SD201 – Large Dataset Mining

Ali Mammadov
ali.mammadov@ip-paris.fr

Diego Gomez
diego.gomezmijangos@ip-paris.fr

Bora Gökbakan
umit.gokbakan@ip-paris.fr

December 10, 2021

Keywords: Machine Learning, Data pre-processing, Feature Engineering, Model Training

1 Introduction

As the pandemic slowly settles in a significant part of the developed world, air flight is returning to its pre-Covid regularity. With this regularity comes also the increase of traffic, increase in the probability that at least one flight encounters technical or staff related difficulties, which then inevitably leads to a delay in the departure of the flight. Flight delays can be stressful and frustrating to passengers because they sometimes entail missing an important event, such as a Christmas this time of the year.

In the following report we seek to analyze a dataset of the delays that took place in a set of airports in 2019, more specifically we will be focusing on the file titled `train_test.csv`[1]. With the help of this dataset we wish to answer some questions to better understand the problem at stake,

- What characteristics of a given flight are more likely to indicate that this flight will be delayed.
- Which month is responsible for the most delays?
- Which airport is responsible for the most delays?
- Is it possible to create a model that accurately predicts whether or not there will be a delay.

The report is organized following the general structure of a Machine learning Pipeline. We start by describing the Data pre-processing we implemented, we the explored the data while creating a dashboard to visualize the data set, finally we test our data with a variety of models. Afterwards we compare several models to attempt to obtain the best possible score.

[1]https://www.kaggle.com/threnjen/2019-airline-delays-and-cancellations?select=train_test_small.csv

This is followed by a showcase of our results. Finally we conclude on the topic answering our initial questions.

2 Data Exploration

2.1 Data features

The features available in the data set can be found on the Kaggle link. A complete list of the features are as follows:

- `MONTH` The month of the flight's departure
- `DAY_OF_WEEK` Day of the week of the flight's departure
- `DEP_DEL15` 1 if the flight is delayed by more than 15 minutes 0 otherwise
- `DISTANCE_GROUP` Grouping by distance of the flights. This features gives a sense of flights that will flight similar distances.
- `DEP_BLOCK` Time of the day the flight departed. This feature is categorical taking values such as MORNING
- `SEGMENT_NUMBER` Number of the segment the airplane will fly in at departure. A segment is a flight between 2 airports. A flight can have multiple segments in the case of scales
- `CONCURRENT_FLIGHTS` Flights leaving from the airport in the same departure block
- `NUMBER_OF_SEATS` Number of seats on the aircraft
- `CARRIER_NAME` Name of the carrier in charge of the flight
- `AIRPORT_FLIGHTS_MONTH` Average airport flights per month
- `AIRLINE_FLIGHTS_MONTH` Average airline flights per month
- `AIRLINE_AIRPORT_FLIGHTS_MONTH` Average flights per month for airline and airport
- `AVG_MONTHLY_PASS_AIRPORT` Average passengers for the departing airport for the month
- `AVG_MONTHLY_PASS_AIRLINE` Average passengers for airline for month
- `FLT_ATTENDANTS_PER_PASS` Flight attendants per passenger for airline
- `GROUND_SERV_PER_PASS` Ground service employees (service desk) per passenger for airline
- `PLANE_AGE` Age of departing aircraft
- `DEPARTING_AIRPORT` Departing airport
- `LATITUDE` Latitude of departing airport
- `LONGITUDE` Longitude of departing airport
- `PREVIOUS_AIRPORT` Previous airport that aircraft departed from
- `PRCP` Inches of precipitation for day

- SNOW Inches of snowfall for day
- SNWD Inches of snow on ground for day
- TMAX Max temperature for day
- AWND Max wind speed for day

2.2 Visualisations

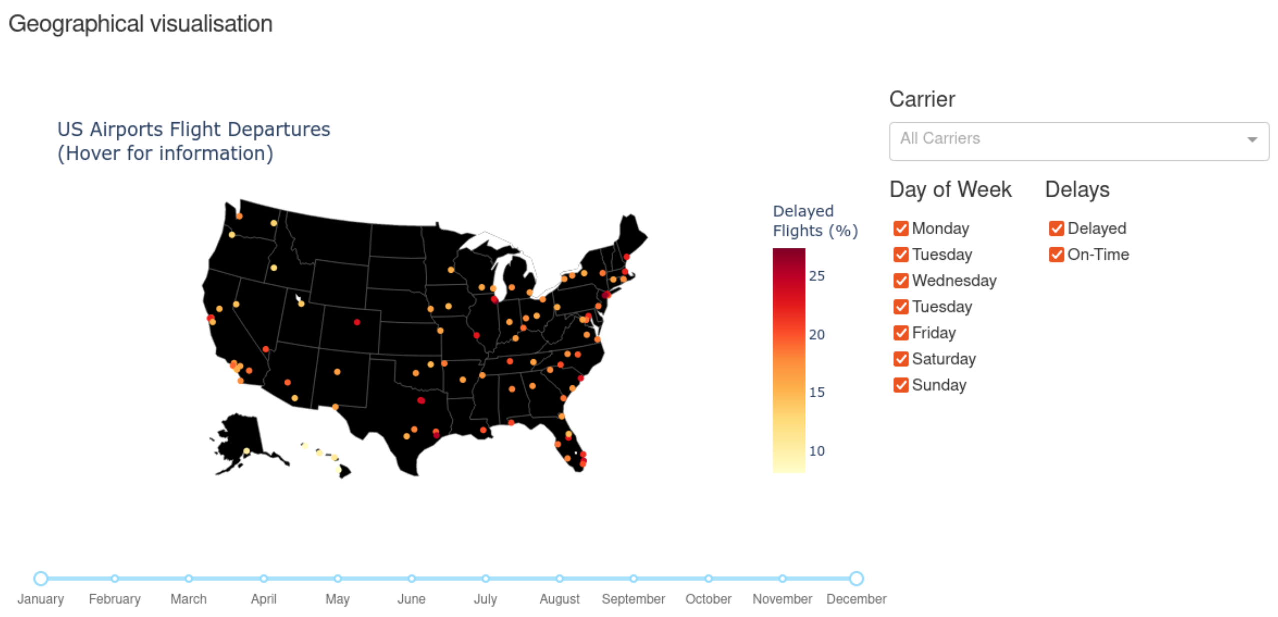


Figure 1: A screenshot from the *Departure Maps* tab of the dashboard

For visualisation and gaining insight into the data, an interactive dashboard was implemented using the *Dash* framework and the *Plotly* library. The service is served in a Docker container on an AWS EC2 instance for high availability.

The dashboard implements a variety of filtering tools in order to see how delays are affected by certain variables, like the day of the week, month, airport, and the carrier. The interactive map is the most comprehensive plot, which gives aggregate data about the airport selected across the selected carriers and months. The dashboard uses a *pandas DataFrame* to manage data filtering and aggregations over selected filters on the fly, and as such, there is some inevitable latency, although this is largely mitigated by using pre-aggregated data where the initial dataset was grouped by the aforementioned variables of control, and the remaining rows were both summed and counted to allow for further aggregations after data filtering. This brought down the original dataset from more than 6 million rows to a more manageable 150 thousand rows.

Plotting the data (Fig. 2) also allows us to answer some of the questions we asked earlier. For instance, the bar chart quickly reveals that August was the month where most flight delays occurred, however, June, a much less busy month, saw a higher percentage of delays.

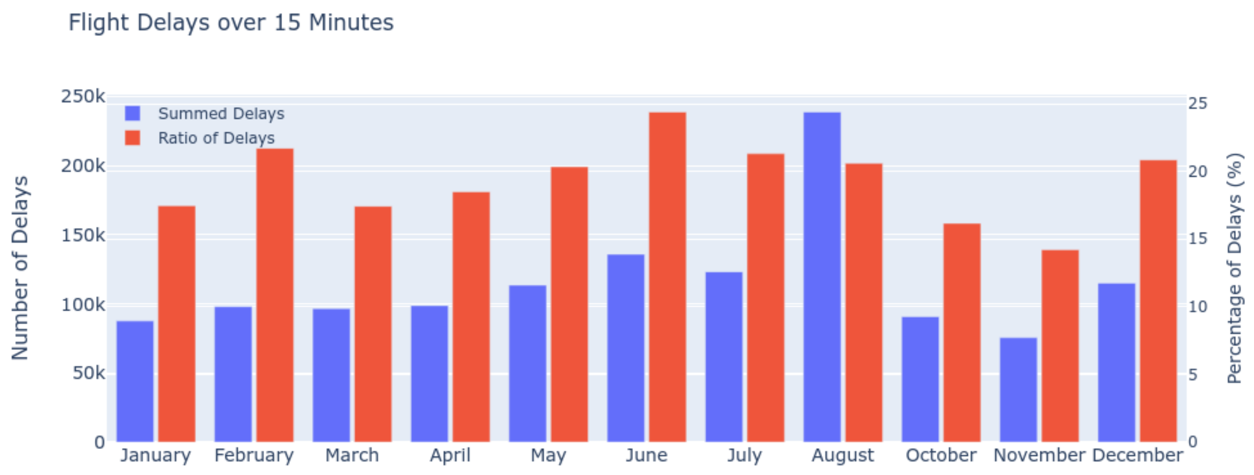


Figure 2: A screenshot from the *Plots* tab of the dashboard

3 Data preprocessing

Our study will be focused on the file titled `train_test.csv`. In the following we will be describing the steps taken to make the dataset more suited to be used by a machine learning pipeline.

3.1 Cleaning

Initially the data present in the file `train_test.csv` is quite raw. Therefore we need to apply some classical cleaning techniques to transform it into usable data.

We apply two techniques to clean the data. Firstly, we encode all of the categorical information. That is for instance features such as `DEP_BLOCK`,`CARRIER_NAME`,`DEPARTING_AIRPORT`,`PREVIOUS_AIRPORT` that correspond respectively to the departing time block (eg. MORNING), the name of the airline carrier, the name of the departing airport and the name of the previous airport (if it exists). Encoding this data makes it so it is understandable by a computer. We decided to use a categorical encode for the feature engineering part (section 3.2)). That is due to the fact that implementing a hot encoding version proved to be too heavy for our limited hardware, even with the use of sparse matrices.

Secondly, we apply scaling techniques to the data to give equal importance to all of the features. Indeed if we do not apply a scaling method some features with naturally higher values such as `CONCURRENT_FLIGHTS` could dominate the model. With a histogram we can observe how our data is distributed visually. After plotting the histograms, our intuition is confirmed, some of the features have a large range of values(see Figures 3, 4). Consequently, we use traditional scaling techniques that we use to solve this problem are the following:

- Standardization.embeds the data in the range [0,1]:

$$Z = \frac{X - X_{min}}{X_{max} - X_{min}} \tag{1}$$

- Normalization. brings the mean to 0 and standard deviation to 1:

$$Z = \frac{X - \mu}{\sigma} \tag{2}$$

  where μ is the mean and σ is the standard deviation.

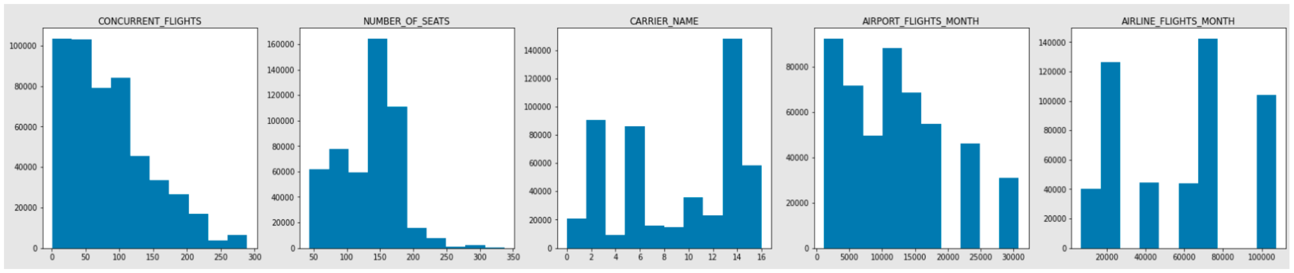


Figure 3: Before cleaning

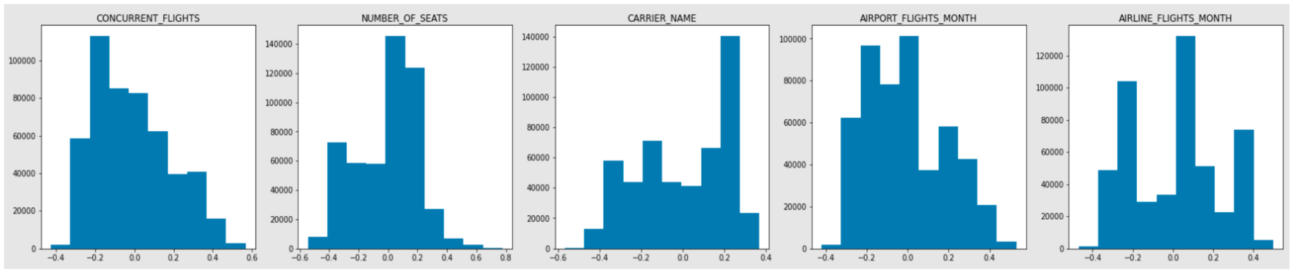


Figure 4: After cleaning

3.2 Feature engineering

With the data cleaned we can proceed to the extraction and selection of the best features. This also allows us to understand the inherit dependencies the data possess. In the following we measure accuracies to respect to the catboosting method later described (section 5).

Firstly, we perform a selection with respect to two different statistical tests, the χ^2 and f-classif tests. The χ^2 test measure the correlation between a given feature and the target value, that is if the value of this test is low, it means it does not provide useful information to the prediction of the target value. In this case the target is the features `DEP_DEL15`. The f-classif test performs the classical ANOVA F-value test that allows us to determine if the feature is statistically significant with respect to the data.

For multiple values of $k \in \{1, \dots, 15\}$ we selected the k best features according to the score. The accuracy for the 2 classes of the target featured were measured, results can be seen in figure 5 and 6. In fact these accuracies correspond to the diagonal of the normalized confusion matrix for each k. In both figures, one can see that there is a very big jump when choosing $k = 3$. This means that with these 3 features the model managed to predict the classes of the data points with a decent test accuracy. The features that correspond to this first peak are `DEP_BLOCK'`, `'SEGMENT_NUMBER'` and `'PREVIOUS_AIRPORT'` for both statistical tests.

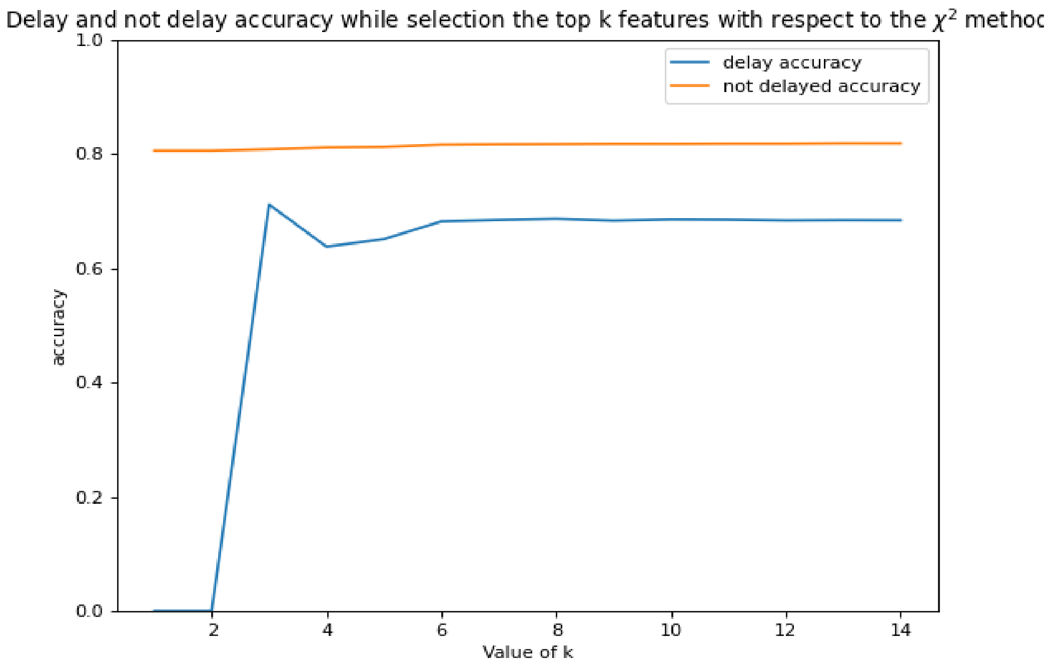


Figure 5: feature selection with the χ^2 method

In the feature extraction step we decided to attempt to apply the classical Principal Components Analysis (PCA) technique. The hope is that our large number of features gets transformed into a lower dimensional space that maximizes the variance. This means that in this new vector space the data that is most different will be better spread out and we hope that this helps the model when fitting. In figure 7 we can see the surprising fact that the whole data set can be predicted via 1 feature. This means that there exists some kind of linear combination of our original dataset that allows us to predict the classes with a good accuracy. Probably a combination of `'DEP_BLOCK'`, `'SEGMENT_NUMBER'` and `'PREVIOUS_AIRPORT'` given our previous findings.

In the following we still decided to use the whole dataset for the following reasons. Firstly, one can see in the jupyter notebook for feature selection that the individual statistical scores of the rest of the features are not negligible. We are hoping that perhaps another model besides the one used in these experiements can be used to better extract useful information

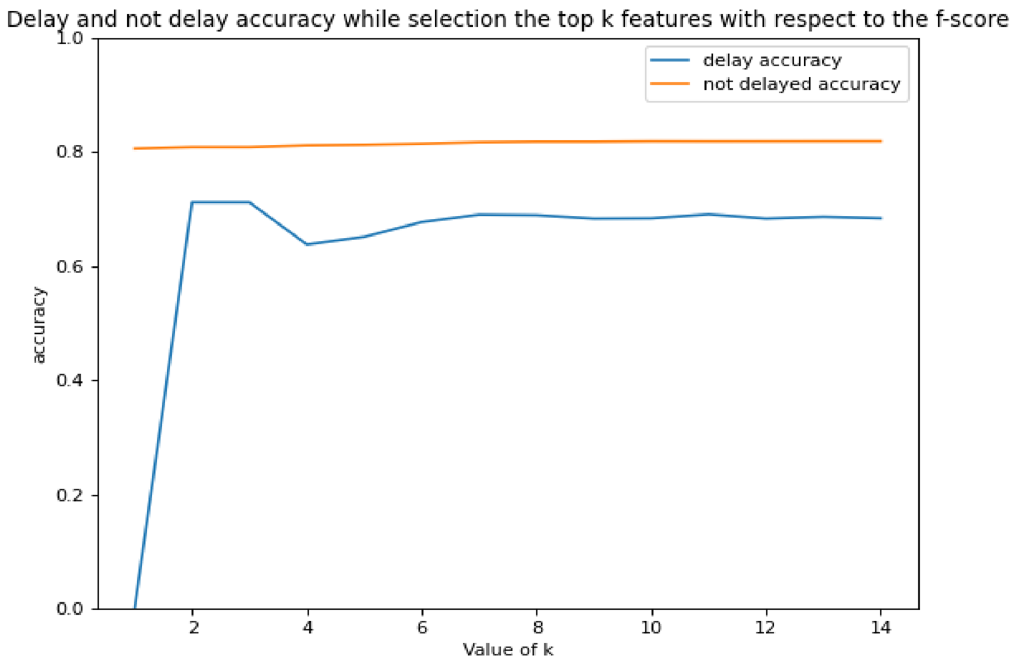


Figure 6: feature selection with the f-score test

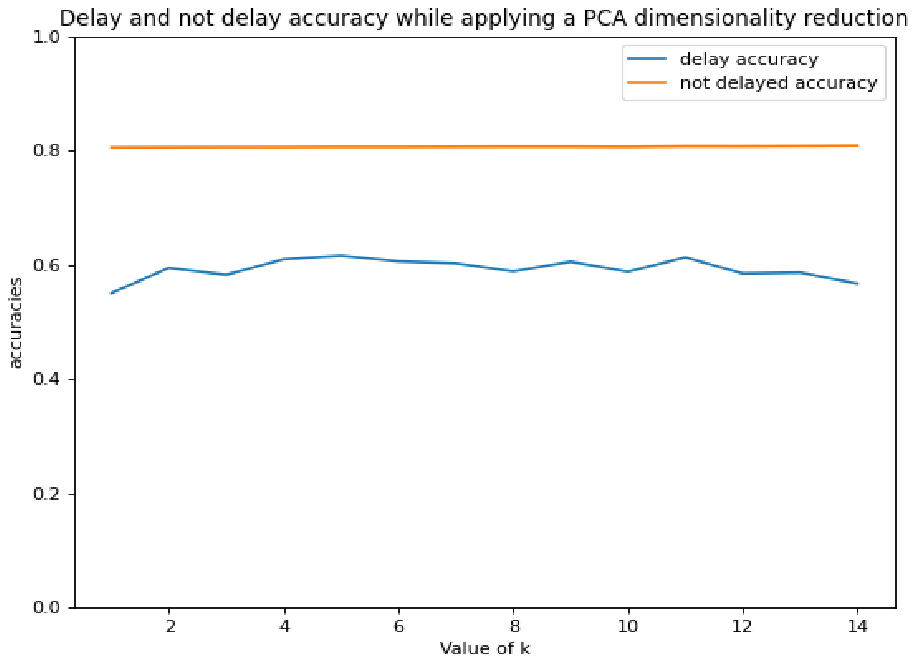


Figure 7: feature selection with PCA.

about these features and thus increase the accuracy. On top of that, some intuitive features were not present here such as snow, precipitation, wind and temperature, we want to keep these features since we believe it will make our model more robust.

4 Clustering

In this project, we also tried to cluster our data set and see a spatial relationship existed between the instances. To cluster the dataset we applied the K-means algorithm and to detect the number of clusters we applied the next three methods:

- Elbow - one of the most used methods to find optimal number of clusters. The main idea of this method is to run the clustering algorithm several times while increasing the number of clusters and compute at each time the difference in the SSE(sum of square error) score of clusters. The most common way to calculate the SSE score is by computing the mean squared distance between cluster centroids and their instances. It was proven in class that the SSE converges under the k-means algorithm. The principle of the Elbow approach is to detect the number K such that the decrease in the SSE score is less important. One can do this visually by looking for an "elbow" in the plot.

- Silhouette - This method is the calculation of silhouette coefficient which is a measure of how similar a data point is within-cluster compared to other clusters. The silhouette score can vary from -1 to 1. The best is 1 and -1 is the worst one.

- Davies-Bouldin - is a method to rate clustering algorithms. This approach is mainly used to evaluate the performance of the K-means algorithm for a given K. In simpler terms, the method is based on the ratio between distance within-clusters and distance between centroids. And the best clustering is that which minimizes the Davies–Bouldin score.

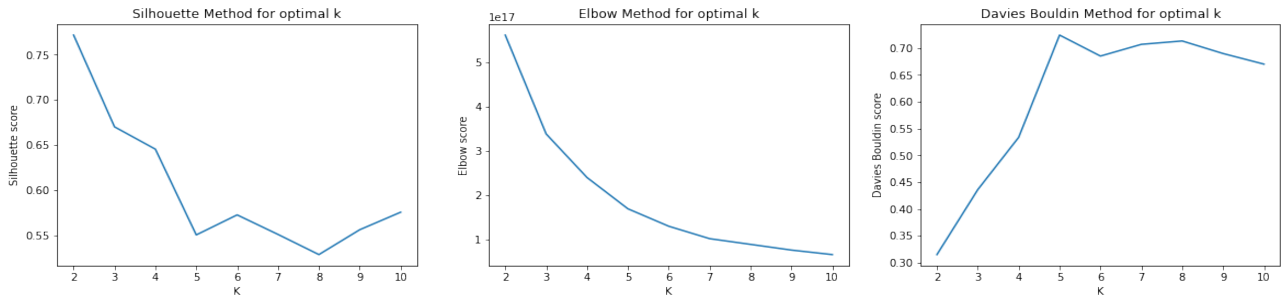


Figure 8: Before cleaning

Unfortunately, the clustering methods we applied did not lead to useful results. Moreover, due to the size and our limited hardware we could not conduct numerous experiments. Thus, we could not explore this method to its fullest potential.

5 Models

5.1 Random forest (RF)

Random forests are a type of machine learning models in which several different decision trees are trained together to provide their own predictions for the input. To obtain the final predicted class, the random forest will take an average or mean of these single predictions. This model has the advantage of avoiding over-fitting compared to normal decision tree methods.

5.2 Extreme Gradient Boosting (XGB)

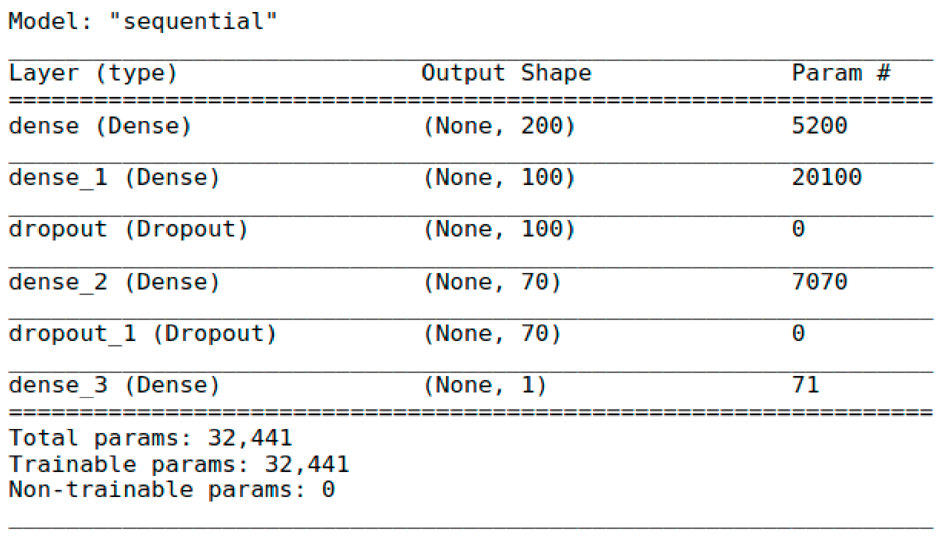
XGB[2] is an implementation of a gradient boosting technique that uses accurate approximations to find a better tree model. It uses a number of tricks that make it extremely successful, especially with structured data. One of the most important is the computation of the second-order gradient of the loss function to provide more information about the gradient direction change and to give a better approximation of the error. At the same time, it leverages advanced regularization (both L1 and L2) which prevents better overfitting.

5.3 Catboost (CatB)

Catboost can be seen as an upgraded version of the Gradient boosting. That is because it involves extra advantages in terms of performance. It is very convenient to solve problems associated with regression and classification. The highlight of this method is that it implements an ordered boosting that avoids overfitting using a permutation-driven alternative. For this reason, Catboost outperforms classical Gradient boosting. On the other hand, it deals with categorical features based on its built-in approach.

5.4 Deep Learning Model

Another model which we used for this dataset is the Deep Neural Network (DNN). For this method, we applied Automated Machine Learning (AutoML) from AutoKeras library [1]. AutoML is the process of automating the way to find the number of layers and the number of nodes for each layer by using random search (in our case) from given search space. Our search space consists of an input layer, several hidden Dense layers, then a Dropout layer and finally an output Dense layer of 1 node corresponding to the Delay column with **sigmoid** activation function which is the best for binary classification task. From this search space, AutoML will find the number of hidden layers, number of nodes for each of them as well as the dropout rate for creating the best architecture for our problem. Based on our experience, the best architecture is on Figure 9. For compiling this model, we used **binary_crossentropy** loss function and Adam Optimizer. We implemented **reduce_on_plateau** callback for decreasing the learning during the training when the current learning will not be able decrease the loss. At the end, we applied the early stopping approach so that our model directly stops when the loss will not be decreased after consecutive p epochs where $p = 10$ is the patience of the callback. This approach prevents overfitting since it does not let the score to increase as a trade-off of regularisation.



```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 200)               5200
_________________________________________________________________
dense_1 (Dense)              (None, 100)               20100
_________________________________________________________________
dropout (Dropout)            (None, 100)               0
_________________________________________________________________
dense_2 (Dense)              (None, 70)                7070
_________________________________________________________________
dropout_1 (Dropout)          (None, 70)                0
_________________________________________________________________
dense_3 (Dense)              (None, 1)                 71
=================================================================
Total params: 32,441
Trainable params: 32,441
Non-trainable params: 0
_________________________________________________________________
```

Figure 9: DL model architecture

6 Computational Results

As the data set was not balanced, to compute results we applied two different methods. The first consisted of simply dropping instances of the imbalanced class. In the second one we use the built-in ”weights” (or ”class_weights”) if sci-kit learn to give greater importance to under represented classes. In our case the delayed flights. The results are summarized in the following table 1.

Model	**Dropping**	**Weighted**
RF	$\begin{pmatrix} 0.639 & 0.404 \\ 0.361 & 0.596 \end{pmatrix}$	$\begin{pmatrix} 0.897 & 0.772 \\ 0.103 & 0.228 \end{pmatrix}$
XGB	$\begin{pmatrix} 0.565 & 0.260 \\ 0.435 & 0.740 \end{pmatrix}$	$\begin{pmatrix} 0.820 & 0.327 \\ 0.180 & 0.673 \end{pmatrix}$
CatB	$\begin{pmatrix} 0.649 & 0.322 \\ 0.351 & 0.678 \end{pmatrix}$	$\begin{pmatrix} 0.903 & 0.694 \\ 0.097 & 0.306 \end{pmatrix}$
DL	$\begin{pmatrix} 0.561 & 0.272 \\ 0.439 & 0.728 \end{pmatrix}$	$\begin{pmatrix} 0.898 & 0.707 \\ 0.102 & 0.293 \end{pmatrix}$

Table 1: Model Confusion Matrices on Test dataset

And one interesting thing we would like to mention, it is the results of the Catboost model with inverted weights, $\begin{pmatrix} 0.809 & 0.101 \\ 0.191 & 0.899 \end{pmatrix}$. We applied inverted weights to other models also, but any of them gave these results.

7 Conclusion and Future Work

In the context of this project we have chosen a flights data set. We have cleaned it, measured the most important features, and created a model that provides decent predictions for the delays of these flights. With all of these processes done, we can now answer the questions at stake stated during the introduction.

- What characteristics of a given flight are more likely to indicate that this flight will be delayed? According to section 3.2 the features that are most likely to indicate a delayed flight are the time of the departure(`'DEP_BLOCK'`), the segment of the fight the plane is going to start at departure (`'SEGMENT_NUMBER'`), and the previous airport the plane visited(`'PREVIOUS_AIRPORT'`).

- Which **month** and **airport** are responsible for the most delays. A univariate analysis tells us that, the month of June is where delays are most likely, with a percentage of 24.39%. Although more delays happened in August, precisely 238,822, this month also had many more flights, so delays are not as likely.

  Looking at the map also gives the airport with the highest likelihood of delays for any given configuration. Using the full dataset, we observe that *Chicago Midway International* has the highest percentage of delays, at 27.24%, although the total number of flights is rather low there.

- Is it possible to create a model that accurately predicts whether or not there will be a delay? According to section 5, and the table 1 the problem is far from being solved. But we believe that these consists good results for a starting base.

Even though we have managed to answer the questions we originally posed, through the analysis of this data set more interesting questions and hints were raised. For instance, we were suspicious to the fact that according to section 3.2 the weather is not such an important factor in delays. Indeed this makes sense when one sees that a according to this data set a flight is delayed if it leaves 15 min after its original departure time. This means that long, weather caused, delays will then be drowned by small management centered delays.

The amount of flights leaving airports all over the world every day is immense. This data set just focuses on a small percentage of them. Future work can be done by extending the study of this problem. For example, by slightly changing the problem to try to predict the kind of delay the flight will be subject of, and even the exact time of delay. This can be achieved by also extending the data set to other airports worldwide, and adding relevant features.

8 Code

Our code is publicly available in the following GitHub repository. The dashboard is also publicly available at the following link.

References

[1] *AutoKeras*. URL: https://autokeras.com/.

[2] *XGBboost*. URL: https://xgboost.readthedocs.io/en/stable/.