BABEŞ-BOLYAI UNIVERSITY CLUJ-NAPOCA

FACULTY OF MATHEMATICS AND COMPUTER SCIENCE

SPECIALIZATION: COMPUTER SCIENCE IN ENGLISH

DIPLOMA THESIS

Intelligent Flight Delay Prediction

Supervisor

Prof. Dr. Onet-Marian Zsuzsanna

Graduate

Gheorghiu Ioana

2020

UNIVERSITATEA BABEŞ-BOLYAI

CLUJ-NAPOCA

FACULTATEA DE MATEMATICĂ ŞI INFORMATICĂ

SPECIALIZAREA: INFORMATICĂ ÎN LIMBA ENGLEZĂ

LUCRARE DE LICENŢĂ

Prezicerea Înârzierilor Curselor Aeriene de Linie folosind Rețele Neuronale

Conducător ştiinţific

Lect. Dr. Onet-Marian Zsuzsanna

Absolvent

Gheorghiu Ioana

2020

Abstract

Air Transportation Industry reached a historical high peak of demand during 2019, with planes overcrowding the airspace as never before seen. The whole system comprised of both human resources in airports' gates, air traffic control towers or safety agencies as well as software promises to take everyone from point A to point B in no time, smallest cost and with all the safety measures. The equation of success for any flight carrier is frequently wrecked by delays with impact on each and every passenger.

There are plenty of different factors which can influence the way a flight performs, taking all of them into consideration being a meaningful task for a Machine Learning model. The purpose of the thesis is to gather the right subset of the data needed to make a meaningful delay prediction with data only available at the time of booking the flight. The neural networks studied approach the problem as a binary as well as a multi-class classification, grouping the flights as arriving on time, delayed or cancelled. The resulting model is accessible from both a REST Service and a Chrome Extension installed in the browser as a helpful tool in deciding upon the right flights to book on a website.

The first chapter presents the theoretical concepts on which machine learning is based, making a thorough overview of the most important terms and concepts. A strong emphasis is made on artificial neural networks which make the subject of the proposed approach. The second chapter starts with a presentation of methods already addressed in the literature for the concerned problem. Next, an in depth description of the data gathering process is made, with detailed description of the scripts merging flight historical data with weather measurements such that a valid dataset is constructed for the training of the model. The last section of the chapter goes through the whole process that was performed before the actual training, from finding the right inputs, handling the class imbalance issue, and up to possible methods for hyperparamter optimization. These are followed by the actual implementations and trials of the neural network, its evaluation and a geographical visualisation of the trained embedding weights. The last chapter provides descriptions about the software applications making use of the machine learning model, that is the REST Service and the decision helper tool implemented as a Chrome Extension.

The performance of the neural network under 16 relevant inputs containing flight schedules and weather confirms the high complexity of the prediction task. The final model reached a rather modest performance as far as machine learning metrics are concerned, but can still be used as a useful decision tool for regular users who want to

be able to pick the flights with the lowest probability of delay.

This work is the result of my own activity. I have neither given nor received unauthorized assistance on this work.

Date
14th of June 2020

Gheorghiu Ioana

Contents

Chapter 1

Introduction

1.1 Air Transportation Industry

Recent years have shown an increasing number of demand and supply in the world of air transportation with tens of millions of scheduled departures carrying billions of passengers only in 2018, with a 3.5% and respectively 6.4% increase when comparing to 2017 [17]. Considering the limitations of the airspace capacity, the sky is indeed the limit, aviation systems and air traffic controllers being challenged on a daily basis to meet the customer's expectations. The air transportation industry relies on the premises of safety and speed, carriers being required to perform on time under the safest conditions in order to maximize both customer experience and incomes.

One of the most important indicators of airlines and airports is the frequency of delays or cancellations. As shown by the Bureau of Transportation Statistics (BTS) data, 20% of the scheduled commercial flights, having as an origin or destination the United States of America, were either delayed or cancelled in 2019 [24]. The leading cause of flights arriving late, as stated by the BTS, is an aircraft arriving late which is strongly connected to the given weather conditions. Other categories show air carrier delay and national aviation system delay as important players which implies that a better management of the resources could minimize certain delays.

1.2 Proposed Contribution

With delays causing significant costs for all of those consuming the air transportation systems, the need of delay predictions is high. The study of the main causes identifies some main directions: one is observing certain models of delay propagation over the course of a day, another is analyzing the impact of weather and another is finding patterns for certain airports and carriers. Given the vast amount of data and the imprecise correlation between the inputs, a deep learning approach is proposed. Its purpose is to predict flight delays well ahead of time in order to help regular potential passengers decide between possible flights. The link between classes of the same categorical inputs will also be analysed through trained weights from embedding layers.

Chapter 2

Artificial Neural Networks

2.1 Machine Learning

Machine Learning is a data analysis method that automates the learning process of a computer without it being explicitly programmed to do so. It is a sub type of **Artificial Intelligence** in the sense that it is able to mimic cognitive functions in humans and is ultimately able to analyse data and make decisions better than a human could.

The manner in which Machine Learning approaches this problem is by analysing the given data, being inherently correct to say that the growing volumes and varieties of data available with the Digital Age feeds the interest to meet new challenges with machine learning methods. Greater computational powers at lower prices are the other two main reasons why more and more solutions replace the human intervention in fields that would otherwise have little to do with Computer Science.

One of the first examples of learning systems to be brought to the public's attention was Samuel's Checkers Player which proved to be better at winning the game than the author itself. The creation of this system dates back to the 1950s [19], making the result of successfully tackling an issue of this kind with mathematical models and algorithms a remarkable step. The key element to its player's success was the experience it earned by continuously competing against itself. A more recent definition incorporates experience as one of the 3 key factors of Machine Learning:

> *"A computer is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E."* [16]

The class of tasks T mentioned in the definition can be any requirement that is to be

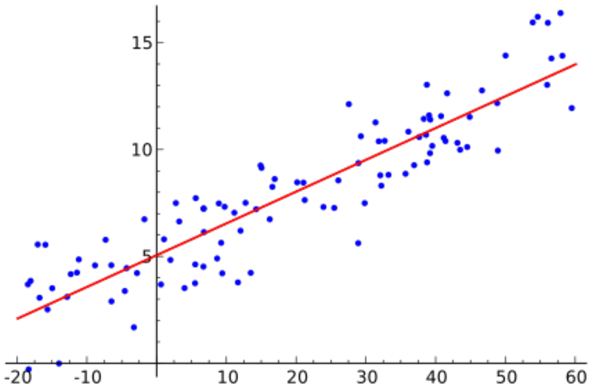


Figure 2.1: Linear Regression

assigned to the Machine Learning model. The requirements or problems that models can solve divide the algorithms typically applied into more learning styles, 3 of them being discussed in the following subsections.

2.1.1 Supervised Learning

Supervised Learning is the task in which data used in training the model contains pairs of both input and expected output, this kind of data being called labelled data. The model is fit on the given data in order to be ultimately able to correctly predict a correct output when given a set of unlabelled data. The way in which the model works is that given a set of N training examples of the form $(x_1, y_1), ..., (x_n, y_n)$, where x_i is the feature vector and y_i is it's label, it gradually tunes a function g:X->Y such that the cost computed between the outputted y and the expected one is minimum. This learning style is categorized into "regression" and "classification" problems based on the type of output desired.

Regression Problems

When the values to which the inputs are mapped are continuous, that is real values, integers or floating points, a regression algorithm is needed. Oftentimes, the skill of the model or the error given by the prediction is computed using root mean squared error $= \sqrt{(\frac{1}{n}) \sum_{i=1}^{n} (\hat{y}_i - y_i)^2}$, where $\hat{y}_1, \hat{y}_2, ..., \hat{y}_i$ are the values predicted by the model, while $y_1, y_2, ..., y_i$ are the actual values and n is the number of observations. A plotted example of linear regression, where the solutions can be described by a straight line, can be seen in Figure 2.1. In the given context, predicting the number of minutes a flight would be delayed by is a regression problem.

Classification Problems

In the case that the input has to be mapped to two or more classes or categories, a classification algorithm is used. Depending on the number of discrete values, the problems are called Binary Classification Problem or Multi-Class Classification Problem. Having problems where inputs are given multiple labels can also happen, these being called Multi-Label Classification Problems. One way of approaching the problem is having the output value as a continuous variable representing a conditional probability using a threshold classifier y(x) at 0.5.

If $y(x) \geq 0.5$, predict $y = 1$

If $y(x) < 0.5$, predict $y = 0$

There are multiple types of classification algorithms that can be found under the Machine Learning umbrella: from Naive Bayes Classifiers which assume that every pair of features being classified is independent of each other, to Decision Trees and Random Forest, and can also be tackled with Neural Networks.

2.1.2 Unsupervised Learning

Unsupervised Learning is the type of learning in which the input data does not contain any kind of expected output. Being fed a set of inputs of the form $x_1, x_2, ...x_n$, where x_i is still a feature vector, the machine is not given any feedback from the outside during its learning process. The machine is thus asked to find patterns in what initially seems to be unstructured noise [3].

The main use of this kind of learning style is Clustering, a method of dividing the input data into a number of groups, called clusters, such that points in a group are more similar to each other than to points in other clusters, the features being used to classify similarity not being preset.

2.1.3 Reinforcement Learning

Reinforcement Learning situates somewhere between the above mentioned types, being oriented to accomplishing a certain, known goal, but not being fed the right answer after each move. The goal is a rather complex one, finding the right path to attain it consisting of a succession of state and action pairs that consequently lead to maximizing the objective. The main use case of this type of learning is the ability to correlate the states and corresponding actions taken by the algorithm with the delayed

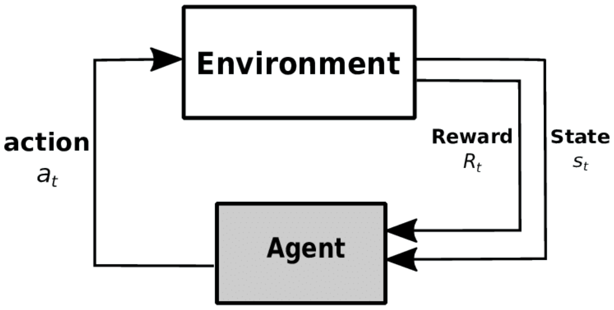


Figure 2.2: Linear Regression

results that they produce. There is a set of concepts that help in understanding the way this type of algorithms work:

- an **Agent** is the one that given a state and an environment decides to take a certain action
- an **Action** is usually chosen by the agent from a discrete set of possible actions as a response to a given **State** which in turn is a configuration of the situation in which the agent finds itself, incorporating all the elements with which it can interact
- the **Environment** consists of the world in which the agent operates and responds to the agent's moves by compiling the subsequent state and responding with certain rewards for the agent.The **Reward** is the element that guides the agent into actually learning the difference between what is good and what is bad

In application, the concept of environment can be seen as a video game or as the world itself, agents being an abstraction of either players of characters in the game, or robots, from household robots to traffic lights.

2.2 Artificial Neural Networks

When searching for a way to construct an algorithm that can process high amounts of information and address complex problems, people started to look at perhaps the most familiar and spectacular mean of intelligence, that is, the human brain. Building an algorithm that imitates the brain emerged as the Artificial Neural Network widely used today in a variety of problems. Being firstly developed in the 1980's, it only raised popularity in the last decade due to the high computational power it requires and offered lately by more and more regular computers. The network itself is not necessarily an algorithm, rather a universal instrument for multiple algorithms which combined can specialize on a certain subject or problem. An artificial neural network is based on a set of interconnected units known as artificial neurons. Each connection is meant to send processed information from one neuron to the other.

2.2.1 Architecture and Learning

Similarly to a biological neuron, the artificial neuron was thought as a computational unit that takes a number of units, applies certain computations on them to be ultimately sent forward through some output wires to other neurons.

The Perceptron

Based on this basic understanding of the neuron's operating principle, McCulloch-Pitts (MCP) proposed the very first mathematical model of an artificial neuron back in 1943 [15] and it can be described by the following rules: it has a number N of binary inputs $x_i \in \{0, 1\}$, a threshold real value θ which determines the result and a single binary output $y \in \{0, 1\}$ computed by the formula:

$$output = \begin{cases} 0 & \sum_{i=1}^{i=n} x_i \leq threshold \\ 1 & \sum_{i=1}^{i=n} x_i > threshold \end{cases}$$

The θ value is the one deciding whether the neuron fires or stays at rest. Stacking multiple neurons of this type, more complex functions can be represented, still, the MCP neuron has multiple drawbacks that lead to years and years of stagnation in the field. Some of them being that the function needs to be hard-coded by the user, there is no way to represent the XOR function and all inputs contribute in an equal manner to the output.

More than a decade after this discovery, the American psychologist Frank Rosenblatt managed to develop the so called **perceptron**, a major improvement over the MCP neuron. His achievement was a result of relaxing some of the MCP's rules by allowing any types of inputs and allowing the result to take only percentiles of inputs into consideration. This type of logic unit is illustrated in Figure 2.3. Given inputs $x_1, x_2, ..., x_n$ to the input layer, each is multiplied with the corresponding weight $w_{1j}, w_{2j}, ..., w_{nj}$. The sum of the products is then given as input to a certain activation function whose result is $f(\sum_{i=1}^{i=n} w_{ij}x_i - \theta)$ that is outputted to the output layer. The *activation function* represents a linear or non linear mapping of the input to the output and is some continuous or discontinuous function mapping real values into either of the

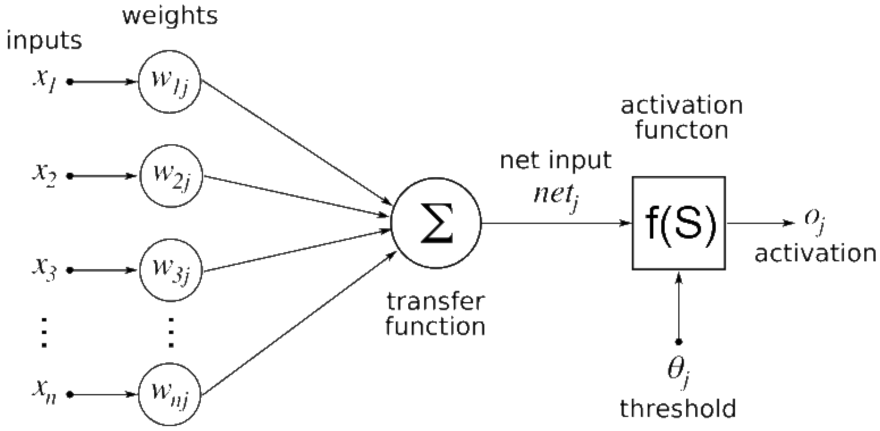


Figure 2.3: Artificial Neuron; *Source:* [26]

intervals [0, 1] or [-1, 1] [7]. θ is a threshold or bias which allows the activation function curve to be shifted up or down.

2.2.2 Activation Functions

Activation functions are a decisive part of the neural network architecture. They determine how the input is mapped to the output and the computational efficiency of the model. Activation functions also play an important role into its ability to converge, thus determining the overall accuracy that the model can be trained to acquire. These functions are attached to each and every neuron in the network and decide whether the neuron 'fires' or not. There are three types of activation functions that can be used.

Binary Step Function

The binary step function is the threshold-based activation function used in the very first artificial neuron designed by MCP. Based on the threshold, if the value is above or below that certain value, the neuron is activated and sends forward the exact same signal.

Linear or Identity Activation Function

The f(x) = x function creates an output signal that is directly proportional the output. What makes this one better than a binary step function is that it allows infinite outputs. The drawbacks are still an impediment since the derivative of the identity function is a constant, making backpropagation impossible and having multiple layers being irrelevant, all of them being equivalent to only having one. A neural network using this function can be resumed to a linear regression model.

Nonlinear Activation Functions

Nonlinear activation functions allow the network to build complex mappings between the input and the output and solve the two problems of the linear activation function.

- The **Sigmoid or Logistic Activation Function** curve is an S-shape. It only exists in the (0, 1) interval, making it especially useful for models that need to predict probabilities. The function is differentiable, making it suitable for backpropagation, but causes a vanishing gradient problem for very high or very low X values. The **Softmax Function** is a more generalized function used for multiclass classification.
- **Tanh or hyperbolic tangent Activation Function** is similar to the the Sigmoid function but maps values to the (-1, 1) range, making the negative inputs to be mapped strictly to negative values.
- **Rectified Linear Unit (ReLU) Activation Function** currently holds the most popularity. It is half-rectified, making it look like a linear function but still having a derivative. One major issue about ReLU is that given negative or values approaching zero, this activation function turns these values into zero, backpropagation not being able to help the model learn
- **Leaky ReLU** tries to handle the Dying ReLU problem by creating a small positive slope for negative values, thus enlarging the range to (- inf, inf). Still, negative values might not be correctly predicted.
- **Parametric ReLU** attempts again to solve the Dying ReLU problem by providing the slope of the negative part as an argument.
- **Swish** is a self-gated activation function developed by researchers at Google that based on experiments shows an improved performance over the ReLU function. It is simply defined by $f(x) = x * sigmoid(x)$

2.2.3 Input data and Embedding Layers

The use of Neural Networks as universal approximators for problems involving wide areas of interest brought into focus the need of representing various types of inputs in a computer friendly and meaningful way. While having to deal with numerical data might be a straight forward task, informing a neural network that some words are nice while others are mean requires a bit of extra data customization. Data that can be fed to a neural network can be found in all shapes and sizes, but is generally split into 2 main categories: continuous and categorical.

Continuous Inputs

Those inputs that can take any value within a finite or infinite range are said to be continuous. Some examples for the current problem are scheduled elapsed time of a flight, or weather measurements like dry bulb temperature, precipitations, visibility, wind speed. Since even numerical values can differ vastly from feature to feature, preprocessing input data helps the neural network converge faster and reduces the chances of it getting down to a local minimum. The most popular transformations applied to continuous inputs are Normalization and Standardization. Normalization usually refers to rescaling values to fit a certain range (typically [0,1] or [-1,1]), while Standardization implies mean removal and variance scaling.

Categorical Inputs

Inputs that can take a fixed, limited and usually known number of values are known as categorical. An easy way to view these types of data is to think of its possible values as classes of a certain domain. Examples of categorical features describing flight schedules are airports, aircrafts, day of the week or month, code of the flight carrier. Literature brings a number of techniques handling this sort of data.

The simplest way to approach categorical values is by encoding the values into numerical ones, that is mapping each possible class in the finite set to a numerical value. This brings a number of issues, one of the most important being that it artificially creates a certain misleading bond between those classes that are mapped to close numerical values.

A technique that solves the problem brought by regular encoding is **One-Hot-Encoding**. This method maps each possible class to a new column that is a binary indicator of the class, taking the value 1 for the current class and 0 for the rest. Deciding to apply this method for a feature like carrier code having for example 10 possible classes, creates transforms the $carrier - code$ column into 10 new columns: carrier-code-A, carrier-code-B, carrier-code-C, etc. The binary values further activate certain neurons in the neural network, thus achieving the task of correctly feeding any kind of categorical data, with the cost of both high number of columns and computations.

Embedding layers are currently the most efficient approach to handling categorical data, solving the issues of sparsity and computational heaviness met in One-Hot-Encoding. The way it does this is by implementing a distributed representation of the weights that would be classically used in matrix multiplications between the One-Hot-Encoded vectors and a matrix of weights. Embedding Layers are represented by a single matrix having one line for each possible unique class of the feature and a variable number of columns. This technique enhances high dimensionality of each class of the feature by mapping the categorical values to vectors of floats of customizable sizes instead of simple labeling values. This way, each column of the embedding can be though of as a certain characteristic. Given an embedding layer of airports across the United States and 4 columns, each row represents the concept of an airport in a 4 dimensional space, thus holding a richer semantic meaning. As an example, the first column could be seen as the airport's overall performance under bad weather during winter or maybe crowdedness during the warm season, while another column could be a link to how prone the geographical location of the airport is to delays caused by low visibility or winds. The fundamental idea behind the high dimensionality of both the embedding layers as well as neural networks is that these numbers are not required to have one exact meaning that brings value to our understanding of the concept, but rather an abstract meaning that influences the numbers behind the final predictions of the model.

Chapter 3

Prediction of Delays in Air Transportation

3.1 Literature Examples

The subject of flight delay prediction is highly addressed in literature and approached both as a regression as well as a classification problem. The main focus on the problem is on the way in which delays propagate through the network, since an improved schedule could also benefit the marketing and budget of flight carriers.

Popular machine learning methods proposed for this problem are Random Forest and Decision Trees analysing data about flights in combination with weather data. [2] presents an approach using different thresholds for delay: 15 and 60 minutes after departure, which achieves 74.2% accuracy and 71.8% recall, respectively 85.8% accuracy and 86.9% recall. [8] goes further with the research on the impact that weather, specifically sea level pressure, has on flight delays and the correlations between them and compares different predictive models like SVM Classifer, Gradient Boosting Classifer, Random Forest Classifer and AdaBoost. A notable contribution that uses 'a stateof-the-art boosting tree based regressor: LightGBM' accomplishes good results and introduces Airport GPS Trajectories Data, a private dataset holding ground GPS tracks on LAX airport [21]. Moving towards machine learning model implementations using Neural Networks, the problem of delay propagation is usually approached using Recurrent Neural Networks. In [12] a Long-Short-Term-Memory RNN was used to model day-to-day sequences of departure and arrival delays for an individual airport. Another sequencing approach was made by [23] using a Genetic Algorithm Version of an algorithm that also takes into consideration seasonality of flying trends.

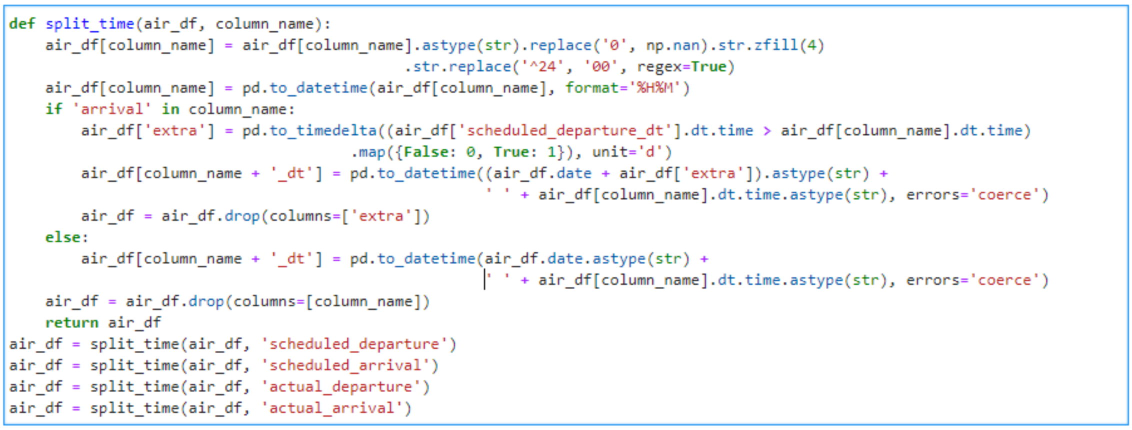
What most of the existing approaches have in common is low geographic coverage and constraints on possible flying routes. Predictions that are limited to up to a couple hours before departure or arrival of the flight are also quite approached. Deep Learning solutions are quite unpopular and do not make the subject of many important publications.

3.2 Data Gathering

An important part of preparing the data sets for the machine learning model was gathering relevant and current data holding enough features such that the model has the chances to learn what makes a flight arrive too late.

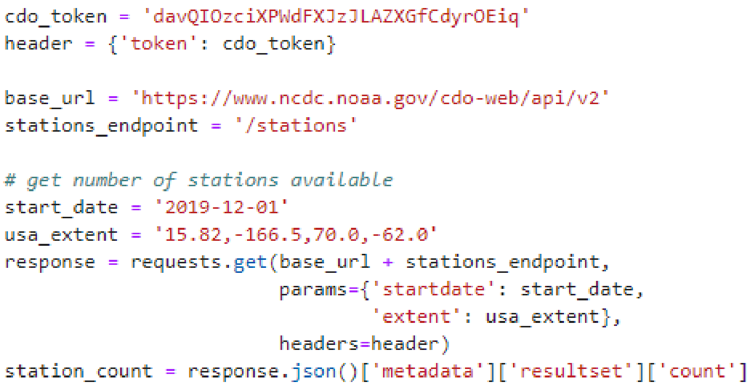
The Bureau of Transportation Statistics administrated by the United States of America under the Department of Transportation holds enormous amounts of data updated on a monthly basis under the title 'Airline Service Quality Performance 234' and is available since 1987. A monthly file contains data about carriers within 1% or more of the total domestic scheduled service passenger revenues. Data is available under .asc file having one line for each flight and more than 70 columns holding data about the flight, scheduled times, airports, number of minutes of delays due to a number of causes like security delays, late aircraft arrival delay etc. One big issue of using this data was that it holds no header, the contents being really difficult to understand and match with what their possible titles could be. Out of all the columns, only the known and most relevant for the task are kept. The date column is afterwards used in order to transform the columns holding time data regarding the scheduled and actual departure and arrival times into datetime type. This way it will be easier to merge the correct weather data. The code implementing this can be seen in Figure 3.1. Finally, the date column is split into 3 columns: month, day and weekday. The column indicating that the flight was cancelled shows either a cancellation code or an empty space for the cases in which the flight was not cancelled.

Weather data joined to the flight data was also retrieved from sources administrated by the United States of America, from the National Oceanic And Atmospheric Administration. The dataset used, called Local Climatological Data, provides a synopsis of climatic values for a single weather station over a specific month among stations across the country. Firstly data about the stations is downloaded via an API provided by the Climate Data Online Web Services and administrated by the National Climatic Data Center. The API available at URL 'https://www.ncdc.noaa.gov/cdo-web/api/v2' ex-



```
def split_time(air_df, column_name):
    air_df[column_name] = air_df[column_name].astype(str).replace('0', np.nan).str.zfill(4)
                                            .str.replace('^24', '00', regex=True)
    air_df[column_name] = pd.to_datetime(air_df[column_name], format='%H%M')
    if 'arrival' in column_name:
        air_df['extra'] = pd.to_timedelta((air_df['scheduled_departure_dt'].dt.time > air_df[column_name].dt.time)
                                      .map({False: 0, True: 1}), unit='d')
        air_df[column_name + '_dt'] = pd.to_datetime((air_df.date + air_df['extra']).astype(str) +
                                                      ' ' + air_df[column_name].dt.time.astype(str), errors='coerce')
        air_df = air_df.drop(columns=['extra'])
    else:
        air_df[column_name + '_dt'] = pd.to_datetime(air_df.date.astype(str) +
                                                      ' ' + air_df[column_name].dt.time.astype(str), errors='coerce')
    air_df = air_df.drop(columns=[column_name])
    return air_df
air_df = split_time(air_df, 'scheduled_departure')
air_df = split_time(air_df, 'scheduled_arrival')
air_df = split_time(air_df, 'actual_departure')
air_df = split_time(air_df, 'actual_arrival')
```

Figure 3.1: Transforming time columns into datetime

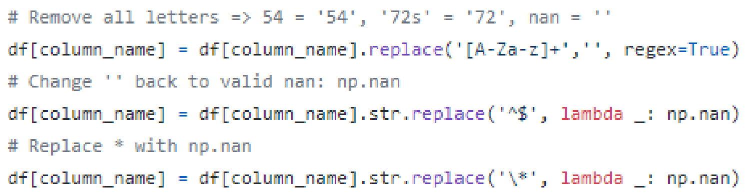


```
cdo_token = 'davQIOzciXPWdFXJzJLAZXGfCdyrOEiq'
header = {'token': cdo_token}

base_url = 'https://www.ncdc.noaa.gov/cdo-web/api/v2'
stations_endpoint = '/stations'

# get number of stations available
start_date = '2019-12-01'
usa_extent = '15.82,-166.5,70.0,-62.0'
response = requests.get(base_url + stations_endpoint,
                        params={'startdate': start_date,
                                'extent': usa_extent},
                        headers=header)
station_count = response.json()['metadata']['resultset']['count']
```

Figure 3.2: Example of call to Climate Data Online Web Services

poses an endpoint dedicated to filtering stations. Data downloaded contains general features like id, name, geographical location, elevation about all the filtered stations. An example of request getting the number of total stations can be seen in Figure 3.2. These are selected using a rectangular range on the map given by the geographical coordinates ’15.82,-166.5,70.0,-62.0’ and having weather data starting from a given date. The data is stored in a Comma Separated File (csv) file for further access.

There are more than 13,000 stations transmitting data and while it is not uncommon that each airport has its own weather station, weather station names do not match with the three letter code of the airport that is found in the flight dataset. For this reason, more data about airports is required. BeautifulSoup package from python was used to scrape all airport codes on the page of Bureau of Transportation Statistics and match the codes with more consistent data available for download on the open source website https://ourairports.com/. The most important feature in this dataset is the geographical location since the next script matches airports with weather stations by trying to find the closest station in terms of geographic distance. Given 2 pairs of

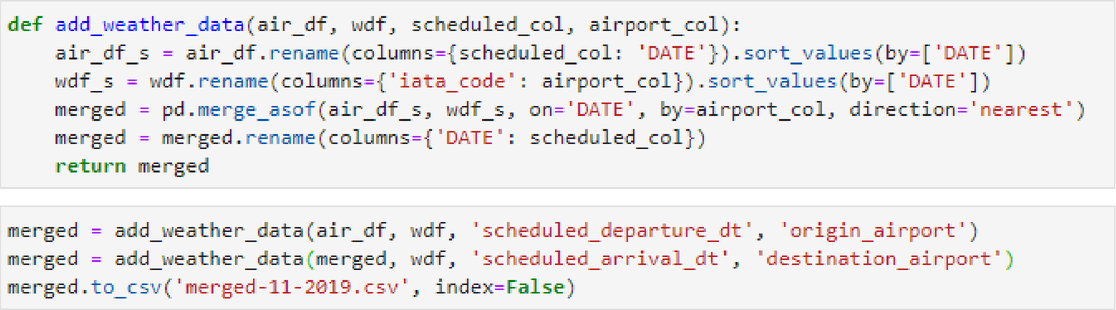


```
# Remove all letters => 54 = '54', '72s' = '72', nan = ''
df[column_name] = df[column_name].replace('[A-Za-z]+','', regex=True)
# Change '' back to valid nan: np.nan
df[column_name] = df[column_name].str.replace('^$', lambda _: np.nan)
# Replace * with np.nan
df[column_name] = df[column_name].str.replace('\*', lambda _: np.nan)
```

Figure 3.3: Sample of code cleaning weather recordings

latitude and longitude, the distance is computed using the **geodesic** distance which is the shortest distance on the surface of an ellipsoidal model of the earth [11]. The distance is computed using the python package geopy. Ultimately, a csv file with data about airports (IATA code, name, geographical location) and their closest weather station (code, name, etc.) is created. Finally, the actual weather data registered only by the stations matched with airports can be downloaded. Firstly, the csv files for all stations that can be seen on a page of the National Oceanic And Atmospheric Administration Website do not have the exact same filename as the ID of the station that registered the data in the file, but a construction of a prefix followed by the station ID. so in order to match the station IDs with the correct filenames, the page is scraped and all the links in the hyperlink tags <a> are retrieved and mapped to their ID based on suffix similarity. The filenames that were mapped to station IDs matching airports in the dataset are then parsed and each file is downloaded and cleaned such that only temperature, precipitation, wind speed and visibility features are kept. Some of these features also contain noisy, sometimes incomplete data, a value followed by an 's' is for example a 'suspect value', according to the Local Climatological Data Documentation, while 'T' represents 'trace precipitation amount or snow depth, an amount too small to measure'. These recordings are cleaned using code like the one in Figure 3.3. All these values are transformed to numerical so that the data is cleaned before it enters the Neural Network, while missing values are completed with average values from neighbouring data.

Having the weather data downloaded and marked by the three-letter IATA code of the airport brings the two datasets only a couple of lines of code away as can be seen in Figure 3.4. This dataset is finally ready to meet the Neural Network. An overview of the elements involved in the process of gathering all the data can be see in Figure 3.5.



```
def add_weather_data(air_df, wdf, scheduled_col, airport_col):
    air_df_s = air_df.rename(columns={scheduled_col: 'DATE'}).sort_values(by=['DATE'])
    wdf_s = wdf.rename(columns={'iata_code': airport_col}).sort_values(by=['DATE'])
    merged = pd.merge_asof(air_df_s, wdf_s, on='DATE', by=airport_col, direction='nearest')
    merged = merged.rename(columns={'DATE': scheduled_col})
    return merged
```

```
merged = add_weather_data(air_df, wdf, 'scheduled_departure_dt', 'origin_airport')
merged = add_weather_data(merged, wdf, 'scheduled_arrival_dt', 'destination_airport')
merged.to_csv('merged-11-2019.csv', index=False)
```

Figure 3.4: Merging flight data with weather data

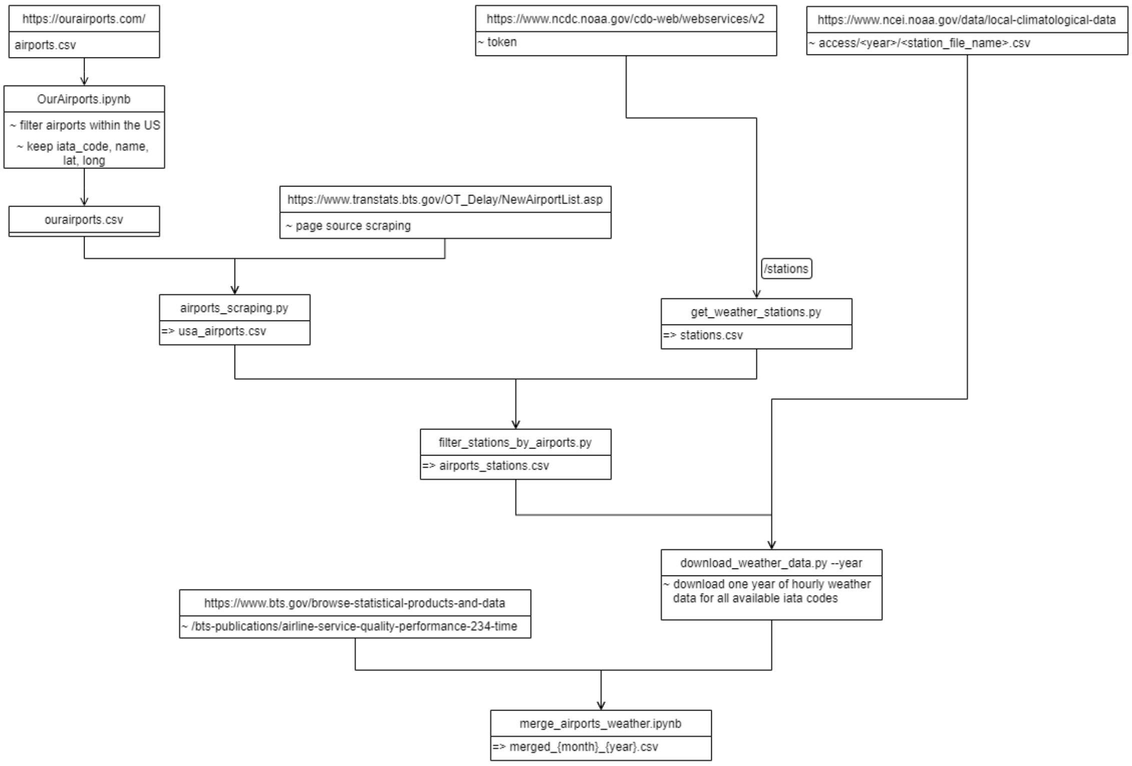

https://ourairports.com/
airports.csv
OurAirports.ipynb
~ filter airports within the US
~ keep iata_code, name, lat, long
ourairports.csv
https://www.ncdc.noaa.gov/cdo-web/webservices/v2
~ token
https://www.ncei.noaa.gov/data/local-climatological-data
~ access/<year>/<station_file_name>.csv
https://www.transtats.bts.gov/OT_Delay/NewAirportList.asp
~ page source scraping
/stations
airports_scraping.py
=> usa_airports.csv
get_weather_stations.py
=> stations.csv
filter_stations_by_airports.py
=> airports_stations.csv
download_weather_data.py --year
~ download one year of hourly weather data for all available iata codes
https://www.bts.gov/browse-statistical-products-and-data
~ /bts-publications/airline-service-quality-performance-234-time
merge_airports_weather.ipynb
=> merged_{month}_{year}.csv

Figure 3.5: Dependencies among scripts used for data gathering

3.3 Proposed Model

What follows is a study over various Neural Network architectures and methods that were tried in order to solve the task of flight delay prediction. The problem can be approached in two different ways considering the output of the model: regression problem or classification. The proposed models implement binary or multi-class classifications. For the binary classification problem a division between on-time and delayed or cancelled flights is applied, while the multi-class classification problem takes into consideration all three classes: on-time, delayed by more than 15 minutes and cancelled.

3.3.1 Data Used

The data set used for this prediction is described by 16 features and is labelled by the delay in minutes at arrival. Examining the data available for the month of December 2019 shows a total of 677687 flights highly imbalanced on the labels: 535006 (79%) are on time, 135537 (0.2%) delayed by 15 minute or more and only 7144 (0.01%) cancelled. The task for the models is to predict the 2 unrepresented classes

The inputs fed to the Neural Network for the flight prediction task can be split into two main classes from a practical point of view: categorical and continuous.

Categorical Features

The categorical data set is comprised of 6 features:

- carrier code: a two-letter code describing the flight carrier. The data set used contains codes representing 10 flight carriers operating in the United States of America
- day: day of the month, integer numbers from 1 to 31
- weekday: day of the week, starting with 0 - Monday to 6 - Sunday
- month: integer numbers from 0 to 11
- origin and destination airport: three-letter 'IATA' code of airports in the United States of America assigned by the International Air Transport Association

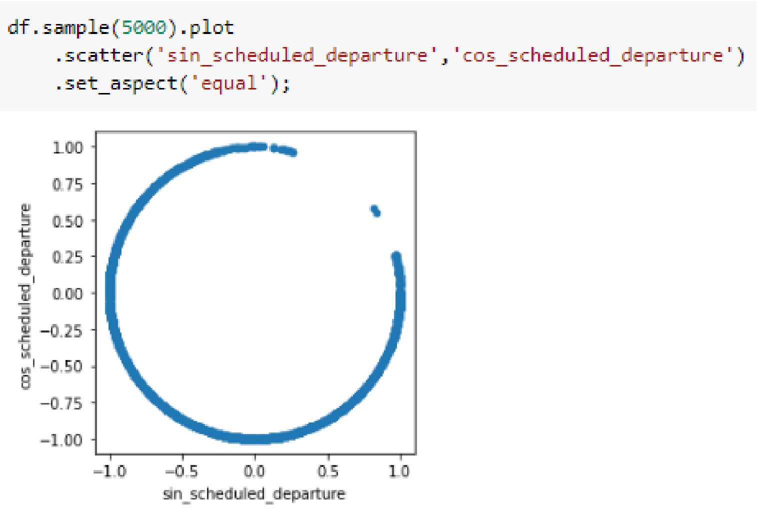
Continuous Features

The continuous features are mostly related to weather measurements, but also contain data regarding time.

- scheduled elapsed time: the flight duration in minutes
- scheduled departure and arrival: the time of departure expressed in minutes from midnight
- dry bulb temperature at airport X at departure time and dry bulb temperature at airport Y at arrival time
- precipitation amounts in inches at airport X at departure time and at airport Y at arrival time
- visibility as horizontal distance an object can be seen and identified given in whole miles at airport X at departure time and at airport Y at arrival time
- wind speed in miles per hour miles per hour at airport X at departure time and at airport Y at arrival time

Values from the continuous subset of the dataset have various means and distributions which can be hard for the neural network to comprehend. In order to bring all the different features to a common scale while also persisting their meaning, various scaling and encoding techniques can be applied.

Time data, for example, is inherently cyclical and the model should also be aware of that. Presenting the scheduled time of departure or arrival, i.e. the time in hours and minutes, as the number of minutes past midnight cancels the cyclical form of the feature. A flight scheduled to depart at 00:05 AM does indeed face similar conditions to a flight departing at 00:15 AM and this can be also understood in the number of minutes past midnight (5 vs 15), but a flight departing at 11:57 PM is at a distance of more than 1420 minutes from these two flights, even though the overall conditions must be quite similar since the real time difference is less than 30 minutes. In order to correctly represent this form of cyclical data, the feature can be split in 2 dimensions, deriving a sine as well as a cosine transformation from it using the formula below for each sample i in the dataset, where mpm = minutes past midnight, 1440 = minutes per day.



```
df.sample(5000).plot
    .scatter('sin_scheduled_departure','cos_scheduled_departure')
    .set_aspect('equal');
```

Figure 3.6: Plotting cyclical data

$$\begin{cases} f_{sine,i} = sin(2 * \pi * mpm_i/1440) \\ f_{cosine,i} = cos(2 * \pi * mpm_i/1440) \end{cases}$$

Plotting the newly created sine and cosine features for 5000 random samples of the dataset scatters the scheduled departure times of the selected flights on a trigonometric clock representing a 24 hour time frame. It can be easily observed in Figure 3.6 that flights from midnight up to around 6 in the morning are much fewer. The same algorithm should not be applied for the scheduled elapsed time of a flight since it imposes no cyclicity.

While scheduled time data now lies between [0, 1] thanks to the sine and cosine functions, all the other continuous features are still varying firstly in units (since they are weather data), and consequently in magnitudes and ranges. Feature scaling, defined as "a method used to standardize the range of independent variables or features of data" [27], can be easily applied such that all features can be treated equally. The scaler used in the case of these models was a min-max scaler which translates each individual feature such that it fits the predefined range from 0 to 1.

The input nodes of the model are separated into two branches. The first branch deals with the categorical data and maps each input to an Embedding Layer. Each Embedding Layer has a depth defined by the formula $min(floor(x/2), 50)$ [10], where x is he number of unique classes in a feature. The outputs of all the embedding layers are afterwards concatenated and sent forward to Dense layers. A graphical view of the branch dealing with categorical inputs can be seen in Figure 3.7. The second branch

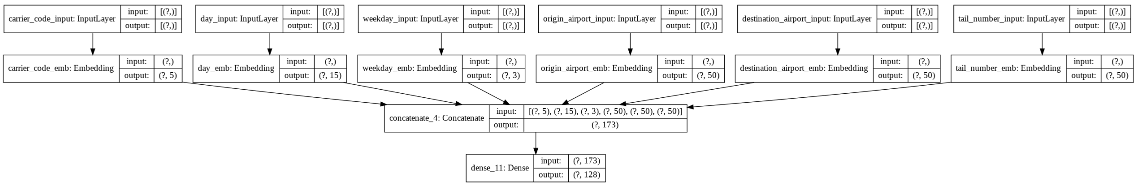

carrier_code_input: InputLayer | input: [(?,)] | output: [(?,)]
day_input: InputLayer | input: [(?,)] | output: [(?,)]
weekday_input: InputLayer | input: [(?,)] | output: [(?,)]
origin_airport_input: InputLayer | input: [(?,)] | output: [(?,)]
destination_airport_input: InputLayer | input: [(?,)] | output: [(?,)]
tail_number_input: InputLayer | input: [(?,)] | output: [(?,)]
carrier_code_emb: Embedding | input: (?,) | output: (?, 5)
day_emb: Embedding | input: (?,) | output: (?, 15)
weekday_emb: Embedding | input: (?,) | output: (?, 3)
origin_airport_emb: Embedding | input: (?,) | output: (?, 50)
destination_airport_emb: Embedding | input: (?,) | output: (?, 50)
tail_number_emb: Embedding | input: (?,) | output: (?, 50)
concatenate_4: Concatenate | input: [(?, 5), (?, 15), (?, 3), (?, 50), (?, 50), (?, 50)] | output: (?, 173)
dense_11: Dense | input: (?, 173) | output: (?, 128)

Figure 3.7: Categorical Inputs Branch

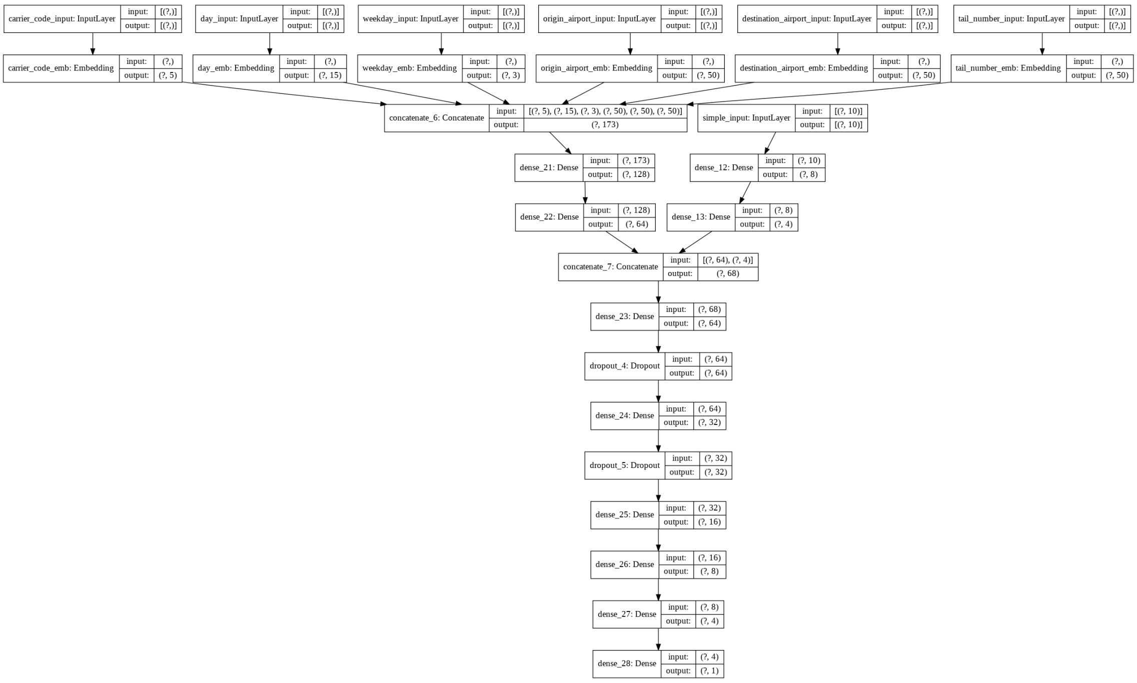

carrier_code_input: InputLayer | input: [(?,)] | output: [(?,)]
day_input: InputLayer | input: [(?,)] | output: [(?,)]
weekday_input: InputLayer | input: [(?,)] | output: [(?,)]
origin_airport_input: InputLayer | input: [(?,)] | output: [(?,)]
destination_airport_input: InputLayer | input: [(?,)] | output: [(?,)]
tail_number_input: InputLayer | input: [(?,)] | output: [(?,)]
carrier_code_emb: Embedding | input: (?,) | output: (?, 5)
day_emb: Embedding | input: (?,) | output: (?, 15)
weekday_emb: Embedding | input: (?,) | output: (?, 3)
origin_airport_emb: Embedding | input: (?,) | output: (?, 50)
destination_airport_emb: Embedding | input: (?,) | output: (?, 50)
tail_number_emb: Embedding | input: (?,) | output: (?, 50)
concatenate_6: Concatenate | input: [(?, 5), (?, 15), (?, 3), (?, 50), (?, 50), (?, 50)] | output: (?, 173)
simple_input: InputLayer | input: [(?, 10)] | output: [(?, 10)]
dense_21: Dense | input: (?, 173) | output: (?, 128)
dense_12: Dense | input: (?, 10) | output: (?, 8)
dense_22: Dense | input: (?, 128) | output: (?, 64)
dense_13: Dense | input: (?, 8) | output: (?, 4)
concatenate_7: Concatenate | input: [(?, 64), (?, 4)] | output: (?, 68)
dense_23: Dense | input: (?, 68) | output: (?, 64)
dropout_4: Dropout | input: (?, 64) | output: (?, 64)
dense_24: Dense | input: (?, 64) | output: (?, 32)
dropout_5: Dropout | input: (?, 32) | output: (?, 32)
dense_25: Dense | input: (?, 32) | output: (?, 16)
dense_26: Dense | input: (?, 16) | output: (?, 8)
dense_27: Dense | input: (?, 8) | output: (?, 4)
dense_28: Dense | input: (?, 4) | output: (?, 1)

Figure 3.8: Full Model

exposes a single entry point accepting inputs on multiple neurons for the continuous features.

The two branches are concatenated and followed by multiple Dense layers using ReLU activation function and one Dropout layer used for minimization of overfitting. Making a binary prediction, i.e. splitting the data between two classes: delayed and on-time flights, is achieved by closing the Neural Network with a Dense layer that is using the sigmoid activation function and having a single neuron returning a probability for the flight to be delayed. Another approach is to change the activation function of the last Layer to softmax, thus the output will show a probability of the flight to be classified into either of the three classes: cancelled, delayed or on-time. A full view of the binary classifier Neural Network having both input branches can be seen in Figure 3.8.

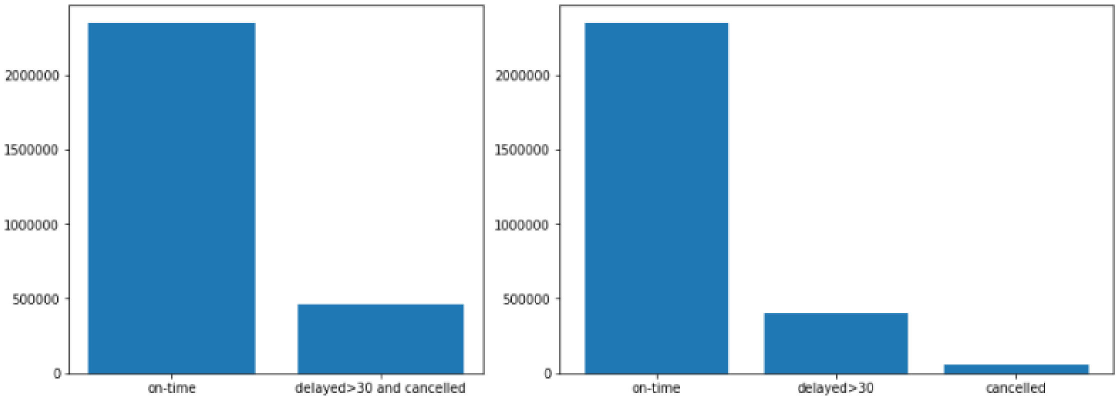


Figure 3.9: Class Imbalance

3.3.2 Handling imbalanced dataset

One of the great impediments imposed by the nature of the problem was the highly imbalanced dateset in which the number of examples in one class greatly outnumbers the number of samples in the other 2 classes, as showed in Figure 3.9 which contains flights from 4 months during the warm season. In order to tackle this issue, there are a number of techniques that can been applied.

The first approach addresses the way in which the training is performed, how the performance is measured after each feed forward and as a result how all the weights are tweaked such that the least represented classes have a higher importance. After a set of data is fed into the neural network, the log loss is computed by measuring the difference between the predicted value and the expected one. Mentioning a dictionary mapping of class weights at training time, makes the algorithm apply the given weights to the samples of each class, such that bad predictions in an important but under represented class have an impact on the overall loss that is higher than the one they would normally have. This behaviour can be easily achieved in Keras giving as parameter to the fit method the class weights. According to the documentation, this method can be used to tell the model to pay more attention to least represented classes [6]. These are computed by the formula below, given that total is the number of all the samples, and counter is the number of samples being labeled with the given label.

$$class\_weight_{label} = \begin{cases} log(mu * total/counter_{label}) & log(mu * total/counter_{label}) > 1 \\ 1 & log(mu * total/counter_{label}) \leq 1 \end{cases}$$

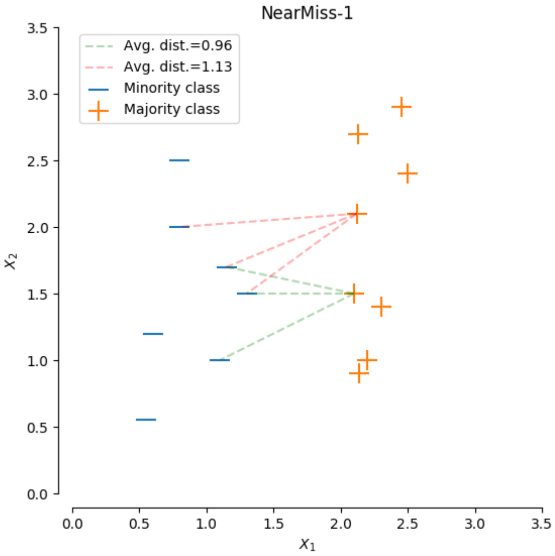


Figure 3.10: Version 1 of the Near Miss Method

Under/Over Sampling

A simple approach into balancing the dataset before it reaches the model is to randomly remove examples of the over represented class, called undersampling, or to duplicate random samples of the under represented class, that is oversampling. The grounding idea is that the model should be as skilled in determining any of the classes, independent of the number of samples labelling it. Unfortunately, a random approach into doing this can lead to unwanted loss of important data. More powerful approaches also take into consideration what samples are more or less representative such that it would make sense to remove or use them for new, artificial samples.

An possible under-sampling technique is called Near Miss which introduces some heuristic rules in order to select samples and has 3 versions. The first one selects the positive samples for which the average distance to the N closest samples of the negative class is the smallest. The second one selects the positive samples for which the average distance to the N farthest samples of the negative class is the smallest. The third one is a 2-steps algorithm. First, for each negative sample, their M nearest-neighbors will be kept. Then, the positive samples selected are the ones for which the average distance to the N nearest-neighbors is the largest [14]. A graphical representation of the algorithm used in the first version can be seen in Figure 3.10. This method showed really good results in training data but lacked precision on validation data.

A popular oversampling method is called Synthetic Minority Over-Sampling Technique (SMOTE) [5]. Given a dataset having s samples with f features, synthetic data

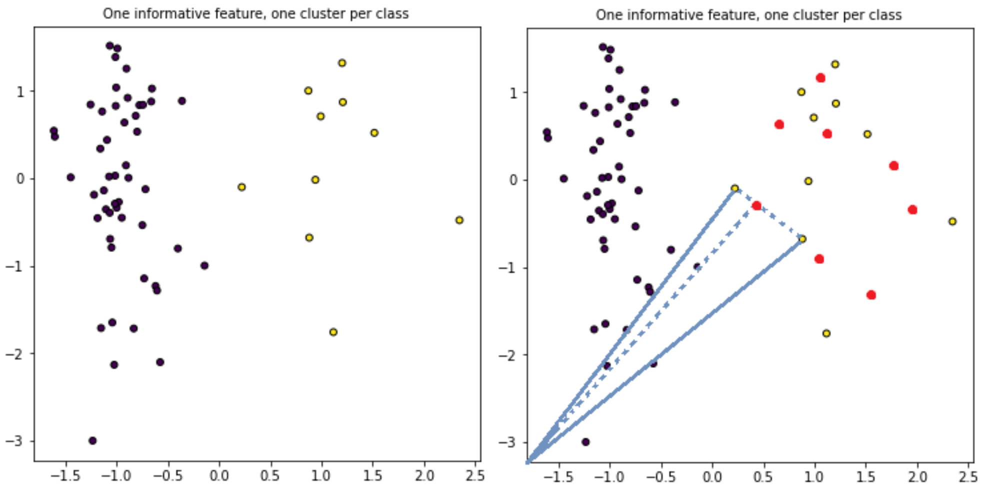


Figure 3.11: SMOTE

points are created in order to oversample the minority class. Firstly, one sample of the minority class is taken and its k nearest neighbours are computed. Secondly, for each of these neighbours, the difference between the vector of the sample and the vector of the neighbour is multiplied by a number between 0 and 1, the resulting point is the synthetic data point. A visual representation can be seen in Figure 3.11, where the red point is a synthetic sample of the minority class. In order to better illustrate the concept, the minority class can be thought of as delayed or cancelled flights and for simplicity, only two features will be considered: departure time and visibility, i.e. the horizontal distance at which an object can be seen. The first sample can be a flight departing at 8:00 AM having a low visibility of 4 miles out of 10. A near neighbour is a flight departing at 10:00 AM with a visibility of 7 miles, much better than the first one, but the flight is still delayed, which could be due to a number of causes, one example can be a late flight arrival due to earlier delays caused by low visibility. There is no data about any flight departing at 9:00 AM, but chances are that it will still be delayed, and the visibility should be something in between the two, so a synthetic data point could be a delyed flight departing at 9:00 AM on a visibility of 5 miles.

An under-sampling technique that also takes into consideration neighbourhoods of the data points is called Tomek links [22]. The links refer to pairs of samples from different classes that are their own nearest neighbours. A more mathematical definition of the pair is (x_i, x_j), where $x_i \in S_{min}$, $x_j \in S_{max}$ and $d(x_i, x_j)$ is the distance between x_i and x_j, then (x_i, x_j) is a Tomek link if there is no x_k such that $d(x_i, x_k) < d(x_i, x_j)$ or $d(x_j, x_k) < d(x_i, x_j)$ [28] since it can be concluded that the samples overlap and are noisy or right at the border between the clusters. The underlying idea is to refine the boundary between the classes by either removing both of the samples or removing only the one in the majority class.

A powerful approach for the problem of balancing highly imabalanced datasets is a combination between the two techniques, firstly combined in a paper from 2003 [1] and usually used in literature alongside the combination between SMOTE and Edited Nearest Neighbours. More precisely, SMOTE is firstly applied in order to over-sample the minority class and balance the dataset, then Tomek links are identified and samples of both the minority and the majority class are removed which does not impose a real problem since so many synthetic minority samples were already created. The combination resulted in a reduction of false negative predictions at the cost of more false positives which is acceptable for the flight delay prediction problem.

These techniques can be easily applied with the implementations provided by the imblearn package in python [14]. It comes out with out of the box implementations of combined over- and under-sampling techniques (SMOTETomek, SMOTEENN) as well as the possibility to manually combine any kind of techniques.

3.3.3 Hyperparameter Optimization

Hyperparameters are those parameters contributing to the architecture and learning process of the neural network that are declared previously to training and cannot be learned, as opposed to node weights which are tuned throughout fitting. The goal of the hyperparameter optimization is to find the ideal set of parameters that minimize the loss function of the neural network after training it on a given number of epochs for a given number of trials. The algorithm has to search for all the parameters in a predefined search space for each of the adjustable parameter. This search is a meta procedure on top of the neural network itself, making the task even more resource consuming than a typical training, but can benefit a lot to the final performance.

There are multiple methods to search the space for the optimal hyperparameters and the simplest and most straightforward one is to randomly combine values from the given search space. This is simply called a Random Search and can be applied without the hopes of rapidly finding an optima. Random Search needs to implement an exhaustive search in order to be sure that it found a good result. A more efficient technique that is able to find the best combination of hyperparameters is called Bayesian Optimization which uses a separate model that has the validation loss computed as a function of the hyperparameters to be tuned. Fitting it to a number of trials can make it tell where the global optima is.

3.3.4 Implementation

The input data is available in csv files containing data about flights for a month of a year. In order to load and efficiently manipulate this type of files, a data analysis library named pandas was used [9],[25]. Pandas is a tool build on top of the Python programming language and specifically on top of numpy library, a python package for scientific computing. The data structure that represents the actual data is called a dataframe and is a two-dimensional tabular data structure. This data structure allows fast manipulation and ways to easily overview the data concerned. Using pandas, only the columns of the dataframe relevant for the neural network are kept, and rows that have missing data are removed. Based on the cancelled code column and the arrival delay, a new 'output' column is created that represents the labels on which the training is made.

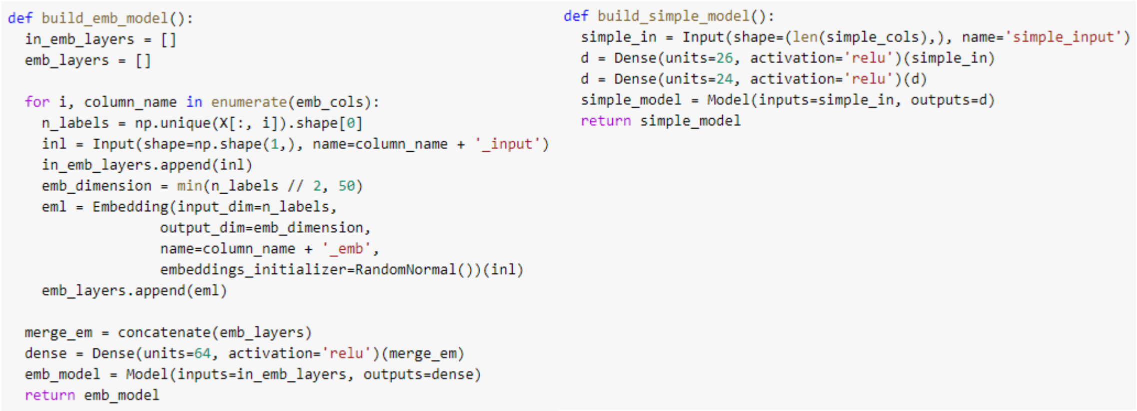
Further, the loaded data is split into categorical or continuous data based on the type of the feature. The data is pre-processed using tools made available by the scikit-learn library [18] [4]. This project, also build on top of Python programming language, provides a multitude of tools for any type of machine learning tasks. In what follows, tools from the preprocessing module were used in order to encode or normalize data. The categorical features are encoded using the OrdinalEncoder, a class that maps each unique category in the data given to a numerical data. This is used since embedding layers do not directly accept data of all forms, for example bytes, which makes feeding airport codes as strings impossible. The continuous features are transformed using a MinMaxScaler, which scales and translates values for each feature such that they fall in the given range, which is by default set to (0, 1). The multiple encoders as well as the scaler are saved to separate files such that they can be used in order to transorm data that will be used for further predictions.

The data is afterwards split into training and validation data, the latter representing 20% of the whole data. In order to do this, another tool made available by the scikit-learn package under the module model_selection is used, namely train_test_split, which also shuffles the dataset before splitting which is a needed operation since data is chronologically ordered. The training data is then resampled using the SMOTETomek technique using the imblearn package which provides methods popular in the literature for balancing imbalanced datasets [14]. The class weights that will be fed to the neural network at training are also computed before the resampling is executed such that a real view on the imbalance between the classes is obtained.

The Neural Network model and all its features are implemented using Keras, a deep learning framework build on top of Tensorflow 2.0. It provides a simple and straightforward API that allows users to easily build any types of neural networks and add numerous features. Building neural networks can be done in two ways in Keras. One is by using the Sequential Model, which allows a single entry point and a single exit point, output. The second method is through the Functional API, which provides more flexibility in the way that layers are chained, making multiple branched neural networks with multiple inputs and outputs possible. Layers that compose the neural network can be easily created by providing a name for the layer, the shape of the input and the number of output neurons. For Dense layers, an activation function must also be specified, while for dropout layers, the dropout rate must be specified.

As far as categorical inputs are concerned, the current implementation requires that each input is mapped to a separate embedding layer. One input layer is created for each of categorical input, these layers being followed by embedding layers which are afterwards merged using a concatenation layer that combines all weights into a single large vector. The continuous features can be fed into the network through a single entry point having a number of rows equal to number of continuous features that is also merged with the large vector resulting from the embedding layers. The code for the two intermediate models that make up the input branches can be seen in Figure 3.12. The two models handling the different inputs forward their outputs to a third final model that is only comprised of hidden dense layers and a dropout layer which are simply chained up until the last layer activated by a sigmoid or softmax function.

After the architecture of the model is defined, its loss functions as well as training metrics and optimizer are set with a call to the compile method. In the case of the binary classification model, a binary cross entropy loss function is chosen, whereas in the case of the multi classifier, a categorical cross entropy loss function is chosen. A compiled model is ready for training which in Keras is done by calling the method fit on the model with plenty of possible parameters. The required parameters are the training features and labels and the number of epochs of training. In order to better understand the performance of the model, validation data can also be passed as parameter with features and labels. A parameter for the class weights can also be passed such that the loss function is computed in a way that does not neglect the minority classes. A useful parameter is the list of callback which are functions or classes called after each epoch is finished and enables some sort of flexibility on the process of training the model. For example, an early stopping callback can monitor a certain metric and stop the training when the learning process does performs under given criteria and restore the model to



```
def build_emb_model():
  in_emb_layers = []
  emb_layers = []

  for i, column_name in enumerate(emb_cols):
    n_labels = np.unique(X[:, i]).shape[0]
    inl = Input(shape=np.shape(1,), name=column_name + '_input')
    in_emb_layers.append(inl)
    emb_dimension = min(n_labels // 2, 50)
    eml = Embedding(input_dim=n_labels,
                    output_dim=emb_dimension,
                    name=column_name + '_emb',
                    embeddings_initializer=RandomNormal())(inl)
    emb_layers.append(eml)

  merge_em = concatenate(emb_layers)
  dense = Dense(units=64, activation='relu')(merge_em)
  emb_model = Model(inputs=in_emb_layers, outputs=dense)
  return emb_model
```

```
def build_simple_model():
  simple_in = Input(shape=(len(simple_cols),), name='simple_input')
  d = Dense(units=26, activation='relu')(simple_in)
  d = Dense(units=24, activation='relu')(d)
  simple_model = Model(inputs=simple_in, outputs=d)
  return simple_model
```

Figure 3.12: Intermediate Models

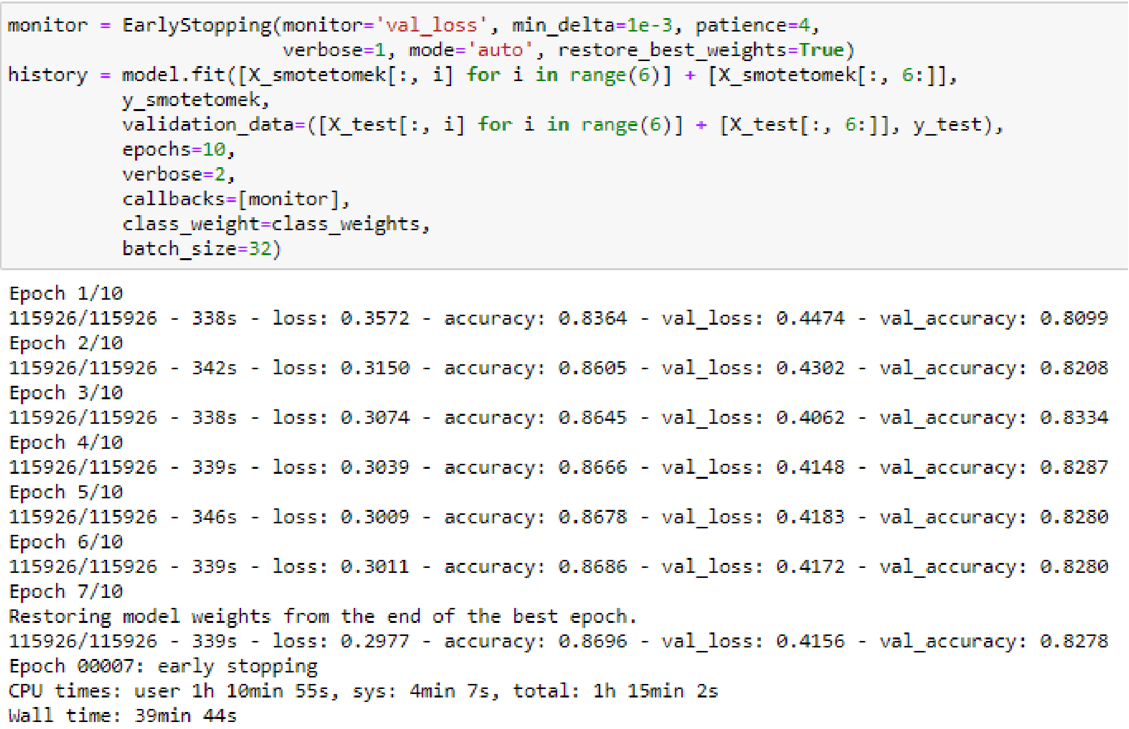
the its best shape. The function call to train the model can be seen in Figure 3.13.

3.3.5 Evaluating the Model

The principal metric used in classification is the confusion matrix, a N x N matrix, where N is the number of classes to be predicted by the model. It shows a summary of the predictions made by the model, categorized both on the correctness as well as the class. On the main diagonal, the number of correct predictions for each class is depicted. On all the other cells of the matrix, the numbers are described as the number of samples being labelled as the label of the column, but predicted with the label of the row, as can be seen in Figure 3.14. An important note is that the positive class in binary classification is the class that should be predicted, say delayed or cancelled flights, while the negative class is, for example, flights arrived on time.

Various metrics can be further computed based solely on the confusion matrix:

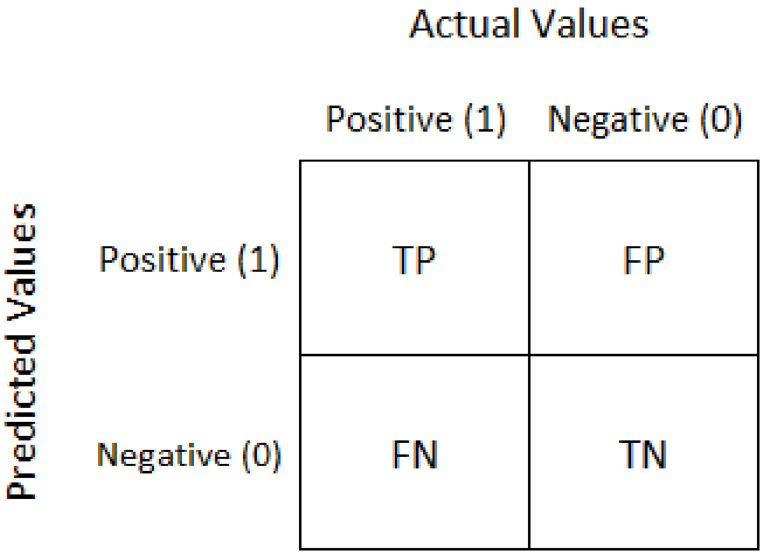
- accuracy: $(TP + TN)/(TP + TN + FP + FN)$, i.e. the number of correct predictions divided by the total number of samples
- precision: $TP/(TP + FP)$, out of all the classes predicted as positive, how many were actually positive; a good measurement for problems where the cost of false positives is high
- recall: $TP/(TP + FN)$, out of all the positive classes, how many were predicted correctly; conversely, it is a good measurement for problems where the cost of false negatives is high
- f1 score: $2 * precision * recall/(precision + recall)$, similarly to accuracy, f1 score is a function of both precision and recall which does not take into consideration



```
monitor = EarlyStopping(monitor='val_loss', min_delta=1e-3, patience=4,
                        verbose=1, mode='auto', restore_best_weights=True)
history = model.fit([X_smotetomek[:, i] for i in range(6)] + [X_smotetomek[:, 6:]],
          y_smotetomek,
          validation_data=([X_test[:, i] for i in range(6)] + [X_test[:, 6:]], y_test),
          epochs=10,
          verbose=2,
          callbacks=[monitor],
          class_weight=class_weights,
          batch_size=32)
```

```
Epoch 1/10
115926/115926 - 338s - loss: 0.3572 - accuracy: 0.8364 - val_loss: 0.4474 - val_accuracy: 0.8099
Epoch 2/10
115926/115926 - 342s - loss: 0.3150 - accuracy: 0.8605 - val_loss: 0.4302 - val_accuracy: 0.8208
Epoch 3/10
115926/115926 - 338s - loss: 0.3074 - accuracy: 0.8645 - val_loss: 0.4062 - val_accuracy: 0.8334
Epoch 4/10
115926/115926 - 339s - loss: 0.3039 - accuracy: 0.8666 - val_loss: 0.4148 - val_accuracy: 0.8287
Epoch 5/10
115926/115926 - 346s - loss: 0.3009 - accuracy: 0.8678 - val_loss: 0.4183 - val_accuracy: 0.8280
Epoch 6/10
115926/115926 - 339s - loss: 0.3011 - accuracy: 0.8686 - val_loss: 0.4172 - val_accuracy: 0.8280
Epoch 7/10
Restoring model weights from the end of the best epoch.
115926/115926 - 339s - loss: 0.2977 - accuracy: 0.8696 - val_loss: 0.4156 - val_accuracy: 0.8278
Epoch 00007: early stopping
CPU times: user 1h 10min 55s, sys: 4min 7s, total: 1h 15min 2s
Wall time: 39min 44s
```

Figure 3.13: Model Training



		Actual Values	
		Positive (1)	Negative (0)
Predicted Values	Positive (1)	TP	FP
	Negative (0)	FN	TN

Figure 3.14: Confusion Matrix; *Source:* [20]

true positives which in most circumstances is not something to be truly important from a business point of view

- Receiver Operating Characteristic (ROC) Area Under the Curve (AUC) score: the ROC curve is a plot of the false positive rate versus the true positive rate, while the area under this curve can be seen as a summary of the skill of the model
- Precision-Recall AUC score: this curve is more useful in the context of imbalanced datasets, especially when the rate of true negatives is not as important

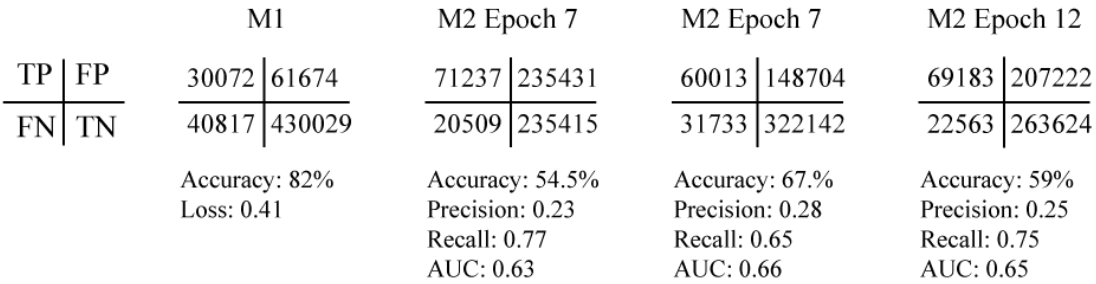
The next models were both trained on data as of months May to July 2019 included.

Binary Classifier

The data fed to the Neural Network configured to return a binary output receives as training data a dataset balanced using the SMOTETomek technique and tested on an imbalanced set. The delay threshold in minutes was set to 30 minutes. The model was trained with class weights set such that the delayed and cancelled flights are in focus.

The first model reached a good training accuracy of 86% and a loss of 0.3 with a validation accuracy of 82.78% and validation loss of 0.41 after also running a hyperparameter tuning algorithm. The high difference between the losses is due to the training dataset being balanced, while the test dataset remained imbalanced. The confusion matrix for M1 in Figure 3.15 gives a more clearer insight into what the model learned. It can be seen that the model has a positive tendency to predict the negative class, making it hopeful that most of the flights will not be delayed. While this aspect lifts up the accuracy and casts down the loss, it does not make the model especially helpful from a practical point of view. Having a greater number of false negatives means that flights that were delayed in reality were predicted by the model as not delayed. As a result, downgrading the rate of false negatives is a preferable goal to achieve than simply rising the accuracy.

The second model was trained with a different class weights parameter that would emphasise the importance of the positive class even more than in the first case. For the training set containing 91746 positive samples and 470846 negative samples, the resulting class weights were 3.048, respectively 0.59 and were computed by simply dividing the total number of samples to the number of samples of the class and dividing the result to two so that the magnitude of the loss does not jump too high. After a



		M1		M2 Epoch 7		M2 Epoch 7		M2 Epoch 12	
TP	FP	30072	61674	71237	235431	60013	148704	69183	207222
FN	TN	40817	430029	20509	235415	31733	322142	22563	263624
		Accuracy: 82% Loss: 0.41		Accuracy: 54.5% Precision: 0.23 Recall: 0.77 AUC: 0.63		Accuracy: 67.% Precision: 0.28 Recall: 0.65 AUC: 0.66		Accuracy: 59% Precision: 0.25 Recall: 0.75 AUC: 0.65	

Figure 3.15: Evolution of the confusion matrix across models and epochs

first round of 7 epochs, the model understood the intention of lowering the rate of false negatives and, as it can be seen in Figure 3.15, the number of false negatives is under half of the number of true positives, far better than the ratio in the first model, where the number of false negatives was higher than the one of true positives. Now the problem is that the model doesn't really make a really good difference between true negatives and false positives, the numbers being quite the same: 235415 vs 235431. This is also signaled by the precision metric which is at the low value of 0.28, while the recall is high, a good sign for problems where the cost of false negatives is high. The training for the next epochs was done by lowering the weight of the positive class and, as an expected result, the number of false negatives jumped, but so did the number of true negatives, so the model now specialized a little bit more into predicting flights arriving on time. This came with the cost of lowering the recall and improving the precision as well as the accuracy. For the last epochs, the class weights were brought to the initial values and the results felt quite in the middle of the two previous cases. While the accuracy is low, 59.15%, the model became quite pessimistic and prefers to classify a flight as delayed, when it won't actually be delayed, than the opposite. This is, from a practical point of view, a more desired behaviour since users would benefit more from being able to choose from flights that are predicted with a higher confidence to arrive on time and flights that that are predicted with a lower confidence to delay.

Multi-Class Classifier

The initial split between only two classes, even though the real world problem imposes 3 classes was done in order to compensate on the high imbalance in the dataset. Asking the classifier to identify between more classes than just two can help in some cases though, so a model classifying flights as being on time, delayed or cancelled was also tested. Training the model without any handling of the imbalance problem, that is without under-/over-sampling or class weights resulted in a 81% accuracy and 0.47 loss, but high numbers of false negatives.

In the second trial, class weights were computed and fed to the model at training time. For 1881755 on-time, 323277 delayed and 45333 cancelled flights, the corresponding class weights were 0.39, 2.32 and 16.54 respectively. This resulted in a 67% accuracy after 20 epochs.

Applying the SMOTETomek algorithm in order to over-sample the minority classes and to under-sample afterwards together with the class weights did not help the model much. While the number of false positives is not too high, the number of false negatives is high although the class weights should have balanced the outcome. The final metrics are: 40% accuracy, 0.42 precision and 0.25 recall.

3.3.6 Further results

A personal curiosity throughout the research and development of this flight delay prediction model was the impact of using embedding layers as a way to handle categorical inputs. For this specific problem, these were origin and arrival airport, days of the week or month, months, carrier codes, etc. While embedding layers were firstly used as a mean to represent vocabulary, considering their actual representation in memory and from a mathematical point of view, they are no more than one hot encoded inputs with a twist for diminishing computational heaviness, as discussed in 2.2.3. After developing the model I was curious to visualise in some way the contents of the embeddings and found that the most relevant input in terms of mapping would have to be the airports. Tensorflow offers a helpful way for graphically representing high dimensional data through their TensorBoard Embedding Projector tool available online at "https://projector.tensorflow.org/".

In order to visualise an embedding, two files must be loaded: one that contains the weights and one that contains the metadata, that is the name of each embedding row. Both of these files must be found in tsv(Tab-Separated-Values) file format. These can be easily saved from an embedding layer that is part of a trained Keras model. After saving the data from the origin_airport embedding layer, which contained one row for each airports, that is a total of 356, and 50 columns, that is the dimensionality of the embedding, the files are loaded in the projector and the points are scattered along a three-dimensional plot, as can be seen in Figure 3.16. Searching for a specific point, in this case airport, is simple and can be done by writing a pattern to match with the labels given in the metadata file. Once a point is selected, the graph highlights

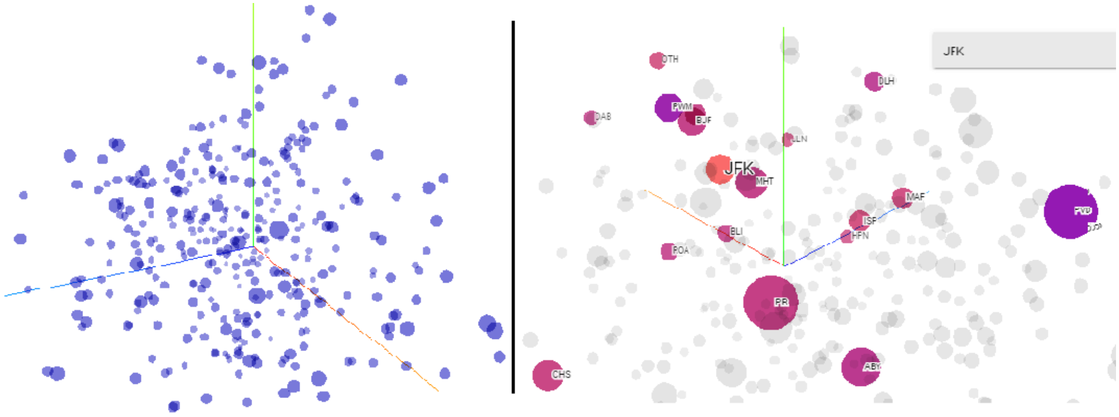
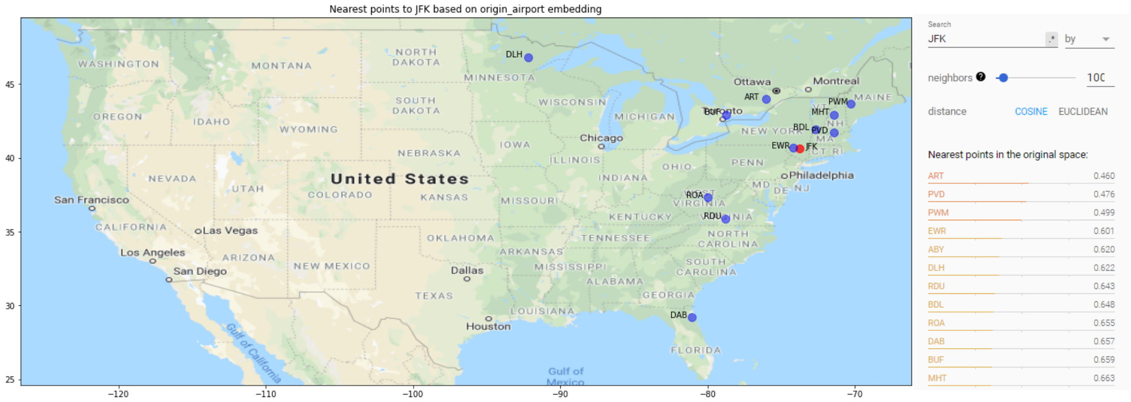


Figure 3.16: Visualisation of the origin_airport embedding in Tensorboard Embedding Projector



Search

JFK

by

neighbors 100

distance COSINE EUCLIDEAN

Nearest points in the original space:

ART	0.460
PVD	0.476
PWM	0.499
EWR	0.601
ABY	0.620
DLH	0.622
RDU	0.643
BDL	0.648
ROA	0.655
DAB	0.657
BUF	0.659
MHT	0.663

Figure 3.17: Geographical plot of the nearest points to JFK airport based on embedding values

some of closest neighbours, the distance between the points being computed as either a cosine similarity or an euclidean distance. Searching for 'JFK' airport, highlights the results from the left image in Figure 3.16 and produces a list of closest neighbours as can be seen in Figure 3.17. A gratifying result was that plotting the 12 closest neighbours of JFK airport, in terms of distance computed using the values in the embedding layers retrieved airports that are also in the proximity of the JFK airport geographically speaking, although the model does not get any geographical data as input in the training phase.

Chapter 4

Practical Application

The application is composed of 2 separate components: the machine learning model that predicts possible flight delays and a user interface that enables users to make use of the model. The final purpose of the project is to enable regular users to get a glimpse of possible statuses of certain flights, that is to know whether the flight will arrive on time, delayed by more than 30 minutes or cancelled.

The machine learning algorithm proposed is based on a Deep Neural Network. It receives as input various attributes describing historical registered flights as well as the weather at the time when the flight took place, some of which are: flight carrier code and number, origin and destination airport, aircraft tail number, scheduled departure and arrival as well as weather related values such as temperature, precipitation, pressure, visibility and wind. The model is a multi-class classifier matching each flight with one class of the following: on-time (arriving before the scheduled arrival time or within the first 14 minutes after), delayed (arriving anywhere later than 15 minutes after the scheduled arrival time) and cancelled (flights that never departed).

The user interface is a web application that enables users to enter new input data unknown to the model and see predictions on the given flights. The weather component that should also be fed as input can also be configured either manually, by entering some weather presets (rainy, foggy, sunny), weather historical means given a period of time or by using a weather forecast API.

4.1 Web Server

In order to expose the functionalities provided by the machine learning model and be able to use them for real world predictions, a web server responding to a GET request was developed. The Framework used for the development of the server is python-based Django which provides an interface to the most important features for an application: responds to HTTP requests and exposes the connection to an SQLite Database with an Object-Relational Mapping.

The HTTP GET request for predicting a flight delay must contain 5 parameters in order for the application to be able to gather all the information the model needs in order to do a prediction: carrier code (a two letter code describing the airline operating the flight), origin and destination airport codes (three letter codes describing the airports) and datetime of departure and arrival having the format 'dd/mm/yy HH:MM'. Upon receiving such a call, the application gathers weather data for the given airports at the given times. The weather is retrieved from a weather API exposing endpoints for forecast weather (up to 15 days from the current datetime) and historical summaries (monthly summaries based on decades of weather measurements). The API is provided by https://www.visualcrossing.com/ and allows for up to 250 requests per day. If the flight requested takes place at any time within 15 days from the current time, the real forecast is requested from the API, otherwise, the prediction will be made with historical summaries for the month in which the flight takes place. In order to minimise the number of requests made to the weather API, the data retrieved is cached into a database which provides two classes containing weather data about a certain airport: temperature, wind speed, precipitation and visibility. When the application received the request, it first checks whether the data is available in the database, and afterwards, if nothing relevant was found, it then makes a request to the Visual Crossing API. After the weather data is gathered, the flight time is computed taking into consideration the different timezones of the times given for departure or arrival. All the inputs are then fed into the model and its prediction, a percentage of how likely the flight is to be delayed) is returned as a response to the request. A visual representation of the workflow can be seen in figure 4.1. The Resources component acts as an efficient provider of information loaded from static files at the time when the first request is made, loading the objects needed in order to preprocess the inputs before feeding them to the neural network, then loading the model and its weights, making it ready for predictions and afterwards loading a file containing mappings from airport codes to geographical coordinates. The coordinates are used as parameters for the weather API.

Components of the Django application were tested using a python package called pytest [13]. Fast, on-request access to the resources on disk was tested, as well as the

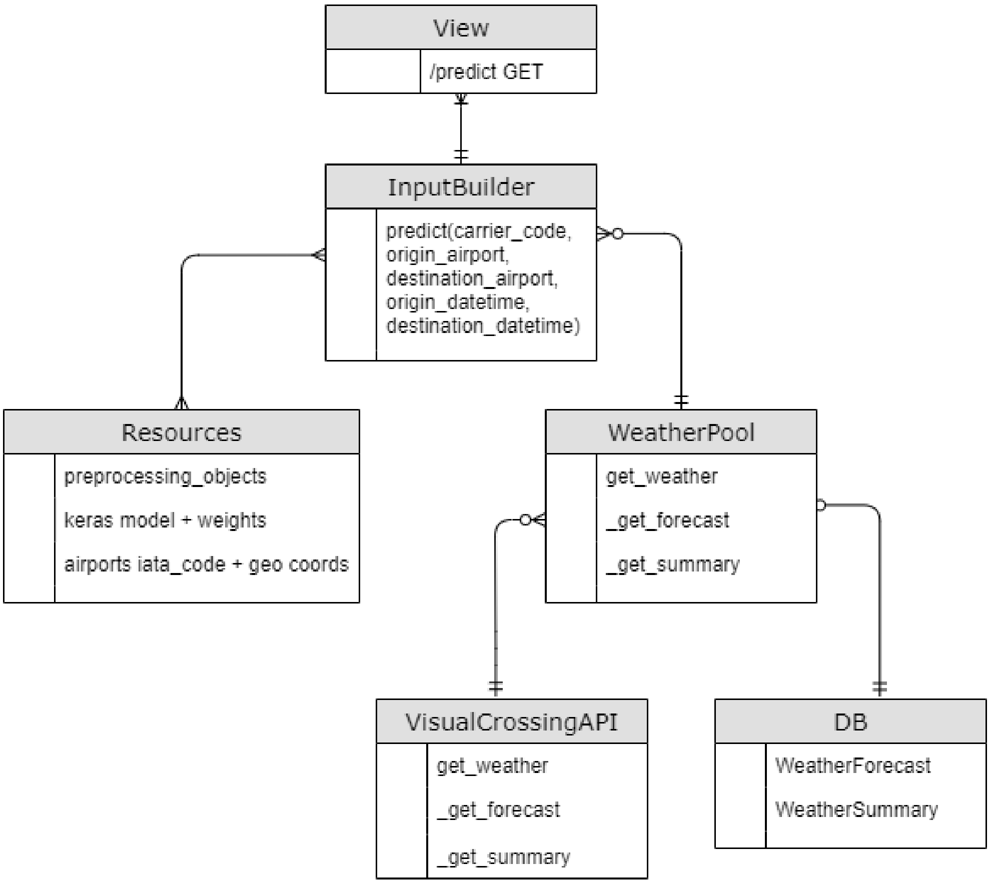


Figure 4.1: REST API application

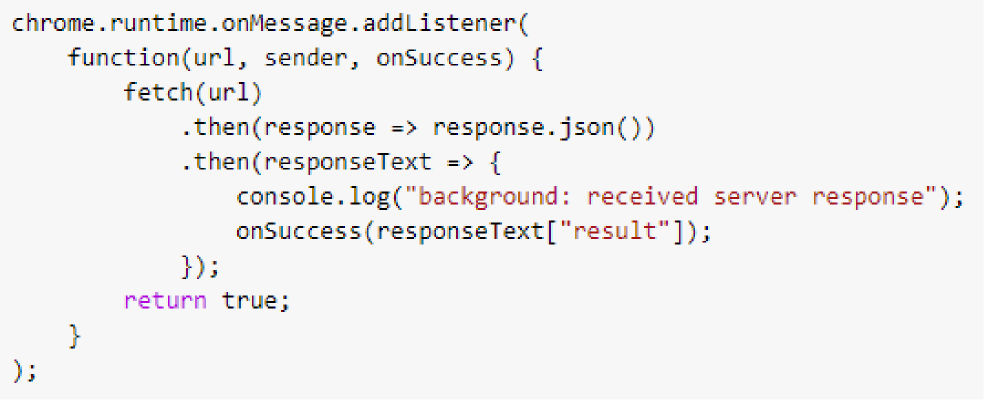
loading and usage of pre-fit pre-processing scikit-learn objects and keras model. Other util methods within the application were also tested for correctness.

4.2 Chrome Extension

Chrome Extensions are intended to be small software programs having a single purpose in customizing the experience of browsing the internet. These are specifically created for the Google Chrome Browser. As stated by the chrome documentation, they allow users to add functionality and behaviour on top of websites in order to tailor the experience. These extensions are developed on top of popular web technologies: HTML, CSS and JavaScript.

4.2.1 Components of Chrome Extensions

In order to start creating an extension, one manifest.json file must be created which contains general information: name of the extension, its version, a description of it and



```
chrome.runtime.onMessage.addListener(
    function(url, sender, onSuccess) {
        fetch(url)
            .then(response => response.json())
            .then(responseText => {
                console.log("background: received server response");
                onSuccess(responseText["result"]);
            });
        return true;
    }
);
```

Figure 4.2: Background Script in a Chrome Extension

a manifest version. The manifest is a mean to declare additional functionalities and components that the extension should include.

In order to provide the extension with the possibility to respond to certain event triggers, should they be clicking on the extension thumbnail or any changes in a browsed page, background scripts must be provided. Declaring the existence and access to such scripts, is done in the manifest file under the "background" key, mentioning as values a list of scrips and a "persistent" key which is recommended to be set to false. In order to react to an event, the background page must declare a listener to "chrome.runtime.onMessage". The lifetime of background scripts can be viewed from the chrome task manager. A listener for events in Chrome extensions is a function that receives as parameters a message, the sender of the message and a response function that can be called when the job that the background scrip is required to do, for example a request, is finished. The execution of the function must be terminated with a return true statement, but further asynchronous calls can be scheduled within the function, before returning. If the job of the background script is to make a request to some external service, then a asynchronous call can be made and it will be executed upon response, even if the function call is ended. A background script that implements this behaviour can be see in Figure 4.2. A request to the provided URL is made and the response is interpreted as json, if the request is successful, then the function provided as onSuccess is called with the contents of the json under the "result" key.

Another component of a Chrome Extension is defined in the manifest via the key "content_scripts", which are CSS or JavaScript files that can be run on pages that match a list of URLs declared within the key "matches". This way, a JavaScript script that needs to retrieve data from a specific set of websites and further send an external request via the background script, can be declared here. If the script has any other dependencies, for example other Javascript packages, like Jquery for example, they can also be listed here. Under the CSS key, the list of CSS files required to change the style of the desired pages can be mentioned.

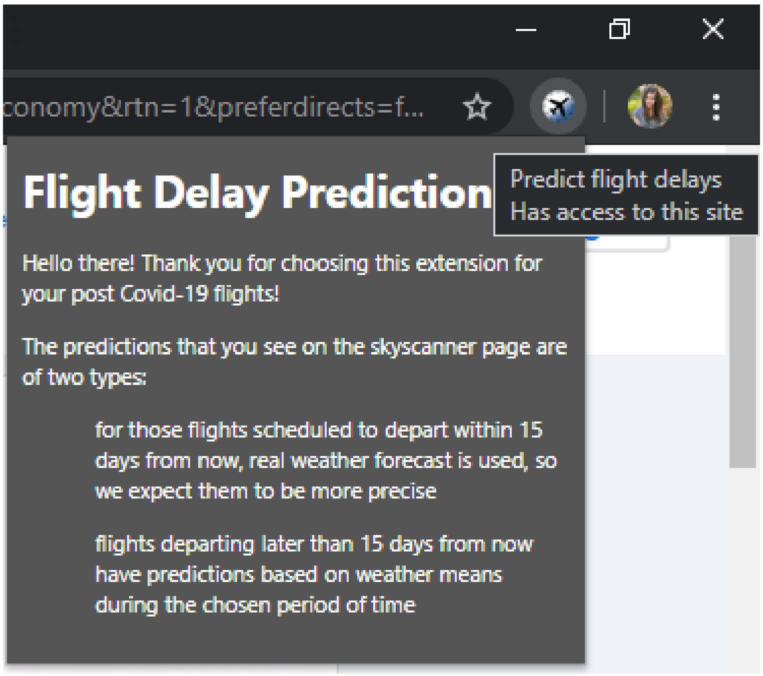
In order to be able to use most functionalities provided by the Chrome API, the manifest must also declare its intent through a set of permissions. These are also a safety measure such that users can be aware of underlying behaviour of the extensions. The values given inside the "permissions" field can be previously declared strings that can be found in the documentation and give access to certain chrome features, or can be URL patterns that give access to wanted hosts. For example, an extension that needs to make requests to a locally deployed server on the 8000 port, must provide the "http://127.0.0.1:8000/*" string to the list of permissions. This way, requests from within the browser to the server will be permitted.

4.2.2 Flight Delay Prediction within www.skyscanner.com

The Chrome Extension allowing users of the Chrome Browser to use the Machine Learning model that predicts flight delays is specifically developed to run on the www.skyscanner.com website accessed via a Chrome Browser. Skyscanner is a metasearch tool that compares and finds routes for a big number of companies around the world, having 100 million users per month. The extension adds a subtle but effective functionality of informing users searching for flights operating in the United States of America about the probability for the searched flights to be delayed. This tool can help potential customers make better decisions between multiple flights and avoid those flights with higher chances of being delayed.

Google Chrome extensions can be either installed from the Chrome Webstore that can be found at https://chrome.google.com/webstore/category/extensions, or manually loaded from a local directory. In order to load a local extension, a user must go to the chrome menu in the upper right corner of the browser, under the 'X' closing button and press the three vertical dots. Under the 'More tools' menu, the 'Extensions' option can be clicked. This option opens a new tab that displays all the currently installed extensions and further options. In the upper left side, the 'Load unpacked' button can be pressed and a File Explorer opens. The user must navigate to the folder containing the extension and click 'Select Folder'. This will load and activate the extension which now also appears with a thumbnail to the left of the three vertical dots menu, like in Figure 4.3.

A user of the website can search flights by entering a route (origin and destina-



Flight Delay Prediction

Hello there! Thank you for choosing this extension for your post Covid-19 flights!

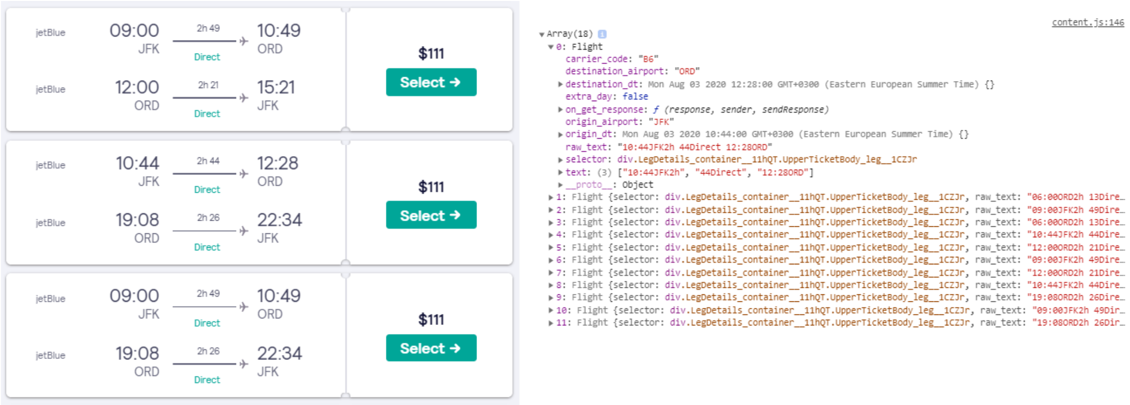
The predictions that you see on the skyscanner page are of two types:

- for those flights scheduled to depart within 15 days from now, real weather forecast is used, so we expect them to be more precise
- flights departing later than 15 days from now have predictions based on weather means during the chosen period of time

Predict flight delays
Has access to this site

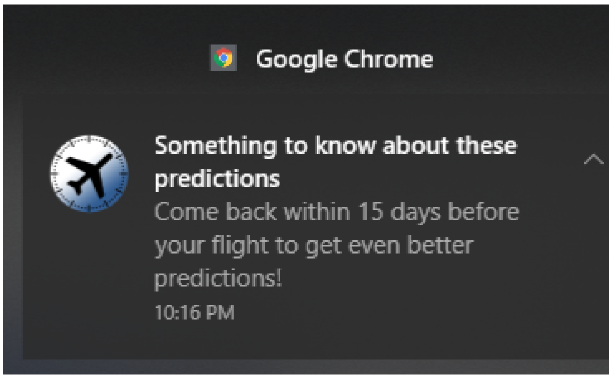
Figure 4.3: Extension Popup loaded in Chrome

tion) and the dates of the desired flight. When the user hits the search button, the extension retrieves the dates of the flights (departure, as well as returning date in the case that the user is searching for round trip flights). As results are generated on the page, the extension is notified via a MutationObserver, a feature that detects changes in the DOM page, useful in this case as the website continuously provides new results without navigating to new pages. For each direct route (which doesn't have intermediate airport stopovers), the extension creates new instances of the Flight class. The information is retrieved at instance creation from the web page source via selectors which provide the text displayed in raw format. For example, a flight selector "div.LegDetails_container__11hQT.UpperTicketBody_leg__1CZJr" provides the raw text "10:44JFK2h 44Direct 12:28ORD" which describes a direct flight departing at 10:44 from JFK airport and arrives at 12:28 at ORD airport and has a scheduled in-flight time of 2 hours and 44 minutes. The raw text can also provide information about flights being operated by other flight carriers, for example "09:40JFK2h 47Direct 11:27ORDOperated by Endeavor DBA Delta Connection". For regular flights, the carrier is extracted from the "alt" attribute of the logo placed on the left side of the route. Figure 4.4 shows some of the elements extracted from the skyscanner page. After the carrier code, the airport codes and the dates are retrieved, each flight instance sends a message to the background script with the parameters and binds a class function that should be called as the response is ready. This function creates a span attached to the



```
                                                                                    content.js:146
▾Array(18) ℹ
  ▾0: Flight
      carrier_code: "B6"
      destination_airport: "ORD"
    ▸destination_dt: Mon Aug 03 2020 12:28:00 GMT+0300 (Eastern European Summer Time) {}
      extra_day: false
    ▸on_get_response: ƒ (response, sender, sendResponse)
      origin_airport: "JFK"
    ▸origin_dt: Mon Aug 03 2020 10:44:00 GMT+0300 (Eastern European Summer Time) {}
      raw_text: "10:44JFK2h 44Direct 12:28ORD"
    ▸selector: div.LegDetails_container__11hQT.UpperTicketBody_leg__1CZJr
    ▸text: (3) ["10:44JFK2h", "44Direct", "12:28ORD"]
    ▸__proto__: Object
  ▸1: Flight {selector: div.LegDetails_container__11hQT.UpperTicketBody_leg__1CZJr, raw_text: "06:00ORD2h 13Dire…
  ▸2: Flight {selector: div.LegDetails_container__11hQT.UpperTicketBody_leg__1CZJr, raw_text: "09:00JFK2h 49Dire…
  ▸3: Flight {selector: div.LegDetails_container__11hQT.UpperTicketBody_leg__1CZJr, raw_text: "06:00ORD2h 13Dire…
  ▸4: Flight {selector: div.LegDetails_container__11hQT.UpperTicketBody_leg__1CZJr, raw_text: "10:44JFK2h 44Dire…
  ▸5: Flight {selector: div.LegDetails_container__11hQT.UpperTicketBody_leg__1CZJr, raw_text: "12:00ORD2h 21Dire…
  ▸6: Flight {selector: div.LegDetails_container__11hQT.UpperTicketBody_leg__1CZJr, raw_text: "09:00JFK2h 49Dire…
  ▸7: Flight {selector: div.LegDetails_container__11hQT.UpperTicketBody_leg__1CZJr, raw_text: "12:00ORD2h 21Dire…
  ▸8: Flight {selector: div.LegDetails_container__11hQT.UpperTicketBody_leg__1CZJr, raw_text: "10:44JFK2h 44Dire…
  ▸9: Flight {selector: div.LegDetails_container__11hQT.UpperTicketBody_leg__1CZJr, raw_text: "19:08ORD2h 26Dire…
  ▸10: Flight {selector: div.LegDetails_container__11hQT.UpperTicketBody_leg__1CZJr, raw_text: "09:00JFK2h 49Dir…
  ▸11: Flight {selector: div.LegDetails_container__11hQT.UpperTicketBody_leg__1CZJr, raw_text: "19:08ORD2h 26Dir…
```

Figure 4.4: Skyscanner interface and elements extracted



Google Chrome

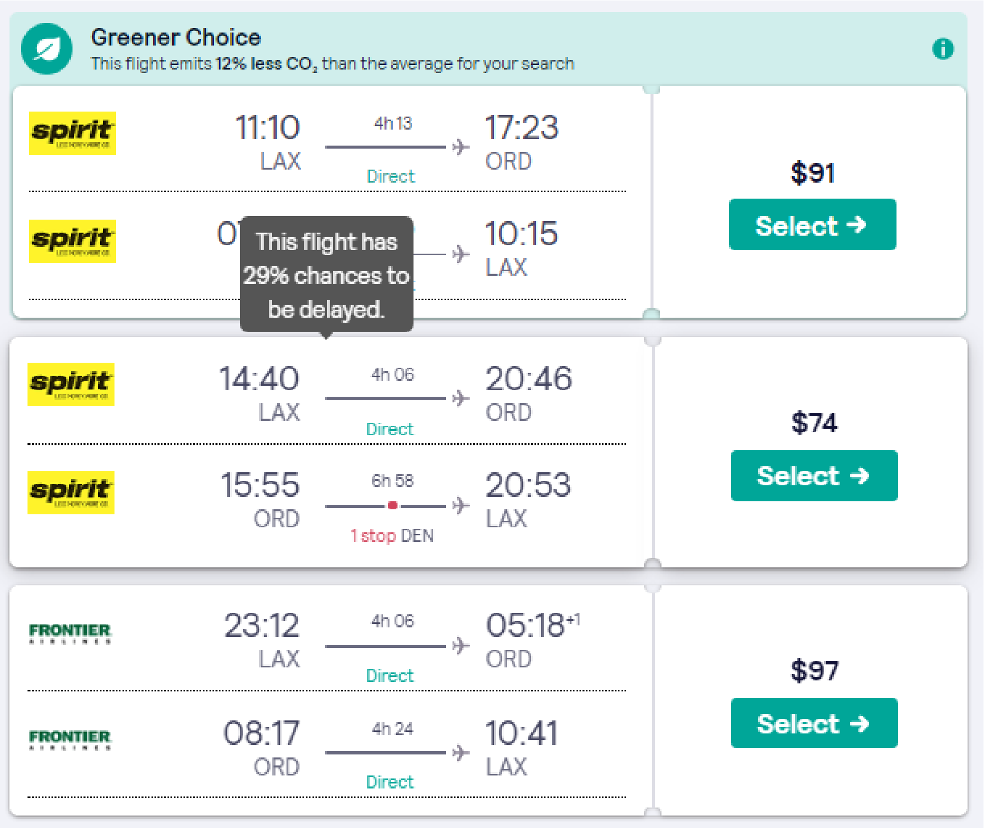
Something to know about these predictions

Come back within 15 days before your flight to get even better predictions!

10:16 PM

Figure 4.5: Notification about the predictions

selector of the concerned flight, containing a text and the delay probability provided by the server. The span is only visible when the user hovers a flight. Figures 4.6 and 4.7 show the behaviour and also reveal that a Spirit flight during the day has a lower chance of delay than a night flight with Frontier. Predictions for flights departing later than 15 days from the current time are computed with weather summary data, as opposed to real forecasts, this aspect can have an important effect on the overall prediction. As a result, when a user searches for later dates, a notification is created by the extension informing the user about the types of the predictions, as can be seen in Figure 4.5.



Greener Choice

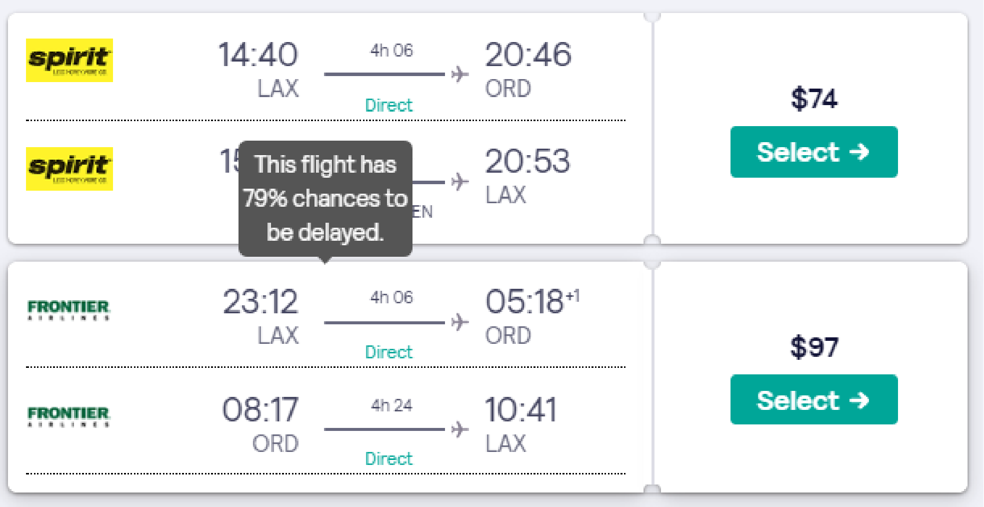
This flight emits **12% less CO$_2$** than the average for your search

spirit	11:10 LAX	4h 13 Direct	17:23 ORD	$91	Select →
spirit	0		10:15 LAX		

This flight has 29% chances to be delayed.

spirit	14:40 LAX	4h 06 Direct	20:46 ORD	$74	Select →
spirit	15:55 ORD	6h 58 1 stop DEN	20:53 LAX		
FRONTIER	23:12 LAX	4h 06 Direct	05:18^{+1} ORD	$97	Select →
FRONTIER	08:17 ORD	4h 24 Direct	10:41 LAX		

Figure 4.6: Extension Behaviour on hover



spirit	14:40 LAX	4h 06 Direct	20:46 ORD	$74	Select →
spirit	15	EN	20:53 LAX		

This flight has 79% chances to be delayed.

FRONTIER	23:12 LAX	4h 06 Direct	05:18^{+1} ORD	$97	Select →
FRONTIER	08:17 ORD	4h 24 Direct	10:41 LAX		

Figure 4.7: Extension Behaviour on hover

Chapter 5

Conclusion

The underlying causes of flight delays are numerous and mostly the results of chained errors in what has to be a perfectly choreographed spectacle of both people and machines striving to arrive ahead of schedule to make sure they make it on time. The clear snowball effect affecting everything in the whole system of air traffic has driven people to analyse and make predictions based on the succession of events and the more or less certainty that delays propagate to further delays by feeding either past events in the given day at the connected airports or previous flights of the concerned aircraft. These were analysed using methods like Random Forest, SVM Classifier, Gradient Boosting Classifier or Recurrent Neural Networks.

The proposed problem was to predict flight delays later in time based on data that can be known well ahead of time to anyone willing to book a flight. This implied the inability to rely on any real time delays that might propagate to further delays, and to rather analyse general trends and capabilities of flight carriers, airports and effects of certain days of the week or weather to any possible flying route. A deep learning approach was used thanks to the high amounts of data available and the scale to which the problem rises in a hope that other patterns causing delays might exist apart from delay propagation or extreme weather. A binary model as well as a three-class classifier were built and achieved modest results as far as accuracy and other typical metrics are concerned, but results that might still benefit a regular user when given the possibility to choose from multiple flights when presented with delay predictions. A REST service was built that can respond to a request holding only little information about the flight: carrier, airports on the route and the times when the flight takes place, data that can be found on any website when searching to book for a flight. The service takes care of making further requests to obtain relevant weather data. In order to also make the model easily reachable, a chrome extension was build that, when installed in the browser, automatically displays delay predictions for the routes that result from a search on the www.skyscanner.com website.

In closing, more research can be done in improving the model by applying state of the art methods for neural networks, applying different imbalanced datasets handling techniques or even by feeding more data about flights.

Bibliography

[1] Gustavo Batista, Ana Bazzan, and Maria-Carolina Monard. Balancing training data for automated annotation of keywords: a case study. pages 10–18, 01 2003.

[2] Loris Belcastro, Fabrizio Marozzo, Domenico Talia, and Paolo Trunfio. Using scalable data mining for predicting flight delays. *ACM Trans. Intell. Syst. Technol.*, 8(1), July 2016.

[3] Olivier Bousquet, Ulrike von Luxburg, and Gunnar Ratsch. *Advanced Lectures On Machine Learning: ML Summer Schools 2003, Canberra, Australia, February 2-14, 2003, Tubingen, Germany, August 4-16, 2003, Revised Lectures (Lecture Notes in Computer Science).* SpringerVerlag, 2004.

[4] Lars Buitinck, Gilles Louppe, Mathieu Blondel, Fabian Pedregosa, Andreas Mueller, Olivier Grisel, Vlad Niculae, Peter Prettenhofer, Alexandre Gramfort, Jaques Grobler, Robert Layton, Jake VanderPlas, Arnaud Joly, Brian Holt, and Gaël Varoquaux. API design for machine learning software: experiences from the scikit-learn project. In *ECML PKDD Workshop: Languages for Data Mining and Machine Learning*, pages 108–122, 2013.

[5] Nitesh Chawla, Kevin Bowyer, Lawrence Hall, and W. Kegelmeyer. Smote: Synthetic minority over-sampling technique. *J. Artif. Intell. Res. (JAIR)*, 16:321–357, 01 2002.

[6] François Chollet et al. Keras. https://keras.io, 2015.

[7] K.-L Du and M.N.s Swamy. *Perceptrons.* 12 2014.

[8] Noriko Etani. Development of a predictive model for on-time arrival flight of airliner by discovering correlation between flight and weather data. *Journal of Big Data*, 6:1–17, 2019.

[9] Simon Hawkins. pandas-dev/pandas: Pandas 1.0.4, May 2020.

[10] Jeremy Howard and Sylvain Gugger. Fastai: A layered API for deep learning. *Information*, 11(2):108, 2020.

[11] Charles F. F. Karney. Algorithms for geodesics. *Journal of Geodesy*, 87:43–55, 2013.

[12] Y. J. Kim, S. Choi, S. Briceno, and D. Mavris. A deep learning approach to flight delay prediction. In *2016 IEEE/AIAA 35th Digital Avionics Systems Conference (DASC)*, pages 1–6, 2016.

[13] Holger Krekel, Bruno Oliveira, Ronny Pfannschmidt, Floris Bruynooghe, Brianna Laugher, and Florian Bruhin. pytest x.y, 2004.

[14] Guillaume Lemaître, Fernando Nogueira, and Christos K. Aridas. Imbalanced-learn: A python toolbox to tackle the curse of imbalanced datasets in machine learning. *Journal of Machine Learning Research*, 18(17):1–5, 2017.

[15] Warren S Mcculloch and Walter Pitts. *A logical calculus of the ideas immanent in nervous activity.* Univ. Press, 1943.

[16] Tom M. Mitchell. *Machine Learning.* McGraw-Hill Science/Engineering/Math, 1997.

[17] International Civil Aviation Organization. The world of air transport in 2018, annual report of the council.

[18] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[19] A. L. Samuel. Some studies in machine learning using the game of checkers. *IBM Journal of Research and Development*, 3(3):210–229, July 1959.

[20] Sarang Narkhede. File: Confusion matrix. [Online; accessed 14-June-2020].

[21] Wei Shao, Arian Prabowo, Sichen Zhao, Siyu Tan, Piotr Koniusz, Jeffrey Chan, Xinhong Hei, Bradley Feest, and Flora D. Salim. Flight delay prediction using airport situational awareness map. In *Proceedings of the 27th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*,

SIGSPATIAL '19, page 432–435, New York, NY, USA, 2019. Association for Computing Machinery.

[22] Ivan Tomek. Two modifications of cnn. *IEEE Transactions on Systems, Man, and Cybernetics*, 6, 11 1976.

[23] Yufeng Tu, Michael O Ball, and Wolfgang S Jank. Estimating flight departure delay distributions—a statistical approach with long-term trend and short-term pattern. *Journal of the American Statistical Association*, 103(481):112–125, 2008.

[24] Bureau of Transportation Statistics U.S. Department of Transportation. Understanding the reporting of causes of flight delays and cancellations, March 2020.

[25] Wes McKinney. Data Structures for Statistical Computing in Python. In Stéfan van der Walt and Jarrod Millman, editors, *Proceedings of the 9th Python in Science Conference*, pages 56 – 61, 2010.

[26] Wikipedia contributors. File: Artificial neural network. [Online; accessed 12-June-2020].

[27] Wikipedia contributors. Feature scaling — Wikipedia, the free encyclopedia, 2020. [Online; accessed 12-June-2020].

[28] Wikipedia contributors. Oversampling and undersampling in data analysis — Wikipedia, the free encyclopedia, 2020. [Online; accessed 12-June-2020].