Flight Delay Predictor

Machine Learning Project

Milestone 3

Mariam ElGhobary
CSCE Department
The American University In Cairo
m.elghobary@aucegypt.edu
SID: 900211608

Youssef Elhagg
CSCE Department
The American University In Cairo
yousseframi@aucegypt.edu
SID: 900211812

I. INTRODUCTION

In our previous milestone, we conducted appropriate data-cleaning and preprocessing, ending up with a finalized dataset consisting of 481,895 rows and 4073 columns. This preparation made our dataset ready for the next step realized in this milestone whereby we come to decide on which training model would best-suit our application. This milestone was dedicated to training our dataset using various supervised learning models and analyzing the performance, pros and cons of each followed by our final chosen model and any related justifications.

II. PRE-TRAINING PREPARATION

A few integral steps and decisions had to be made prior to the execution of the core objective of this milestone which is the running of the models. Those include:

A. Set Splitting

We created a splitting script which uses sklearn's **train_test_split** function to create randomized train and test sets split at an 80:20 ratio respectively. Hence our Training set consisted of 385,516 data points whilst our test set consisted of a total of 96,379 data points.

B. Label Binning

Since our finalized dataset had the label **DepDelay** as number of minutes, we had to discretize our label to be able to run the classification models and not be restricted by the regression models. From the previous milestone it was clear that our data follows a lognormal distribution and is strongly skewed towards the left (0). Thus the discretization via percentiles would have created very uneven label distributions with the first few percentiles holding almost all of our data and the higher percentiles bearing very few points and hence most models would have probably predicted all points as the same lower percentile label. Therefore we had to come up with a better form of discretization that produced decent label distribution amongst data points, as well as, bearing meaningful splits when it comes to the application itself. Hence we decided to bin our label into 4 bins using a dedicated binning script with the bins/classes numbered 0-3 and they are divided as follows:

- **Class 0:** This is corresponding to delays from 0-15 minutes and this essentially resembles "no delay" as according to the United States Federal Aviation Administration (FAA) this is the span allocated to no delay [1].
- **Class 1:** This is corresponding to delays from 16-45 (inclusive) minutes and this essentially resembles "little delay". This range was also seen to be utilized by the FAA's Air Traffic Control System Command Center [2].
- **Class 2:** This is corresponding to delays from 46-120 minutes. Starting from 2 hour delays, simple airline compensation begin to be distributed in the forms of free snacks, refreshment and other simple services and so it appeared to us as an appropriate breaking point for our final 2 labels [3].
- **Class 3:** This is the final label corresponding to delays of more than 2 hours.

Class distributions amongst training and test data are as follows:

```
##### Training Set #####
Class 0 (<= 15 mins): 46697
Class 1 (15 to 45 mins): 168836
Class 2 (45 to 120 mins): 131077
Class 3 (> 120 mins): 38906

##### Testing Set #####
Class 0 (<= 15 mins): 11718
Class 1 (15 to 45 mins): 42216
Class 2 (45 to 120 mins): 32447
Class 3 (> 120 mins): 9998
```

C. Metric Choice

1. Metrics computed for classification models: **Accuracy** , **Precision**, **Recall** and **F1-Score**. Accuracy measures the ratio of correctly predicted instances out of all instances. Precision resembles the ratio of true positives out of all instances predicted to be positive by the model. Recall, on the other hand, computes the ratio of true positives over actually positive instances. F1 is the harmonic mean of the precision and recall giving each of them equal weights.

   These scores were outputted for each class as well as in the form of overall macro and weighted averages.

2. Metrics computed for regression models:

   Goodness of fit or **R^2** Score was selected as our metric where the value of 1 indicates a perfect fit, zero indicates the fit is not better than predicting the mean and a negative number indicates that we are better off predicting the mean.

III. MODEL RUNNING AND ANALYSIS

1. KNN

- **Experimental Setup**

  Firstly, **GridSearchCV** was used to determine the best k. Ideally, Ks should be tested from one to the root of our data points, but as that is such a high number running on such a range would not be feasible in terms of time or computer memory. Instead we ran on Ks 1 to 5. 5-Fold cross validation was also implemented within our grid search and the best k turned out to be 5.

  Test Data was then passed to the **predict**() function and metrics were computed using predictions and test labels to analyze the model's performance.

- **Parameter Choice**

  For the GridSearch the KNN classifier was passed in along with our **KFold** object (5-folds) and the default scoring method of accuracy was left as is. We also added a verbose=2 parameter to output progress and time taken for each of the 25 fits.

  When it comes to running the final KNN model we used sklearn's **KNeighborsClassifier** and set default parameters where the metric is euclidean. The only inserted parameter was the n_neighbors parameter which was set to our best k which was 5.

- **Performance Evaluation**

  Overall, scores were fairly low with an accuracy of 0.39 and macro averages of remaining metrics being in the 0.27-0.29 range whilst weighted averages of a slightly improved 0.36-0.39 range. Looking at per-class scores only label 1 which is 15-45 minutes produces more acceptable scores which would make sense as it is the label with the most data points. The remaining 3 labels had incredibly low scores thus bringing down overall scores significantly. Label 3 had the lowest scores where recall and f1 were very close to 0.

  A more detailed breakdown of scores is shown below:

	precision	recall	f1-score	support
0	0.18	0.15	0.16	11718
1	0.45	0.61	0.52	42216
2	0.36	0.30	0.33	32447
3	0.16	0.04	0.06	9998
accuracy			0.39	96379
macro avg	0.29	0.27	0.27	96379
weighted avg	0.36	0.39	0.36	96379

- **Pros of KNN**
  - Simple implementation
  - Structure allows easy handling of multi-class data
  - KNN can be applied to data from any distribution and is able to approximate very complex ones locally

- **Cons of KNN**
    - High computational complexity and memory usage hence very time-consuming
    - Low scores for our dataset
    - KNN tends to struggle with excessively high dimensionality hence our cleansed data of nearly 4,000 columns poses challenges for the model

2. Decision Trees

- **Experimental Setup**
    - Classification

    Sklearn's **DecisionTreeClassifier()** was initialized and fed the train data and the binned label.

    - Regression

    Sklearn's **DecisionTreeRegressor()** was initialized and fed the train data and the original label in its numerical form.

    5-Fold cross validation was used in the fitting process of both the classification and regression models as it can reduce performance bias, optimize hyperparameters and reduce overfitting, hence improving our model overall. No other form of tuning/optimization was implemented.

    Test Data was then passed to the **predict**() function and metrics were computed using predictions and test labels to analyze each model's performance.

- **Parameter Choice**
    - Classification

    Sklearn's classifier uses an optimized version of CART. Default parameters were used for example the default **criterion** parameter for splitting was 'gini', **max_depth** was set to 'None' meaning nodes will be expanded until all leaves are "pure" or until all leaves contain less than **min_samples_split** parameter which by default is set to 2.

    - Regression

    Sklearn's regressor model also uses an optimized version of CART. The default parameters for the regression model are essentially the same as the classifier model apart from the default value of the **criterion** parameter being set to 'mse'.

- **Performance Evaluation**
    - Classification

    Overall, scores were fairly low with an accuracy of 0.39 and macro averages of remaining metrics being in the 0.31-0.32 range whilst weighted averages of a slightly improved 0.38-0.39 range. Looking at per-class scores all classes had very low scores none of which exceeding the 0.5. Label 1 which is 15-45 minutes has the highest scores of 0.48-0.5. However, an observation made compared to other model scores is that the difference between Label 3's scores and the rest of the scores here was much lower hence the gaps of performance between classes overall here was reduced.

    A more detailed breakdown of scores is shown below:

	precision	recall	f1-score	support
0	0.17	0.16	0.17	11718
1	0.48	0.50	0.49	42216
2	0.38	0.38	0.38	32447
3	0.23	0.20	0.22	9998
accuracy			0.39	96379
macro avg	0.32	0.31	0.31	96379
weighted avg	0.38	0.39	0.38	96379

    - Regression

    An R^2 score of nearly -0.5 implies that predicting the mean would have been slightly better than the model's performance. Hence the model performance is clearly poor overall.
    R^2:

```
-0.5457513275766954
```

- **Pros of Decision Trees**
    - Low running time
    - Can handle non-linear relationships well
    - Can handle binary columns efficiently which is required for our heavily one-hot -encoded data

- **Cons of Decision Trees**
    - Highly Prone to overfitting however CART's pruning attempts to solve this issue
    - Despite good handling of binary columns, a large number of one-hot encoded columns may lead to the model struggling to determine feature importance

3. Random Forest

- **Experimental Setup**
    - Classification

Sklearn's **RandomForestClassifier()** was initialized and fed the train data and the binned label.

    - Regression

Sklearn's **RandomForestRegressor()** was initialized and fed the train data and the original label in its numerical form.

5-Fold cross validation was used in the fitting process of both the classification and regression models as it can reduce performance bias, optimize hyperparameters and reduce overfitting, hence improving our model overall. No other form of tuning/optimization was implemented.

Test Data was then passed to the **predict**() function and metrics were computed using predictions and test labels to analyze the model's performance.

- **Parameter Choice**
    - Classification

Sklearn's **RandomForestClassifier** uses an optimized version of CART for all of its decision tree classifiers. We used the same default parameters as those for the **DecisionTreeClassifier** from Sklearn. The only difference is that the **RandomForestClassifier** uses the **n_estimators** parameter (number of decision trees) , which by default is set to 100.

    - Regression

Sklearn's **RandomForestRegressor** also uses an optimized version of CART for all of its decision tree regressors. It generally uses all the same default parameters, but the **criterion**, which is set to "squared_error" by default.

- **Performance Evaluation**
    - Classification

Overall, the model's performance was mediocre. All the weighted averages were close to 0.5, because classes 1 (15 to 45 mins) and 2 (45 mins to 2 hours), which have the highest scores, also have the highest distribution. However, classes 0 (0 to 15 mins) and 3 (more than 2 hours), had the lowest scores, which could be because of their low distributions.

Compared to all other models, the random forest classifier performed the best.

A more detailed breakdown of scores is shown below:

	precision	recall	f1-score	support
0	0.31	0.02	0.03	11718
1	0.49	0.78	0.60	42216
2	0.44	0.36	0.40	32447
3	0.61	0.08	0.14	9998
accuracy			0.47	96379
macro avg	0.46	0.31	0.29	96379
weighted avg	0.46	0.47	0.42	96379

    - Regression

The R^2 score of the model was close to 0, which would imply that the model yielded scores essentially as good as predicting the mean:

```
0.08964064245552128
```

- **Pros of Random Forest**
    - Random Forests are less prone to overfitting.
    - Random Forests can handle categorical variables well, which can be beneficial if factors influencing flight delays are categorical (like airline or airport).
    - Provide measures of feature importance, which can be useful for understanding which factors are most influential in predicting flight delays.
- **Cons of Random Forest**
    - Random Forests can create a large number of deep trees, leading to a model that requires a lot of memory and is slow to evaluate.
    - Random Forests can be biased towards variables with more levels. Therefore, the variable importance scores from Random

Forest are not reliable for this type of data.

4. Linear Regression

- **Experimental Setup**
  Sklearn's **LinearRegression()** was initialized and fed the train data and the original numerical label. 5-Fold cross validation was used in the fitting process of both the classification and regression models as it can reduce performance bias, optimize hyperparameters and reduce overfitting, hence improving our model overall. No other form of tuning/optimization was implemented.

  Test Data was then passed to the **predict**() function and metrics were computed using predictions and test labels to analyze the model's performance.

- **Parameter Choice**
  The linear regression model does not have any hyperparameters to specify or initialize the classifier with.

- **Performance Evaluation**
  The R^2 is a very large negative number implying the model is performing much worse than predicting the mean. Hence the model fails to capture any meaningful relationship between the features and the label and possibly overfits.

  R^2:

  ```
  -16204181595016.0
  ```

- **Pros of Linear Regression**
  - Simple model which is computationally easy to implement and utilize
  - Low running time
- **Cons of Linear Regression**
  - Very poor metrics for our Dataset
  - Assumes a linear relationship between the features and the label hence will significantly struggle to capture more complex nonlinear relationships in the data which may lead to underfitting.

5. Logistic Regression

- **Experimental Setup**
  Sklearn's **LogisticRegression()** was initialized and fed the train data and the binned label. 5-Fold cross validation was used in the fitting process of both the classification and regression models as it can reduce performance bias, optimize hyperparameters and reduce overfitting, hence improving our model overall. No other form of tuning/optimization was implemented.

  Test Data was then passed to the **predict**() function and metrics were computed using predictions and test labels to analyze the model's performance.

- **Parameter Choice**
  Sklearn's **LogisticRegression** uses default values for its parameters when initializing the model. For example, the **max_iter** parameter (maximum number of iterations), which by default is set to 100. We set the **multi_class** parameter to "ovr" (one-vs-rest if there are more than 2 classes). Also, according to Sklearn, the **solver** should be set to "newton-cholesky" if the number of rows is much greater than that of the columns, and if there are a lot of one-hot-encoded features with rare values, which correspond to our case.

- **Performance Evaluation**
  Overall, the model's performance was mediocre. All the weighted averages were close to 0.4, because classes 1 (15 to 45 mins) and 2 (45 mins to 2 hours), which have the highest scores, also have the highest distribution. However, classes 0 (0 to 15 mins) and 3 (more than 2 hours), had the lowest scores, which could be because of their low distributions.

  A more detailed breakdown of scores is shown below:

	precision	recall	f1-score	support
0	0.25	0.01	0.02	11718
1	0.47	0.77	0.58	42216
2	0.40	0.34	0.37	32447
3	0.30	0.01	0.02	9998
accuracy			0.45	96379
macro avg	0.36	0.28	0.25	96379
weighted avg	0.40	0.45	0.38	96379

- **Pros of Logistic Regression**
  - Less computationally intensive than some other methods like SVM or Random Forest, making it a good choice for problems with a large number of observations.
  - Less sensitive to noise and overfitting and can perform well even if some predictors are correlated.
  - Provides coefficients that can be interpreted as the effect of a one-unit change in the predictor variable on the odds of the outcome, holding all other predictors constant.
- **Cons of Logistic Regression**
  - Assumes a linear decision boundary. It might not perform well when the decision boundary is not linear, which corresponds to our case.
  - Assumes that the predictors are independent of each other, which might not always be the case in a flight predictions problem.

6. Neural Networks

- **Experimental Setup**

  We used Tensorflow's Keras library to build an MLP (Multi-Layer Perceptron) Classifier and Regressor, with one hidden layer. We chose to run the MLP model using Tensorflow instead of the one provided by Sklearn because of Tensorflow's **Scalability**: TensorFlow is designed to handle large datasets and complex models, making it suitable for deep learning. **Flexibility:** TensorFlow provides more flexibility in designing complex models. You can easily design an MLP with multiple hidden layers, different types of layers (convolutional, recurrent, etc.), and various activation functions. The model was fed the train data with the label once as binned and once in its initial numerical form. 5-Fold cross validation was used in the fitting process as it can reduce performance bias, optimize hyperparameters and reduce overfitting, hence improving our model overall. No other form of tuning/optimization was implemented.

  Test Data was then passed to the **predict**() function and metrics were computed using predictions and test labels to analyze the model's performance.

- **Parameter Choice**

  The initialization of Tensorflow's model is a bit different from all other Sklearn models. It involves manually adding the layers, and choosing the appropriate number of neurons for each layer. As there were no default values for them, we chose to add one hidden layer (since we have less than 500k rows) and 256 neurons for that layer. The hidden layer's activation function is "relu" by default. As per the output layer and the compilation of the model (compilation is not the actual fitting of the model, it is just for initialization), they are a bit different for the Classifier and Regressor:
  - Classifier:
    - **Hidden Layer:** the number of neurons for the hidden layer corresponds to the number of classes (4 in our case). The activation function is "softmax", which is the default.
    - **Compilation:** the loss function used for the compilation is "categorical_crossentropy", which is the default for a classification problem.
  - Regressor:
    - **Hidden Layer:** There is only one neuron for the hidden layer as the output is only 1 continuous value. No activation function by default.
    - **Compilation:** the loss function used for the compilation is "mse", which is the default for a regression problem.

  Both the classifier and the regressor used the "adam" optimizer. We also included an

EarlyStopping object and set the patience to 20. This means that the fitting will stop when a certain monitored quantity ("val_loss" by default) does not improve for 20 iterations. This ultimately helps in avoiding overfitting. Some other default parameters were used when fitting: **epochs** (or iterations) which are set to 100 by default, and **batch_size** which is set to 32 by default.

- **Performance Evaluation**
    - Classification

Overall, scores were not high due to the model's failure to successfully predict classes 0 and 3. The scores for precision, recall, and f1-score were all 0 for these 2 classes. Class 1 (15 to 45 mins) performed the best which would make more sense since our data distribution was higher for that range.

A more detailed breakdown of scores is shown below:

	precision	recall	f1-score	support
0	0.00	0.00	0.00	11718
1	0.46	0.91	0.61	42216
2	0.42	0.16	0.24	32447
3	0.00	0.00	0.00	9998
accuracy			0.45	96379
macro avg	0.22	0.27	0.21	96379
weighted avg	0.34	0.45	0.35	96379

- Regression

The R^2 score was also not good as it is closer to 0 than 1 implying that the model yielded scores essentially as good as predicting the mean:

```
0.05237978049936287
```

- **Pros of MLP**
    - Non-linearity, which can be beneficial if the factors influencing flight delays are non-linearly related.
    - MLPs can be configured with different numbers of hidden layers and neurons to adjust the model complexity based on the problem at hand.
    - MLPs can handle noise in the input data, which is present in our case.
- **Cons of MLP**
    - MLPs with many layers/neurons can overfit to the training data, leading to poor generalization to unseen data.
    - MLPs can be difficult to train effectively.
    - MLPs are sensitive to the scale of the input features, so careful preprocessing of data is required.

7. Naive Bayes

- **Experimental Setup**

Sklearn's **GaussianNB()** was initialized and fed the train data and the binned label. 5-Fold cross validation was used in the fitting process of both the classification and regression models as it can reduce performance bias, optimize hyperparameters and reduce overfitting, hence improving our model overall. No other form of tuning/optimization was implemented.

Test Data was then passed to the **predict**() function and metrics were computed using predictions and test labels to analyze the model's performance.

- **Parameter Choice**

Default parameters were used and so the **priors** parameter is by default set to 'uniform', meaning that it is assumed that each class is equally likely. The second parameter **var_smoothing** is by default set to 1e-9. This parameter adds a small value to the variances of all features to stabilize computation which is what is meant by "smoothing".

- **Performance Evaluation**

Overall, scores were fairly low with the lowest accuracy out of all models at 0.29 and macro averages of remaining metrics being in the 0.27-0.31 range whilst weighted averages of a slightly improved 0.29-0.38 range. Looking at per-class scores only label 1 which is 15-45 minutes produces the highest relative scores but they were still very low with the highest being precision at 0.5. The remaining 3 labels had slightly lower scores but the gaps between them were not very significant apart from label 3 which had a recall notably higher than its f1 and precision scores at 0.44 which is the highest recall for label 3 out of all models. This implies that the classifier predicts more instances as positive (for

label 3) or predicts most borderline points as positive, decreasing false negatives and increasing false positives. The low precision could also be justified by imbalanced class distributions as label 3 has the smallest number of instances and so there is a large number of negative (non-label 3) instances and a relatively small number of positive instances decreasing true positives.This could mean that even a small number of false positive predictions can substantially reduce precision.

A more detailed breakdown of scores is shown below:

	precision	recall	f1-score	support
0	0.16	0.29	0.21	11718
1	0.50	0.31	0.38	42216
2	0.37	0.21	0.27	32447
3	0.14	0.44	0.22	9998
accuracy			0.29	96379
macro avg	0.29	0.31	0.27	96379
weighted avg	0.38	0.29	0.31	96379

- **Pros of Naive Bayes**
  - Simple and fast to implement and run
  - Due to computational simplicity the model is highly scalable and can handle datasets with a large number of data points and a large number of features which is the case for our dataset
- **Cons of Naive Bayes**
  - Low metrics
  - The model is very sensitive to prior probabilities especially when the dataset is imbalanced or the class distribution is skewed. This can lead to significant prediction bias and our data is in fact very skewed towards the left (0)

8. SVM

- **Experimental Setup**

  According to Sklearn, we should use the **LinearSVC** and **LinearSVR** libraries for large datasets (larger than tens of thousands). This is because these libraries are more optimized to handle large datasets, which train the models much faster.
  - Classification

  Sklearn's **LinearSVC()** was initialized and was fed the train data and the once binned label.
  - Regression

  Sklearn's **LinearSVR()** was initialized and fed the train data and the original label in its numerical form.

  5-Fold cross validation was used in the fitting process of both the classification and regression models as it can reduce performance bias, optimize hyperparameters and reduce overfitting, hence improving our model overall. No other form of tuning/optimization was implemented.

  Test Data was then passed to the **predict**() function and metrics were computed using predictions and test labels to analyze the model's performance.

- **Parameter Choice**
  - Classification

  Sklearn's **LinearSVC** library uses default values for its parameters when initializing the model. The default values include the **loss** function which is set to "squared_hinge" by default. The **multi_class** strategy, which is set to "ovr" (one-vs-rest, if there are more than 2 classes) by default. The **max_iter** variable (maximum number of iterations to run), which is set to 1000 by default.
  - Regression

  Sklearn's **LinearSVR** library also uses default values for its parameters when initializing the model. It generally uses the same default parameters apart from the **loss** function, which we set to "squared_epsilon_insensitive". It also obviously does not have the **multi_class** parameter since it is a regression model.

  In addition, both models set the **dual** parameter to False. According to Sklearn, if the number of rows is much larger than the number of columns, **dual** should be set to False.
- **Performance Evaluation**
  - Classification

  Overall, scores were not high due to the model's failure to successfully predict classes 0 and 3. The scores for precision, recall, and f1-score were all 0 for these 2 classes. Class 1 (15 to 45 mins) performed the best which would make more sense

since our data distribution was higher for that range.

A more detailed breakdown of scores is shown below:

	precision	recall	f1-score	support
0	0.00	0.00	0.00	11718
1	0.46	0.85	0.60	42216
2	0.41	0.24	0.31	32447
3	0.00	0.00	0.00	9998
accuracy			0.45	96379
macro avg	0.22	0.27	0.23	96379
weighted avg	0.34	0.45	0.37	96379

- Regression

The R^2 score was also not good as it is a negative number, implying that the model yielded scores essentially as good (or even worse) as predicting the mean:

```
-0.01595606236714242
```

- **Pros of SVM**
  - SVMs are effective when the number of features is large, which can be beneficial if there are many factors influencing flight delays.
  - Especially in high-dimensional space, SVMs are less prone to overfitting compared to some other algorithms.
  - SVMs are designed to find the hyperplane that maximizes the margin between classes, which can lead to more robust predictions.
- **Cons of SVM**
  - A relatively small number of mislabeled examples can dramatically decrease the performance of the SVM.
  - Training an SVM can be computationally intensive, particularly for larger datasets.

IV. COMPARATIVE ANALYSIS OF BEST MODELS

Seeing as our regression models yielded very poor scores indicating that we would have been much better off predicting the mean, we narrowed down our choices to the classification models.

Although many models yielded relatively acceptable scores for the most prominent labels which were 1 and 2, we disregarded models with zeroed scores for labels 0 and 3 so that the model is not functioning as if it were a binary classification model.

Amongst models fitting such criteria, we prioritized models with higher weighted average scores over macro-average scores as our label distribution was quite uneven and so factoring this into our scores is essential.

Based on the points mentioned above, we concluded that the best three models were:

1. Decision Trees

Decision trees yielded one of the higher sets of weighted metric scores with non-zero scores for labels 0 and 3. Out of our best 3 models its scores for labels 0 and 3 were not only relatively high but also quite close to each other.

Decision trees also deal well with data with high complexities and many categorical values which is essentially the case for our dataset.

Although weighted scores are high and gaps are not significant the individual scores do not reach a significant value with the highest per class score being label 1's precision at 0.5. So essentially we have mediocre performance for all labels.

We are reminded with the score breakdown once again below as a reference:

	precision	recall	f1-score	support
0	0.17	0.16	0.17	11718
1	0.48	0.50	0.49	42216
2	0.38	0.38	0.38	32447
3	0.23	0.20	0.22	9998
accuracy			0.39	96379
macro avg	0.32	0.31	0.31	96379
weighted avg	0.38	0.39	0.38	96379

2. Random Forest

Random Forest yielded the higher sets of weighted metric scores. Scores are significantly higher overall than decision trees and logistic regression.

Like decision trees, it deals well with data with high complexities and many categorical values which is essentially the case for our dataset.

Here we note that the scores for labels 0 and 3 although non-zero both have a very low recall compared to precision. This implies that the model is more conservative on making positive predictions for these labels leading to higher false negatives and lower false positives.

However we see that some labels here have much higher relative scores with the highest being label 1's recall at 0.78 followed by label 3's precision at 0.61.

We are reminded with the score breakdown once again below as a reference:

	precision	recall	f1-score	support
0	0.31	0.02	0.03	11718
1	0.49	0.78	0.60	42216
2	0.44	0.36	0.40	32447
3	0.61	0.08	0.14	9998
accuracy			0.47	96379
macro avg	0.46	0.31	0.29	96379
weighted avg	0.46	0.47	0.42	96379

3. Logistic regression

Logistic regression also yielded one of the higher sets of weighted metric scores.

Similarly to random forest we can see that although labels 0 and 3 had non-zero scores, their recall was very low relative to precision.

One of logistic regression's biggest perks is that it is less computationally expensive than random forest however there is a clear trade-off between this and the outputted scores.

	precision	recall	f1-score	support
0	0.25	0.01	0.02	11718
1	0.47	0.77	0.58	42216
2	0.40	0.34	0.37	32447
3	0.30	0.01	0.02	9998
accuracy			0.45	96379
macro avg	0.36	0.28	0.25	96379
weighted avg	0.40	0.45	0.38	96379

V. CHOSEN MODEL

We decided to choose Random Forest as our final model as it yielded the highest overall weighted metrics and highest accuracy whilst outputting non-zeroed scores for labels 0 and 3.

Random forests also tend to be less susceptible to overfitting than decision trees due to the bagging methods and feature randomness in the creation of each tree.

There is a running-time trade-off but here we prioritized higher scores as the accuracy of scores in flight delay prediction is integral. Although decision trees had better scores for labels 0 and 3 with less significant gaps, we chose random forest as its scores for labels 1 and 2 are much higher. We acknowledge that prediction scores of all labels are significant but we decided that as long as the labels have non-zero scores we will prioritize the model with higher scores for labels 1 and 2 as they are the most frequent delay ranges and are going to be the labels of highest interest for our application's users.

VI. REFERENCES

[1] "Flight cancellation and delay." Wikipedia. Accessed April 7, 2024. Available at: https://en.wikipedia.org/wiki/Flight_cancellation_and_delay

[2] "Flight Delay Information - Air Traffic Control System Command Center." Federal Aviation Administration. Accessed April 7, 2024. Available at: https://www.fly.faa.gov/flyfaa/usmap.jsp?legacy=true

[3] "Flight delay 2 hours: Your rights as a passenger." Flightright. Accessed April 7, 2024. https://www.flightright.com/blog/flight-delay-2-hours-claims

Github Repo:
https://github.com/useframi1/Machine-Project.git