Statistics and Machine Learning Final Report

Kwaku Bimpong and Robert Wetten

2025-04-25

Contents

Introduction

Problem Description

In this project, we address the problem of predicting flight delays for departures from Pittsburgh International Airport (PIT). Using flight data from 2022 and 2023, our goal is to develop a predictive model that forecasts whether a flight will experience a delay of 15 minutes or more. The prediction must be based solely on information available before the scheduled departure, simulating real-world scenarios in which decisions must be made without hindsight. Our approach involves data cleaning, feature engineering, and the application of the random forest algorithm to create an accurate and robust classifier. Ultimately, predictions are evaluated based on the Area Under the Curve (AUC) metric, rewarding models that effectively distinguish between delayed and on-time flights. This write up documents our methodology, findings, and reflections throughout the modeling process.

Data

The data we are using comes from the Airline On-Time Performance Data provided by the Bureau of Transportation Statistics of the U.S. Department of Transportation. This data contains information on the flight's time/date, airline, origin, destination, departure performance, and more. As mentioned above, a lot of our process involves data cleaning, and as such, we discuss how we handled this data and what features to retain below. To start, we combine the flight data in the 2022 and 2023 data sets to enlarge our training data. Afterwards, we remove the NA values of the DEP_DEL15 variable so that we only include instances

where we know whether the flight is delayed or not, allowing us to accurately train our data based on the true results.

Before displaying our features and getting into modeling, we decided that we would first take time to clean this data and add some variables that could be useful. To start, we knew that many of the variables that described delayed flights wouldn't be available to us in the data we were trying to predict, and for this reason, we decided to remove these variables from our combined data set. Next, since we are only concerned with departing flights from PIT, we decided to filter our data to just these departures, as we could not come up with a sensible way to pair arriving flights with their correspondent departing flights (if they exist).

As a next measure, we noted that many of the variables in the data set were redundant to other variables. For example, the variables `DEST_STATE_FIPS`, `DEST_STATE_NM`, and `DEST_WAC` are all redundant to `DEST_STATE_ABR` because they contain a 1:1 mapping. This was the case for a number of other variables such as much of the origin information and distance information. Finally, we decided to remove all of the data on the time of the flight except for the `FL_DATE` variable, which summarized this information. After removing the variables mentioned above, we were left with a combined dataset that contained 11 variables, pictured in Table 1.

Table 1: Variables remaining after removing redundancies.

Variable
FL_DATE
OP_CARRIER
DEST_AIRPORT_ID
DEST_CITY_MARKET_ID
DEST
DEST_STATE_ABR
CRS_DEP_TIME
DEP_DEL15
CRS_ARR_TIME
CRS_ELAPSED_TIME
DISTANCE

With the major cleaning out of the way, we decided to use our `FL_DATE` variable to create some more variables. In particular, from our intuition about airports, we felt that more delays would be likely on weekends, holidays, an may have some relationship with how far along in the year the flight took place. For that reason, we added 3 variables to our dataset: `is_weekend`, `is_holiday` (based on the top 5 most major holidays), and `day_of_year`.

Exploration

Univariate EDA

Before beginning our analysis, we performed extensive exploratory data analysis, first to understand the structure of our data, and second to gain insights that would guide our modeling. As a first measure, we display the distribution of our response variable, `DEP_DEL15`, which is a binary feature explaining whether the departure from the Pittsburgh airport is delayed by greater than 15 minutes (`DEP_DEL15` = 1) or not (`DEP_DEL` = 0).

In Figure 1, we simply highlight the huge class imbalance. This imbalance will be important to note as we proceed in our supervised analysis. It also suggests that we should search for a classifier that is better than the naive classifier that always predicts no delay (`DEP_DEL15 = 0`). For reference later, this classifying technique would produce an accuracy of 84.57% across all of the labeled data.

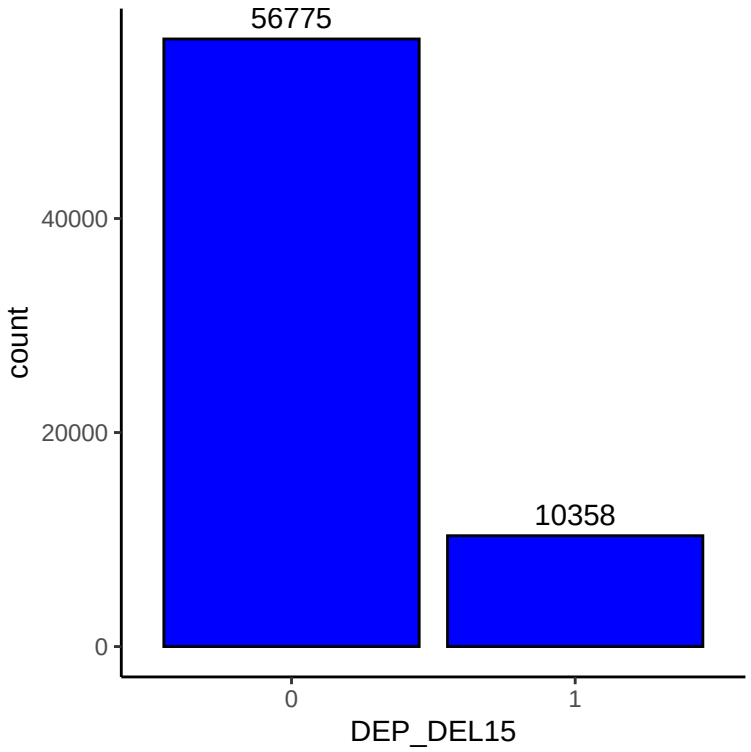

Figure 1: Distribution of DEP_DEL15, we notice a huge class imbalance. The number of on time flights is over 5 times greater than the number of on delayed flights.

Figure 2 displays the distributions of a couple quantitative predictors that we believe to be useful: flight distance and the estimated flight time. The figure shows that the distributions are somewhat similar. Additionally, through visualizing this, we were able to find problematic values for the estimated flight time, which pointed to having negative estimates. These rows were removed as there seems to be no way to impute these possibly wrongly entered values.

Bivariate EDA

With some of the big variables explained, we will now analyze the bivariate distributions between some of the predictors and our binary response variable. Figure 3 displays the bivariate distributions between `DEP_DEL15` and both of `DISTANCE` and `CRS_ELAPSED_TIME`. The figures show similar trends.

Next, we consider the fact that flights may get delayed based on the carrier. Additionally, we hypothesize that flights may be delayed based on where they are flying to. To investigate this visually, we plot the percentage of delayed flights across carrier and destination airport. These results are shown in Figure 4.

Overall, our EDA indicated some striking trends between our predictor variables and the binary response of whether or not the flight was delayed. For this reason, we were happy to have included these final variables and moved on to building a supervised learning model in order to make predictions.

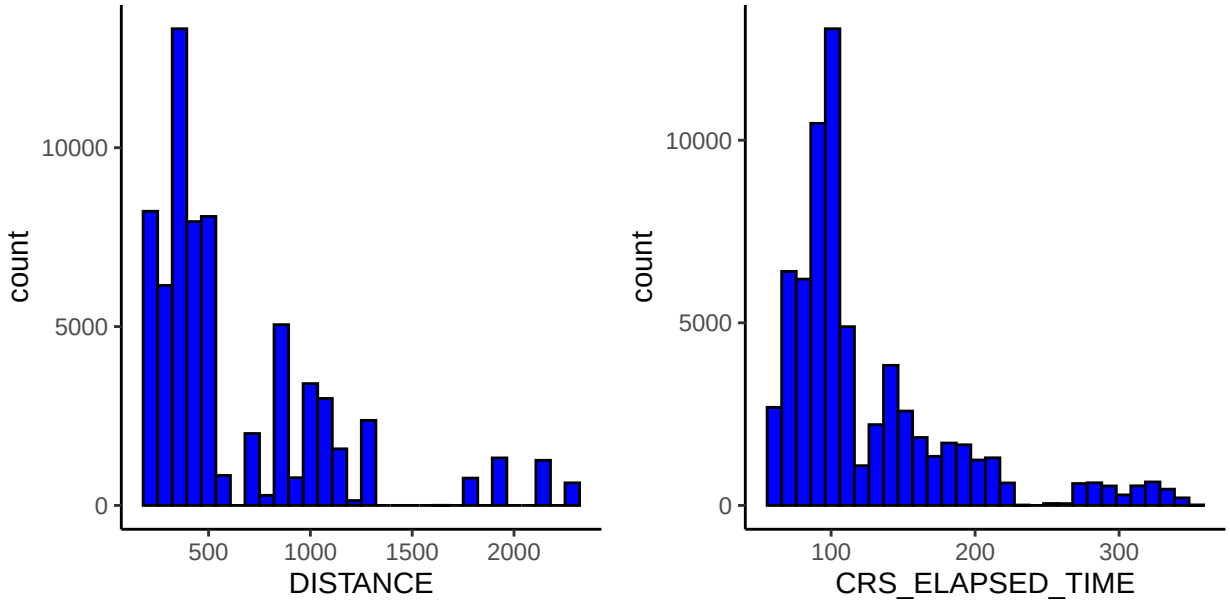

Figure 2: Distributions of DISTANCE and CRS_ELAPSED_TIME, we notice that many flights have a distance below 500 miles, with some flights reach all the way to over 2000 miles. Additionally, most estimated times are within the 0 - 120 minute range. We hypothesize that longer flights may have more delays.

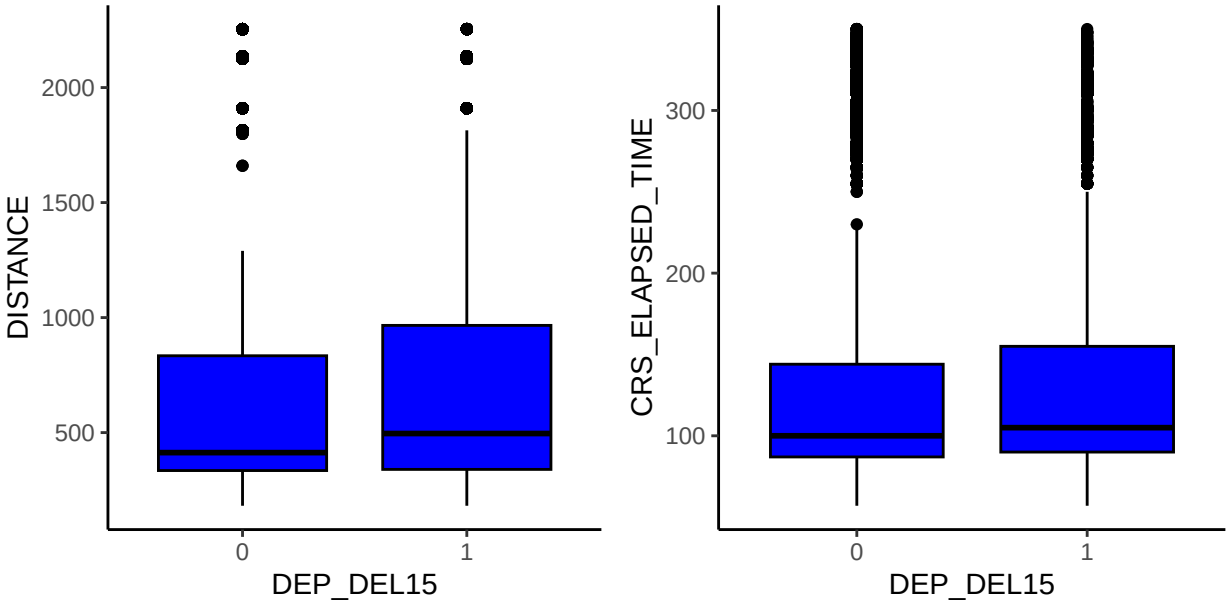

Figure 3: Distributions of DISTANCE and CRS_ELAPSED_TIME versus DEP_DEL15, we notice a slight trend in that the delays (DEP_DEL15 = 1) seem to be slightly associated with larger distances and longer estimated times. Furthermore, before plotting, we noticed a problematic result with the CRS_ELAPSED_TIME that there are some negative values. These were counterintuitive and can't be updated easily, so we removed these two rows from our data.

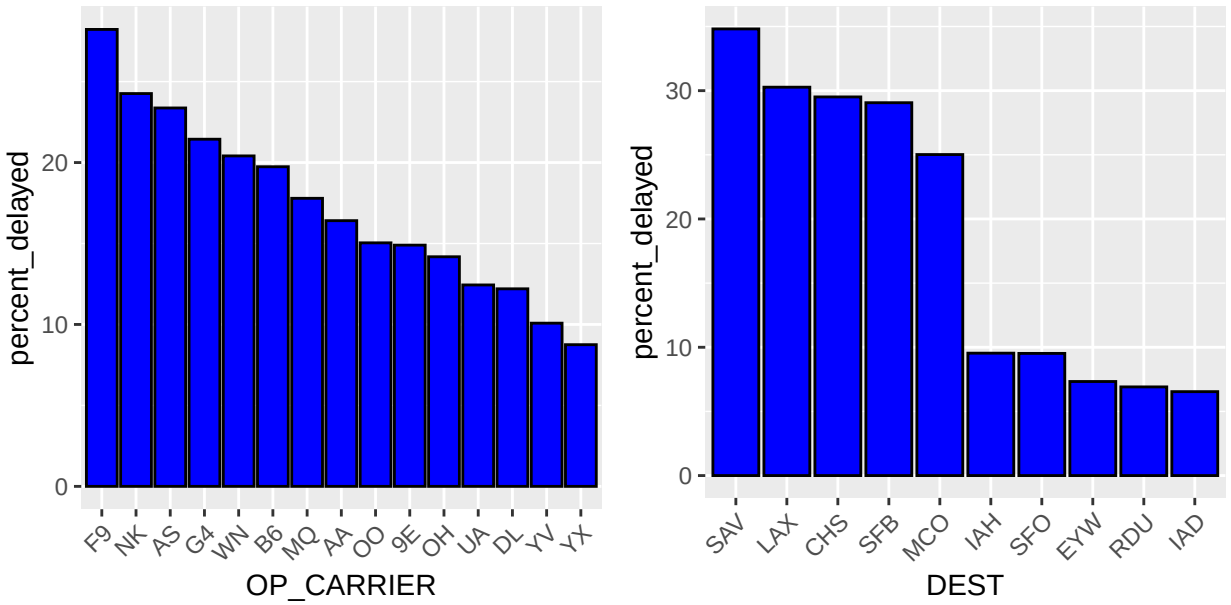

Figure 4: Percentage of delayed flights by carrier. We see that Frontier, Spirit, and Alaska Airlines have a high percentage of delays while Republic Airways and Mesa Airlines are late way less frequently. Additionally, we see that Dulles, Raleigh-Durham, and Key West are almost never delayed while Savannah is often delayed

Supervised Analysis

Setup

Before trying any modeling techniques, we split our combined data set (the dataset containing both the 2022 flight data and the 2023 flight data with the updated variables) into a training set and a test set. The training set contained 80% of the instances within the set and the testing set contained the remaining 20%.

Initial Trial

Our initial trial was a logistic regression model with lasso regularization, with the thoughts that any unnecessary variables would be removed from our model due to the smoothing parameter. In particular, lasso regularization can reduce a variable's coefficient to 0 because instead of the simple least squares minimization, lasso regularization adds an L1 penalty that causes the minimization problem to reduce each coefficient that isn't important:

$$\text{Minimize } \sum_{i=1}^{n}(y_i - \widehat{y_i})^2 \rightarrow \text{Minimize } \sum_{i=1}^{n}(y_i - \widehat{y_i})^2 + \lambda \sum_{j=1}^{p}|\beta_j| \tag{1}$$

In order to find the optimal lambda value for our regularization, we used 10 fold cross validation. Specifically, after storing our predictors in a matrix X and our response in Y, we ran `cv.glmnet(X, Y, alpha = 1, standardize = F, intercept = F)`, and then used this to find the optimal `lambda.min` and finally found our model using `glmnet(X, Y, alpha = 1, lambda = lambda.min, standardize = F, ntercept = F)`. Once having our final model, we predicted on our test set, and ultimately got poor performance, so we decided to scrap using logistic regression with the lasso regularization.

Final Trial

After trying that, we decided to restart our model building and sought to use an ensemble learning method: random forests. Random forests is a model that builds many decision trees and then once all are created, in order for the model to pick a class, it takes the majority class label that is predicted form the trees.

In particular, random forests first randomly samples from training data with replacement in order to create a new dataset of the same size as the original, but with possibly very different rows due to the replacement in the sampling. Once these new datasets are made, the algorithm randomly chooses a random subset of features and decides the best splits to make using mutual information, only considering the chosen features. Finally, the tree continues to grow until a desired level, which is usually when all samples below to a class. The beauty of the random forests model is that it will repeat this entire process for a large amount of trees in order to generate accurate predictions. Once these trees are created, they will be compared together and the majority class for a new instance will be set as the label for the prediction. Although each tree is weak on its own because of the high randomness, when many trees are combined to make the prediction, it works quite well.

We chose to use the random forests algorithm to build our model because it handles the mixed data types in our set (we have both discrete and continuous quantitative variables, and categorical variables). Additionally, due to the individually noisy structure where one tree does not have too much predictive power, the model is robust to outliers. Finally, after visualizing the data, none of the variables were even remotely "normally" distributed, so random forests felt like a safe choice since there wouldn't be obvious violations being broken (like the linearity assumption in logistic regression or the normality assumption in linear regression).

The final variables we chose for our model (as touched on above in or data section) are displayed in Table 2.

Table 2: Final variables in model

Variable
OP_CARRIER
DEST_AIRPORT_ID
DEST_CITY_MARKET_ID
DEST
DEST_STATE_ABR
CRS_DEP_TIME
DEP_DEL15
CRS_ARR_TIME
CRS_ELAPSED_TIME
DISTANCE
is_weekend
day_of_year
is_holiday

Finally, after our building our model with 500 trees, Table 3 displays our model performance on the training data.

Table 3: Confusion matrix and error rates from random forests model

	0	1	class.error
0	44335	1053	0.0232000
1	7356	962	0.8843472

Additionally, Table 4 displays some more key metrics about our most informative variables, day_of_year, estimated depart time, and estimated arrive time:

Table 4: Top 3 important variables

	MeanDecreaseAccuracy
day_of_year	159.48706
CRS_DEP_TIME	59.01568
CRS_ARR_TIME	56.25679

Our model performed close to baseline on our held out testing data. Below displays the confusion matrix on the testing set:

```
##            Reference
## Prediction     0     1
##          0 11088  1793
##          1   299   247
```

Finally, with our model created and evaluated on the training and testing sets, we loaded in the guess data, adapted it to match our data and added variables as we did above, and created guesses.

It appears that the class imbalance proved to be a true struggle for our model because it fails to correctly figure out when a flight will be delayed.

Analysis of Results

Prediction performance by flight type

On our held-out test set, the random forest classifier produced the following confusion matrix:

	Predicted On-Time (0)	Predicted Delayed (1)
Actual On-Time (0)	11 088 (TN)	1 793 (FP)
Actual Delayed (1)	299 (FN)	247 (TP)

- **Specificity (TN rate):** 86.1 % (11 088 / 12 881)
- **Sensitivity (TP rate):** 45.2 % (247 / 546)

Well-predicted flights
- **Off-peak departures:** Early-morning and late-evening flights are correctly classified as on-time in the vast majority of cases.
- **Low-delay carriers/destinations:** Carriers with historical delay rates below 10 % (e.g., Republic Airways, Mesa Airlines) and routes to airports like DCA (Washington–Reagan) and RDU (Raleigh–Durham) show high true-negative rates and few false alarms.

Poorly-predicted flights
- **High-delay carriers:** Frontier and Spirit Airlines (baseline delay rates > 20 %) suffer many false negatives because the model underweights their elevated prior probability of delay.
- **Mid-day and long-haul flights:** Flights departing during midday "rush" windows or covering distances above 1 000 miles display greater variability in actual delays than our features capture, leading to lower true-positive detection.

Adapting to continuous delay prediction

To predict the **continuous** delay duration (in minutes) rather than a binary threshold:

1. **Response:** Use `DEP_DELAY` as the numeric target.

2. **Model:** Train a regression model (e.g., random-forest regression or gradient-boosted trees) optimized for MAE or MSE.

3. **Features:** Engineer variables capturing extreme-delay drivers, such as:
   - Cumulative delay minutes accrued by the carrier earlier in the day.
   - A weather-severity index at departure time.

4. **Evaluation:** Measure performance under a loss function aligned with stakeholder preferences (e.g., MAE for equal weighting of errors or Huber loss for robustness).

This preserves information lost in binary thresholding and quantifies expected waiting times more precisely, at the cost of handling heteroskedasticity and skewed residuals.

Cost-sensitive decision rule

Let
- $p = \Pr(\text{delay} \mid x)$ be our estimated delay probability,
- d be the expected delay duration (in hours),
- C be the fixed cost of missing the flight,
- r be the hourly waiting cost for arriving early.

Two actions:
- **Arrive exactly on time:** expected cost $= p \times (r\,d)$.
- **Arrive one hour early:** cost $= r$ (guaranteed no missed flight).

Set "arrive early" when

$$p\,C \;>\; r \quad\Longrightarrow\quad p \;>\; \frac{r}{C} \;=\; p^{*}.$$

In practice, if the one-hour wait is not fixed but equals r, more generally:

$$p^{*} = \frac{r\,d}{C + r\,d}.$$

- If $C \gg r\,d$, then p^{*} is low and one arrives early even at modest delay probabilities.
- If $C \ll r\,d$, then p^{*} is high and only very likely delays justify arriving early.

This decision threshold aligns predictions with real-world trade-offs between waiting time and the penalty of missing a flight.