applied filter for US only weather stations during our join strategy since the Airlines dataset contains only domestic US based flights.

Throughout this project, we focused on Departure Delays (as opposed to Arrival Delays), since our initial EDA showed that arrival delays were less common than departure delays. This intuitively makes sense as airlines can adjust the speed of the aircraft if they are running late to ensure they do not arrive at the destination late. Additionally, arrival delays are dramatically impacted by larger scale weather patterns (i.e. headwind/tailwind) that are not captured in our weather dataset.

To start the project, we will be using the provided airlines and weather database along with Airport and Airline data from openflights.org for time zones and geographic locations. Columns within airlines and weather that have more than 80% of null input and columns that is not needed for predicting departure times will be dropped from the analysis. There are a number of features in the Airlines dataset that are gathered during departure as listed below, which are highly correlated with departure delays. However, since these features would not be available to our model prior to departure, they cannot be used for inference. For this reason, we have excluded these features from our model.

- Taxi Out
- Wheels Off
- Wheels On
- Taxi In
- Arrival Delay
- Actual Elapsed Time

We will also use an external dataset the from openflight.org

Primary Evaluation Metric

We choose F measure with beta = 0.5 (F0.5) as our primary metric for the model, and we will also look at precision and recall as secondary metrics. Using F0.5 allows us to give higher cost penalty to False Positive, since it could highly impact both the airlines as well as the customers. For example, if we falsely predict flights that are not delayed as delayed, the airlines could end up spending major cost for resource reallocation that are not correct. For the customers, they could arrive at the airport late if they think their flights would be delayed, and therefore could miss their flights. Weighted F0.5

score is also suitable for models with imbalanced data, since it does not include the True Negative (the majority class) in its calculation.

$$F0.5\ Score = \frac{1.25 \cdot Precision \cdot Recal}{0.25 \cdot Precision + Recall}$$

$$Precision = \frac{TruePositive}{TruePositive + FalsePositive}$$

$$Recall = \frac{TruePositive}{TruePositive + FalseNegative}$$

Join Strategy

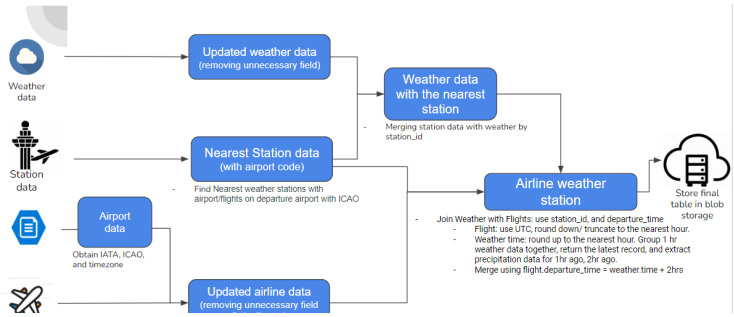
For executing our Join efficiently across three different datasets, we constructed a time window for each flight from two to four hours before the flight's scheduled departure time to combine the flight and weather information and set the window end to two hours before a flight's scheduled departure time to avoid leakage in our time-series weather data. We also chose the start of specific time window by considering the average time interval between different weather observations and how much the dataset would grow if we are joining multiple weather observations to each flight. We joined the weather data in this time window from the nearest weather stations to a flight's departure airport onto each flight using the ICAO code, giving a complete picture of the weather conditions around each flight in our dataset. The weather reports' time and distance restraints gave insight on practical meaning and resource availability, including weather observations that would reflect the conditions at the airport while trying to keep the performance costs low for the full data join. We've listed key steps in our join strategy below:

Full Data Joining Pipeline:

1. Flights with airport location: use IATA code
2. Nearest weather stations with airport location (departure airport): use ICAO
3. Weather with nearest stations: inner join to remove irrelevant data (i.e. keep US weather data only), use station_id
4. Weather with flights: use station_id, and departure_time
    - Flight: convert to UTC, Round down or truncate flight to the nearest hour.

- Weather time round up to the nearest hour. Group 1hr weather data together, return the latest record, and extract precipipatation data for 1hr ago, 2hr ago before the weather time as potential additional features.
- Merge using flight.departure_time = weather.time + 2hrs

Detailed pipeline is as follows:



Summarize EDA and Feature engineering

All the datasets provided required substantial amount of cleanup and review. Weather dataset in particular required most cleansing due to its size and number of features that were not readily usable without further imputation. After joining our datasets, it became apparent that null and/or missing values would drive majority of our EDA effort. After iterating on our joining methodology in Phase 2, we were able to reduce the number of null/missing values in our final joined dataset. However, despite reducing the overall number of null values, we still had a significant proportion of observations with at least one null/missing value. It's important to note that these null/missing values did not take into consideration the values that had been reserved for invalid or irregular observations (e.g. weather data with categorical variable values of '9' or continuous variable values of '99999'). This aspect of our dataset was one of the most significant challenge that would require imputation before feeding it into modeling pipeline.

Feature Engineering

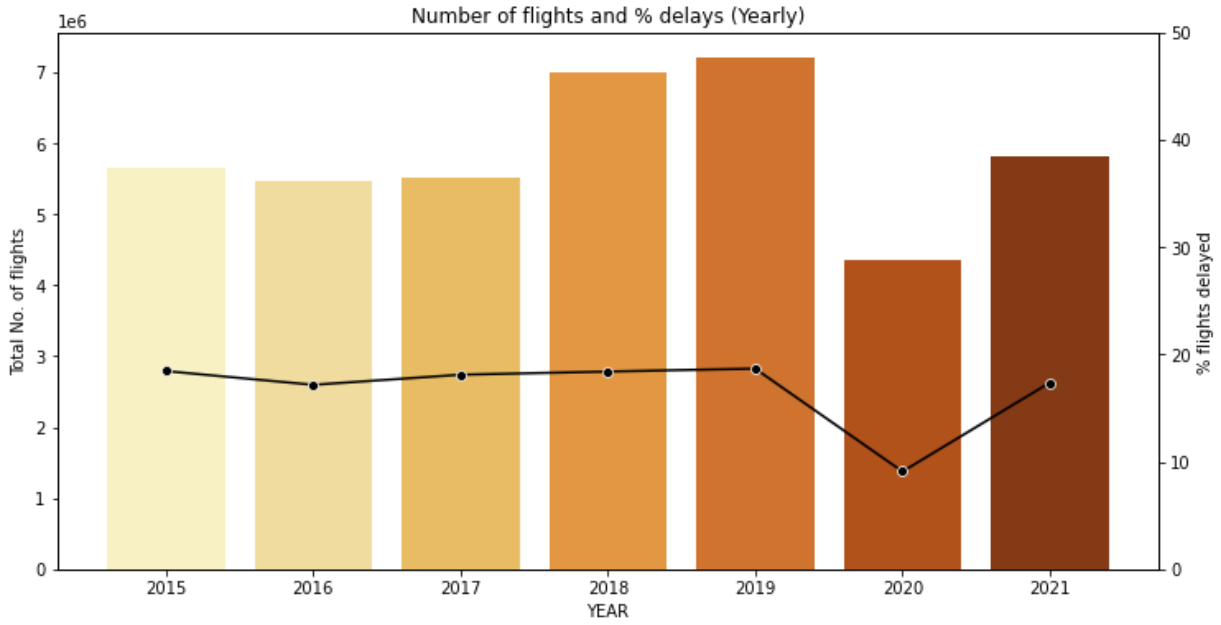
The following features were created and used in our model data:

1. Previous flight delay indicator time feature. This feature shows whether an airplane (using `TAIL_NUM`) was delayed in its previous flight, denoted by `PREV_DEP_DEL15`
2. Percent of delayed flights grouped by airport (`ORIGIN`) and airline (`OP_UNIQUE_CARRIER`) for the last 5 to 20 flights.
3. Ranking of airports that have the most outbound flights and connections to the other airports in the previous year, extracted by PageRank. This indicator represents the importance of an airport in the network. 2016 to 2021 uses pagerank number of the previous year, 2015 uses pagerank number of the same year.
4. Covid: External dataset with number of active COVID cases by state in the previous month.
5. Holiday indicator: External dataset with return indicator for US national holidays within +/- 2 days before and after each US public holidays.
6. Time feature: Created time groups for the planned departure time, i.e., morning (0-9am), late morning (9am-12pm), noon (12-16pm), afternoon (16-20pm), evening (20pm-0am).
7. Weather features: extracted current weather condition from text, create dummy variables for each weather condition (i.e., blowing snow, freezing rain, tornado, thunder, etc.), and sum up these events as an indicator of severe weather at orgin and destination airport.

Number of Flights per year

We started our EDA process by looking at how many flights occured in the US per year from 2015-2021 and what percentage of those flights were delayed. Figure 1 below highlights this information:

Figure 1: Number of flights per year

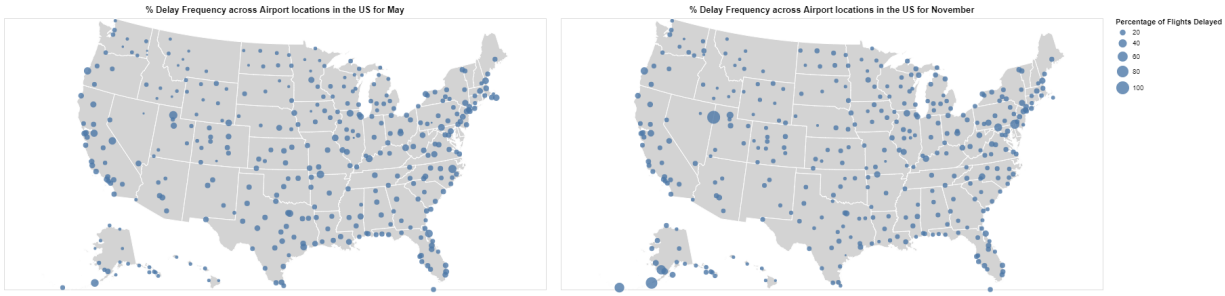


As seen from the plot, the average percentage of flights delayed from 2015 - 2019 is pretty consistent between ~ 16-19%, which matches the statistics shared by US Federal Aviation Administration (FAA) Data source (https://www.transtats.bts.gov/homedrillchart.asp). Additionally, the total number of flights and subsequent delays were signigicantly less in 2020, which makes sense due to the onset of COVID impacting domestic travel.

Flight delays across US

Next, we were interested in understanding the percent delays across different US airports each month to identify which airports are most impacted by delays. We were also able to visualize delays at major airports during peak travel months of summer and holiday season. Figure 2 shows percent delays across US airports for the month of May and November.

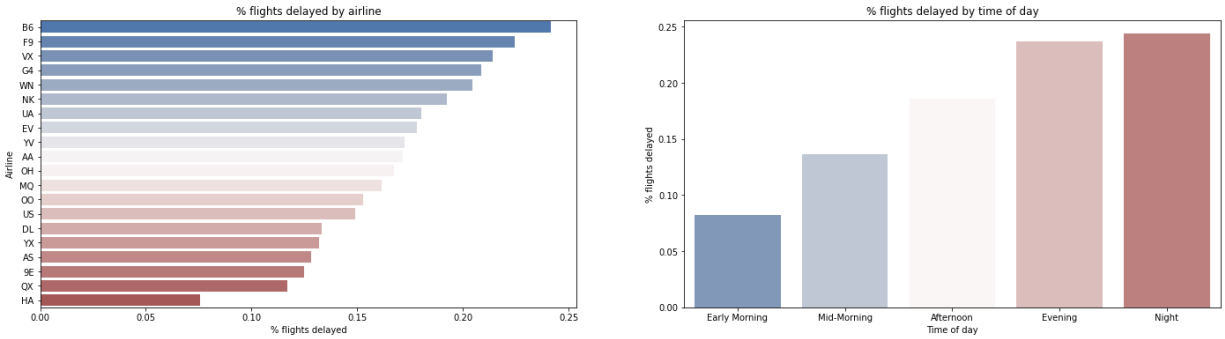
Figure 2: Percent delays across US airports



Flight delays by airline and time of day

Our team was also interested in understanding percentage of flights delayed by airline and time of day in order to gain some useful insights to help us prepare for our modeling pipeline. Figure 3 shows percentage flights delayed across different airlines and time of day when they are delayed.

Figure 3: Percent delays across airlines and time of day



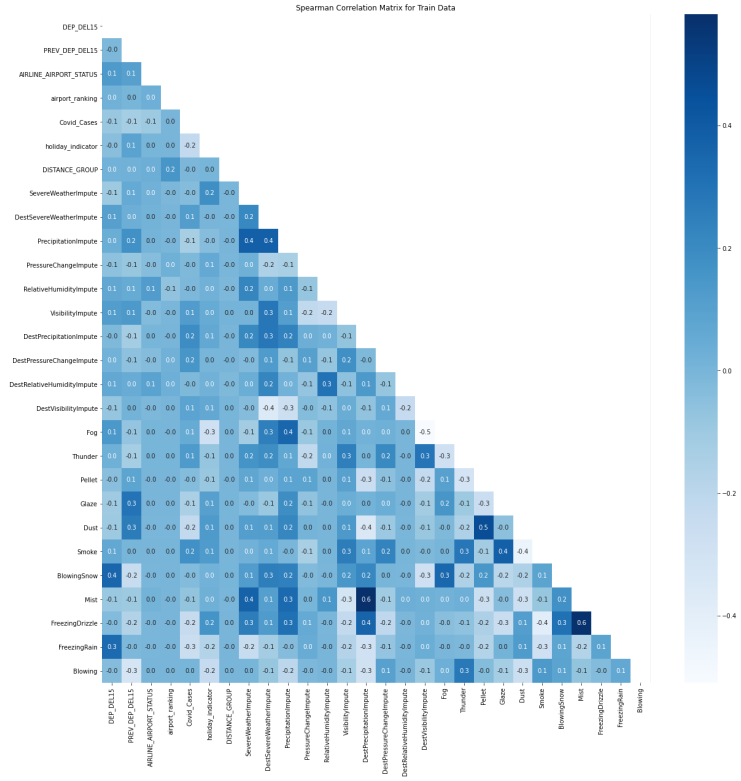
As seen from the visualization above, JetBlue, Frontier and Virgin America has the highest flight delay percentage compared to all other airlines while Hawaiian and Horizon Air (Part of Alaska airlines) has the best on time performance. Additionally, majority of flights are delayed in the afternoon and at night which makes sense since these delays are usually impacted by prior delays for the same flight `TAIL_NUM`. This was an important finding as part of the EDA process as we created an additional feature for prior flight delay that was used in our modeling efforts.

Feature Correlation and Distribution

In order to better understand the relationship between features in the datasaet and dependent variable, we created Spearman correlation matrix as shown in Figure 4, between the target variable `DEP_DELAY15` and the input variables in order to check if there is any non-linear relationship between them. Based on the matrix, majority of our features have low correlation with flight delay, however both weather features blowing snow and freezing rain at the origin airport are highly correlated at 0.3 - 0.4 with the dependent variable, which is an indication of severe weather condition. The rest of features have 0.1 or lower correlation with flight delay. This helped us select these severe weather conditions and useful features for our modeling efforts. The 51

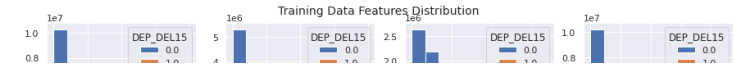
features selected in our model included these variables in the correlation matrix together with some additional categorical features such as day of week, departure month, and departure time block.

Figure 4: Features Spearman Correlation



The feature distribution plot in Figure 5 shows most of our features have Poisson distribution with the exception of relative humidity and pressure change that have quite normal (Gaussian) distribution. We used Min-Max scaler to normalize these data before training different machine learning models, especially for the algorithms using gradient descent for learning. Min-Max scaler also does not change our feature distribution, it just put the variable values within a 0-1 range. This distribution also shows significant imbalance between majority and minority class in our data, which required us to implement oversampling and undersampling techniques to improve the imabalance.

Figure 5: Features Distribution



Algorithm Summary

In this section, we take a closer look at different algorithms we implemented in order to predict departure delays. We will summarize Logistic regression, Random Forest & XGBoost models and underlying mathematics.

1. Logistic Regression

We implemented Logistic Regression as our baseline model for binary classification, with our target variable indicating whether a flight is delayed at least 15 mimutes (y = 1) or not (y = 0). Logistic regression is a simple stochastic model that could effectively classify binary targets that are linearly separatable, therefore it serves well as a baseline model. In order to prepare raw data for logistic regression model, all the categorical columns were encoded using custom functions. For all the categorical features with less categories, we implemented one hot encoding technique. For time series data, custom Cross Validation function was used while ensuring there is no data leakage. We used **Binary Cross Entropy (BXE)** as loss function for this model, given by:

$$J(\theta) = -\frac{1}{m}\sum_{i=1}^{m}(\mathbf{y_i} \cdot log(\hat{\mathbf{y}_i}) + (1 - \mathbf{y_i}) \cdot log(1 - \hat{\mathbf{y}_i}))$$

We also look at Ridge Binary Cross Entropy (RBXE) and Elastic Net loss function for our Logistic Regression Model. The loss function for RBXE function is as follow:

$$J(\theta) = -\frac{1}{m}\sum_{i=1}^{m}(\mathbf{y_i} \cdot log(\hat{\mathbf{p}_i}) + (1 - \mathbf{y_i}) \cdot log(1 - \hat{\mathbf{p}_i})) + \lambda \cdot \sum_{j=1}^{n}\mathbf{w}_j^2$$

2. Random Forest

As part of our experimentation strategy, we wanted to explore if a random forest classifier could achieve better performance with our model when compared to baseline. Random forest is a classification algorithm consisting of many decisions trees. It uses bagging and feature randomness when building each individual tree to try to create an uncorrelated forest of trees whose prediction by committee is more accurate than that of any individual tree. We implemented this model using PySpark's `RandomForestClassifier` in order to test random forest classification on our feature

set. Hyperparameter tuning included experimenting with different number of trees in the forest, and the maximum depth of those trees. Based on our experimentation, we identified that models with more trees and deeper trees outperformed those with less trees and shallow trees, but we had to make a trade-off on performance and runtime. Trees deeper than 20 levels took over two hours to train, and models with more than 50 trees ran into a similar issue.

3. XGBoost

XGBoost (eXtreme Gradient Boost) is a gradient boosted algorithm that is similar to Gradient Boosted Trees, but contains a number of optimizations. XGBoost follows a boosting frameworkby creating trees and iteratively improving them using the residuals from each tree for a new predictor until an optimal solution is reached. However, there are a number of ways that XGBoost optimizes the framework to produce potentially better results, faster. The XGBoost objective function is trying to minimize the difference between the actual label and the predicted label, which is given by:

$$L^{(t)} = \sum l(y_i, \hat{y}_i^{(t-1)} + f_t(x_i)) + \Omega(f_t)$$

4. Multilayer Perceptron (MLP)

Based on the feedforward artificial neural network, the MLP model consists of multiple hidden layers of nodes. Each layer is fully connected to the next layer in the network. We trained MLP models with two different model architectures, a simple and a more complex layers to evaluate the model performance. All the hidden nodes map inputs to outputs by a linear combination of weights with a sigmoid activation function, and the output layer use softmax function. Similar to logistic regression, we use a Log loss function for optimization

Describe novel approaches taken

As part of our continued experimentation, we explored other Machine Learning Models and built the algorithm for Multitask learning and also attempted Ensemble Model.

Multitask learning

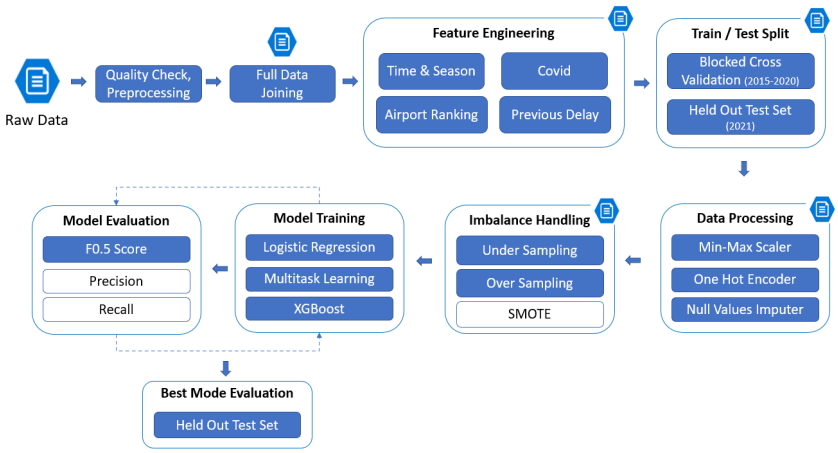
- Model approach: extends binary classification (delay/ not delay) using Logistic Regression combining with continuous value prediction using Linear Regression (the number of delayed minutes). With the intention that learning of delayed minute magnitude could improve the learning of classification, and our model is also less likely to overfit training data.
- Loss function: we use combination loss function for this model. the algorithms was built with different regularization options, including L1, L2 and Elastic Net. For example, the combination loss function with L2 regularization is as follows:

$$J(\theta) = \frac{(1-\alpha)}{m}\sum_{i=1}^{m}(\hat{\mathbf{y}_i}^{(r)} - \mathbf{y_i}^{(r)})^2 - \frac{\alpha}{m}\sum_{i=1}^{m}(\mathbf{y_i}^{(l)} \cdot log(\hat{\mathbf{p}_i}^{(l)}) + (1-\mathbf{y_i}^{(l)}) \cdot log$$

Machine Learning Pipeline

ML Pipeline Flow chart:

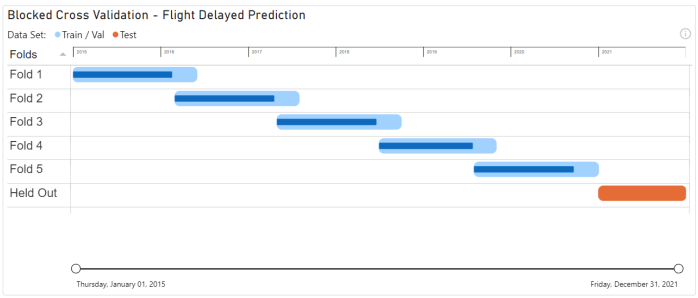
Our Machine Learning Pipeline includes the following steps



Cross Validation Data Splitting:

We use Blocked time-series Cross Validation method to reduce data leakage while training:

- Training & Validation data: we split 2015-2020 data into 5 folds, each fold has 17 months of data (14 months for training, 3 months for validation).
- Test data: 2021 data was held-out for final validation and not used during training.



Data Processing:

The following methods were used for data processing pipeline:

- Imputer: replace with the station's averaging monthly value for weather data that has less than 5% of missing value. Replace with 0 for other weather data that has more than 20% of missing value. We assume that 0 means the event does not happen, i.e. no rain, no snow.
- Categorial One Hot Encoder
- Categorial Feature Hasher to 20 dimensions
- Min Max Scaler
- Variance Threshold Selector for Feature Selection
- PCA for Feature Extraction

Imbalanced Data Handling

Resampling methods are designed to add or remove examples from the training dataset in order to change the class distribution. Once the class distributions are more balanced, it helps our machine learning classification algorithms/models to fit successfully on the transformed datasets

Following methods were used in our project:

- **Undersampling:** The target classes in our dataset are highly imbalanced. The majority of flights in the dataset based on our EDA are not delayed, which means that majority class for our target variable significantly outweighs the minor class. This introduced very high bias into our baseline and initial models towards non-

delayed flight predictions resulting in poor predictive power, since majority of the training examples were representative of non-delayed flights. In order to mitigate this bias, we introduced additional feature engineering into our modeling pipeline to under sample the majority "non-delayed class". We created algorithms for Undersampling that performed under sampling on majority class. After running the models with revised features and Undersampling, we noticed that both precision and recall scores for all our models were better balanced.

- **Oversampling:** We created algorithms that performed over sampling on minority class, and so we do not have to remove data from the majority class. The advantage of this method is having more data for training that could outperform undersampling, more data could also help to reduce underfitting or overfitting problem in our model. However, the challenge of oversampling is big data and so we could not run with heavy grid search cross validation or complex model. To mitigate this, we performed feature selections and only included the most

Discuss performance and scalability concerns

Model Experiments - Hyperparameter Tuning

Cross validation performance summary:

- Baseline Model:

Model	Features	ModelHyperparameter	Cross Validation Performance	Blind Test Set Performance	Training Time
Baseline - Logistic Regression Minimum Feature Engineering	56	No regularization	F0.5 Score: 0.003 Precision: 0.354 Recall: 0.0007	F0.5 Score: 0.005 Precision: 0.37 Recall: 0.001	4.21 mins

- Best Model:

Model	Features	Model Hyperparameter	Cross Validation Performance	Blind Test Set Performance	Training Time
XGBoost Advanced Feature Engineering Undersampling	51	max_depth=5 n_estimators=50	F0.5 Score: 0.71 Precision: 0.80 Recall: 0.48	F0.5 Score: 0.47 Precision: 0.47 Recall: 0.50	26.0 mins

- Models with Undersampling Data & Advanced Feature Engineering:

Model	Features	Best Model Hyperparameter	Cross Validation Performance	Blind Test Set Performance	Training Time
Ridge Logistic Regression	51	regParam=0.0001 threshold=0.5	F0.5 Score: 0.68 Precision: 0.75 Recall: 0.50	F0.5 Score: 0.43 Precision: 0.41 Recall: 0.53	5.21 mins
Elastic Net Logistic Regression	51	regParam=0.0001 threshold=0.65 elasticNetParam=0.3	F0.5 Score: 0.68 Precision: 0.77 Recall: 0.46	F0.5 Score: 0.45 Precision: 0.44 Recall: 0.49	7.93 mins
XGBoost	**51**	**max_depth=5 n_estimators=50**	**F0.5 Score: 0.71 Precision: 0.80 Recall: 0.48**	**F0.5 Score: 0.47 Precision: 0.47 Recall: 0.50**	**26.0 mins**
Random Forest	51	max_depth=20 n_estimators=100	F0.5 Score: 0.71 Precision: 0.81 Recall: 0.48	F0.5 Score: 0.48 Precision: 0.48 Recall: 0.48	2.25 hrs

Model	Features	Best Model Hyperparameter	Cross Validation Performance	Blind Test Set Performance	Training Time
Multitask Learning	51	learningRate=0.5 alpha=0.7 regParam=0.0001 elasticNetParam=0.3 earlyStopping=0.5 nSteps=60	F0.5 Score: 0.62 Precision: 0.64 Recall: 0.54 RMSE=64.9	F0.5 Score: 0.29 Precision: 0.25 Recall: 0.57 RMSE: 48.0	13.02 hrs

- Models with Oversampling Data & Advanced Feature Engineering:

Model	Features	Best Model Hyperparameter	Cross Validation Performance	Blind Test Set Performance	Training Time	Ev Ti
Ridge Logistic Regression	51	regParam=0.0001 threshold=0.5	F0.5 Score: 0.67 Precision: 0.73 Recall: 0.49	F0.5 Score: 0.43 Precision: 0.41 Recall: 0.53	16.80 mins	7.
Random Forest	51	max_depth=15 n_estimators=30	F0.5 Score: 0.69 Precision: 0.74 Recall: 0.49	F0.5 Score: 0.55 Precision: 0.70 Recall: 0.29	4.12 hrs	12 m
XGBoost Top 10 Most Important Features (1 categorical feature)	**14**	**max_depth=5** **n_estimators=30**	**F0.5 Score: 0.69** **Precision: 0.80** **Recall:0.45**	**F0.5 Score: 0.48** **Precision: 0.48** **Recall: 0.49**	**33.6 mins**	**3.**

Model	Features	Best Model Hyperparameter	Cross Validation Performance	Blind Test Set Performance	Training Time	Ev Ti
Multilayer Perceptron 14 - 2 - sigmoid - 2 - softmax Top 10 Most Important Features (1 categorical feature)	14	blockSize=256 stepSize=0.05	F0.5 Score: 0.70 Precision: 0.77 Recall: 0.50	F0.5 Score: 0.47 Precision: 0.46 Recall: 0.53	24.9 mins	3.
Multilayer Perceptron 14 - 8 - sigmoid - 4 - sigmoid - 2 - softmax Top 10 Most Important Features (1 categorical feature)	14	blockSize=128 stepSize=0.05	F0.5 Score: 0.70 Precision: 0.77 Recall: 0.50	F0.5 Score: 0.46 Precision: 0.45 Recall: 0.54	32.7 mins	3.

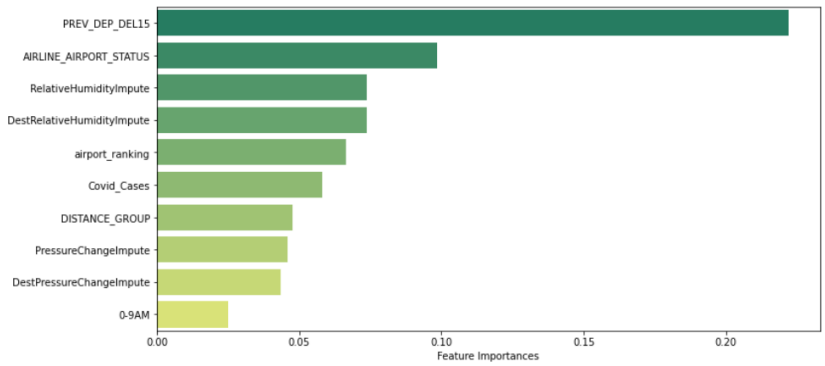
- Model Experiments with Feature Selection & Feature Extraction:

  We used the following methods for model experiments:
  - Feature selection with feature importances: we used the top 10 important features extracted from Random Forest model result. As we have expected, these most important features were mainly generated from our feature engineering step. With the most important feature is the previous flight delay indicator, follow by airlines and airport delay status within the last 5-20 flights.

Other features include airport pagerank, and some weather measures which were listed in Figure 6. We also tested removing the most important feature to see how much it impacts our model performance.

- Feature Hasher: transformed all categorical features available in the dataset (including all airport codes and airline codes) to a vector of 20 dimensions.
- Variance threhold: selected features with at least 0.5% variance for training.
- Pricipal Component Analysis (PCA): applied feature extraction with 10 PCA components for training.

Figure 6: Feature Importances



Model experiments results:

Model	Features	Best Model Hyperparameter	Cross Validation Performance	Blind Test Set Performance	Training Time
Ridge Logistic Regression Feature Selection - Top 10 Most Feature Importances	14	regParam=1e-5 threshold=0.5	F0.5 Score:0.68 Precision:0.74 Recall:0.52	F0.5 Score:0.42 Precision:0.40 Recall:0.55	9.35 mins

Model	Features	Best Model Hyperparameter	Cross Validation Performance	Blind Test Set Performance	Training Time
Ridge Logistic Regression Feature Selection - Removing Top 1 From Top 10	13	regParam=1e-4 threshold=0.5	F0.5 Score:0.62 Precision:0.68 Recall:0.45	F0.5 Score:0.36 Precision:0.34 Recall:0.48	12.5 mins
Ridge Logistic Regression Feature Selection - Categorical Feature Hasher 20 & Top 10	34	regParam=1e-5 threshold=0.5	F0.5 Score:0.68 Precision:0.73 Recall:0.53	F0.5 Score:0.41 Precision:0.38 Recall:0.55	30.0 mins
Ridge Logistic Regression Feature Selection - Variance Threshold 0.5%	20	regParam=1e-5 threshold=0.5	F0.5 Score: 0.68 Precision: 0.73 Recall: 0.52	F0.5 Score: 0.41 Precision: 0.39 Recall: 0.54	51.5 mins
Ridge Logistic Regression Feature Extraction - PCA - 10 Components	10	regParam=1e-5 threshold=0.5	F0.5 Score: 0.65 Precision: 0.79 Recall: 0.38	F0.5 Score: 0.43 Precision: 0.45 Recall: 0.39	40.4 mins

Discussion

- Our baseline model has very poor performance. After performing feature engineering, handling data unbalanced and fine tuning model hyperparameters

with regularization to reduce overfitting, the model performance has increased significantly.
- Our best model with high F0.5 score and efficiency is XGBoost with undersampling. Even though, Random Forest performs slightly better, it takes a lot more time for training than XGBoost.
- Multitask learning does not perform as good as a traditional logistic regression model and it takes very long time for training. The threshold for early stopping implementation was not reached with slow convergence of the loss function.
- Oversampling with XGBoost could only be performed with less features, we've selected the top 10 most important features from Random Forest undersampling for this model. The performance of oversampling model is in the same range of undersampling model. However, with the advantage of having less data for

Summary on Limitations, Challenges, and Future work

Limitations:

- **Oversampling Model:** With large data, our cross validation with XGBoost model could not been run with the full features, and returned out of memory error. Our alternative approach for this was training with less features to reduce the dimension (using the top 10 most important features). However, the model performance was not much different from undersampling model. This shows better predictive power from Oversampling even with much less features compared to Undersampling since there is more training data available. However, we had to make tradeoff on features due to limited computational resources available.

- **Multitask Learning Model:** Our current algorithms using Spark RDD to calculate gradient descent and update the model weights take very long time to process. We were not able to perform hyperparameter tuning for this model, and use the same hyperparameter tuning result from Logistic Regression instead. Early stopping algorithms were implemented, and training should be stopped early when the current validation loss is within the threshold of the last 3 losses' averaging value. However, our model took long time to converge, we could only trained for 60 iterations and it could not reach the early stopping threshold.

Challenges:

Primary challenge we faced during this project was to work with such a large dataset. So far, our experience has been dealing with small data that allows us to expriment with different models in much faster and easier way due to less computational resources being used. In this project, we had to finetune our data and filter out non-important features, execute efficient hyperparameter turning prior to running each model to make sure we have reasonable training time for each model we wanted to experiment with. In addition to that, we also had to spend time on understanding the raw dataset given to us and what each of the feature means to our model.

`RandomGridSearch`: In order to find better evaluation metrics, we planned to add a random grid search, which will help us find better hyperparameters.

Class imbalances can make the model favor negative classification. If the imbalance is higher than the existing distribution, oversampling similar to what we used in our project (Random Oversampling or Ideally SMOTE) should be in place to adjust the class distribution.

In our quest in the pyspark documentation we couldn't find an official implementation of a random grid search function similar to ParamGridBuilder. We found one implementation of this function from an article and try it out. However, we couldn't train our models using a RandomGrid search due to memory constraints. We only could do it for the logistic regression model and didn't get results with.

Another challenge is our models suffered from overfitting and localized on blocked cross validation training data. Therefore, the models' prediction on unseen 2021 data show a high drop in F0.5 score, precision and recall. To mitigate this issue, we could try splitting the data in smaller folds (3-4 folds) and bigger data in each fold, with validation data spread in a bigger window to cover every months and seasons. Or another approach is using rolling window cross validation. However, due to time constrained we were unable to implement these methods.

Data Leakage:

Data leakage could happen when our model is fed with additional information which allow the model to learn something that it should not know. For example, in time series modeling, we could potentially create data leakage if using information of the future data to predict and evaluate the past. Or any feature with its value that is not actually available at the time we want to use the model to make a prediction could create leakage to our model. To prevent data leakage in our model, we used historical data (i.e., weather data and previous flight delay data) that are at least 2hrs before the departure time as selected features.

Potential leakage in our model: The pagerank airport information could has some leakage for 2015 training data in our model since we do not have 2014 data, but the rest of the years from 2016 - 2021 we used pagerank from the previous year to reduce data leakage issue. To impute missing weather data for features having less than 5% null value, we used the averaging monthly values from that station, which were extracted from the entire training data. Giving the low percentage of missing values, and the weather data do not provide strong predictive power to the model, this data leakage is negligible.

Future Work:

- Additional Feature Engineering: departure airport with most delayed last year / last month using pagerank with custom weight. This mostly requires building the pagerank algorithms from scratch, since GraphFrames is not the best tool to use for this work.
- Further hyperparameter tuning on features to calculate mean, median and/or mode to reduce similar feature length
- We would also like to add additional external features like average airport traffic depending on the departure airport, since we think similar feature can improve the predictive power of our model even further
- We would like to narrow down list of important features even further and implement a Neural Network model with 2 to 3 hidden layers and experiment with different activation functions (ReLU, etc.)
- Created a custom random search or Bayesian search for cross validation, this will allow us to expand hyperparameter tuning to larger sets and window with more complex models.

- Implement SMOTE for handling unbalanced data, our oversampling results showed similar performance with undersampling models. Using SMOTE to generate synthetic data for the minority class based on its ditribution and nearest neighbors information could potentially help improve our models' learning.
- Implement rolling window cross validation or split the data in smaller folds (with bigger data) to help reduce overfitting. However, the rolling window approach could potentially increase data leakage in our model, and could potentially not applicable for oversample or SMOTE implementation.
- Further improve the efficiency of Multitask learning model and tried to implement with Spark DataFrame, or inherit and create custom parameters/ methods from the current Spark MLLib classes.
- Trouble shoot how to save the model weights and results in blob storage or create

Project Constraints / Scalability Challenges

Scalability/Time complexity - at the start of the project, after completing initial EDA on the airlines and weather datasets, we were faced with the challenge of optimizing our join strategy such that we would be able to process the data within a reasonable amount of time. Even given the substantial cluster computing resources we used during the course of our project, we quickly found that if we did not optimize the transformations, the pipeline would not be scalable. Our process was to first develop the ideal solution if we did not have to worry about time complexity and scalability, which resulted in a join on multiple complex conditions (such as the absolute value difference between latitudes and longitudes of airports and weather stations being within a specified distance). This took us 2-3 hours to execute on the 3-month dataset, so from there we set about finding ways to join the data in a similar fashion but in a much more efficient manner. We utilized bucketing to optimize the joins and achieve effectively the same result, but within a matter of 5-10 minutes instead of 2-3 hours. This made developing the rest of the pipeline, including iterating over feature engineering experiments, much more feasible.

Conclusion

Using our metric of F0.5 score, we saw incremental improvement from our baseline Logistic Regression model to Random Forest, XGBoost and Multilayer Perceptron Neural Network model by applying Advanced Feature Engineering and Hyperparameter tuning. While we were able to see significant improvements, we were not able to get our F0.5 score to very high levels (we generally had a ceiling around F0.5 = 0.7 for our best models XGBoost & MLP). We also found that feature engineering is incredibly important to model performance, since we saw greater gains from creating new features such as Airport PageRank indicating most outbound connections for a specific airport, Airline and Airport delay status for last 5-20 flights and whether a flight's previous flight was delayed 2 hours before the flight. The tuning of hyperparameters was important for us to control the models from exhibiting bad behavior such as overfitting, but major improvements in performance came from creating new features that were going to be helpful to predict the target variable. It was also interesting to see how the model performed differently when we ran it on the full, 4-year dataset. When we used 1-year data for Cross Validation for faster performance, we saw the models overfit, but when using the full dataset we see that the model was able to generalize better, due to the reduced variance caused by the increase in data.

Successes and Takeaways

We were able to pre-process and clean up the large database which speeds up the joining of the full database for analysis. Storing the database on block storage allows us to search through the database using SQL or other methods at ease. During the process of joining the full database, we brought in the airport data to help decipher the location acronyms linking the weather data, stations, and airline data altogether. The limitations of the data, its format, and the limitations of the clusters also provides us a real life situation when conducting a machine laerning project. Understanding each models, its applicability, and adding weight to our analysys helped us improved the F0.5 scores, precision, and recall.

In summary, this project was a culmination of all different course concepts that we learned. The primary goal was to apply ML algorithms at Scale and this project allowed us with the opportunity to not only the apply these algorithms at scale, but also gave us an opportunity to work with real-world raw data that is not cleaned and ready to be fed in a ML pipeline. Also, the theme of this project to cater our solution for a customer from a Business context gave us renewed appreciation for importance of project management, communicating efficiently with peers, resource allocation based on cluster availability and overcoming challenges related to work distribution.

Concepts applied in this Project

- *Embarassingly Parallel/Scalability:* As mentioned earlier, much of this course's theme revolves around creating Scalable algorithms that can be applied efficiently across large datasets. This consideration was crucial for modeling. For example, it was much easier to model using Logistic Regression compared to Random Forests and XGBoost. Both Random Forest and XGBoost models are much more computationally expensive than Logisitic Regression. This observation became apparent during our experimentation when adding more features to our models. As we added more features in our pipeling, both Random Forest and XGBoost models frequently crashed due to memory and network shuffle constraints, especially when using Oversampling. In contrast, additional features increased the training times of Logistic Regression models, but memory and network shuffle issues were not encountered. We encountered issues with XGBoost and Random Forest models with many features even when scaling our cluster to the maximums allowed.
- *Lazy Evaluation in Spark:* This project also required us to reinforce our learnings from earlier in the semester related to Spark's lazy evaluation, since we had to take into consideration which operations were transformations versus actions to make sure we utilize our computational resources efficiently. As an example, we ensured our pipeline calls out action commands such as `display()` after majority of dataframe transformations and/or feature imputations were complete.
- *Graph Algorithms:* Flights from individual airports naturally form a graph network from which we were able to extract information relevant to our problem. Further, thinking about outbound flights as a graph led to creation of one of our top 10 important feature named `airport_ranking` during feature engineering, which

helped improve predictive power of our model. Using that, we created a PageRank feature that effectively captured the traffic outbound from each airport. We hypothesized that this would help us because it would provide information about potential traffic congestion at airports that might cause delays, which turned out to be a good assessment as reflected in our model discussion above.

- *Multitask Learning:* By implementing this algorithm from scratch using Spark RDD

APPENDIX

Notebooks:

- Data cleaning & Joining: notebook ()
- Feature Engineering: notebook ()
- End-to-end Data Processing, Cross Validation Train/ Test Split & Model Pipeline: notebook ()
- Imbalance Data Handling:
  - Under Sampling: notebook ()
  - Over Sampling: notebook ()
- Baseline Logistic Regression Model: notebook ()
- Ridge Logistic Regression - Undersampling: notebook ()
- Elastic Net Logistic Regression - Undersampling: notebook ()
- XGBoost - Undersampling: notebook ()
- Random Forest - Undersampling: notebook ()
- Multi-task Training - Undersampling: notebook ()
- XGBoost - Oversampling: notebook ()
- Multilayers Perceptron - Oversampling: notebook ()
- Random Hyperparameter Tuning: notebook ()
- Model Experiments with Feature Selection and Feature Extraction: notebook ()