Assignment II
Data preprocessing and initial modelling

Marc Janer Ferrer
Marc Camps Garreta

April 17th
Machine Learning I

UNIVERSITAT POLITÈCNICA DE CATALUNYA
Data Science and Engineering

Contents

1 Introduction

Note: for the moment we have decided that we are not going to use the whole dataset since there are a lot of rows and computations take very long, that's why all the graphs and data provided in this report are only taken from a subset from our dataframe. More specifically, by executing the following lines of code.

```
flights = pd.read_csv('flights.csv')
flights = flights.sample(frac=0.03, random_state=42)
```

In this section of the project we are going to proceed with data cleaning and some initial modeling. In respect to our initial report, we've changed a bunch of features since we noticed that they were not accurate for what we wanted to predict. Therefore, we are going to make a review of our features before cleaning.

Table 1: Nature of predictor features

Variable	**Nature**	**Brief Description**
Arrival delay	Continuous	quantity in minutes
Month	Categorical	from 1 to 12
Day	Categorical	from 1 to 31
Day of week	Categorical	from 1 to 7
Airline	Categorical	airline tag
Origin airport	Categorical	IATA format
Destination airport	Categorical	IATA format
Scheduled Arrival	Continuous	hour format
Departure Time	Continuous	quantity
Departure Delay	Continuous	quantity
Taxi out	Continuous	quantity in minutes
Wheels off	Continuous	quantity in minutes
Scheduled Time	Continuous	quantity
Distance	Continuous	quantity

2 Data Cleaning

To start with, we'll provide the overall schedule that we will follow in order to clean our dataset:

1. **Identify and remove duplicates**: Conduct a check for duplicate rows within the dataset and remove any duplicates detected.

2. **Handle missing data**: Determine the presence of missing data and develop a plan to manage it. You may choose to either remove any rows containing missing data or impute missing values.

3. **Verify data consistency**: Conduct a thorough examination of the data to identify any erroneous, inconsistent or outliers values that do not conform to expected patterns or ranges.

4. **Address categorical data**: For any categorical data present, formulate a strategy for encoding it to ensure it can be used in the model. A common approach is to use one-hot encoding.

5. **Normalize data**: Standardize the data to ensure that features with large ranges do not dominate the model's predictions. This is achieved by scaling the features to have similar ranges.

2.1 Identify and remove duplicates

This is the easiest part of data cleaning since we can use *Python* functions that will automatically do so for us. For instance we can compute the following:

```
flights = flights.drop_duplicates()
```

In our case, we did not have duplicates so the dataframe remained the same.

2.2 One-hot encoding

Once we've treated all of our categorical data and transformed some continuous variables to categorical, we one-hot encode all of these variables using the

get_dummies() function of pandas.

2.3 Handle missing data

This stage of the process, is again simple. In order to handle the missing data the first thing that one thinks of is about NaN values. Which, is a true, however there may also be some empty values in string format or some many other possibilities that need to also be treated to avoid fooling the algorithms of training. Taking into account that we have a huge amount of rows (over $5M$), we decided to directly delete all of the rows that contained any NaN value. We pass from 5819079 to 5714008 samples. In other words, we keep the 98.2% of the samples. Next step is to check if there is some placeholder in some column that is also missing data (such as an empty space). However after exhaustively checking the data frame, we did not observe any other missing data.

2.4 Verify data consistency

After eliminating rows that contained missing data, we now proceed to treat values that may not be as NaN but maybe have been erroneously collected and have non-coherent values. To do so a good practice is to check the output of the describe method for dataframes in the form of heatmap.

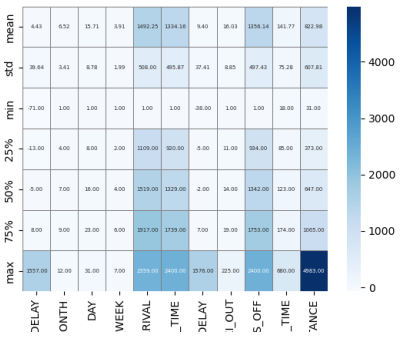


Figure 1: Heat map of the describe method

A good way of finding some values that are not correctly written is to check

minimums and maximums. For instance, in DEPARTURE TIME and WHEELS OFF have 2400 as maximum, which should be either 0000 or 2359, so we decided to encode it as 2359 in the following way:

```
flights.apply(lambda row: row.set_value("WHEELS_OFF", 2359)
   if row["WHEELS_OFF"] == 2400 else None, axis=1)
flights.apply(lambda row: row.set_value("DEPARTURE_TIME",
   2359) if row["DEPARTURE_TIME"] == 2400 else None, axis=1)
```

A part from that we where unable to find any other. We could consider the maximum distance of the flights we have as erroneous however it is not the case since the longest national US flight is Hawaiian from Honolulu to Boston, registering 5,095 miles (8,199km).

On the other hand, we could also consider the ARRIVAL DELAY containing very big values that maybe were bad recorded. The following are the 10 biggest delays:

```
    [1557.0, 1456.0, 1323.0, 1238.0, 1167.0, 1140.0, 1116.0,
        1034.0, 1019.0, 1017.0]
```

However since there are a bunch of reported flights to be delayed about that quantity we consider that they may be important to take into account for predictions, so we won't modify these rows.

A part from that we also needed to change the formatting of the AIRPORTS, both of destination and origin, since some of the rows, had airports in a weird format different than that of $IATA$ so we deleted those rows in the following way.

```
# Delete all of the rows that contain integer values
flights = flights[~flights['ORIGIN_AIRPORT'].apply(lambda x:
    isinstance(x, int))]
flights = flights[~flights['DESTINATION_AIRPORT'].apply(
   lambda x: isinstance(x, int))]
# Delete all of the rows that contain numerical values in
   the string
flights = flights[~flights['ORIGIN_AIRPORT'].apply(lambda x:
    str(x).isnumeric())]
flights = flights[~flights['DESTINATION_AIRPORT'].apply(
   lambda x: str(x).isnumeric())]
```

To end with, in order to delete some possible outliers that we are not observing at the moment we use the $Z - score$ in order to delete those, that is:

$$z = \frac{x - \mu}{\sigma} \tag{1}$$

All of the values laying out of a threshold (in our case 3) are considered outliers and the whole row is deleted.

2.5 Handle categorical data

This may be one of the most important stages of the cleaning. In this section the aim is to either modify the categories of some categorical data and to transform some continuous data to categorical.

2.5.1 Arrival delay

To start with recall that the variable arrival delay is encoded with the quantity of minutes of delay (in negative if the flight gets before the scheduled arrival). However our goal is to do binary-classification, so we will transform this column onto another column $DELAYED$ that contains a 1 if delayed and a 0 otherwise. We assign a 0 to negative values or 0, and a 1 to positive values (delayed flights).

2.5.2 Month

Instead of having 12 categories (one for each month of the year) we thought it could be interesting to divide the year in quarters (only three categories). So $\{1, 2, 3, 4\} \rightarrow 1$, $\{5, 6, 7, 8\} \rightarrow 2$ and $\{9, 10, 11, 12\} \rightarrow 3$.

2.5.3 Day

Insead of having 31 categories for every day of the month, we divide it into fortnights such that : $[1, 15] \rightarrow 1$, $[16, 31] \rightarrow 2$. In this way we reduce the number

of categories and we still have some sense of what point of the month we are in, since normally, at the end of the month there are less flights because of people having less money to spend.

2.5.4 Day of week

In this case instead of having 7 categories, we thought of just having 2 of them, indicating in-week days and weekends. That is: $[1, 5] \rightarrow 1$, $[6, 7] \rightarrow 2$. That keeps the fact that people usually travels in non-working days and simplifies the categories.

2.5.5 Airlines

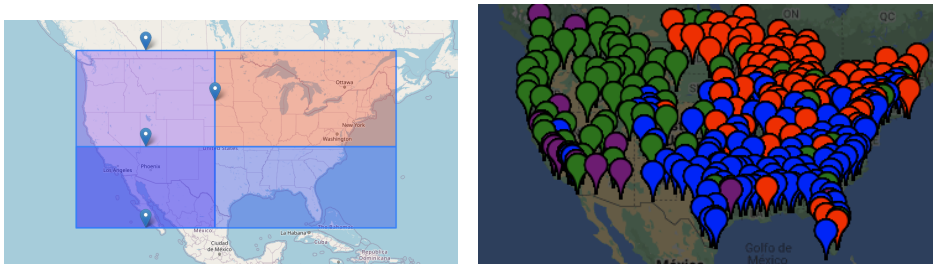
In order to treat airlines we thought that a good option was to divide the airlines into three categories. Major, low-cost and regional, in such a way that we could retain some of the airline information but without the need of having a huge amount of categories. The assignation is the following:

```
major_airlines = ['DL', 'AA', 'UA', 'US']
low_cost_airlines = ['WN', 'NK', 'F9', 'B6', 'VX']
regional_airlines = ['EV', 'OO', 'MQ', 'AS', 'HA']
```

2.5.6 Airports

This was the more elaborated and difficult step to take. There were a vast amount of categories since there are a lot of airports in the US. So we decided to approximately divide the airports in the following four groups (considering their latitude and longitude). The following image shows an approximation of our idea.

And after getting the longitude and latitude for every airport we managed to agroupate the flights in four different groups: BOTTOM_RIGHT, BOTTOM_LEFT, UPPER_LEFT and UPPER_RIGHT. It is important to add that this is done for both the origin airport and the destination airport.



(a) Folium generated image for airport segmentation

(b) Airport segmentation

Figure 2: Airport feature treatment visualization

2.5.7 Scheduled arrival & departure time

Both explanatory features in this case are in time format (*for instance: 450 means 4:50 a.m*). And to make it easier for interpretation we simplify it to whether it is during the day or during the night. Let t_i be a certain time stamp for x_i, then if $t_i \in [600.0, 1800.0) \rightarrow Daytime$ and if $t_i \in [1800.0, 600.0) \rightarrow Nighttime$.

2.6 Normalize data

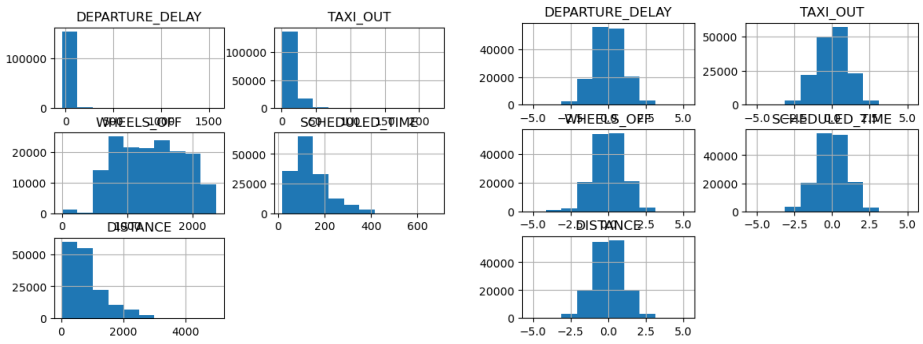
To end with the treatment of our variables now, the ones that are continuous need to be normalized because they can have different scales and units, which can cause problems in some machine learning algorithms. There are many ways in which we can normalize our data. In our case we will use the **quantile transformer**.

The **quantile transformer** works by dividing the data into quantiles, or equally sized bins, based on the rank of each data point within the feature. Then, it maps the values of each quantile to a uniform or normal distribution, based on the specified output distribution. This process ensures that the transformed data has a similar distribution across all features, which can be useful for some machine learning algorithms.

To do so we use the following piece of code:

```
# we normalize continuous variables
from sklearn.preprocessing import QuantileTransformer
# Initialize transformer with number of quantiles and output
    distribution
transformer = QuantileTransformer(n_quantiles=100,
   output_distribution='normal')
# Apply transformation to continuous columns
for col in continuous:
    flights[col] = transformer.fit_transform(flights[col].
       values.reshape(-1, 1))
```

And our continuous variables suffer the following transformation:



(a) Continuous variables before transformation

(b) Continuous variables after transformation

Figure 3: Normalization of continuous variables

3 Initial modelling

To start with, we group our data to a part of training and test. In order to do some initial modelling we are going to adjust a Logistic Regression model using the sklearn module from python. And to end with we make test how good our

model is doing by predicting the test samples. Getting an accuracy of 84%. For completeness, since the code is not that long we can add it in here:

```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y,
    test_size=0.2, random_state=42)
# Initialize the model
logreg = LogisticRegression()
# Fit the model to the training data
logreg.fit(X_train, y_train)
# Make predictions on the testing set
y_pred = logreg.predict(X_test)
# Compute the accuracy of the model's predictions
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy:.2f}")
```

It is a very straightforward modelling however it gives good predictions. We also have to take into account that we could train the model with much more data, but for the moment we'll keep working with a subset of data for reducing computational effort since we consider that the subset of data that we are using is already very representative (156940 samples).