Live Aircraft Destination Prediction using Machine Learning & Adversarial Attacks

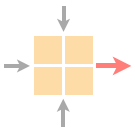
Bachelor Thesis

Author(s):
Lindner, Mathis Kurt

Publication date:
2022-08-03

Permanent link:
https://doi.org/10.3929/ethz-b-000568043

ETH zürich



Networked Systems
ETH Zürich — seit 2015

Live Aircraft Destination Prediction using Machine Learning & Adversarial Attacks

Bachelor Thesis

Author: Mathis Lindner

Tutor: Martin Strohmeier

Supervisor: Prof. Dr. Laurent Vanbever

August 3, 2022

Abstract

As a continuation of uncovering threats posed in aviation that are due to ADS-B not being secure, we looked into how a non-commercial aircraft user's privacy can be compromised in real-time with publicly available data. Since there are many ways to predict the destination of an airplane whilst in the air we investigated this matter using a long short-term memory and a gradient booster model. With the gradient booster model, depending on the planes operator, we successfully achieved an accuracy of 0.70 to 0.89, 30 minutes before the aircraft lands.
In addition, with adversarial machine learning attacks, we deluded the gradient booster model, while keeping the attacker's capabilities as omnipotent but still achievable as possible. We managed to misdirect the model with a naive based and targeted based poisoning attack. The most effective attack on the model seemed to be the targeted one, it had the greatest dissimulation with the least data manipulation.

Contents

Chapter 1

Introduction

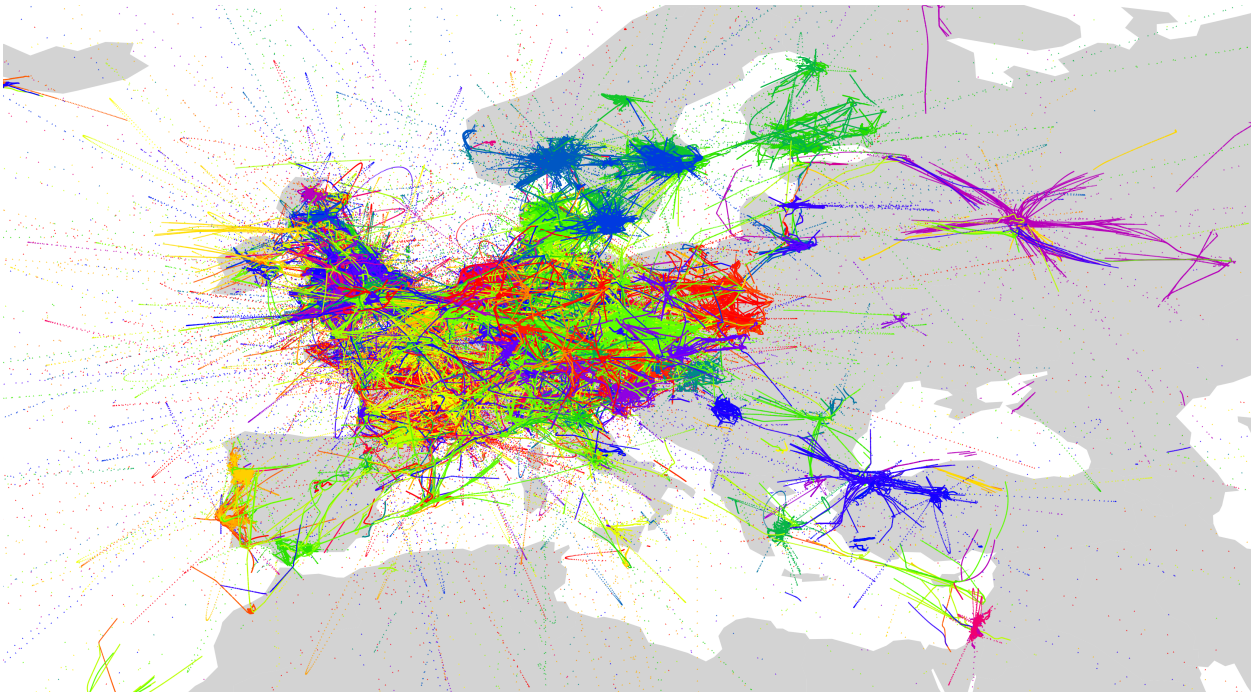


Figure 1.1: Info-graphic of flight paths recorded over Europe by OpenSky from 2022

1.1 Motivation

Nowadays we can track aircraft in real-time all over the world thanks to dedicated websites such as Opensky & Flightradar24 [7, 1]. This already opened new privacy issues that lead to sensitive information being exposed. Governmental activities have been uncovered thanks to open-source intelligence [13]. It is a recurring theme, that aircraft owners are concerned that their privacy is being violated by having their location disclosed e.g: In January 2022, Elon Musk asked the owner of the elonjet Twitter account [15], a bot that sends out a tweet every-time Elon Musk's Gulfstream G650 lands at an airport: "Can you take this down? It is a security risk.". This Twitter bot uses the OpenSky database, just as we did. This paper is a continuation of the issues that have been shown on the lack of privacy and security in aviation [10].

1.2 Research Questions & Contributions

We can now ask ourselves

- How extensive is the threat of an uncovering of the destination of a plane while it is still in the air?
- Is there a way for aircraft owners to defend themselves against this attack on their privacy considering automatic dependent surveillance–broadcast & open-source intelligence as they exist today?

This Thesis's primary contributions are

- Successfully predicting the destination of non-commercial planes using a gradient booster model.
- The creation of a live interactive map that predicts aircraft destinations in real-time using the classifier.
- Exposing methods to delude the predictions using adversarial machine learning attacks, suited for an injection.

1.3 Methodology

Machine learning is ubiquitous and not always suitable. We could question the methodology as to why someone would use such models and not just make a simple educated guess, by putting rules oneself as to where and when a plane could land. To put it in another way, one rule could be: if a plane starts descending, we could directly try to predict where the plane may land, based on the heading as soon as the altitude starts decreasing. This alternative would be a tedious task as airspace regulations vary from country to country. Furthermore, different planes have different behaviours, which would complicate it even more. Ultimately, it would be rather challenging to perform as efficiently as a machine learning solution.

1.4 Outline

This Thesis will discuss how the data was acquired, the models that were considered and then how they were implemented. Thereafter, we will consider different attacks on the model. In the results and discussion section we will verify how well the predictions turned out and discuss the discrepancies which occurred. We will also analyse how much the model can be mislead with adversarial machine learning and why some attacks performed better than others. We conclude with additional work that has been done to demonstrate the capabilities of the model and future ideas.

Chapter 2

Background and Related Work

2.1 Aircraft Tracking System

Automatic Dependent Surveillance–Broadcast (ADS-B) is a system that enables tracking planes thanks to transmitters that send out information about the aircraft by themselves and receivers which can be in the air or on the ground. This technology is, among other things, meant to enhance situational awareness for users in the air and for ground receivers such as air traffic control (ATC). The data that is sent out by the aircraft transmitters includes: a unique identifier (ICAO24), the longitude, the latitude et al. [12]

2.2 Lack of Privacy & Security in Aviation

The lack of privacy and security in the aviation industry has been extensively researched.
The security of ADS-B is not assured because the data-links are not encrypted and there is no confidentiality. The paper on uncovering the possibility of utilizing air traffic communications for open-source intelligence [14] exposed a security risk that involves making use of an open-source database such as OpenSky. Moreover, the risk of spoofing an aircraft has been proven to be possible, which allows us to assume access to online databases for our adversarial attacks [10, 4].
The privacy of users is not assured either: there are numerous reports of incidents that already occurred such as the leak of military operations and the tracking of assets belonging to governments and businesses [11].

2.3 Prediction Models

In order to predict where an aircraft will land, we needed a model. A few options were considered in our research. We are assuming the reader's knowledge about recurrent neural networks and decision tree models, more specifically, long short-term memory and gradient boosting [9, 3].
Since the data on flight trajectories is sequential, the idea of utilizing a recurrent neural network seemed natural. Long short-term memory is a deep learning model that has been used mainly for natural language processing, but has its affinities when it comes to time-series regression and classification. LSTM models have been proven to be an effective method to predict the trajectory that a plane will take in form of a regression [18]. LSTM models used to be the state of the art when it came to natural language processing models until 2017, when transformers were introduced with a recognized paper in machine learning: "All you need is attention" [16]. Due to this paper, we considered using a transformer for solving our time series classification problem. After a reasonable

amount of research, this paper on using them on time-series [17] convinced us otherwise. We did not look further into it.
As an alternative to the LSTM model, since decision trees have proven their effectiveness on classification countless times, even more so in flight destination prediction [5] where a random forest algorithm was used, we opted for a gradient boosting model. It is a popular choice when it comes to classification and usually said to outperform the classic random forest technique.

2.4 Adversarial Machine Learning

Adversarial machine learning is a great solution to evaluate the stability and to verify the resistance of a model against a potential attacker that is actively trying to misdirect the predictions. It is known to be an underrepresented technique to legitimize the use of machine learning to solve tasks, while making sure we are providing honest and accurate results [8].
To get feasible ideas as to how to attack our model, we looked into possible adversarial attacks on machine learning models with this survey [2] and noted, that an evasion and poisoning attack were both possible approaches. Our goal had become the creation of a naive-based and targeted-based attack on our model.

So far, predicting trajectories with an LSTM and predicting destinations have been worked on with a random forest algorithm. Instead, we made predictions in real-time using a gradient booster algorithm and then trying to defy those capabilities using adversarial machine learning.

Chapter 3

Data Acquisition and Preprocessing

3.1 Sources of Data

We used several sources of data that could affect the behaviour of our models and are therefore clarified explicitly in this section.

3.1.1 Aircraft Database

To gather trajectories that are beneficial to train our models on, the first step was to choose the aircraft of interest. There are countless planes that all have different purposes of flying. We needed to keep them in separate categories whilst ignoring the aircraft that are pointless to predict: it was necessary to filter out planes that we are not interested in. For us this meant that we had to ignore all the planes that disclose information on where they are planned to land such as commercial flights.Their scheduled flights are publicly made available by various websites such as OpenSky [7] & Flightradar24 [1]. Therefore, commercial flights were the ones were filtered out first. Furthermore, to enhance individual performances of each classifier on different aircraft types, we heuristically clustered the non-commercial planes into four categories: Private, Business, Government/State and Military. This was done thanks to the OpenSky aircraft database, that gave away information on aircraft models and the kind of operator. To better understand the distribution of the aircraft owners in each category, we provide a table below 3.1 and a plot in the Appendix A.2.

Owner Type	Number Of Planes
Military	11882
Private	66366
Business	32687
State/Government	2180

Table 3.1: Aircraft's to Owner Types

The decision of clustering aircraft owner's into different categories was made, based on an attempt to distinguish the different plane behaviors: The flight path, the airports where the planes land on, their general flying altitude and their climbing rates seemed to differ depending on the the owner type. e.g business planes will not land on the same airports than military planes. Therefore we also clustered the airports based on the flights that were made by the different plane owners, according to who landed where. How the aircraft are clustered has an immense impact on how accurate the prediction will be. To justify our claim about the flight paths differing according

to the owner type we looked extensively at their behavior: a large mount of private planes will circle around their starting position and land at their departure airport. This was noticed while gazing over multiple flight paths and then was confirmed by polar plots like these ones 3.1. We can observe that private planes often orbit around the airport, while the business plane flies head on their destination, until it is close to it's desired airport, where it will engage in a landing maneuver. To further support this case, we added 2 flights that were categorised as private in the Appendix. A.4a A.4b

(a) Business flight

(b) Private Flight

Figure 3.1: Polar plots on 2 plane's heading with the time in seconds

3.1.2 Aircraft State Vectors Database

Once we knew which planes and their respective owner we want to track, we gathered their state-vectors from the OpenSky historical database, which included the following features: timestamp, longitude, latitude, heading, altitude and the estimated arrival airport. To get more insight on the the state vectors, some feature extrapolation was performed and since the data from the database was noisy, it had to be cleaned up and re-sampled to further improve performance. We added a table 3.2 and the matching figure to get an idea of the distribution of the flights that were recorded from the beginning of the year 2022, to June 2022. A.3.

Owner Type	Number Of Flights
Military	40180
Private	50268
Business	459746
State/Government	24601

Table 3.2: Number of Flights to Owner Types

3.1.3 Airport Database

To help us with the clustering, we downloaded the airports names, latitude, longitude from the "ourairports" website [6]. The airport database also includes runway lengths and heading for some, but not for all airfields. Unfortunately, not enough airports had all of those specifications which

refrained us from using the specifications as features. We ended up solely using the longitude and latitude. The code to show how we clustered them with the sklearn module is in the appendix C.1. In the paper [5], the authors used 150 clusters, which ended up being effective for us too, as you can observe with the elbow curve that we added in the AppendixA.1. Technically we predict a narrower location since we are working with a different clustering for each owner type, compared to one for all. For a visualisation of those clusters, we added a figure below 3.2 to get a proper impression of the granularity. More specifically, these are the clusters that were created for the private flights that landed in Europe.

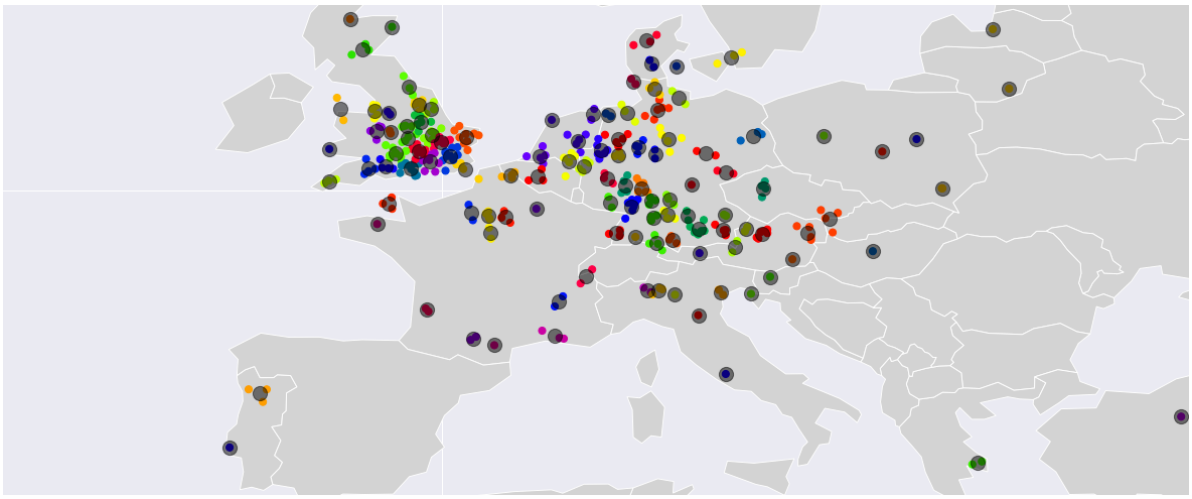


Figure 3.2: The clustering of the airports used by private flights in Europe

3.2 Data Cleaning

After acquiring the data, analysing it and trying to create a fitting model, we determined that several cleaning processes were necessary. We came back on this matter several times because some issues were discovered only later on.

3.2.1 Excluding Outliers

We noticed that some of the flight trajectories contained data points that were clearly noise. An example is provided in the appendix A.5a. The question then came up, as to how to take care of those outliers. We looked at several options on how to treat them, and chose to heuristically filter out the state vectors, that had a speed that did not seem within reason. More specifically, the speed that we calculated in the feature extrapolation helped us to filter out unrealistic distances that could not have been performed in a short amount of time: any two coordinates that were recorded some time apart that resulted in a speed over 300m/s were discarded. Even if the algorithm discarded one too many data-points, with our re-sampling and interpolation, we ended up getting a data-set, that was more uniform than before.

3.2.2 Re-sampling and Interpolation

Originally, some flights contained a data-point every second, other flights had significant gaps that could be interpolated. To reduce the variability of the granularity of the state vectors, we first tried

to re-sample the data to get a data-point every 15 seconds and interpolate gaps up to 5 minutes. The initial choice of a sample rate of 15 seconds was made based on the OpenSky [7] live API's reliability: When trying to get the live state vectors from the OpenSky database, we managed to get a response approximately every 15 seconds. This granularity turned out to be too much data for not enough entropy. We ended up choosing a sample rate of 1 minute, which fairly reduced the data-set's size and the time of training. It was a necessary reduction to perform the adversarial attacks later on. The data size was reduced from originally 50GB to 7GB. The re-sampling and interpolation results of one flight can be found in the appendix. A.5

3.2.3 Feature Extrapolation

To ease the process of classification for the models, we chose to extrapolate new features. We ended up not keeping all of the features, because some did not make a significant difference. Moreover, they were clogging the pipeline for our adversarial attacks instead of giving us better predictions. We tried adding features such as

- mapping the seconds/minutes/hours onto sinusoidal functions to use the temporality of the landing of aircraft's, thanks to the timestamps of the data-points
- the distance between a data-point and the previous one thanks to the longitude, latitude and the haversine distance equation 3.1
- the ground speed of the aircraft, thanks to the distances between the points and the timestamps
- the GPS heading of the aircraft, thanks to longitude and latitude of 2 data-points and this formula 3.2

$$d = 2R \arcsin \sqrt{\sin^2 \frac{\Delta\varphi}{2} + \cos\varphi_1 \cos\varphi_2 \sin^2 \frac{\Delta\lambda}{2}} \tag{3.1}$$

$$\beta = atan_2(\cos\lambda_2 \cdot \sin\Delta\phi, cos\phi_1 \cdot \sin\phi_2 - \sin\phi_1 \cdot \cos\phi_2 \cdot \cos\Delta\lambda) \tag{3.2}$$

with:

- R being the earths radius, which was chosen to be 6,371 km
- λ being the latitude
- ϕ being the longitude

After looking through feature importance plots, we ended up keeping these features 3.3. In other words, these features ended showing to be the most valuable ones.

Longitude	Latitude	Heading	Altitude	Ground Speed	Vertical Rate

Table 3.3: Features used for the prediction

Chapter 4

Implementation

4.1 Models

The paper on flight destination deanonymization by an online attacker [5] suggested the idea of using a deep-learning model, more specifically a recurrent neural network. A popular choice of RNN is the long short-term memory model. We invested non-negligible time and effort designing and training the deep learning model with various parameters, but we ended up choosing the gradient booster model to perform the adversarial machine learning attacks.

4.1.1 Long Short Term Memory

Knowing that LSTM models perform well when it comes to time series modeling and classification, we tried tweaking different parameters to solve our problem. We originally chose a different prediction domain to make our first evaluations on the models. Our first approach implemented a prediction on the top 10, 100 and 1000 most flown to airports with all owner types for one classifier without clustering them. For the features, we chose to use the ones mentioned prior 3.3 and for the length of the sequence, we chose 3 data-points 15 seconds apart. For the sizes of the hidden layers, when trying to predict flights to the top 100 flown to airports, we tried a few different dimensions from 32 to 512. To predict the top 1000 airports we tried hidden layers of sizes 512 to 2048. Connected to the last hidden layer, we added 1 more dense layer of the same size as the airports, to give it another possibility to diversify its prediction. To make sure our prediction was as accurate as possible, we also tried designing the LSTM network with 2 connected dense layers at the end but noticed, that one was enough and a second one just slowed down the learning capabilities of the model. In other words, the depth of the network inferred enough information to predict the final airport. We added a snippet of the implementation in the appendix C.3.

4.1.2 Gradient Booster

Since gradient boosting is one of the most popular methods for classification, we thought it would be a great opportunity for us to work with it. Indeed, it worked as expected, it turned out to be a viable option to test our adversarial machine learning attacks. For the features, we flattened three data-points that followed each-other, to simulate a realistic opportunity to predict the destination: with just one we would have less information, with more than 3 we would have to wait longer to get results in a live prediction scenario. The gradient boosting was performed on the data-set that was re-sampled to be one state vector per minute. For the definite implementation, we used the python XGBoost library and tried a few learning rates and max depths. We came to the conclusion

that a learning rate of 1e-5 was necessary and for the maximal depth, the default of 6 fulfilled the requirements. Although, after examining the business flights, we noticed that a greater depth was a necessary because of the discrepancies in data-set sizes. We opted for a maximal depth of 10 for the business flights. Fortunately, when saving the classifiers, their size were reasonable.

4.2 Experimental Setup for the Models

4.2.1 Data Streams

We will examine how the training and testing data was fed to the models, to be sure that no confusion arises: We opted for an 80/20 split.

Training Data

The flights did not contain the right amount of state vectors to get n following data-points, which is why we had to omit a few. For each flight path X', containing k state vectors with each containing m features, we ignored the first state vector(s) i of each flight, to be able to reshape them into a $\frac{k-i}{n} \times n \cdot m$ matrix 4.1. In other words if $n \nmid k$, we ignored as many state vectors i as necessary, with their timestamps being as early as possible. For the target vector y, we gave up as many rows as the reshaped matrix X has: $\lfloor \frac{k}{n} \rfloor$. Meaning that one flight path gives us multiple predictions to train our model on, since we have multiple points in time as to when to make our prediction. The python code is in the appendix C.2.

$$X'_{k\times m} = \begin{bmatrix} x_{1,1} & \dots & x_{1,m} \\ \vdots & \ddots & \\ x_{k,1} & & x_{k,m} \end{bmatrix} \xrightarrow{reshaping} X_{\lfloor \frac{k}{n} \rfloor \times m\cdot n} = \begin{bmatrix} x_{i,1} & \dots & x_{i,m\cdot n} \\ \vdots & \ddots & \\ x_{\lfloor \frac{k}{n} \rfloor,1} & & x_{\lfloor \frac{k}{n} \rfloor,m\cdot n} \end{bmatrix} \tag{4.1}$$

Test data

To evaluate our model, we tested the accuracy n minutes before the landing. If there was no data-point to evaluate the model n minutes before, we allowed the prediction to be made on data-points up to 5 minutes before the time we originally wanted to assess the accuracy on. We are assuming that the prediction did not change drastically. if we did not proceed in that way, we would not have had enough data-points to evaluate our model and fully assess its capabilities. To dissimulate any confusion, we have included a graphical representation to show the minimal and maximal ranges from where we sampled the state vectors from.

4.2.2 Classifier Experiments

After having set up the data streams, we proceeded with designing the experiments. Our first goal was to prove that we can build an accurate model to predict the destinations of the planes early enough.

We first set up tests to compare the LSTM and GB model to choose with which one of them we would proceed with our Adversarial Attacks. Those tests included testing the accuracy on the 100 and 1000 most flown to airports 30 minutes before landing. We also based our analysis on their confusion matrices to understand the limitations of each model.

Subsequently, we chose the best performing model and to further prove its capabilities, we tested its accuracy on all planes every 10 minutes 2 hours prior landing which then can give us the legitimacy

to attack the model.
Following this, we then investigated how the best classifier judges the importance of each feature, to forge the most effective dissimulation using adversarial attacks.

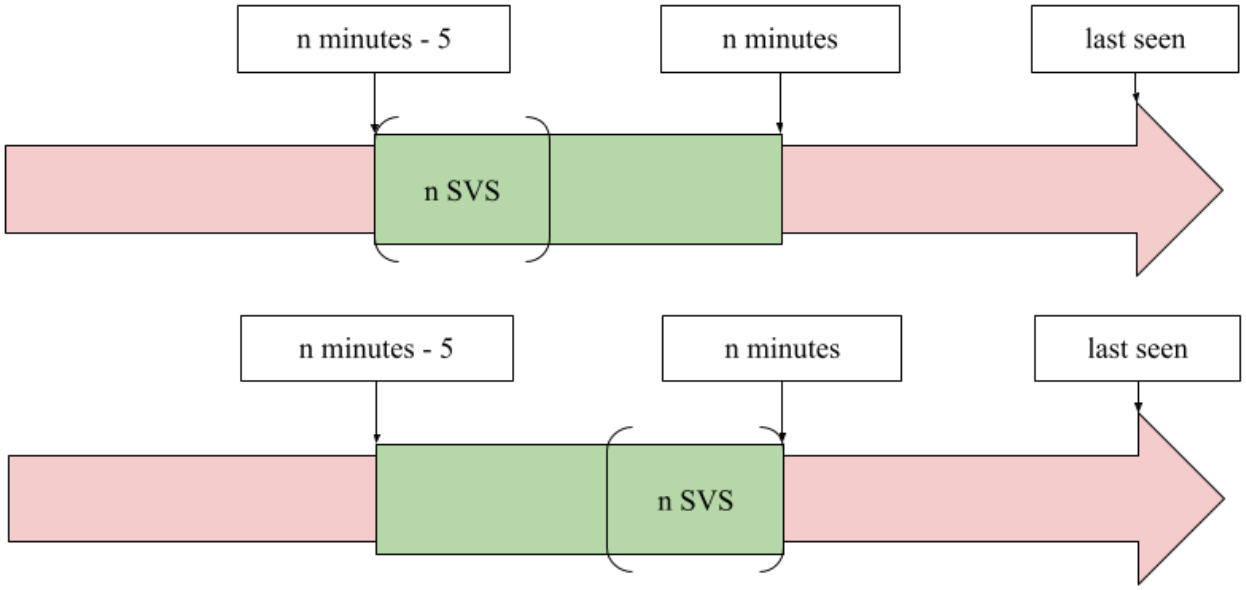


Figure 4.1: testing data timeline graphics

Chapter 5

Adversarial Attacks

We now know that it should be possible to predict where a plane is going to land before it actually does. There are numerous ways to deceive machine learning models to let them wrongfully predict data. We will here take a look at two types of adversarial attacks that could potentially be made on our model. Each time we set an adversarial attack scenario, the attackers capabilities will be mentioned.

5.1 Evasion Attack

An evasion attack is the concept of trying to throw off a model, after it was trained, as if it was already deployed. It is about checking if we can miss-direct its predictions by changing the inputs. In our case it we would be altering our path as much as possible to get a wrong prediction. We could pose this threat as a white- and black-box attack. If we know exactly how the model works we can more easily throw off the prediction. More specifically, if we have information on the airports or clusters that are used, We can try to perfectly retrace a path that would typically fly to one cluster and change direction only at the very end. For the most effective attack and to not lose too much time altering our path to imposture another flight, we can choose to head towards one destination that is close to actual destination.Even if we do not know how the clusters were built or which flights were chosen for the training data, we can make educated guesses by looking on websites that are publicly accessible such as the "ourairports" website[6] where over 70,000 airports are referenced including their coordinates.
We did not perform evasion attacks on our model because we concluded that solely delaying the prediction, would not outweigh the cost in energy and time that would be induced by this attempt.

5.2 Poisoning Attack

A more viable option would be altering the database with a poisoning attack which assumes that we can inject new flights, which does not seem too out of reach, as we have discussed in the related work section. Subsequently, there were a few ways to misdirect the model. A poisoning attack can be done as a white box or black box attack. Just as we mentioned in the evasion attack section, it would not make much of a difference, since the data on where airports are located, is publicly available. Although, if you know how the clusters have been set-up, this would give you more insight, as to how to falsify the data as effectively as possible. We created two data-sets to test our model: a naive poisoned and a targeted poisoned training data-set and then compared how both attacks would affect the prediction on the test data.

5.3 Experimental Setup for the Adversarial Attacks

5.3.1 Naive Poisoning Attack on our Model

To test the stability of the model against such attacks, we created a naive poisoned training data-set and then monitored how the model's performance changed. Naively injected, in our case, is the idea of multiplying all the flights that went to one random destination y_i, without the attacker having a specific target in mind. We solely wanted to test how the model's accuracy changed on the test set, after a portion of the training data-set had been fabricated. We actively chose to sample flights from only one cluster: if we randomly duplicated all the possible flights, we would have obtained the same accuracy and the model would not have been affected. The naive poisoning algorithm is given as a pseudo code here 1 and in the appendix in python C.4. The algorithm randomly chooses a target destination y_i and samples n times from the flights $\in X$ that are supposed to land at y_i. To make the relationship between the added flights and the original size of the data-set clearer, we will test for the amount of poisoned flights $\frac{|P|}{|P|+|X|}\%$,
We now can vary the amount of poisoned flights by changing n and compare the new scores with the baseline scores and determine the stability of our classification.

- X being the training data, y the true destination
- $f(X) = y$, y being the destinations of each flight X
- n being the number of poisoned flights
- $|P| = n$ a bag of the the fabricated flights
- X_{new} the new bag of flights with duplicated and original flights

Algorithm 1 Pseudo-code Naive Poisoning

```
y_i ∈ y                                   ▷ random destination y_i
while n ≠ 0 do
    X_k, f(X_k) = y_i                     ▷ X_k sampled flight from flights that land at y_i
    P ← X_k
    n ← n − 1
end while
X_new ← X ∪ P
```

5.3.2 Targeted Poisoning Attack

When using the word targeted, we are referring to the desired destination of the attacker. Supposed that an attacker wants to fly to a destination y_i, in an attempt to miss-direct the model, he will inject an amount n of flights, that landed to the k closest clusters to y_i.
To achieve the most promising results, we found the k closest possible destinations y_j, in a radius ϵ and $y_j \in D, j \neq i$. Then, we randomly sampled n flights that flew to any of the destinations in D. This implies that we will sample the data-points from all the flights that landed close enough to y_i. Finally, we looked at how many flights were then wrongfully labeled and how many n flights it took to achieve this. This attack should be more effective, since the confusion of the destination could happen rather quickly if a noticeable amount of flights land close to the desired destination. The classifier may over-fit on the destinations D around y_i.

Algorithm 2 Pseudo-code Targeted Poisoning

$y_i \in y$ ▷ y_i being the attackers desired ending location
  $y_j \in y, j \neq i$ **do**
    while $k \geq 0$ **do**
      if $||y_j - y_i|| \leq \epsilon$ **then** ▷ ϵ smallest radius that still includes k clusters
        $k \leftarrow k - 1$
        $D \leftarrow y_j$ ▷ $|D| = k$, k possible destinations
      end if
    end while
  end
  while $n \geq 0$ **do**
    $X_j \leftarrow X$
    if $f(X_j) \in D$ **then**
      $P \leftarrow X_j$
      $n \leftarrow n - 1$
    end if
  end while
  $X_{new} \leftarrow X \cup P$

Chapter 6

Results & Discussion

6.1 Classifiers

We found hints on how to work with an LSTM to predict the destination. We also tested how the gradient booster model performs, depending on how early we try to predict the final cluster. The GB classifier performed better than the LSTM which led us to further investigate its capabilities on predicting the destination as early as possible. We will go over the first results of the LSTM to justify our choice to move forward with the GB model.

6.1.1 LSTM

After thoroughly testing different hidden state sizes with various number of airports, we deduced that an LSTM could be a good approach. Nonetheless, considering that we are not working with enough data-points that follow each-other, defeats the purpose of an LSTM. For the top 100 airports, our best score was 0.51, 30 minutes before landing, obtained with 128 hidden layers. To further analyse the performance of our model, a confusion matrix will be provided in the appendix that includes the results with 128 B.1b and 32 hidden states B.1a. The logarithmic function was applied to the values, to expose the miss-classifications better.
We can observe that the wrong predictions are coherent, since the airports in the square are all in the vicinity of each other geographically. More specifically, the visible square represents a few airports in England, their names can be read on the x axis.
While using the same data-set, we made our first tests with a gradient booster model, which gave us an accuracy of 0.73 which explains our choice to move forward with this classifier, rather than the LSTM.

6.1.2 Gradient Booster

Performance Analysis

As mentioned in the implementation of the gradient booster classifier 4.1.2, we went into a more detailed analysis of its performance, since the first results seemed more promising than the LSTM. For each type of flight, we analysed how early we could predict where it would land, in steps of 10 minutes 6.1. For completion of data, we also added the matching table 6.1. The confusion matrix to each owner is also provided in the Appendix for reference B.2.
As you can observe, we achieved a high accuracy on the predictions of the flights, which therefore gave us a great opportunity to misdirect the classifiers in the adversarial machine learning section.

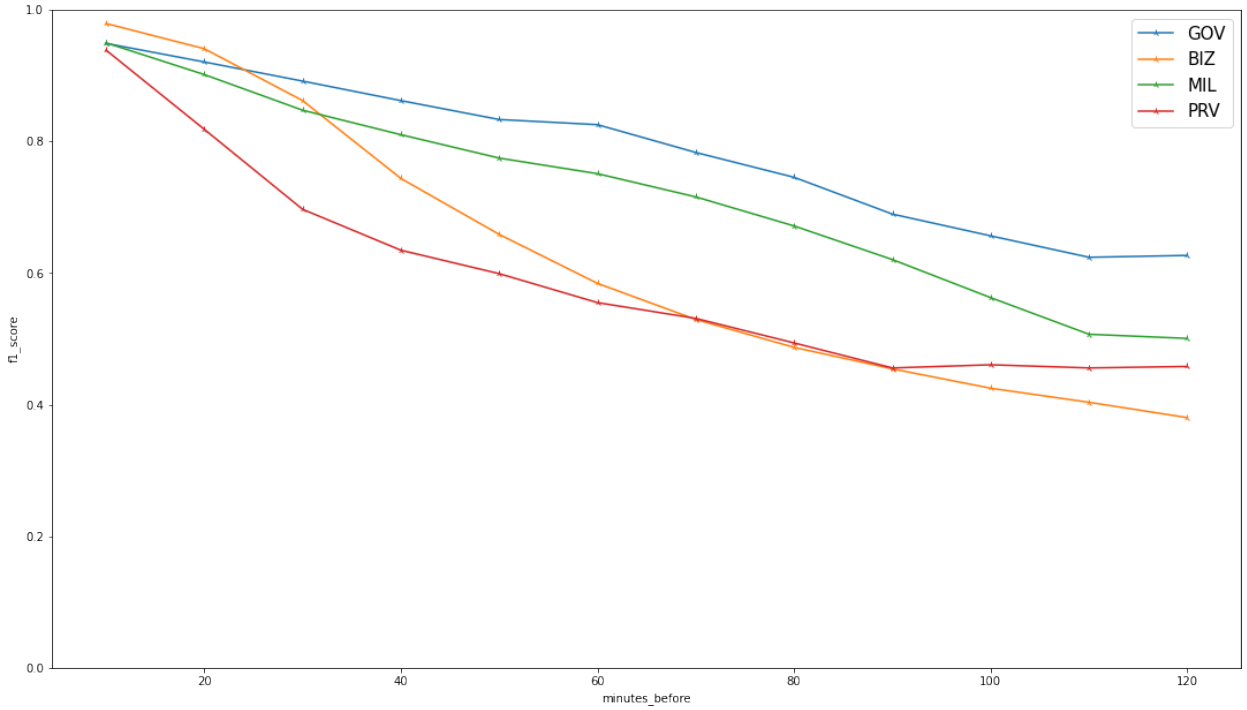


Figure 6.1: Gradient booster performances

We can note that some planes, that were flown by different owner types, had better predictions than others at different times. We think that there are a few reasons for that: The state/government flights might have been easier to predict because of the size of the aircraft subset. We are not referring to the number of flights that were conducted by the owner type, but the number of planes that flew those flights. Having less planes to predict, means less discrepancies in behavior of all kinds: more consistencies in speeds, position, climbing rates. This would also explain why business and private planes are the least accurate 30 minutes and more before landing. We can see in the table 3.1 that both of those categories have more planes to classify than military and government aircraft's.
To explain why the prediction of private flights is the least accurate right before landing, we suggested that it is due to them including more trajectories that simply circle around the airport and then land where they departed from. We have two flights in the Appendix that underline this matter A.4b, A.4a. This makes them difficult to predict, especially when the private flights also have trajectories that simply go from one cluster to another. Therefore, solely assuming that a plane would land where it started from was not enough either to classify the private flights.

Feature Importance

The feature importance of each variable, given by the XGBoost module's function, can be seen in this figure 6.2. The order of the recorded data-points is defined by the number in the feature names: From 0 to 2, with 2 being the last data-point of the 3. As you can observe, the last data-points are the most important ones, more specifically the heading, longitude and latitude. Nevertheless, the other features are also necessary: The combination of the first and last data-points, that are

minutes before	GOV	BIZ	MIL	PRV
10	0.95	0.98	0.95	0.94
20	0.92	0.94	0.9	0.82
30	0.89	0.86	0.85	0.7
40	0.86	0.74	0.81	0.63
50	0.83	0.66	0.77	0.6
60	0.83	0.58	0.75	0.56
70	0.78	0.53	0.72	0.53
80	0.75	0.49	0.67	0.49
90	0.69	0.45	0.62	0.46
100	0.66	0.42	0.56	0.46
110	0.62	0.4	0.51	0.46
120	0.63	0.38	0.5	0.46

Table 6.1: f1 scores by aircraft owner type n minutes before landing

here 2 minutes apart, can apparently give us the biggest insight as to where the plane is going to land. This makes sense, since they can give one the greatest insight about the flight's changes. The altitude's feature importance might not be higher, because it only matters towards the end of the flight: Typically, a plane is at a constant altitude and only at the very end will it descend to it's destined airport. The landing phase is only a short period of the full flight which explains the lack of relevance of the altitude for the prediction.

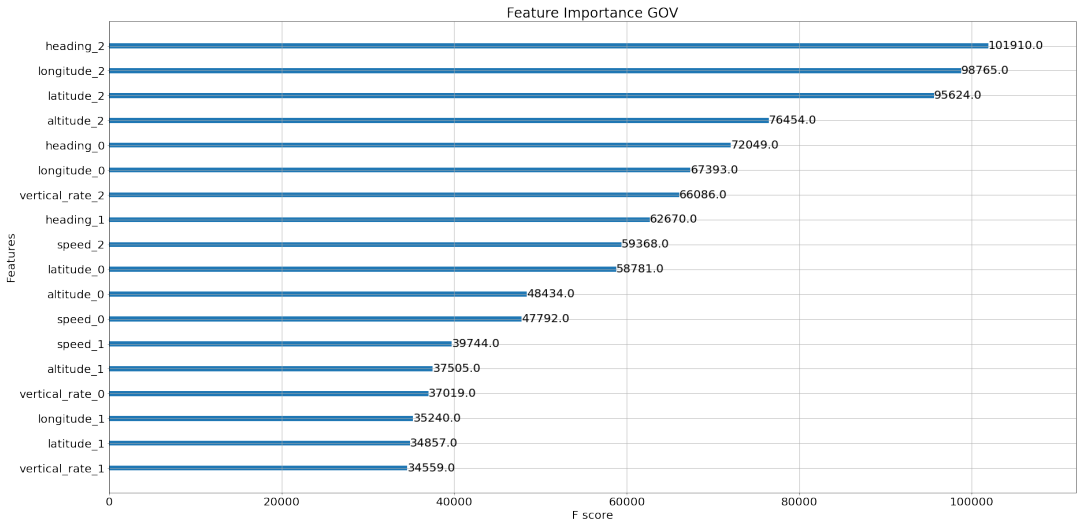


Figure 6.2: feature importance of State/Government flights

6.2 Adversarial Attacks

After having verified that our model performs well, we now are going to analyse the stability of the gradient booster classifier, thanks to the adversarial machine learning attacks discussed prior. We did not proceed with the attacks on all owner types because we are confident that the results will be consistent across the board. Especially since we chose to attack the most stable classifier of all four: The Government/State flights 6.1.

6.2.1 Naive Poisoning

Here we have the results for a progressively poisoned data-set for Government/State owned Aircraft's 6.3. The matching table has been added to the appendix B.1. As expected, we can observe that the more duplicated flights we inject in the training data-set, the worse the model performs. The declining accuracy on the graph of the test data-set's performance can be explained through the increasing amount of fake flights. When we have more than 90% of flights that are bogus, the overall accuracy remains surprisingly high. It is due to how the gradient booster classifier is built: it creates trees that will only miss-classify flights in a specific region, more specifically: around the randomly chosen cluster. We need a substantial amount of flights to mask the real destinations. Moreover, the perturbation is not significant compared to the amount of flights that were added to get this result. It is unachievable to inject that many flights in the database without getting detected.
In the Appendix, we can observe the confusion matrices that showcase how the poisoning affects the miss-classifications 30 minutes before landing B.4. On the most poisoned databases we can observe which cluster was sampled: n°136. We can tell because of the horizontal line that suggest an over-fitting on one specific cluster.

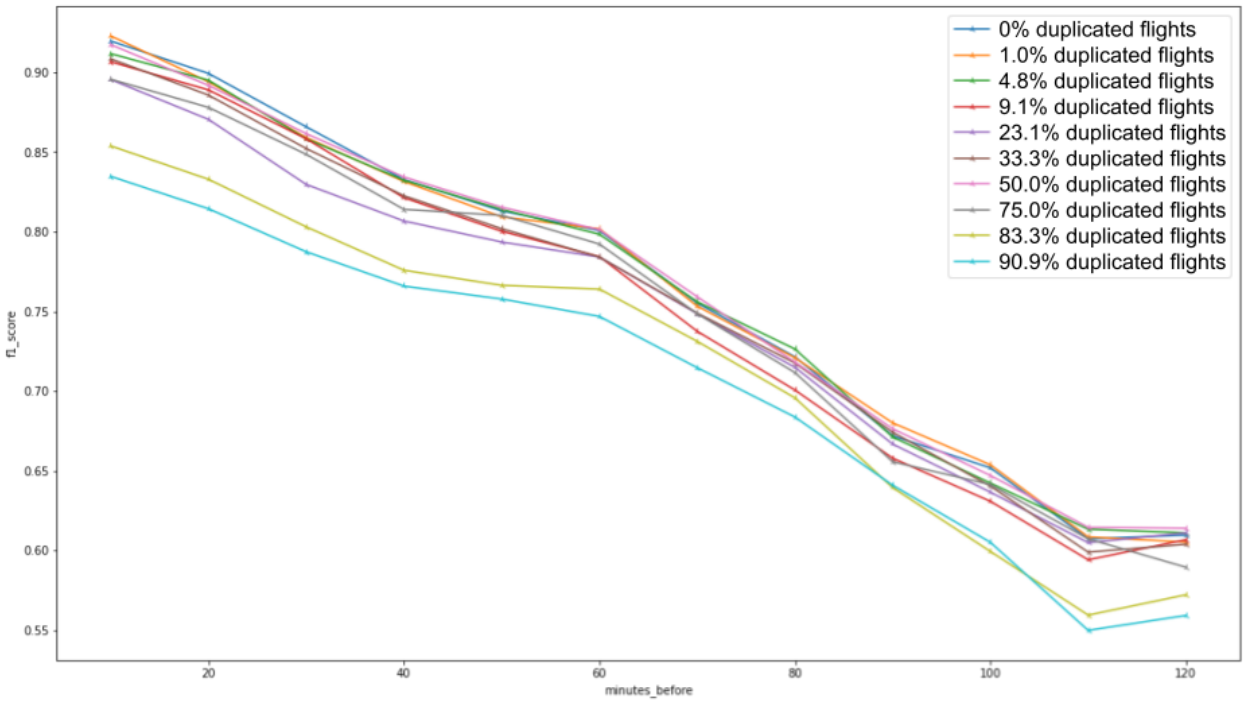


Figure 6.3: Naive poisoning on GOV flights

6.2.2 Targeted Poisoning

For our targeted poisoning attack, we chose $k = 5$ closest clusters to cluster 36, $D = 93, 88, 65, 132, 119$ to get the results in 6.4. As a visual aid to understand how the clusters were set-up, we provided a figure in the Appendix B.5. We chose to poison this cluster specifically, because of its high accuracy when it is not poisoned. This can be confirmed when looking at the curve when 0 duplicated flights were added. We also made sure to have enough clusters around target cluster 36, to have the opportunity to duplicate flights that landed at clusters D. There are exactly 625 flights that landed in cluster 36 in the training data-set, which can help one understand how many flights were added compared to how many flew to the cluster.
We can observe in the plot 6.4 that the targeted attack is much more effective than the naive poisoning attack. When too many flights that landed close to n°36 were duplicated, the gradient booster classifier cannot separate the trajectories anymore. In our case, with 2000 added flights, the classifier's accuracy already dropped to half of its original value. We can clearly attack this model to misguide its prediction with a more reasonable amounts of injected flights compared to the naive poisoning method. The abrupt changes between the curves can be explained by the miss-classification of multiple flights at once: the trajectories of the flights that resemble each-other will trivially be predicted erroneously at once.

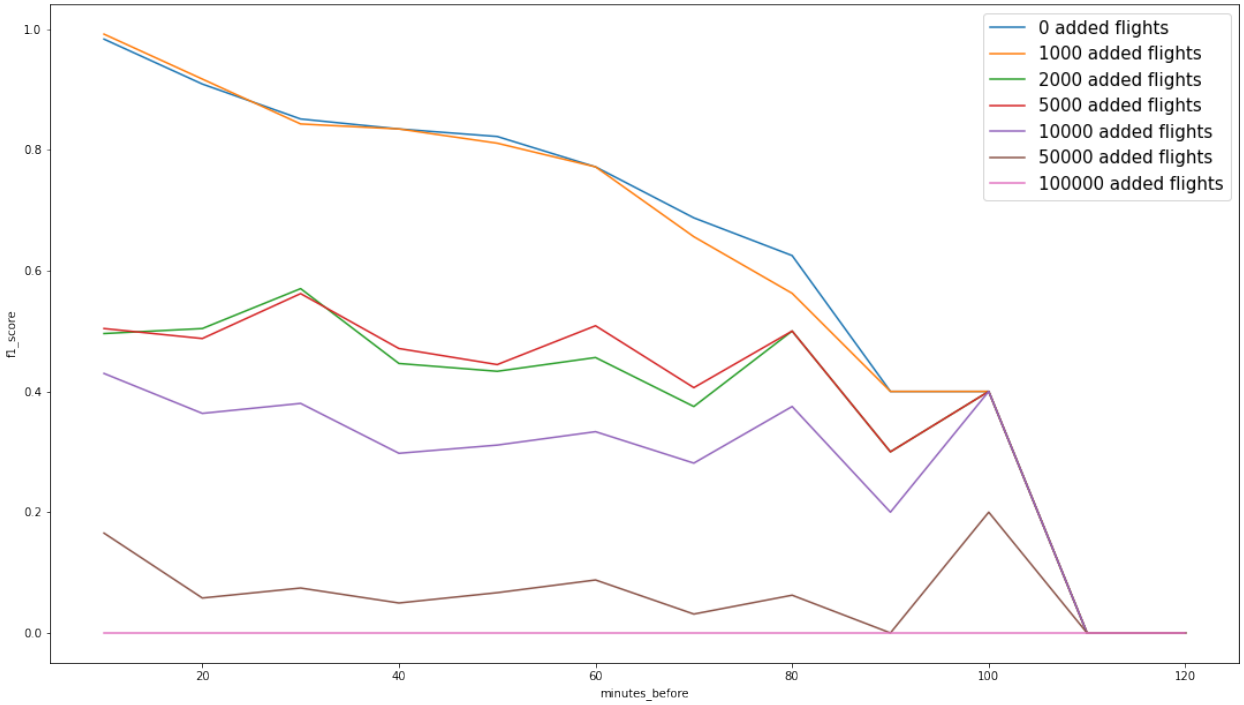


Figure 6.4: Targeted poisoning on cluster 36 in GOV flights

Chapter 7

Conclusion

7.1 Models

The LSTM model was not as effective as we hoped right away. This might be explained by the network not being deep enough to learn the flight paths. We were using three data-points following each-other, which meant we could only have 3 LSTM cells attached to one-other. We did not take more because we thought that if we did, it would defeat the purpose of a rapid prediction, meaning shorter than 2 minutes. Thankfully the gradient booster model gave us the necessary results to predict the aircraft's final destination without needing the departure airports location.
We can say that we successfully created a model that accurately predicts where a plane will land in the near future.

7.2 Adversarial Attacks

Although our model performs well predicting the destination of a plane before landing, we have showed that it can be miss-directed with a reasonable amount of flights that were premeditatedly selected and injected in an open-access, crowd-sourced database. Fortunately, it is impossible to mislead the model on all of its cluster with a naive poisoning because of how the decision trees in a gradient boosting model are built.
We have successfully found an approach as to how to attack the models predictions.

7.3 Live Destination Prediction

As a proof of concept, a live interactive map was created that allows one to identify where planes are forecast to land. From the 100,000 planes that we selected and trained our model on, about 600 of them are typically live in the sky at the same time, depending on the time of the day. The variability is quite high, which is due to flights mainly being held daytime in central Europe and America. Here is a figure of what the maps looks like 7.1. A demonstration is available on youtube here to showcase the predictions that the classifier has made. As you can observe, the model is predicting single airports and not the clusters because it was created before grouping them. The accuracy of the live predictions is then affected, because it is more difficult to predict an airport directly compared to a set of them, thankfully it still displays sensible results. The classifier that was used, gave us an accuracy of 0.6 on our test data-set, 30 minutes before landing on the top 1000 most flown to airports which is remarkable considering some airports are only a few kilometers

away from each-other.
We can say that we successfully created a live interactive map to predict aircraft destinations.

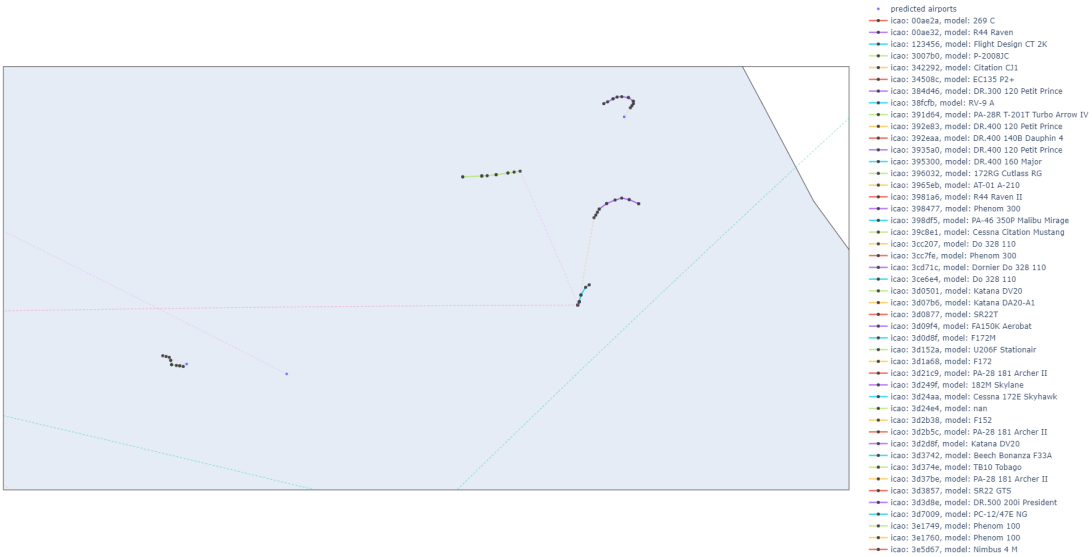


Figure 7.1: Landing Prediction

7.4 Future Work

To conclude the Thesis, we will go over future work that can be done to further improve the prediction and in the matter of adversarial attacks, we will propose extensions, to elaborate the threats.

7.4.1 Future work on the Model

In future studies, we could address the idea of clustering sets of plane identifiers having similar flight paths. As a result, we could create a classifier for each of the plane identifiers clusters, which could lift the accuracy significantly. This is an attempt to better the heuristic approach of clustering the flights into the four categories that we originally chose. From what we could observe in the figure 6.1, the clustering has a big impact: there is a visible discrepancy between the owner-type's accuracy's. This might also limit the transponders that deteriorate the data-set: With a bit of luck, the miss-behaving plane's ADS-B Out transponders would end up in a separate cluster, which would make the learning of accurate routes easier. We did observe that some flights had altitudes that were clearly noise and instead of ignoring them, we now may be able to cluster them together and still make sense of the data.
To further prove and extend the capabilities of the predictions, we could forecast the destination of specific planes: To build on of the idea of the ElonJet account [15], that merely posts the flights that were taken, after the plane lands, we could try to predict where a plane is going to land before it does and when the model is confident enough with its prediction, tweet about it, before the Twitter bot does. This could be achieved using transfer learning, which would help with the

specificity of non-commercial planes often landing at the same airports, while at the same time, fight against the lack of data that comes with using the trajectories of only one plane.

7.4.2 Future Work on Adversarial Attacks

As with any attack and defense scenario, we can raise the attackers capabilities to be as white-box as possible or we can try to make the attacks more complex and precise. If the database operator can easily spot and reject flights that seem to have been duplicated, or simply lightly altered, we could assemble a generative adversarial model and create new flights that resemble as much as possible to the ones that we can already get from the database. In other words: Supposed duplicated flights may be detected by the database operator too easily, even if we induce some noise, or minor differences to make them seem different, one attack could be to create GAN, that creates flights that simulate trajectories by using already existing flights as training data. It could be insightful to see how well we can tell them apart afterwards, that is with how much confidence can the network tell if a generated flight is not real. We could use this as a metric to tell if the database curator could spot if a flight created by the GAN has been fabricated.

Bibliography

[1] AB, F. Flightradar24. `https://flightradar24.com`.

[2] Chakraborty, A., Alam, M., Dey, V., Chattopadhyay, A., and Mukhopadhyay, D. Adversarial attacks and defences: A survey. *CoRR abs/1810.00069* (2018), 4–7.

[3] Chen, T., and Guestrin, C. XGBoost. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (aug 2016), ACM.

[4] Costin, A., and Francillon, A. Ghost in the air (traffic): On insecurity of ads-b protocol and practical attacks on ads-b devices. *black hat USA 1* (2012), 1–12.

[5] Marc Jourdan, Karolis Martinkus, D. R. Flight destination deanonymization by an online attacker.

[6] Megginson, D. Our airports. `https://ourairports.com/data/`.

[7] Schäfer, M., Strohmeier, M., Lenders, V., Martinovic, I., and Wilhelm, M. Bringing up opensky: A large-scale ads-b sensor network for research. `http://opensky-network.org`, 2014.

[8] Siva Kumar, R. S., Nyström, M., Lambert, J., Marshall, A., Goertzel, M., Comissoneru, A., Swann, M., and Xia, S. Adversarial machine learning-industry perspectives. In *2020 IEEE Security and Privacy Workshops (SPW)* (2020), pp. 69–75.

[9] Staudemeyer, R. C., and Morris, E. R. Understanding lstm – a tutorial into long short-term memory recurrent neural networks, 2019.

[10] Strohmeier, M., Lenders, V., and Martinovic, I. On the security of the automatic dependent surveillance-broadcast protocol, 2015.

[11] Strohmeier, M., Martinovic, I., and Lenders, V. *Securing the Air–Ground Link in Aviation*. Springer International Publishing, Cham, 2020, pp. 131–154.

[12] Strohmeier, M., Schäfer, M., Lenders, V., and Martinovic, I. Realities and challenges of nextgen air traffic management: the case of ads-b. *IEEE Communications Magazine 52*, 5 (2014), 111–118.

[13] Strohmeier, M., Smith, M., Lenders, V., and Martinovic, I. The real first class? inferring confidential corporate mergers and government relations from air traffic communication. In *2018 IEEE European Symposium on Security and Privacy (EuroSP)* (2018), pp. 107–121.

[14] Strohmeier, M., Smith, M., Moser, D., Schäfer, M., Lenders, V., and Martinovic, I. Utilizing air traffic communications for osint on state and government aircraft. In *2018 10th International Conference on Cyber Conflict (CyCon)* (2018), pp. 299–320.

[15] Sweeney, J. Elonjet. `https://twitter.com/ElonJet`.

[16] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I. Attention is all you need. *CoRR abs/1706.03762* (2017).

[17] Zeng, A., Chen, M., Zhang, L., and Xu, Q. Are transformers effective for time series forecasting? *arXiv preprint arXiv:2205.13504* (2022).

[18] Zhao, Z., Zeng, W., Zhibin, Q., Chen, M., and Yang, Z. Aircraft trajectory prediction using deep long short-term memory networks. `https://www.researchgate.net/publication/334384222_Aircraft_Trajectory_Prediction_Using_Deep_Long_Short-Term_Memory_Networks`, 07 2019.

Appendix A

Data Appendix

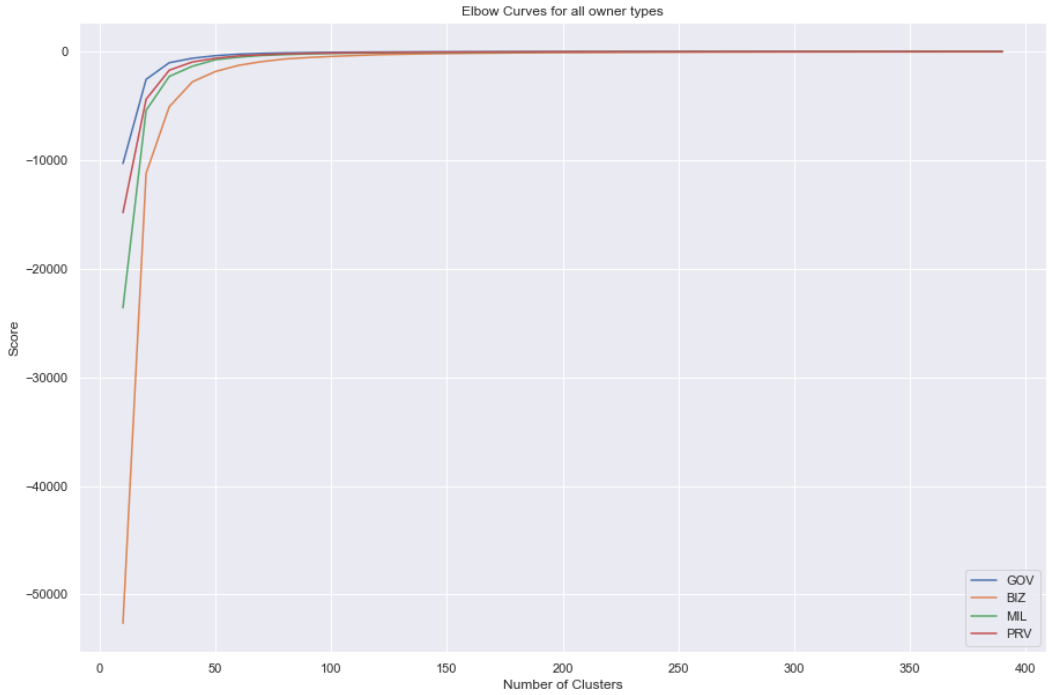


Figure A.1: Elbow curves for all owner types

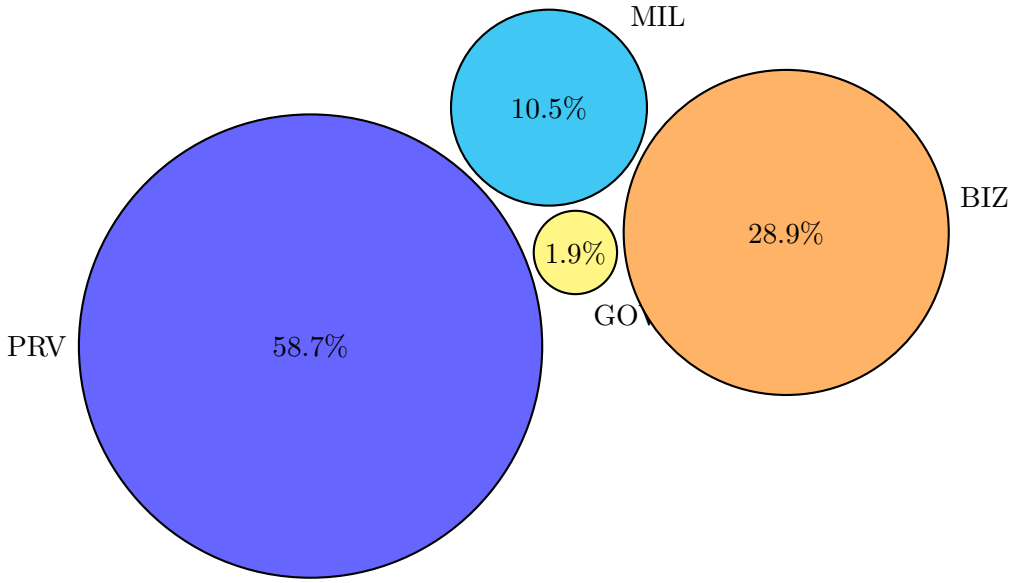


Figure A.2: Proportion of aircraft for each owner type

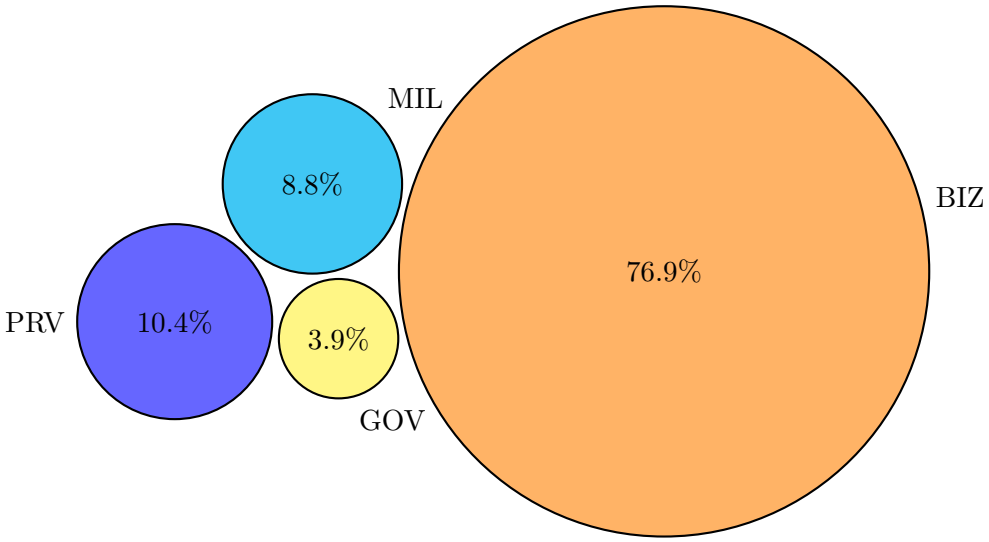
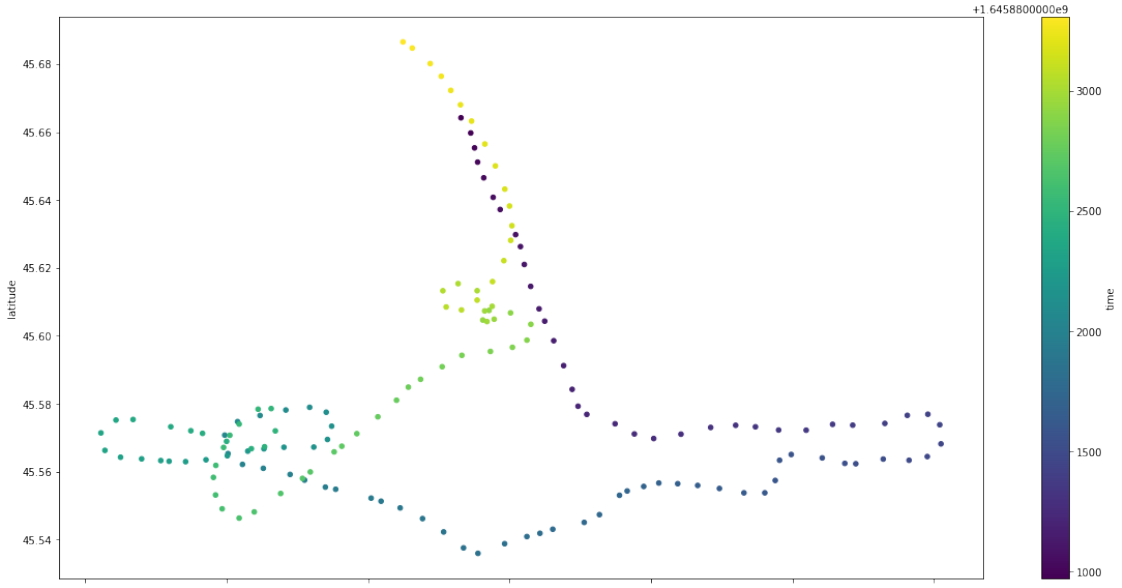
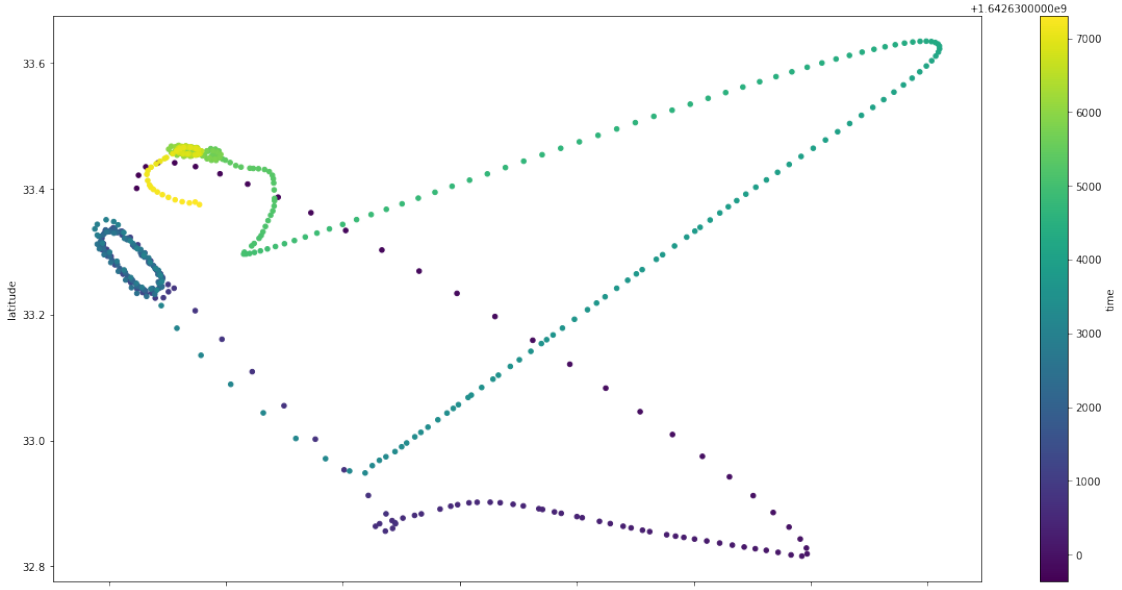


Figure A.3: Proportion of flights recorded for each owner type

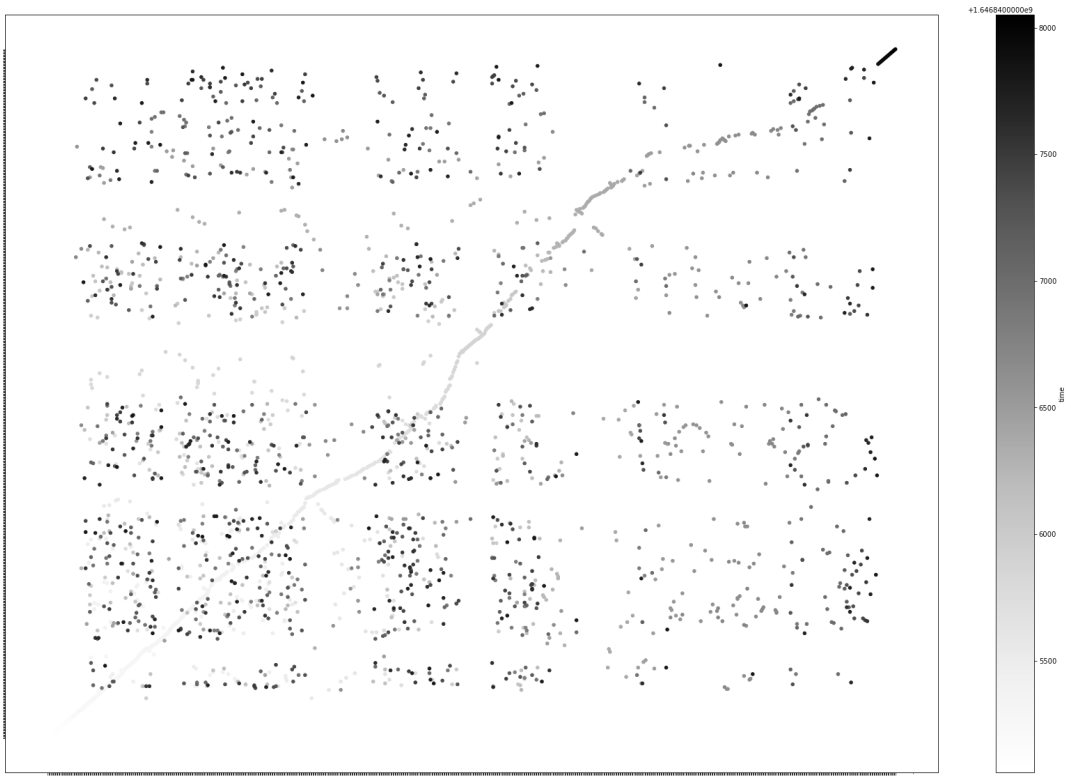


(a) Cessna flight

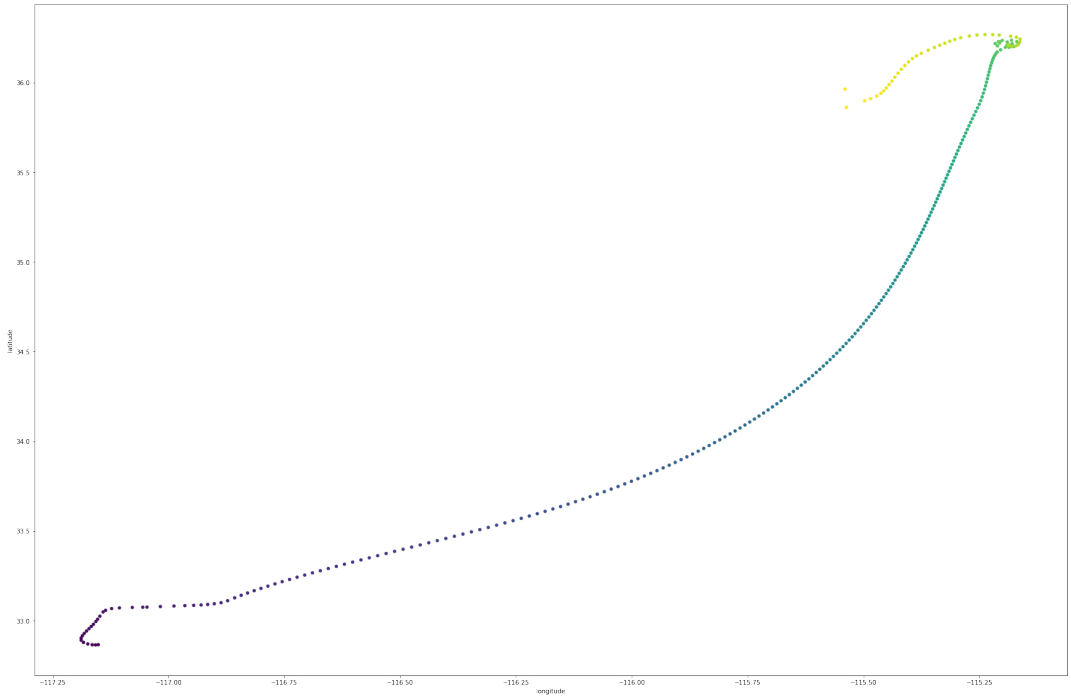


(b) Antonov Flight

Figure A.4: two private flights



(a) noisy data

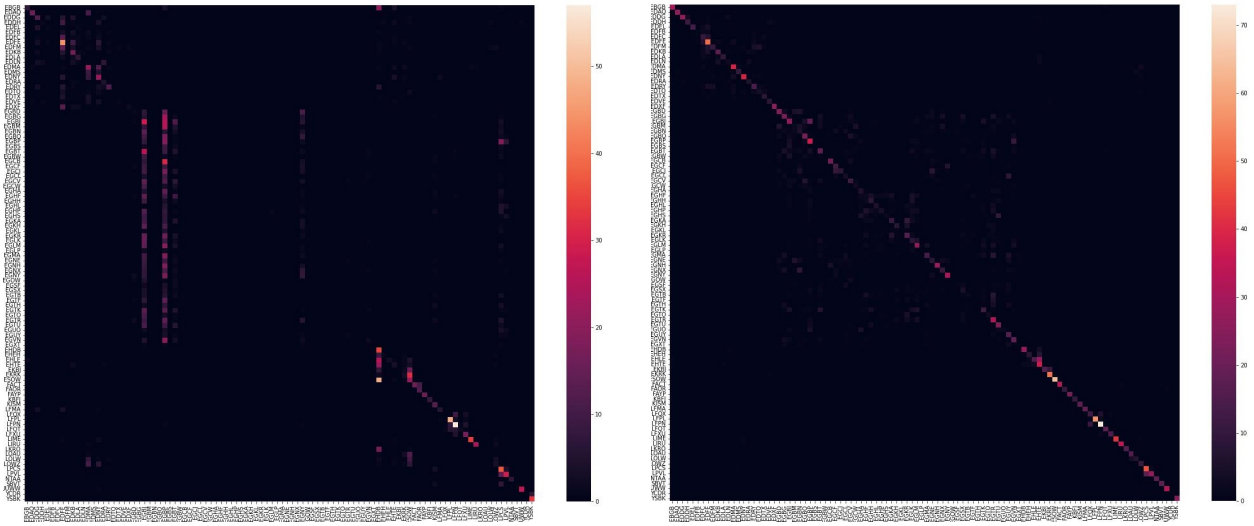


(b) de-noised, re-sampled, interpolated Flight

Figure A.5: data cleaning

Appendix B

Results Appendix



(a) 32 hidden weights

(b) 128 hidden weights

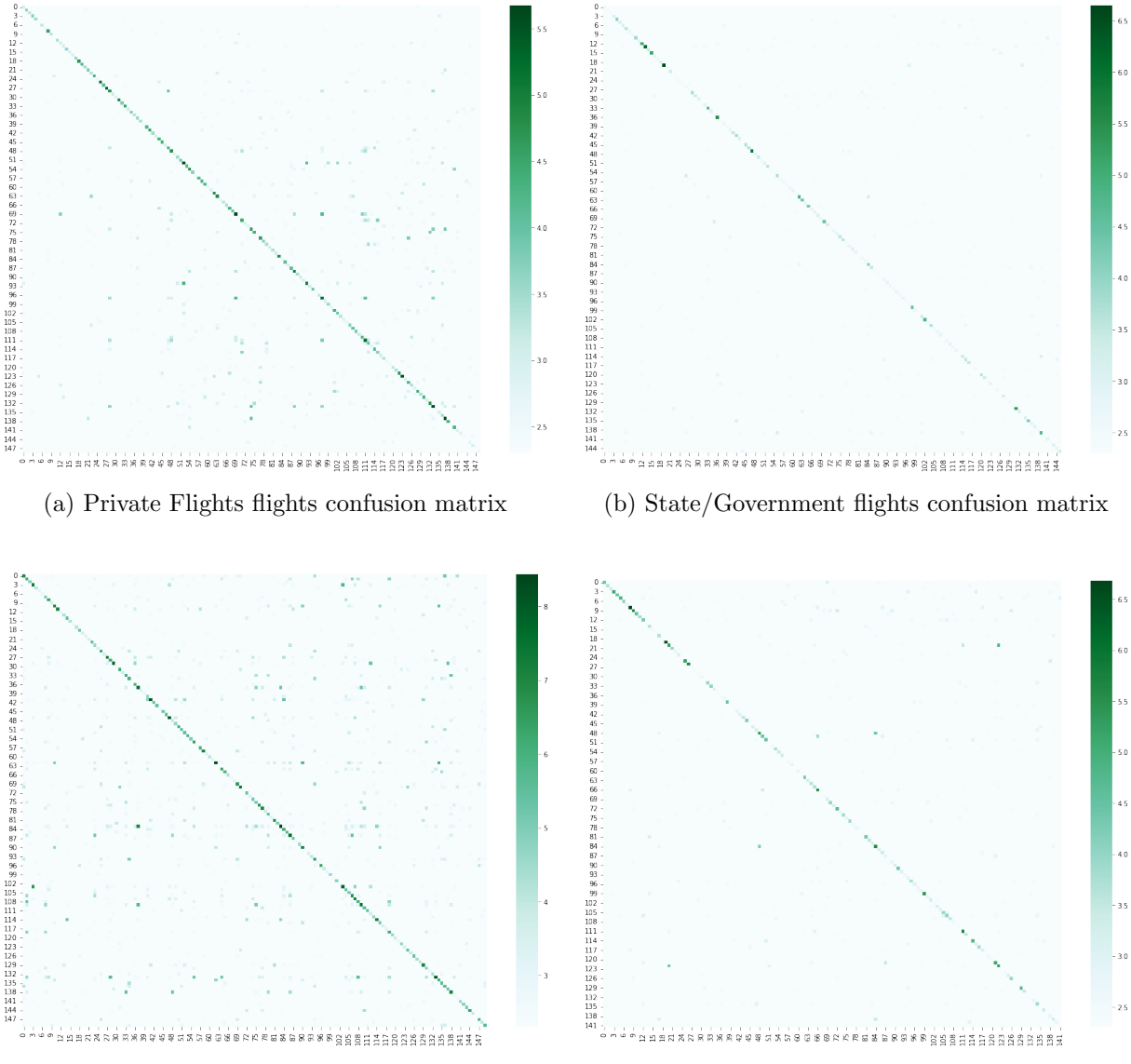
Figure B.1: LSTM confusion matrices 30 minutes before landing

minutes before	amount of poisoned flights									
	0.0	0.01	0.048	0.091	0.231	0.333	0.5	0.75	0.833	0.909
10	0.92	0.92	0.91	0.91	0.9	0.91	0.92	0.9	0.85	0.83
20	0.9	0.89	0.9	0.89	0.87	0.89	0.89	0.88	0.83	0.81
30	0.87	0.86	0.86	0.86	0.83	0.85	0.86	0.85	0.8	0.79
40	0.83	0.83	0.83	0.82	0.81	0.82	0.83	0.81	0.78	0.77
50	0.81	0.81	0.81	0.8	0.79	0.8	0.82	0.81	0.77	0.76
60	0.8	0.8	0.8	0.78	0.78	0.78	0.8	0.79	0.76	0.75
70	0.76	0.75	0.76	0.74	0.75	0.75	0.76	0.75	0.73	0.71
80	0.72	0.72	0.73	0.7	0.71	0.72	0.72	0.71	0.7	0.68
90	0.67	0.68	0.67	0.66	0.67	0.67	0.68	0.66	0.64	0.64
100	0.65	0.65	0.64	0.63	0.64	0.64	0.65	0.64	0.6	0.6
110	0.61	0.61	0.61	0.59	0.6	0.6	0.61	0.61	0.56	0.55
120	0.61	0.61	0.61	0.61	0.61	0.6	0.61	0.59	0.57	0.56

Table B.1: f1 scores by aircraft owner type n minutes before landing

minutes before	flights added						
	0	1000	2000	5000	10000	50000	100000
10	0.98	0.99	0.5	0.5	0.43	0.17	0.0
20	0.91	0.92	0.5	0.49	0.36	0.06	0.0
30	0.85	0.84	0.57	0.56	0.38	0.07	0.0
40	0.83	0.83	0.45	0.47	0.3	0.05	0.0
50	0.82	0.81	0.43	0.44	0.31	0.07	0.0
60	0.77	0.77	0.46	0.51	0.33	0.09	0.0
70	0.69	0.66	0.38	0.41	0.28	0.03	0.0
80	0.62	0.56	0.5	0.5	0.38	0.06	0.0
90	0.4	0.4	0.3	0.3	0.2	0.0	0.0
100	0.4	0.4	0.4	0.4	0.4	0.2	0.0
110	0.0	0.0	0.0	0.0	0.0	0.0	0.0
120	0.0	0.0	0.0	0.0	0.0	0.0	0.0

Table B.2: f1 scores by amount of duplicated flights added n minutes before landing on cluster 36



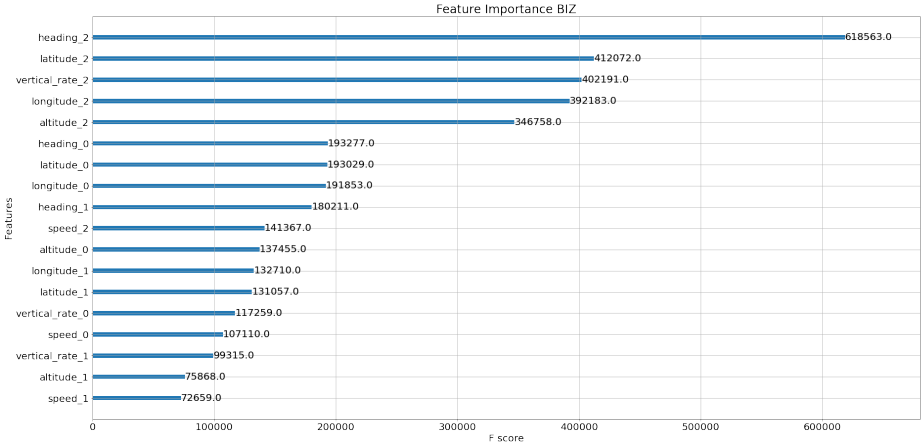
(a) Private Flights flights confusion matrix

(b) State/Government flights confusion matrix

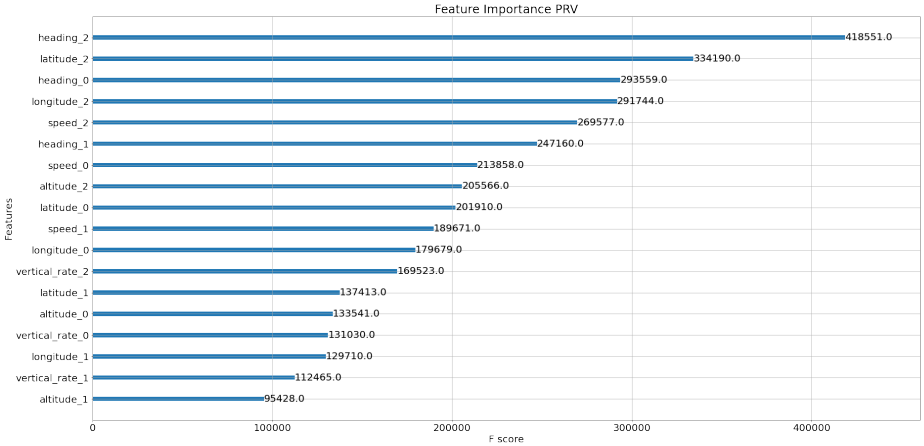
(c) Business flights confusion matrix

(d) Military flights confusion matrix

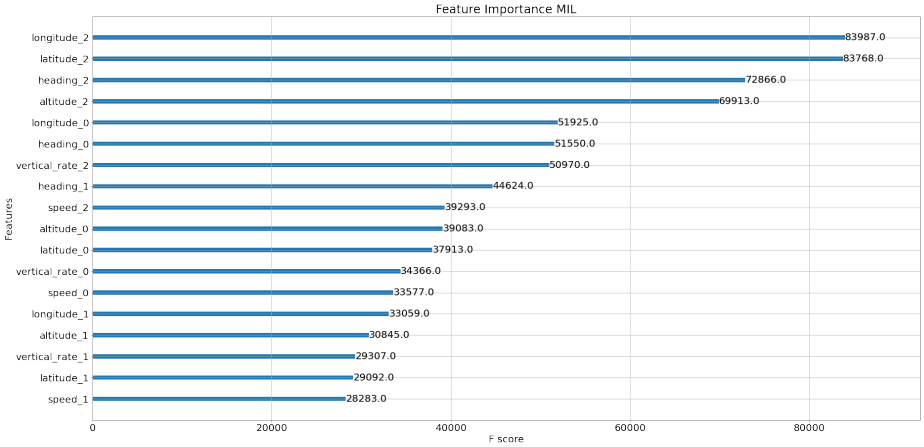
Figure B.2: gradient booster confusion matrices 30 minutes before landing



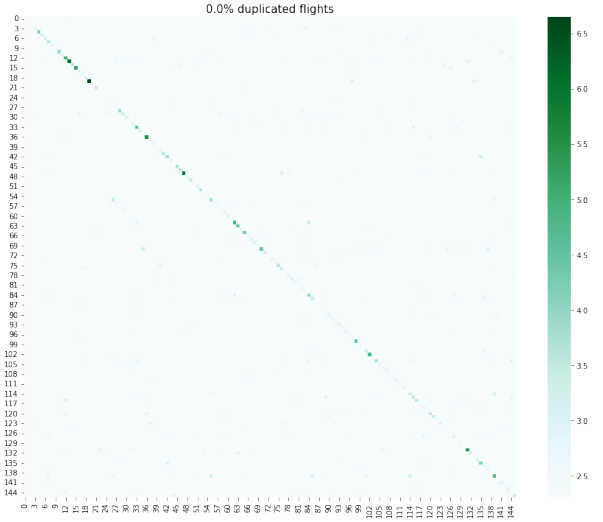
(a) feature importance of business flights



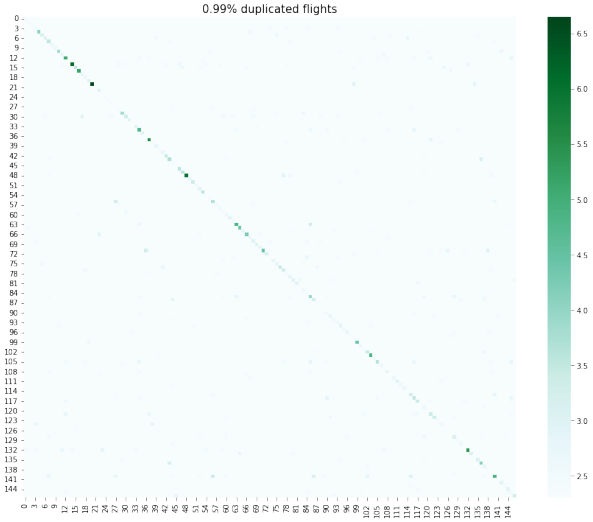
(b) feature importance of private flights



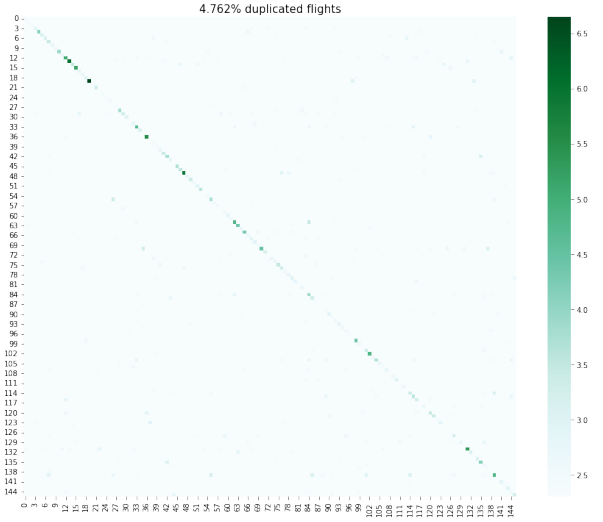
(c) feature importance of military flights



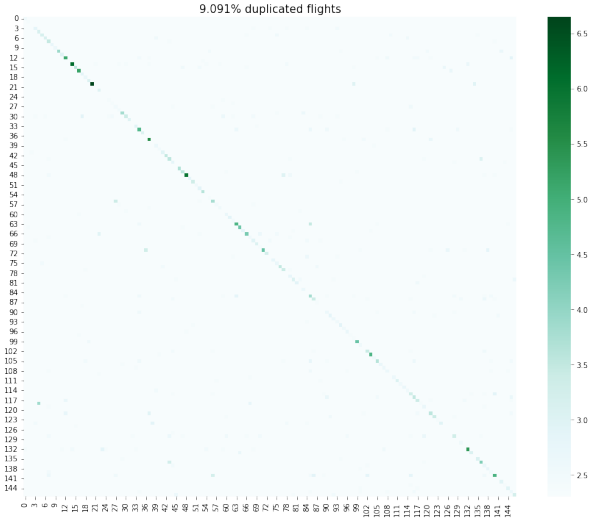
(a) Naive Poisoning Confusion Matrix



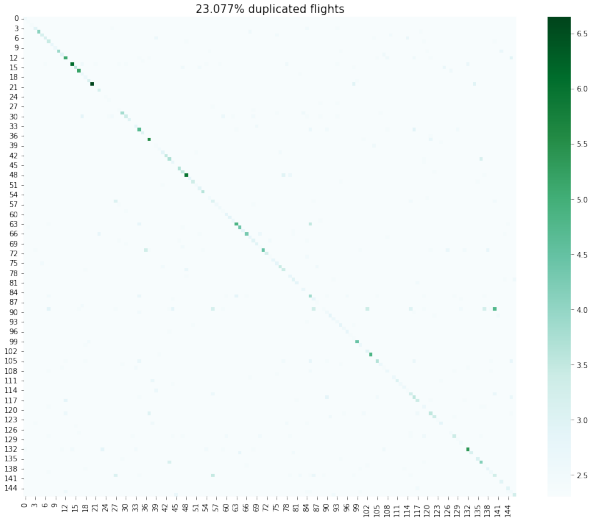
(b) Naive Poisoning Confusion Matrix



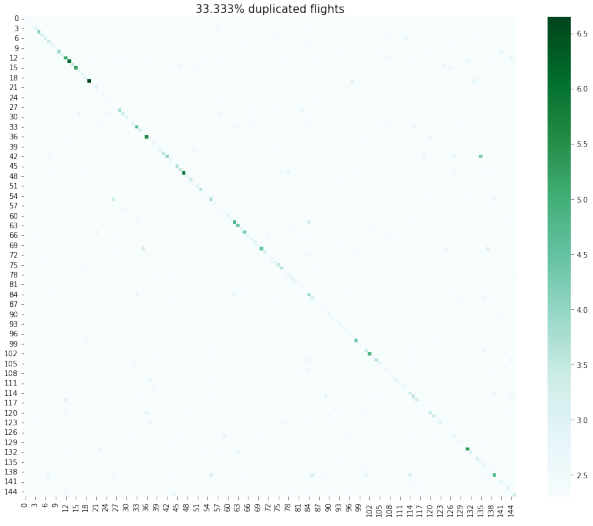
(c) Naive Poisoning Confusion Matrix



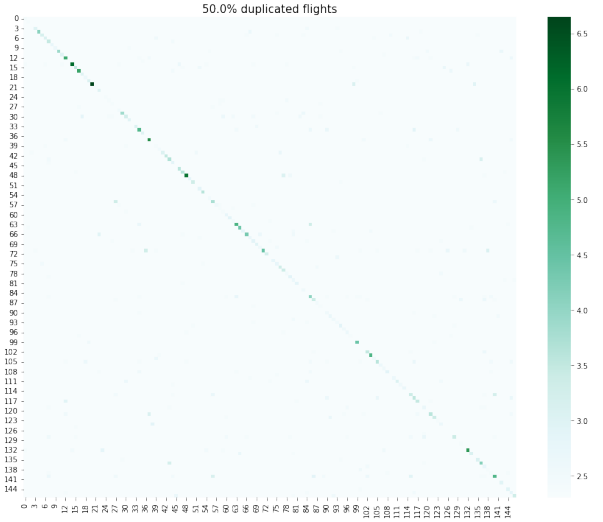
(d) Naive Poisoning Confusion Matrix



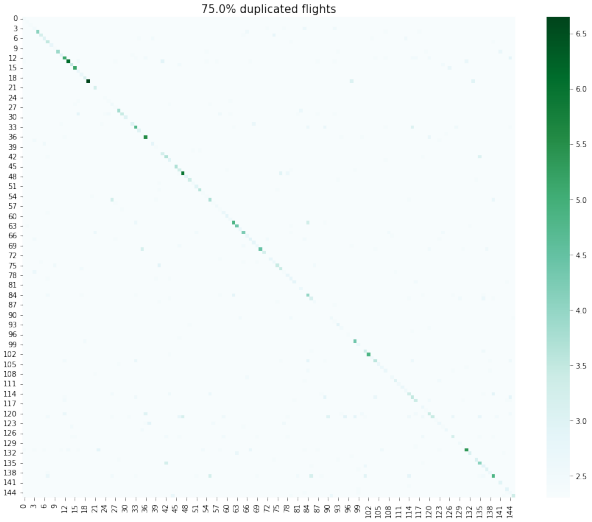
(e) Naive Poisoning Confusion Matrix



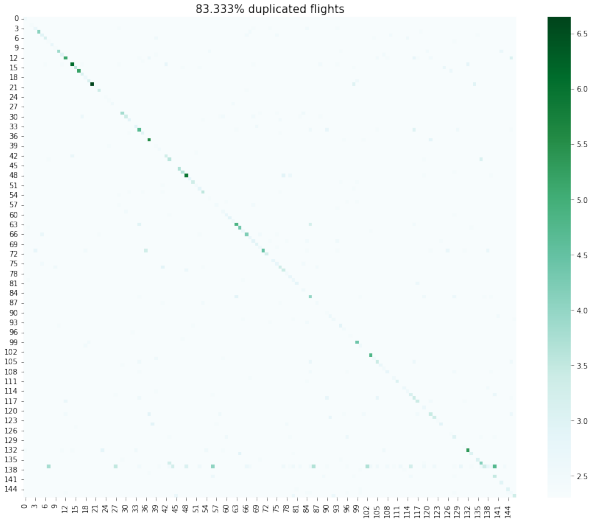
(f) Naive Poisoning Confusion Matrix



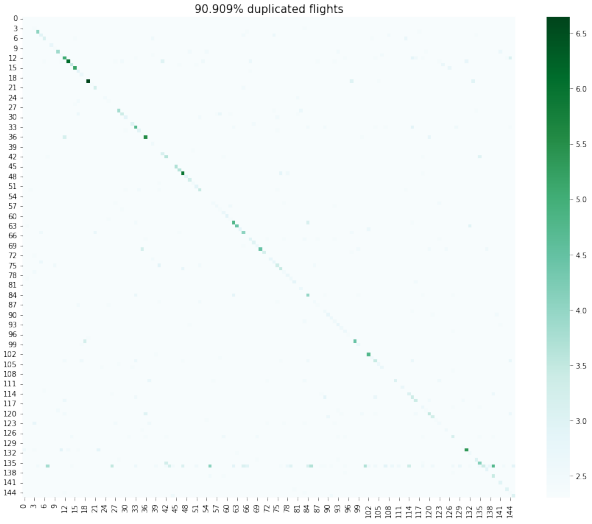
(g) Naive Poisoning Confusion Matrix



(h) Naive Poisoning Confusion Matrix



(i) Naive Poisoning Confusion Matrix



(j) Naive Poisoning Confusion Matrix

Figure B.4: confusion matrices naive poisoning 30 minutes before landing

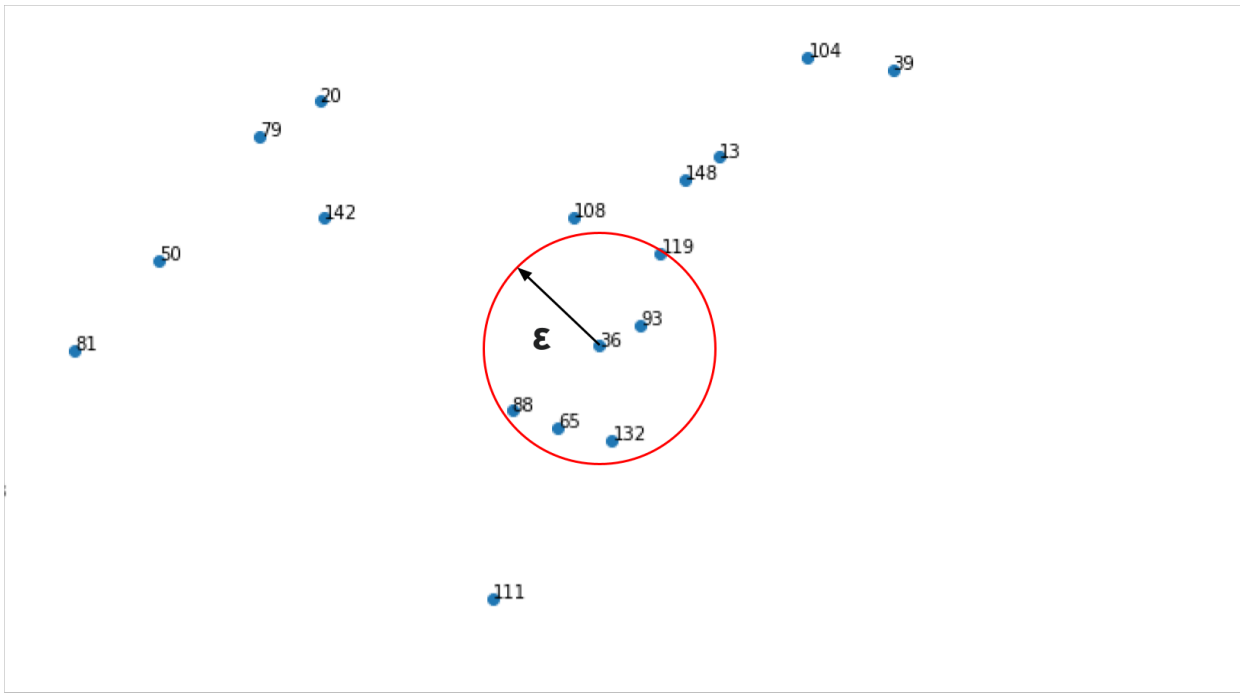


Figure B.5: cluster locations around our target cluster 36

Appendix C

Code Snippets Appendix

```
from sklearn.cluster import KMeans
n_clusters = 150

for OT in owner_types:
    flights = pd.read_csv(f'data/flights/flights_{OT}.csv')
    kmeans = KMeans(n_clusters = n_clusters, init ='k-means++')

    airport_counts = flights['estarrivalairport'].value_counts()
    airport_counts.sort_index()
    df = X[X['ident'].isin(airport_counts.keys())] #keep airport info only for
    flights we fly to
    df.sort_values(['ident'])

    kmeans.fit(df[['latitude','longitude']]) # Compute k-means clustering.
    df['cluster_label'] = kmeans.fit_predict(df[['latitude','longitude']])
    centers = kmeans.cluster_centers_ # Coordinates of cluster centers.
    #mapping for the airports to the clusters
    mapping = pd.Series(df.cluster_label.values,index=df.ident).to_dict()
    #save mapping
    pd.Series(df.cluster_label.values,index=df.ident).to_csv(f'data/airport/
    cluster_mappings/cluster_mapping_{OT}.csv')

    #create file for cluster information: position of center
    cluster_id = np.array([i for i in range(150)])
    cluster_info = pd.DataFrame([cluster_id, centers[:, 1], centers[:, 0]]).T
    cluster_info.columns = ['cluster_id', 'longitude', 'latitude']

    #map the cluster id's to the flights
    flights['airport_cluster'] = flights['estarrivalairport'].map(mapping)

    #save
    cluster_info.to_csv(f'data/airport/cluster_infos/cluster_info_{OT}.csv', index
     = False)
    flights.to_csv(f'data/flights/flights_{OT}.csv', index = False)
```

Listing C.1: clustering of the airports

```
svs = pd.read_csv(f'data/state_vectors/train/svs_{owner_type}.csv')
flights = pd.read_csv(f'data/flights/flights_{owner_type}.csv')
# only keep n_of_dp_flight, (ignore i first)
n_of_dp_flight = (svs.groupby('flight_id').size()/n_of_data_points).apply(floor)*
    n_of_data_points
```

```
modified_groups = []
for name, group in svs.groupby('flight_id'):
    group = group.tail(n_of_dp_flight[name]) #only keep last dp of each flight
    modified_groups.append(group)
svs = pd.concat(modified_groups, axis = 0)
#reshaping to k/m x m*n
reshaped_flights =  pd.DataFrame(svs[features_to_predict].to_numpy().reshape(-1,
    n_of_data_points*n_of_features))
X = pd.concat([reshaped_flights], axis = 1)
#create y according to X and return both
```

Listing C.2: training data influx

```
class LSTM_network(nn.Module):
    def __init__(self, input_size, hidden_size, output_size):
        super(LSTM_network, self).__init__()
        self.input_size = input_size
        self.hidden_size = hidden_size
        self.output_size = output_size

        self.lstm_layer = nn.LSTM(input_size, hidden_size, num_layers=n_layers,
    batch_first=True)
        self.linear_layer_1 = nn.Linear(hidden_size, output_size)
        #self.linear_layer_2 = nn.Linear(output_size, output_size)

    def forward(self, x, hidden):
        output, (hn,cn) = self.lstm_layer(x, hidden)
        output = self.linear_layer_1(output)
        #output = self.linear_layer_2(output)
        return output, (hn,cn)

n_layers=1
hidden_dim  = 128
n_features = len(features)
top_n_airports = 100
model = LSTM_network(n_features, hidden_dim, top_n_airports).double()

epochs = 100
loss_function = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(model.parameters(), lr=1e-5)
```

Listing C.3: LSTM network

```
#naively poison the dataset by duplicating n flights m times to a random
    destination
class naive_poisoned_training_dataset(Dataset):
    def __init__(self, owner_type, n_flights):
        svs = pd.read_csv( f'data/state_vectors/train/svs_{owner_type}.csv')
        flights = pd.read_csv(f'data/flights/flights_{owner_type}.csv').
    drop_duplicates()

        #poisoning
        random_cluster = np.random.choice(flights['airport_cluster'].unique()) #
    return a random cluster number of on of the flights
        duplicable_flights_ids = flights[flights['airport_cluster'] ==
    random_cluster].flight_id

        #sample directly from svs from duplicable flights
        dup_flight_ids = np.random.choice(duplicable_flights_ids, size=(n_flights)
    , replace=True)
```

```
        poisoned_svs_list = []
        for f_id in tqdm(dup_flight_ids):
            poisoned_svs_list.append(svs[svs['flight_id'] == f_id])

        poisoned_svs = pd.concat(poisoned_svs_list, ignore_index=True)
        #add poisoned flights
        svs = pd.concat([svs,poisoned_svs], axis=0).reset_index()
```

Listing C.4: naive poisoning

```
from sklearn.neighbors import DistanceMetric
#return closest clusters to target cluster
def get_n_closest_cluster(owner_type, cluster = 0, n_of_clusters = 1):
    clusters = pd.read_csv(f'data/airport/cluster_infos/cluster_info_{owner_type}.csv')
    clusters['cluster_id'] = clusters['cluster_id'].astype(int)

    dist = DistanceMetric.get_metric('haversine')
    clusters['distances_to_cluster'] = (dist.pairwise(clusters [['latitude','longitude']].to_numpy())*earths_radius)[cluster]
    clusters = clusters.drop(labels = cluster)  #exclude the cluster itself
    return clusters.nsmallest(n = n_of_clusters, columns = ['distances_to_cluster'])['cluster_id'].to_list()


class targeted_poisoned_training_dataset(Dataset):
    def __init__(self, owner_type, cluster_to_jam, n_clusters_to_fake, n_flights):
        svs = pd.read_csv( f'data/state_vectors/train/svs_{owner_type}.csv')
        flights = pd.read_csv(f'data/flights/flights_{owner_type}.csv')

        #poisoning
        clusters_ids = get_n_closest_cluster(owner_type, cluster_to_jam, n_clusters_to_fake)
        duplicable_flights_ids = flights[flights['airport_cluster'].isin(clusters_ids)].flight_id

        dup_flight_ids = np.random.choice(duplicable_flights_ids, size=(n_flights), replace=True)

        poisoned_svs_list = []
        for f_id in dup_flight_ids:
            poisoned_svs_list.append(svs[svs['flight_id'] == f_id])

        poisoned_svs = pd.concat(poisoned_svs_list, ignore_index=True)
        #add poisoned flights
        svs = pd.concat([svs,poisoned_svs], axis=0).reset_index()
```

Listing C.5: targeted poisoning