Predicting Flight Cancellations and Delays

Alexander Meyerowitz

1 Introduction

The problem I am trying to solve is predicting whether flights will be cancelled or delayed. The target variable is the ground truth flight status of the flights in the dataset; this is a ordinal target variable with options: on time, slight delay, medium delay, large delay, and cancelled. Because the target variable is ordinal, this is a classification problem.

This problem is important because it can empower travelers with an understanding of the risk of delay and cancellation to their potential flights. In the age of COVID-19 it is more important than ever to limit the amount of time spent in high-traffic, public places like airports. Having flight significantly delayed or cancelled after arriving at the airport puts travelers' health at unnecessary risk. Doing research on this problem, and having an accurate model for predicting flight status therefore protects the health of travelers.

The dataset contains flight information for the year 2022 up to July. It contains around 5 million rows and 60 columns. Not all of the features are useful for the problem I am trying to solve, and after paring it down to the relevant features the dataset contains 10 columns. The target variable I am predicting is broken into two columns in the dataset itself: cancelled and delay time. Since I wanted to frame this as a classification problem that encompassed both cancellation and delay, I followed Rob Mulla's conversion of these two columns into a single ordinal feature [3].

This dataset was found here on Kaggle.

I will quickly go over the features contained in the dataset: Distance (the number of miles between the origin and destination airports), Airline (the name of the airline of the flight), Dest (the three letter airport code of the destination airport), DestStateName (the destination state), Origin (the three letter airport code of the origin airport), OriginStateName (the state of origin), DayOfWeek (the day of the week, i.e. Monday, Tuesday, etc.), CRSDepTime (the time of departure formatted as a integer HHMM), DepTimeBlk (a categorical version of departure time split into time blocks), Month (the month at the time of departure).

In doing a literature search for uses of this dataset, I found several techniques and performance metrics that can be used as a benchmark in the future. In Daniel Fabien's Kaggle tutorial on predicting flight delays, he solved the problem of predicting how long flights would be delayed (a regression problem rather than a classification problem). He attempted using linear and polynomial regression to solve the problem. With his best model, the number of predictions where the difference between the true delay and the predicted delay were greater than 15 minutes was 5% [1].

In Bhuvan Bhatia's Master thesis, the problem of predicting flight delays was also investigated using this dataset. This was framed as classification problem rather than a regression problem. The following models were investigated: logistic regression, random forest classifiers, and support vector machine classifiers. The best results that were achieved was 77% accuracy, an F1 score above 50%, and precision of 66% using a random forest classifier [2].

2 Exploratory Data Analysis

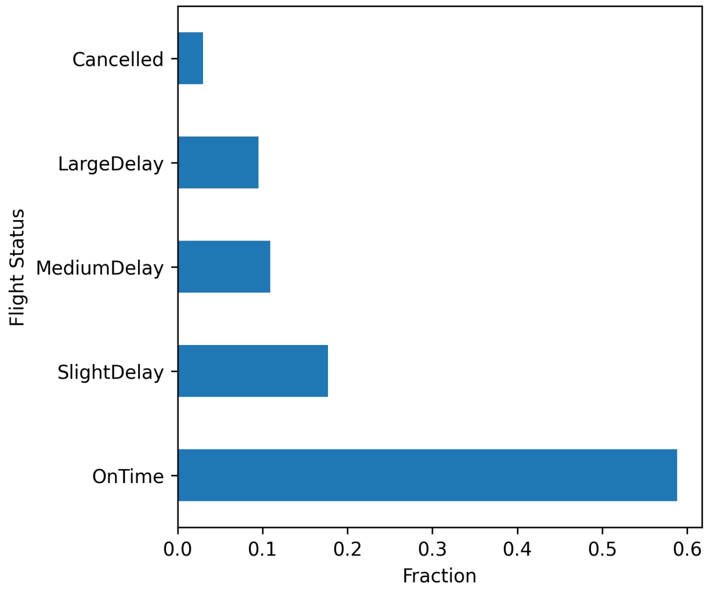

Fig. 1 This graph shows the balances of the target variable.

In Fig. 1 I investigated the balance of the target variable. I found that the it is unbalanced with most flights being on time. This informs how I will split the data in the future because I want to maintain this balance in the train, test, and validation sets so that each set is an accurate representation of the data.

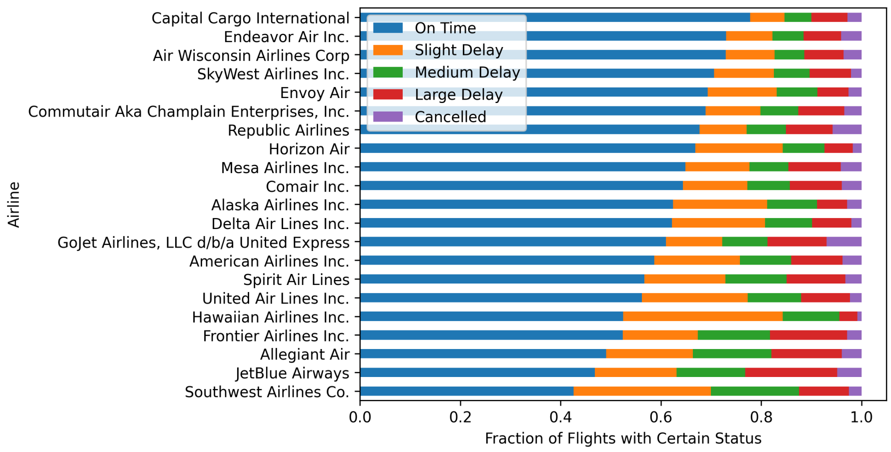

Fig. 2 This graph shows how the break down of flight status varies per airline.

In Fig. 2 I looked at how the break down of flight status varied based on airline. This was particularly interesting because I had not considered that there were different *types* of airlines in this dataset. In fact, there are both passenger (consumer) and cargo airlines in the dataset. It was interesting to discover that cargo airlines are much more reliable and less prone to delay and cancellation than passenger airlines.

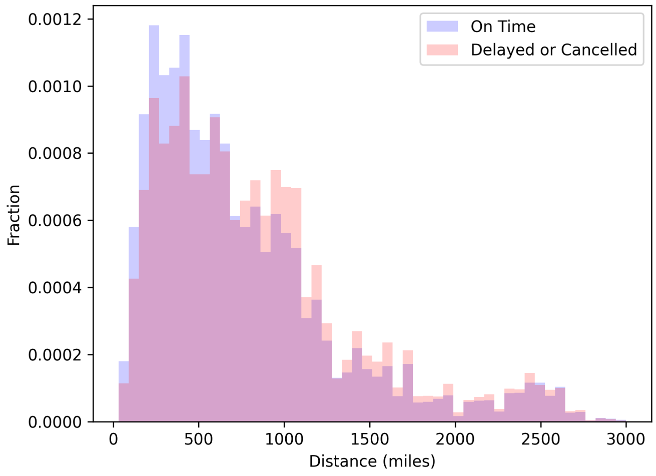

Fig. 3 This graph shows the distribution of distance for flights that are on time vs not on time.

In Fig 3. I looked at the distribution of distance for flights that were on time vs flights that were delayed or cancelled. I went into this with the intuition that longer distance flights would be much more likely to be cancelled. This, however, was not really the case. While the distribution show the hint of this influence, they are mostly the same. This represents less of an influence than I was expecting.

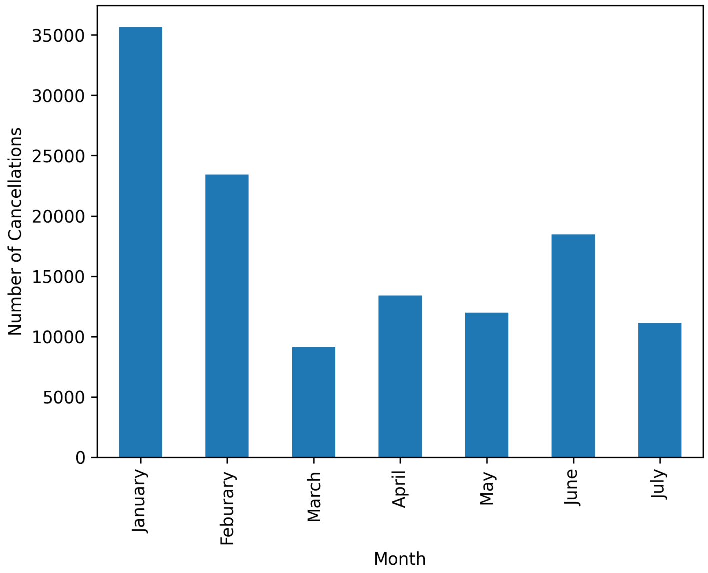

Fig. 4 This graph shows the number of cancellations per month

In Fig 4. I looked at the number of cancellations per month. This was primarily to investigate the spike in cancellation I had heard about in the news over the Summer of 2022. It was interesting to find that though there was a large number of cancellations in June, this paled in comparison to the number of cancellations in the winter months. This could be because of inclement weather, like snow, which makes cancellation much more likely. The uproar over flight cancellations over Summer can be because this is still an outlier for that time period of the year.

3 Method

3.1 Splitting Strategy

My initial plan was to split the dataset using a Stratified K-Fold split because I wanted to maintain the balance of my classes in all of the train, test, and validation splits. This was so that each was representative of the data as a whole, and there wouldn't be a case where one class was not found in one of the sets at all; this best mimics the use case of my eventual ML model because I will feed it a single flight and ask what the flight status is—I will need the model to be trained on all possible flight statuses to accurately predict this case.

In practice the dataset was too large to implement a Stratified K-Fold split, so I opted for a classic random train-validation-test split. I used 95% of the data for training, and 2.5% for both validation and test data. The data is IID and had no time series data. There may be group structure in this dataset depending on the prediction task at hand. For example, you could split groups based on origin destination, flight airline, etc. For my purposes, I did not split on any of these groups.

3.2 Preprocessing

There were occassionally missing values for departure delay when the flight was cancelled, but this was not an issue because of the feature engineering that was done to make this problem in a classification problem (departure delay and cancelled were merged into one ordinal column).

I applied a one-hot-encoder to all the categorical features in my dataset. This meant that Airline, Dest, DestStateName, Origin, OriginStateName, DayOfWeek, DepTimeBlk, and Month were preprocessed this way. This was because they all contained either boolean or string values which not be fed directly into a machine learning model.

I applied a standard-scaler to the numerical features in my dataset. This meant that Distance, and CRSDepTime were preprocessed in this way. Because this values are not between 0 and 1, preprocessing them in this manner makes them more friendly to machine learning models.

After preprocessing with both the one-hot-encoder and the standard-scaler, my dataset has 912 features.

This turned out to be too many features to run a substantial amount of data with. In order to deal with this problem, I dropped categorical features which seemed to have an outsized impact of the number of features without helping predictive power. I dropped "Dest", "DestStateName", "Origin", "OriginStateName". Each of these features had either all 50 states or every airport in the United States which meant a large number of mostly useless columns.

3.3 ML Pipeline

My ML Pipeline takes in a random state and splits the data according to the method in section 3.1. It then preprocesses the data according to section 3.2. After has been prepared for training, I use a ParameterGrid to do find the best hyperparameters for the models I am training. I keep track of the training and validation performance for each model in this loop and only return the one that performed best on the validation data. I run this pipeline for several random states to account for uncertainties due to splitting and due to non-deterministic

ML methods. After I've looped through random states, I calculate the mean test performance and the standard deviation of the test performance.

Once models are trained, I use several methods to understand the trained models. I first generated a confusion matrix to see the pattern of predictions given the true label of the given data. I also optionally generate graphs of different feature importances; these include permutation feature importance, the built-in XGBoost feature importances (only used with XGBoost models), and local and global SHAP feature importance.

3.4 ML Models

I trained four different ML models within my pipeline: an XGBoostClassifier, a RandomForestClassifier, a KNeighborsClassifier, and a LogisticRegression model. This was based on successful methods in previous works (as discussed in the introduction) as well as my intuition over which models would have the highest performance.

I evaluated these models with accuracy and F1 score with a weighted aggregation technique. I used weighted aggregation because it better accounts for imbalanced datasets.

4 Results

4.1 Overview

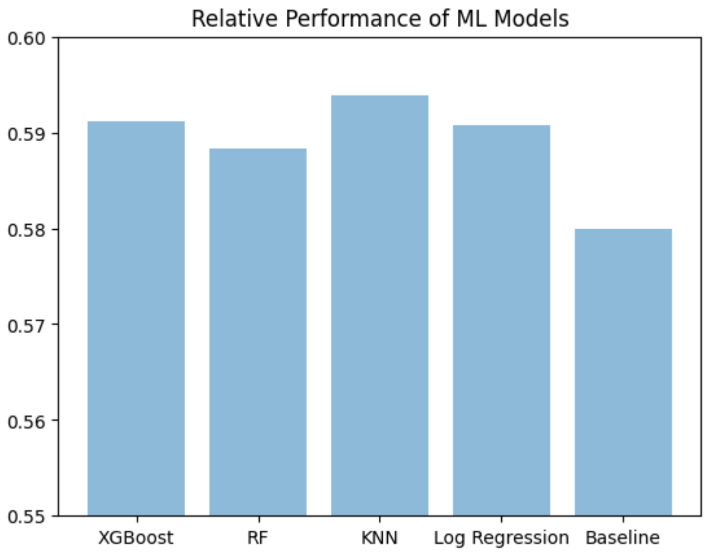

Fig. 5 This graph shows the relative performance of each ML model

Overall the performance of my different methods did not differ significantly from the baseline. Ultimately the amount of data I had (4,000,000 datapoints) significantly hampered the amount of data I was able to input into the models as well as the number of columns I was able to use while doing so. The performance of sklearn's KNeighborClassifier ended up being the best.

4.2 XGBoost Classifier

The XGBoost Classifier achieved an accuracy of 0.5911764705882353 with a standard deviation of 0.011348859708617879. It's f1 score was 0.4688504990160799 compared to a 0.4488056166672677 baseline.

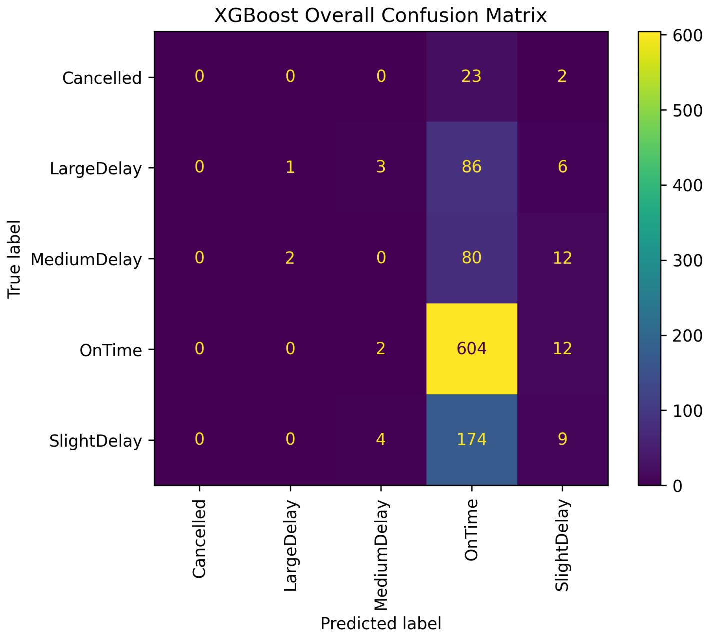

Fig. 6 This graph shows the confusion matrix for the XGBoost model

As seen in the confusion matrix, it more or less learned to always predict that the flight would be on time. At first I was skeptical of whether this was truly learning anything, but on further inspection with different feature importance methods proved to me that it was learning something from the data, even if not being able to achieve the best performance.

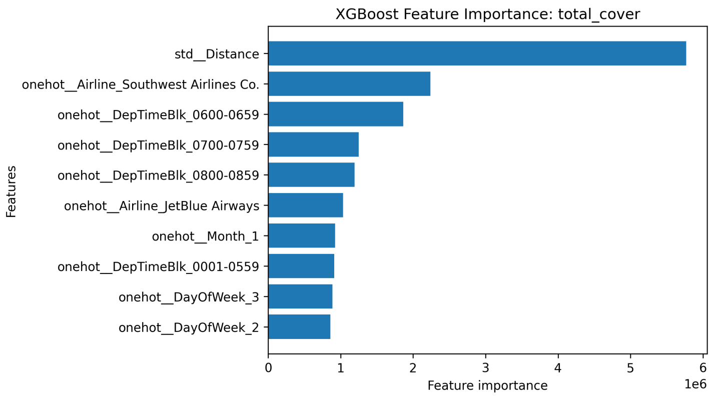

Fig. 7 This shows one built feature importance graph for the XGBoost model

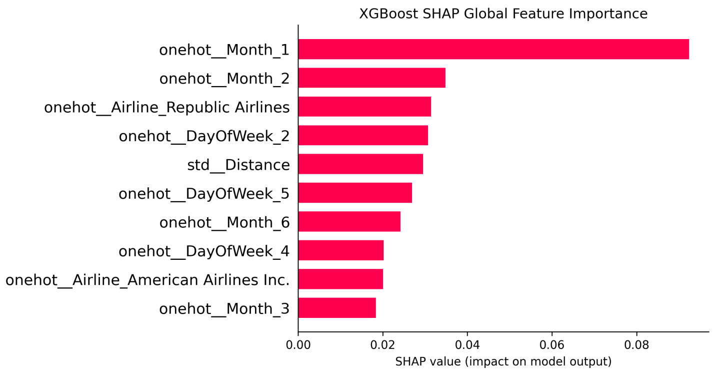

Fig. 8 This graph shows the global SHAP feature importance for the XGBoost model

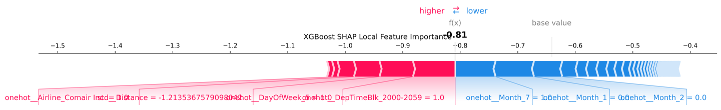

Fig. 9 This graph shows the local SHAP feature importance for the XGBoost model

In the feature importance graphs, we see features like month 1 and 2, and airlines like Republic, Southwest, and American being important to the predictions. This is reminiscent of the EDA step where in month 1 and 2 there were the most cancellations. Additionally the Republic airline is notable in its high on time percentage, while the consumer airlines American, JetBlue and

Southwest are notably not on time very often compared to its peers. Ultimately the models seems to be picking out important features despite the lackluster performance.

4.3 Random Forest Classifier

The Random Forest Classifier achieved an accuracy of 0.5882699097685367 with a standard deviation of 0.003962953132649594. It's f1 score was 0.44253576307667963 and was identical to the baseline.

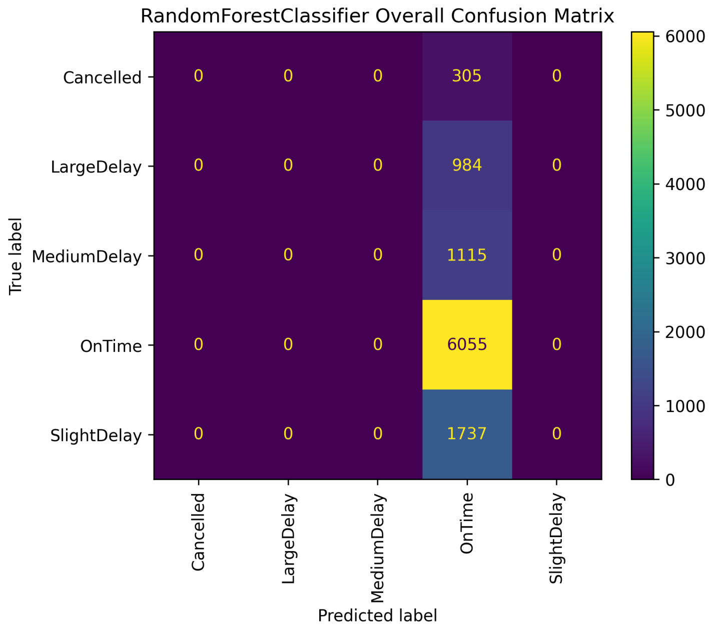

Fig. 10 This graph shows the confusion matrix for the Random Forest model

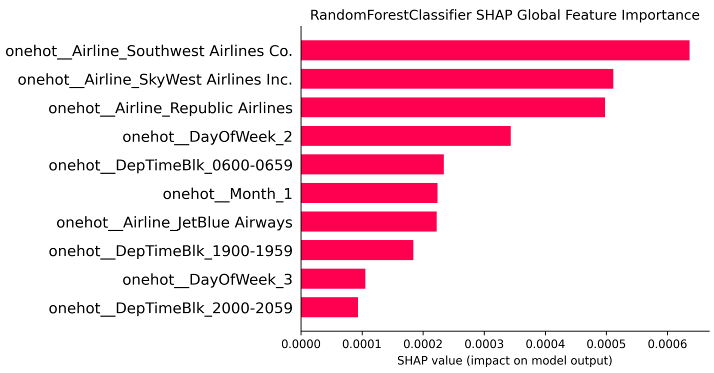

Fig. 11 This graph shows the global SHAP feature importance for the RandomForest model

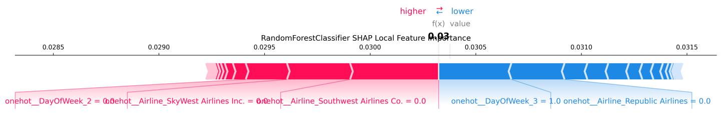

Fig. 12 This graph shows the local SHAP feature importance for the RandomForest model

Even more severely than the XGBoost model, the Random Forest Classifier basically learned to always predict OnTime. Again, however, it does seem to pick out important features to make its predictions.

4.4 K-Neighbor Classifier

The K-Neighbor Classifier achieved an accuracy of 0.5938276448280372 with a standard deviation of 0.0024712633959676107. It's f1 score was 0.47398717689371 29 compared to a 0.44253576307667963 baseline. This was the best performing model that I trained.

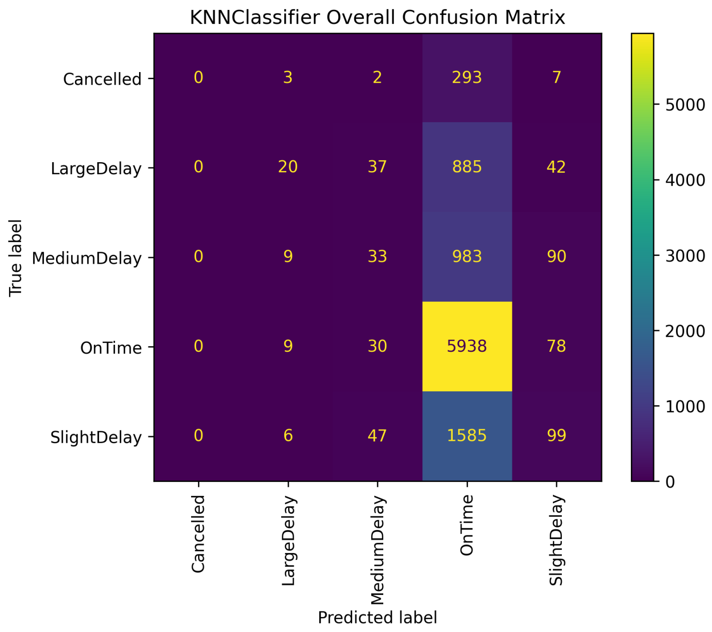

Fig. 13 This graph shows the confusion matrix for the K-Neighbor Classifier

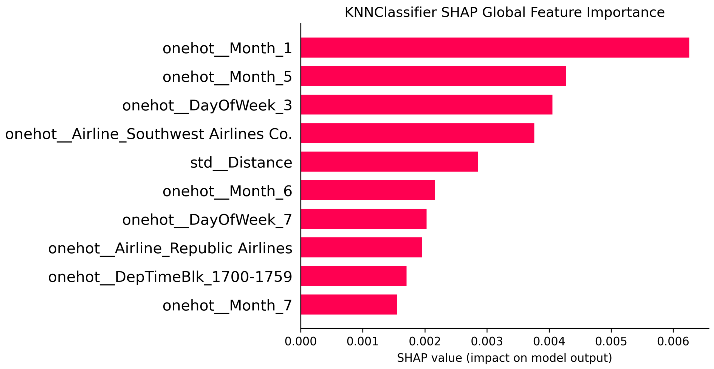

Fig. 14 This graph shows the global SHAP feature importance for the K-Neighbor Classifier

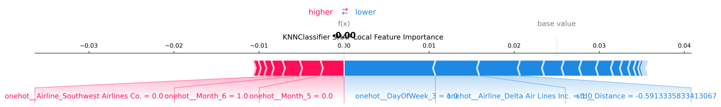

Fig. 15 This graph shows the local SHAP feature importance for the K-Neighbor Classifier

Similar to the XGBoost model, this mainly overfit and learned to always predict OnTime. The model sometimes predicted classes, however, compared to the RandomForestClassifier. The feature importances show that it was using relevant features to make its predictions despite the poor performance. It specifically picked out Month 1 as its most predictive feature, which is the month with the highest cancellations in the training data.

4.5 Logisitic Regression

The Logistic Regression model achieved an accuracy of 0.5938276448280372 with a standard deviation of 0.0024712633959676107. It's f1 score was 0.4592731748994135 compared to a 0.44253576307667963 baseline.

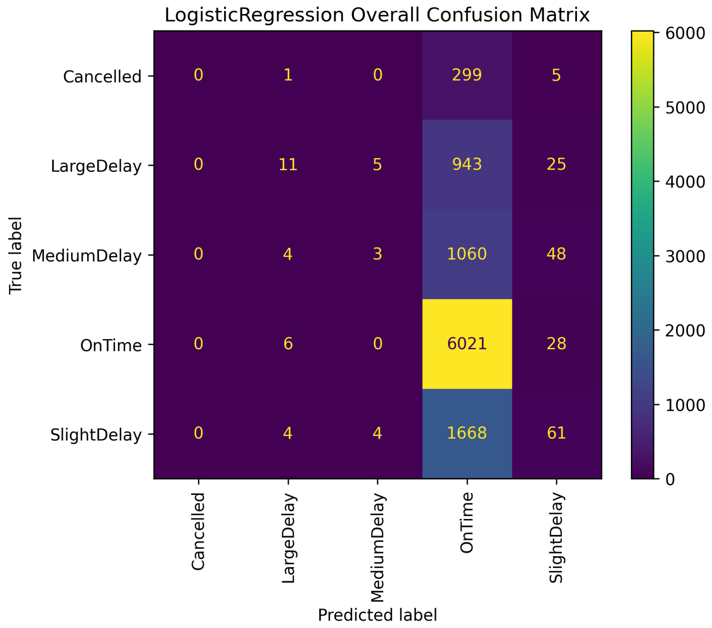

Fig. 16 This graph shows the confusion matrix for the K-Neighbor Classifier

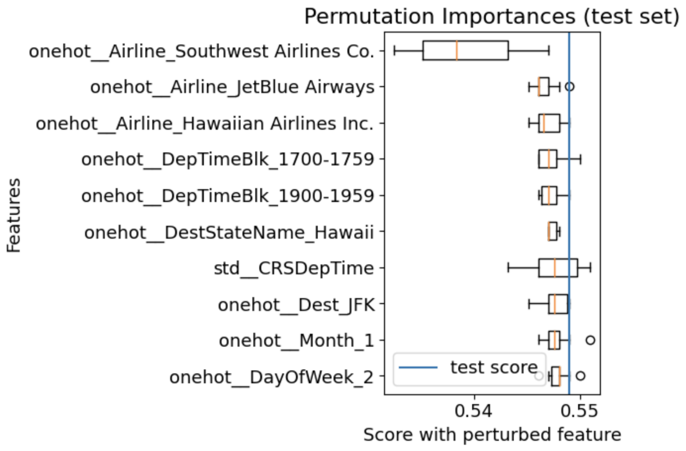

Fig. 17 This graph shows the permutation feature importance for the Logistic Regression model

Similar to the other models, this mainly overfit and learned to always predict OnTime. The permutation feature importance shows that it was using relevant features to make its predictions despite the poor performance. It specifically picked out airlines in the bottom 5 of 'OnTime' percentage as important features. This indicates that it is making predictions that are aligned with the EDA I did at the beginning of the project.

5 Outlook

The main thing I can do in the future to improve this project is to increase the predictive power of the models I am able to generate. The primary avenue of improvement in this direction will likely be to increase the amount of data I am able to use with the models. I was severely hardware and memory limited for this project given the amount of rows and columns in the dataset. I dropped most of the features in order to be able to use any substantial fraction of the rows. Potentially important features like origin and destination were not able to be made use of. With additional data about these origin and destination locations I could potentially reduce the number of options in these feature columns and make them usable. For example, if I could classify them according to the amount of traffic they see on a daily basis, I could preserve some of the important information about them while reducing the number of columns I generate while trying to one-hot encode these columns.

Additionally, I would like to be able to explore Deep Learning as a modeling approach for this problem. This would allow for specifically crafted models that align with the particularities of this problem.

6 Github Repository

My github repository is available here.

References

[1] Fabien, D. (2015). Predicting Flight Delays [Tutorial]. Kaggle. https://www.kaggle.com/code/fabiendaniel/predicting-flight-delays-tutorial

[2] Bhatia, B. (2018). FLIGHT DELAY PREDICTION. Master's Thesis 1-69. https://scholarworks.calstate.edu/downloads/qr46r081g

[3] Mulla, R. (2021). Flight Cancellation Dataset. Kaggle. https://www.kaggle.com/code/robikscube/flight-cancellation-dataset-eda/notebook