# Predicting Flight Delays at Pittsburgh International Airport

Pavel Khokhlov, Christopher Hwang, Gordon Wang

December 7, 2018

# 1 Introduction

Every year, airlines face multi-million-dollar costs due to flight delays caused by extreme weather and non-optimized operations management. Our data comes from the Airline On-Time Performance Data made available through the Bureau of Transportation Statistics of the U.S. Department of Transportation. Specifically, our data set has info regarding commercial flight activity to and from Pittsburgh International Airport for 2015-2016. In addition to this, we used Pittsburgh weather data from NOAA from the weather station at the PIT airport. The problem we are trying to solve is predicting if there will be a flight delay for each departing flight in 2017, using only information available before departure.

# 2 Exploration

It is important to define the variable that we are trying to predict. To distinguish more significant delays from small ones that occur on a regular basis, we define a flight delay as one of at least 15 minutes or more. From there, we examined what variables we thought seemed most impactful on predicting if a flight would be delayed by 15 minutes or more. In general it is more typical for an evening flight to be delayed than a morning flight. This can be seen in the data as well. We grouped our data by hour of expected departure and then computed the percentage of delayed flights in each hour resulting in Table 1.

| hour <int> | x <dbl> |
|---|---|
| 5 | 0.03357030 |
| 6 | 0.05598010 |
| 7 | 0.06458046 |
| 8 | 0.06220363 |
| 9 | 0.08152174 |
| 10 | 0.09896433 |
| 11 | 0.11345276 |
| 12 | 0.12879103 |
| 13 | 0.14630159 |
| 14 | 0.16482583 |

| hour <int> | x <dbl> |
|---|---|
| 15 | 0.18572794 |
| 16 | 0.18628859 |
| 17 | 0.22020833 |
| 18 | 0.24187832 |
| 19 | 0.20893077 |
| 20 | 0.22049180 |
| 21 | 0.21084337 |
| 22 | 0.12903226 |

Table 1: Percentage of delayed flights group by hour of expected departure

Clearly, the percentage of delayed flights by hour is almost a monotonically increasing function in hour. The evening and late evening flights certainly have more delays proportionally than earlier ones by several factors. When we fit our models, the expected departure time ended up being the "most important" feature for our two models.

## 2.1 Data Cleaning

There was a nontrivial amount of data cleaning required for this dataset. In order for all of our joins to work how we would like, we needed all our data to have standardized POSIX datetimes. This required parsing each dataset's date and time columns with regular expressions and combining results before casting. After this, we needed to change the types of some columns such as AIRLINE_ID into factors and etc. This was done for convenience in some cases as well as out of necessity for our models to correctly interpret the feature (e.g. a random forest should not split on AIRLINE_ID as if it is a continuous variable). Another unexpected part of the data cleaning came when we performed predictions on the 2017 dataset because there was an airline in that data whose AIRLINE_ID was not included in the AIRLINE_ID factor of the 2015-2016 data. This became an issue for XGBoost because it is very picky with how the data is structured. We had to change the levels in the training set's AIRLINE_ID factor. It is also important to mention, that the training had to be done on a subset of the data which had flights departing from Pittsburgh. This was partially because that was our goal but also because our weather features would have been difficult to recover for all cities, and might not have given us meaningful signals for prediction in Pittsburgh.

## 2.2 Feature Engineering/ Data Expansion

In addition to the variables provided within the data set, we decided to brainstorm about and add more predictors that we thought contributed to flight delays, such as weather (one hot vectors indicating fog, thunder, precipitation, ice pellets, hail, smoke, snow, high winds, and more), the current flight date's proximity to a US holiday (both days before and after), and the percentage of flights that were delayed 1, 2, 3, and 4 hours prior to the flight's scheduled departing time. Thus, this entailed adding an additional dataset from NOAA for Pittsburgh weather data and a reference for US holiday dates. The join for the NOAA data as well as the weather data was quite seamless because all that was necessary was to join on date.

The most challenging feature to engineer was number/percentage of flight delays X hours prior to the current flight. As mentioned in the cleaning section, it was necessary to have all the flights to have POSIX times for us to be able subtract flights in a standard format. There was not a seamless join or lag that could be performed on the dataframe in vanilla R like the analytic functions in VerticaDB that would give us information about

flights for a certain time window before the current one. This required us to run a simple for loop and search for our desired rows.

The data set that we worked with showed a drastic difference in number of departure flights that were not delayed and delayed. Figure 1 shows the distribution of these flights, where not delayed and delayed are labeled 0 and 1 respectively,

```
    0     1
85734 16179
```

Figure 1: Distribution of Departure Flights Not Delayed and Delayed

The imbalanced nature of the data will cause difficulties for models such as random forest that don't deal well with imbalances.

# 3 Supervised Analysis

In order to find the model that best classifies, we tried numerous predicting methods with various combinations of variables.

## 3.1 Random Forest

With the number of predictors that we have in our data set, the random forest method seemed like an effective option as it would create a multitude of decision trees and narrow down mean predictions for each of the combinations of predictor values. We created multiple models with different mixes of predictors that we thought would contribute to the flight delays most impactfully. After the first model with nearly all our predictors, we examined the variable importance plot of the model and included the most influential ones in the next model. The procedure we had for testing these models involved performing 5-fold cross validation on our training data. The parameter for number of trees was not cross validated, but made large enough so we would not have to spend lots of time tuning it. We also adjusted the sample size in each model to be balanced, since random forest suffers on imbalanced data - this in particular dramatically increased our AUC. After multiple iterations of this process, we were able to narrow down the variables that created the model that performed the best, as can be seen in Figure 2.

```
rf_bal7 = randomForest(DEP_DEL15 ~ MONTH + AIRLINE_ID + CRS_DEP_TIME
                       + AWND + PRCP + SNOW
                       + HOLIDAY_PROX + DAY_OF_WEEK
                       + PCT_DEL_1 + PCT_DEL_2 + DISTANCE,
                       data=train_df, sampsize=c(nsmall, nsmall),
                       ntree=1000)
rf_bal7_guess = predict(rf_bal7, test_df, type='prob')
```

Figure 2: Call to Best-performing Random Forest model

The performance was measured by examining the area under the ROC curve of each model. Figure 3 shows the ROC curve and AUC for the best of these models.

```
Call:
roc.default(response = test_df$DEP_DEL15, predictor = predict(rf_bal7,     test_df,
type = "prob")[, 2], plot = TRUE, col = "purple")

Data: predict(rf_bal7, test_df, type = "prob")[, 2] in 12273 controls
(test_df$DEP_DEL15 0) < 1315 cases (test_df$DEP_DEL15 1).
Area under the curve: 0.6981
```

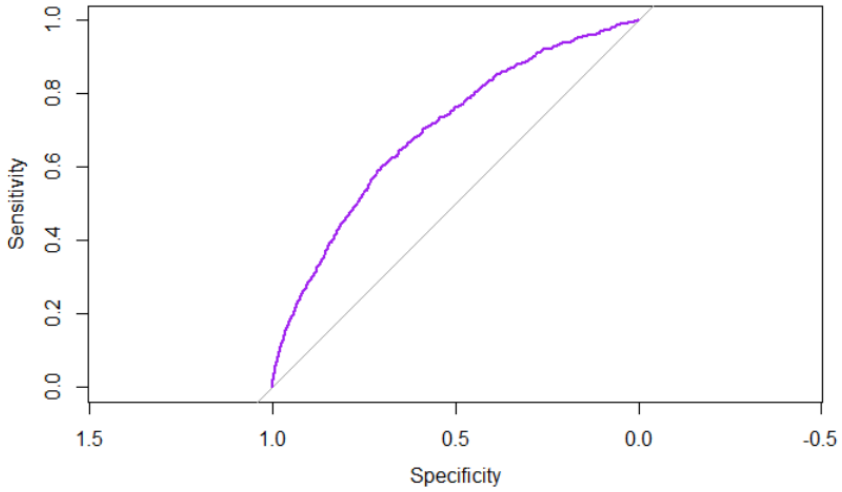


Figure 3: ROC curve and AUC for Best-performing Random Forest Model

The variable importance plot for the best random forest model in Figure 4 shows CRS_DEP_TIME as the most influential variable.

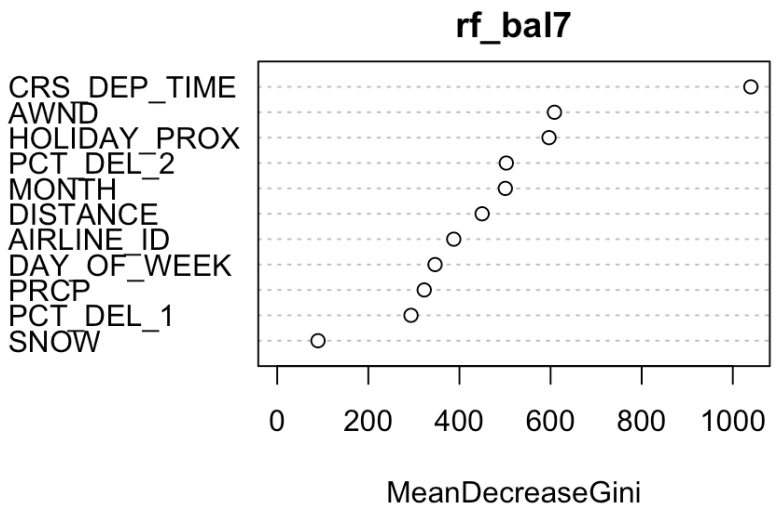


Figure 4: Variable Importance Plot for Best-performing Random Forest Model

## 3.2 XGBoost

Extreme Gradient Boosting (XGBoost) was another effective method for predicting flight delays because it is a good alternative to random forest; gradient boosting generally is more powerful and accurate. Gradient boosting uses shallower, low variance trees and eliminates high bias via additive adjustment for previous error. However, boosting needs careful tuning for its hyperparameters, whereas random forest is tuning free.

In selecting variables for the possible models, we followed the same process in selecting variables for random forest models. For our first XGBoost model, we selected the same exact predictors as we did for our second random forest model. For our second XGBoost model, we selected predictors that we considered most influential, according to intuition and partial dependence plots.

For the hyperparameters, we tuned them in several steps. Beforehand, we trained with default parameters on hold-out test set to check model's baseline AUC performance. Second, we lowered the learning rate (eta) to 0.1. We then determined the optimal number of trees for this learning rate by using stratified 10-fold cross-validation and an early_stopping_rounds value of 10 to stop the boosting after the cross-validated AUC score

doesn't improve after 10 iterations. Afterwards, using caret package in R, with this new optimal tree number as nrounds (max tree number value), we tuned via random search and stratified cross-validation the tree complexity parameters: max_depth and min_child_weight. Following, we tuned in 3 separate steps: sampling parameters sub_sample and colsample_bytree, regularization parameter gamma, and learning rate eta and nrounds. For each of these 3 steps, we used random search/cross-validation to tune and also retuned each time the optimal tree number using early_stopping_rounds to stop boosting before nrounds number of iterations is reached. Finally, we tuned for last time the optimal tree number with cross-validation and early_stopping_rounds and then trained the final model with all of the tuned parameter values. We did this tuning for both XGBoost models and evaluated model performance by testing on hold-out test set and measuring AUC, comparing to baseline performance.

The best model's ROC curve and AUC is shown below in Figure 5.

```
Call:
roc.default(response = test_df$DEP_DEL15, predictor = guess_xgb2,     plot = TRUE,
col = "goldenrod4")

Data: guess_xgb2 in 12273 controls (test_df$DEP_DEL15 0) < 1315 cases
(test_df$DEP_DEL15 1).
Area under the curve: 0.7009
```

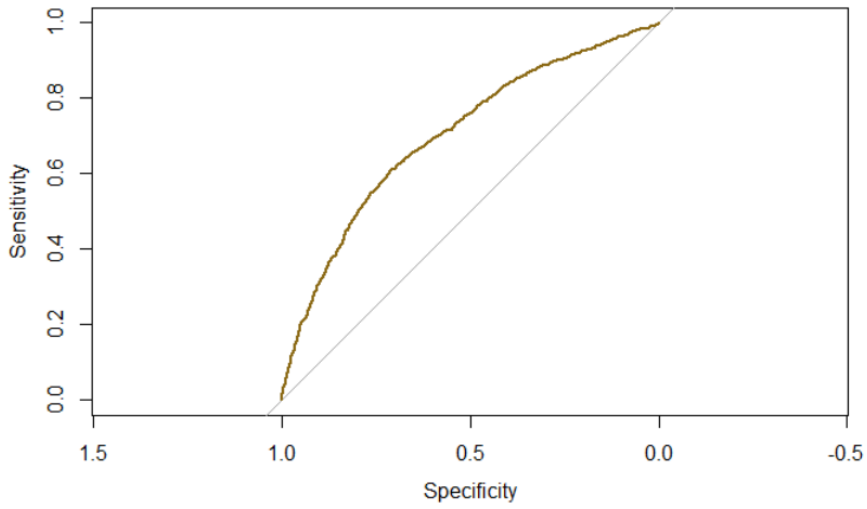


Figure 5: ROC Curve and AUC for Best-performing XGBoost Model

## 3.3 Ensembling

Upon choosing the models that perform best on our hold-out test data set, we created an ensemble of two models: one from random forest and one from XGBoost. The models used were the best performing for their respective method. We took the means of each prediction and found the ensemble model to perform better, as seen in Figure 6.

```
ensemble = rowMeans(cbind(guess_xgb2, rf_bal7_guess[, 2]), na.rm=TRUE)
```

```
Call:
roc.default(response = test_df$DEP_DEL15, predictor = ensemble,     plot = TRUE, col
= "red")

Data: ensemble in 12273 controls (test_df$DEP_DEL15 0) < 1315 cases
(test_df$DEP_DEL15 1).
Area under the curve: 0.7081
```

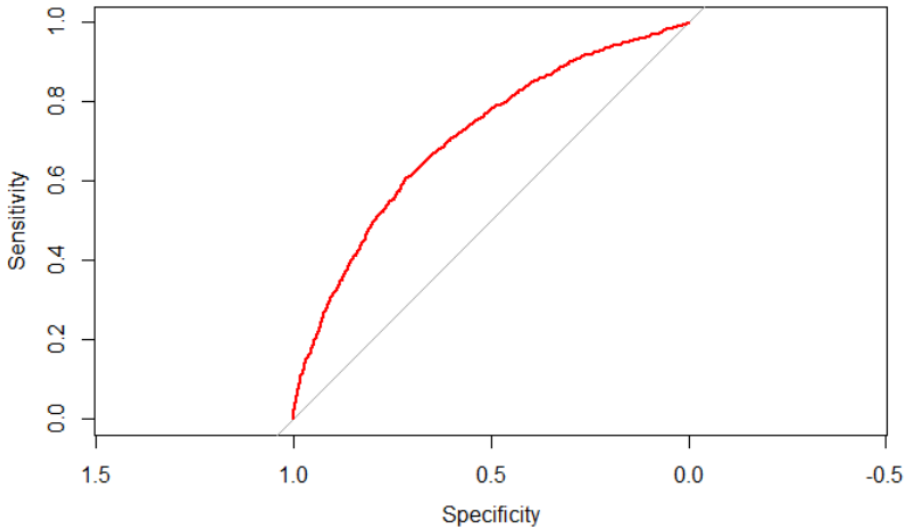


Figure 6: ROC Curve and AUC for Ensemble Model

## 3.4 Prediction

We predicted our AUC to be 0.708 because our out of sample AUC on hold-out test data was 0.708. We chose ensembling because we had already trained two types of models and the ensembling procedure was not very involved (a simple mean). The ensemble model outperformed the rest of the models on the out of sample hold-out test set with a seemingly non-random improvement.

# 4 Analysis of Results

Given the quartile distribution of the predictions, we created a confusion matrix shown below in Figure 7 that compares our predictions with the actual results with a threshold at the 75% quartile value. We used the results to figure out which data points performed well and poorly to do further analysis on their patterns.

```
        0%        25%        50%        75%       100%
0.01746067 0.14095898 0.21553591 0.27869001 0.54335714
```

```
                  real$DEP_DEL15
delay.guesses_    0    1
             0 1035  127
             1  294   92
```

Figure 7: Confusion Matrix for Ensemble Model Performance

Looking at the data points where the model performed poorly on, we see that those flights with a departure date of an average of 1.73 days after a holiday and facing weather conditions of fog, ice fog, or freezing fog performed mostly false positively (falsely predicted a delay) compared to the actual data. On the other hand, flights that the model predicted well with true negatives were those that were on average 17 days away from a holiday mostly with flights that head to the south in Florida, Georgia, and Maryland.

It is intuitively clear that if a flight is certain to be delayed for $n$ hours then we would lose nr if we were to show up as if the flight was on time. Thus, if a flight is delayed for more than $C/r$ hours, then it is worth it to miss the flight. We obviously do not know if a flight will be delayed more than that. If we had a model that said that the probability of being delayed by more than $n$ hours is $p$, then our expected loss by arriving on time would be $pnr$. If $pnr > C$, it follows that we have an expected loss by arriving to the flight on time. It is likely that $C$ is significantly higher than $nr$, but we would still like to arrive, only later. With our model that gives us a probability of delay for more than $n$ hours, we could tune a threshold parameter for the probability that would indicate if we should delay our own arrival by $n$ hours using our data with a loss function dependent on $C$ and $nr$. For example, if the probability of being delayed by 4 hours is greater than our tuned 0.7 then we should arrive 4 hours late and on average, we should minimize our loss.