UCLouvain

École polytechnique de Louvain

Predicting Delays in Brussels' Public Transport

Author: **Catherine Lesuisse**
Supervisor: **Siegfried Nijssen**
Readers: **Pierre Schaus, Guillaume Derval**
Academic year 2019–2020
Master [120] in Computer Science

Acknowledgements

I wish to express my sincere appreciation to my supervisor, Professor Siegfried Nijssen, he guided and encouraged me to when the road got tough and without his patience and assistance, the goal of this project would not have been realized.

I would also like to extend my thanks to the teaching assistants and professors I've had contact with throughout my learning experience at UCL. Your help when answering my sometimes numerous emails and questions is immeasurable.

Finally, I am grateful for the support and encouragement provided by my family and friends throughout my study. In particular André Lesuisse and Matthew Martin.

Abbreviations and Symbols

API	Application programming interface
App	Mobile application
AWS	Amazon Web Services
BCR	Balanced Classification Rate
CV	Cross-Validation
ERD	Entity–relationship diagram
FN	False negative
FNR	False negative rate
FP	False positive
FPR	False positive rate
FR	Functional requirement
GTFS	General Transit Feed Specification
HTTP	Hypertext Transfer Protocol
IDE	Integrated development environment
JSON	JavaScript Object Notation
k-NN	k-nearest neighbors
NB	Naive Bayes
NFR	Non-functional requirement
MLP	Multi-layer Perceptron classifier
OS	Operating System
SDK	Software development kit
STIB-MIVB	Société des Transports Intercommunaux de Bruxelles - Maatschappij voor het Intercommunaal Vervoer te Brussel
SVM	Support vector machine
SVC	Support vector classification
TP	True positive
TPR	True positive rate
TN	True negative
TNR	True negative rate
RFE	Recursive feature elimination
XML	Extensible Markup Language

List of Figures

List of Tables

Listings

Contents

Chapter 1

Introduction

In Brussels there were 417.6 million journeys in 2018 taken via public transport provided by STIB-MIVB.[26] STIB-MIVB, Société des Transports Intercommunaux de Bruxelles - Maatschappij voor het Intercommunaal Vervoer te Brussel, is the local public transport operator in Brussels, Belgium and is responsible for the organisation of the Brussels' metro, trams, and buses. With this statistic increasing every year it becomes more important for both users and service providers that their voyage is without delay and reliable. Not all traffic and delays can be avoided, but it helps to have an insight into when a delay might occur in the network. The goal of this thesis is analyzing the STIB-MIVB network to see if machine learning can be used to predict transit delays.

The "Data Collection" chapter will detail the tools used for data collection from the STIB-MIVB API and how it was automated. What data was collected: GTFS and Vehicle Position, and choices of the line selection from each transit type are explained as well.

In the next chapter, "Vehicle Delay Analysis", describes how vehicle delays will be calculated, statistics of delays on the STIB-MIVB network discovered from the delay data

Once we have our delays calculated from the previous chapter we can start building the machine learning (ML) model in the next chapter "Delay Classification Model". The classification model will make a binary decision by predicting if a delay occurs or not. First features building and selection will be explained and then model selection.

Because this topic has a direct practical usage I explored how how the machine learning models could be integrated into a mobile application for predicting delays in real time. This inspired the development of a mobile app to utilize the results from the machine learning models, discussed in the next chapter "Mobile Application".

Lastly in "Future Improvements", more work which could be done on the project was explained for both the machine learning and mobile application development. Having the opportunity to delve into this project and reflect on it after its conclusion gave me some ideas and insight into where additional progress could be made.

Important prior work on this topic was done by Nikita Marchant in his master's thesis paper "Prédiction des temps de trajet dans un réseau de bus à l'aide de données

historiques"[25]. In this earlier work a k-NN regression model was created to predict the arrival time of a vehicle on singular bus line 95 in comparison with the estimated arrival times STIB-MIVB gives.

Other work on predicting the delays in bus networks was done in Hong Kong by Yu, Lam, and Tam[49] where regressional delay predictions were tested with several ML models such as support vector machine (SVM), artificial neural network (ANN), and k nearest neighbours algorithm (k-NN). Their model aimed to capture the delay times of buses arriving at a stop where each bus is arriving via different routes.

The work by Berger, Gebhardt, Müller-Hannemann, and Ostrowski[5] produced a stochastic model for delay propagation using data from Deutsche Bahn AG. Their model was built to later be applicable to other forms of public transit as well and be suited for online real time applications.

In my paper I will be focusing on how to apply ML to different transit lines and testing several different classification models. This differs from the other papers which focus on regression. In addition the input vector will be using different variables and include non-vehicle features such as time of day, and day of the week of travel.

Chapter 2

Data Collection

This chapter discusses the technologies used to collect this data, how make an API request, the shape of the data, and which subset of lines were selected for analysis. We will collect data for the position of vehicles from servers from the STIB-MIVB over a period of over two months. This is the data which will be used in the machine learning model. The Vehicle Position data contains information on the last stop vehicles part of a transit line have passed, and this will be used to position them in the network. The network information is collected from the GTFS data which is a standardized format for public transportation schedules and associated geographic information.

2.1 Data and Machine Learning Technologies

2.1.1 DigitalOcean

It is impractical to collect data from STIB-MIVB on a laptop as we will need to run an uninterrupted script for over a 2 month period to collect data and a local laptop cannot handle scripts running to process large amount of data. After data collection the raw data collected for this project was a little over 2GB, and at the end of the project 50GB of backup data, data transformations and results were stored. For these reasons a remote server was needed. Choices considered include the server space provided to students by UCL, AWS[4], and DigitalOcean. The server provided by UCL didn't contain enough memory since a lot of information would be stored during the collection of data from STIB-MIVB. AWS is the most popular option[12] with many different solutions available. DigitalOcean was the best fit as it contained all the features I needed and gave me a better user experience than AWS at a more afforable price.

2.1.2 MongoDB

MongoDB[42] is used as the infrastructure for data storage and manipulation. It is a NoSQL document-oriented database program. Another option could have been MySQL[30], a relational database management system. The STIB-MIVB data which is used in the project would fit the form of a relational database, but MongoDB data is stored in JSON

format which is the same format the Vehicle Position data uses and is very scalable. MongoDB is also a newer, very popular,[46] database.

2.1.3 Python3

Data manipulation and collection will be done with Python3. Python3 has a wide variety of libraries and modules available for machine learning making it a great language to use, in addition to its generally easy to understand nature.

Several additional libraries are used:

pymongo[34]	This is used to connect and interact with the MongoDB through python3.
requests[35]	HTTP library which allows the server to make HTTP requests to the STIB-MIVB API for transit data collection.
sklearn[45]	Machine learning library used for the creation and usage of the trained model used for predicting delays.

2.2 STIB-MIVB API

There are several APIs available to collect different types of information from STIB-MIVB. They are the Files, Operation Monitoring and Network Description APIs. Below in the table is a description of information STIB-MIVB releases about their network. The data sets of interest in this project are Vehicle positions and GTFS.

API	Dataset	Description
Files	GTFS	GTFS stands for General Transit Feed Specification, it is a defined format for public transportation schedules and their associated geographic information It is the de facto standard for the representation of its kind of data. The GTFS data is used to build the structure of the public transit network and have the planned scheduled trips available.[13]
Files	Shapefiles	Returns information linked to lines (itineraries) and dots (stops) geometry.
Operation Monitoring	Vehicle positions	Returns real time vehicle positions for given line id's in relation to stops.
Operation Monitoring	Waiting Times	Returns real time waiting times for the next two vehicles of each line passing through the requested stop id's.
Operation Monitoring	Travelers info.	Returns real time messages related to the stops of a specified line(s) passed in the parameters.
Network Description	Stops by line	Return a list of the consecutive stops on a specific line, from the departure point to the end of the line in both directions of travel.
Network Description	Stop details	Returns details of the actual stops on the network. This dataset contains the following information: stop id, stop name in French and Dutch, and the geolocation of the stop.

Using the APIs provided to the public by STIB-MIVB[9] a Python3 script, *get_stib_mongo.py* , was written to access two of the available data sets GTFS (General Transit Feed Specification)[9] and Vehicle positions (real-time)[31]. STIB-MIVB does not make any historical data from GTFS files or Vehicle positions publicly available. For this reason it was necessary to create scripts to collect GTFS and Vehicle positions data over a period of time in order to have information collected to manipulate. A server from DigitalOcean[10] was setup and used to have this script run autonomously for data collection.

The first step to have access to the data is to create an account on their developer site[47]. Afterwards an application on their site needs to be registered and created. Each application needs to subscribe the the APIs of interest prior to creating an access token to make any calls. After subscribing to the APIs of interest the consumer key and consumer secret values are available. These are used to generate time limited access tokens, one access token per consumer key and consumer secret. These access tokens are what give access to make calls to the API and retrieve information. For security reasons all created access tokens are limited in the number of calls made per minute, and each API call has a limited number of information which can be requested. For this reason it is helpful to create several applications to be able to increase the amount of data collected.

To generate these access tokens for use in a python script you first need to copy the

consumer key and consumer secret generate from the STIB-MIVB web-page and base64 encode it, like in the bash example below.

```
echo [consumer key]:[consumer secret] |base64 | head -n 1
```

Listing 2.1: Base64 consumer token generation

This gives you a token which can be used to make a GET request to the STIB-MIVB server for your access tokens. In the example below the python3 library *requests* is used. These access tokens are created after the expiration of the prior so the script always has access to the other APIs.

```
headers = {'Authorization': 'Basic ' +
    base64_consumer_key_consumer_secret}
data = {'grant_type': 'client_credentials'}
response =
   requests.post('https://opendata-api.stib-mivb.be/token',
    headers=headers, data=data)
access_token = json.loads(response.content.decode('utf8'))[
    "access_token"]
```

Listing 2.2: Python example

2.3 GTFS Collection

From the Files API from STIB-MIVB we are able to collect GTFS files in zip format. These zip files contain 9 GTFS files. Six of the files are mandatory for GTFS structure: *agency.txt*, *calendar.txt*,*routes.txt*, *stops.txt*, *stop_ times.txt*, *trips.txt*, while the remaining three are optional and help with scheduling exceptions, mapping routes geographically, and translating names: *calendar_ dates.txt*, *shapes.txt*, and *translations.txt*.

The GTFS data is documented to be updated approximately every 2 weeks and only contains information for a specified time interval. The release dates of new GTFS data are not publicly known so every morning the *GTFS.zip* would be downloaded and the binary compared to the current in use GTFS binary. If the new GTFS zip was different it is assumed to have updated information and the old information is replaced with the new files, and the old is stored.

```
headers = {'Authorization': 'Bearer ' + access_token}
response = requests.get(
    'https://opendata-api.stib-mivb.be/Files/2.0/Gtfs',
       headers=headers)
with open('./STIB_gtfs_timetables/' + new_gtfs_file + '.zip',
   'wb') as fd:
    for chunk in response.iter_content(chunk_size=128):
        fd.write(chunk)
        fd.close()
if filecmp.cmp('./STIB_gtfs_timetables/' + curr_gtfs_file +
   '.zip',
    './STIB_gtfs_timetables/' + new_gtfs_file + '.zip') == False:
```

```
    # update GTFS data to reflect the new updates GTFS files
else:
    # GTFS were not updated, remove new GTFS.zip and keep using
       the current files
```

Listing 2.3: GTFS.zip retrieval from STIB-MIVB

2.4 Vehicle Position Collection

Now that we have GTFS files there is a base to map transit vehicles back to. In order to predict vehicle delays the first step is to now track vehicle position and when they are passing stops. To do this data from the Operation Monitoring API about Vehicle Position will be collected. This API gives access to Vehicle Position data that gives information on which stop a vehicle has passed and how many vehicles from a transit line are currently driving on the network. The vehicle positions in the STIB-MIVB API are updated in real time.

The API request requires a list of at maximum 10 transit lines of interest. Returned is a JSON containing the list of requested lines, for each line there is a list of vehicles currently driving in the network that belong to each line. Each vehicle has a list of their last passed stop, distance away from the last passed stop, and direction of travel is returned.

It is not possible to have information on what specific schedule from the GTFS files a vehicle should be following, so these relative positions will have to be mapped back approximately to the scheduled stop times to predict their scheduled route.

Vehicle Position data was collected by making GET requests through python3s library *requests*, similar to the GTFS data collection.

```
data = {'grant_type': 'client_credentials'}
headers = {'Authorization': 'Bearer ' + access_token}
id_list = list of 10 transit lines
url = "https://opendata-api.stib-mivb.be/OperationMonitoring/" + \
    "4.0/VehiclePositionByLine/" + "%2C".join(transit_id_list)
response = requests.get(url, headers=headers, data=data)
```

Listing 2.4: Vehicle Position retrieval from STIB-MIVB

The output from the Vehicle positions call is a JSON string with vehicle positions. For each line there is an array of vehicle positions and their corresponding information: directionId, distanceFromPoint, and pointId. Below is an example of the output. In addition to this a "date" and "time" variable were added to each JSON line collected to signify the time and date of data collection. This information was then stored in a file named after the collection date containing that days' vehicle positions collected.

```
{"lines":
    [{"lineId":1,
    "vehiclePositions":
        [{"directionId":8161,"distanceFromPoint":0,"pointId":8012},
```

```
        {"directionId":8731,"distanceFromPoint":0,"pointId":8021}]}
    ]}
}
```

Listing 2.5: Vehicle Position output from STIB-MIVB

2.5 Line Selection

There are a total of 72 transit lines in the STIB-MIVB network. Due to the size of the transit network if all lines were selected it could cause potential issues with data storage, it would require the use of additional keys because of API limitations, and cause excess processing times. For this reason a representative subset of metro, tram, and bus lines will be selected to work with. The initial criteria used to select lines were created with the help of an expert, Bruno Seny, a representative from Stratec[36]. Stratec is a consultancy agency specialized in mobility and data analysis. Lines selected were known to pass though congested areas, less congested areas, or experience frequent delays. These different lines were selected so the features will help give good variance of delays for all the lines.

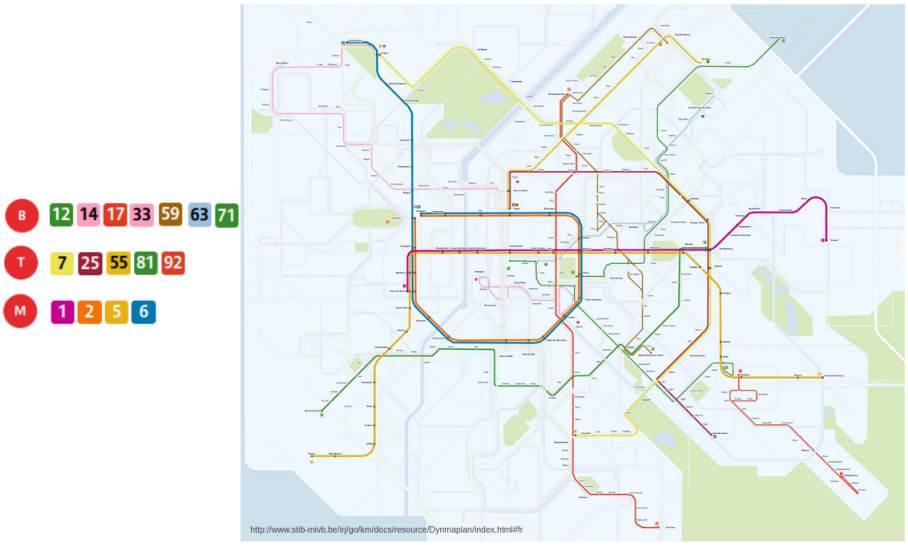


Figure 2.1: STIB-MIVB network with selected lines

Above in Figure 2.1 is a map from STIB-MIVB showing the lines selected for analysis. There are 4 metro lines, 7 tram lines, and 7 bus lines selected. At least one line is covering each corner for a geographical spread of lines and the rest are selected via professional recommendation. Data was collected for a period of several months, but because of occurrences of STIB-MIVB API or script failures not all transits have complete information for each day. For this reason the data which will be used is from between

the dates of July 3, 2019 and August 26th, 2019. These were the dates for which there is vehicle information collected for all selected transit lines with no gaps.

Chapter 3

Vehicle Delay Analysis

In this chapter the vehicle delays will be calculated using the GTFS and vehicle positions collected from STIB-MIVB discussed in the prior chapter, "Data Collection". From these delays it can be seen which attributes are highlighted by the delay statistics that need to be taken into consideration before starting the feature and model building and selection. A database will be built from the data to allow for easier manipulation for schedule retrieval and calculate vehicle delays. Using the database allows for efficient searching of scheduled stop to match to vehicle positions from the network. Once the vehicle delays are calculated statistics of the delays will be looked at.

3.1 Delay Calculation

The first step in delay calculation is to map each vehicle position to the STIB-MIVB transit schedule to see which scheduled stop the vehicle passed and assign the vehicle to that scheduled stop as visited. When making the calls to the Vehicle Positions API the time the API call was made is stored along side the retrieved vehicle positions, as the variable *collection_ date_ time*. This was what allowed the mapping back of vehicles to the schedule from the GTFS files.
Because we don't know for certain which schedule a vehicle from the network should be following there is uncertainty if we are able to select the correct scheduled time for a vehicle. Described in Figure 3.1 is an example of this ambiguity. If a bus arrives at a stop every 30 minutes and something happens which causes a portion of the network to experience erratic shifts in arrival times and a bus only passes by the scheduled time slot 15 minutes early or 15 minutes late, we will not be able to tell which vehicle was actually supposed to be passing at the designated time. In Figure 3.1 it is 3:00pm and a bus had passed stop 2 15 minutes prior and another is coming 15 late, we would have to make a guess as to whether the bus was delayed for this stopped or passed early for some reason.

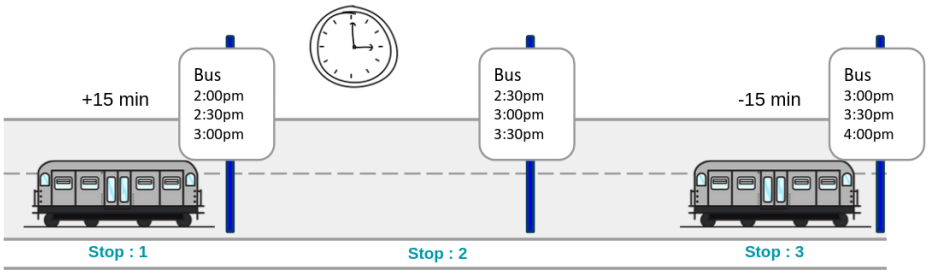


Figure 3.1: STIB-MIVB network ambiguity example

In addition to the above ambiguity, vehicles occasionally disappear from the network and thus they won't be possible to track when attempting to match scheduled stops. Once the data from STIB-MIVB has been collected it is analysed to calculate the delay. A python class called *STIB_API*, in the file *STIB_API.py*, was created to store the data in a MongoDB database and have functions to retrieve, reshape, and analysis the data. Each file from the GTFS zip was given its own collection, with the exception of *agency.txt* and *translations.txt* files since these are irrelevant to the delay analysis. For each date of data collected a *todays_stop_times* collection[8] was created where every scheduled stop is a separate document[11]. Below in Figure 3.2 is a UML model of the database created and used in the script. The blue collections represent the GTFS data and the pink collections are collections I created for the vehicle position data in addition to the *todays_stop_times* utilized to match the vehicles to their scheduled stops for each days group of vehicles.

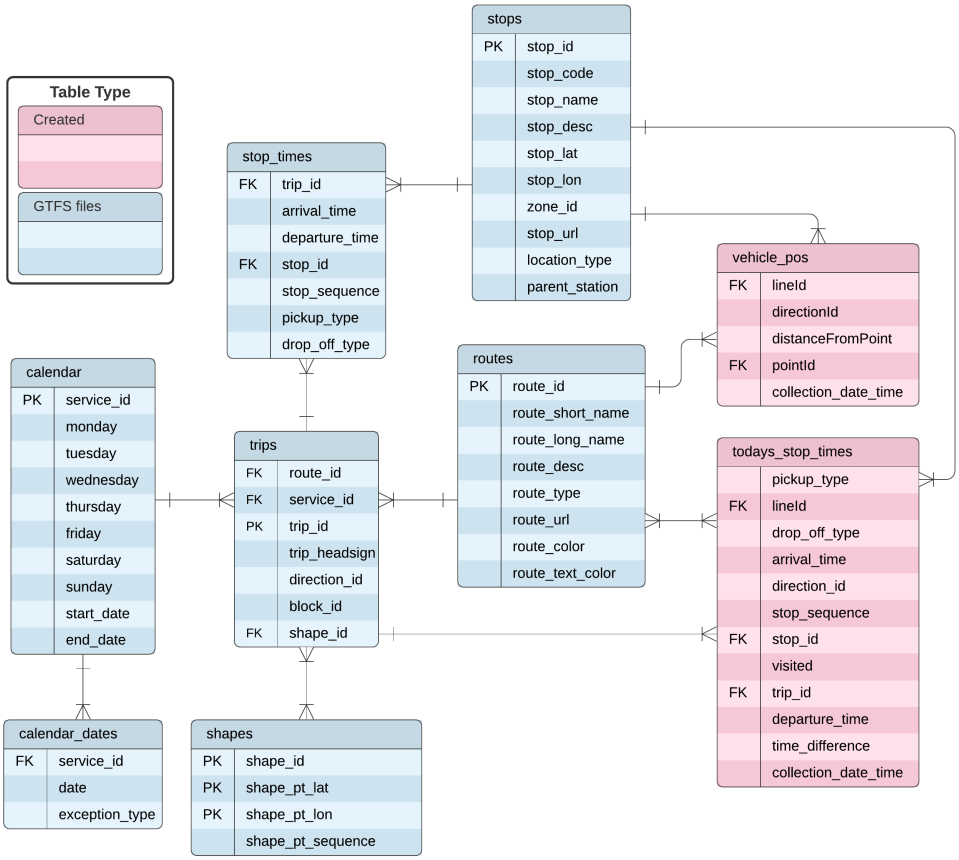


Figure 3.2: STIB_API MongoDB ERD with Crow Foot Notation

The collected vehicle positions of that date were then iterated through. Matches of the smallest time difference were made between the *arrival_time* of scheduled stops and the *collection_date_time* of the collected vehicles. The absolute value of the difference was used, because it is always possible for a transit vehicle to be ahead of schedule and not only late. This time difference value is then joined to the document as the field *time_difference*. This is shown below in Figure 3.3 where the *todays_stop_times* collection shows the original document features before and after a vehicle is matched to it's scheduled stop.

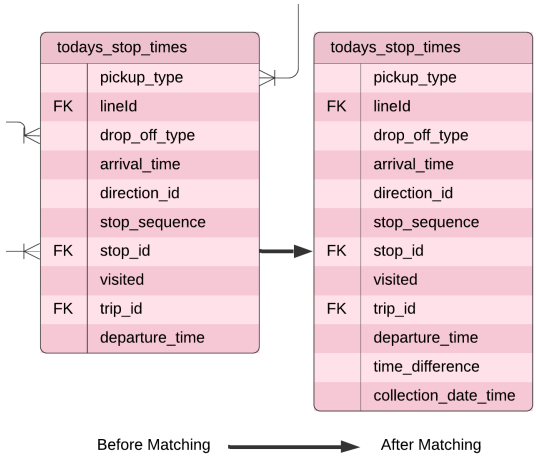


Figure 3.3: *todays_stop_times* MongoDB collection ERD with Crow Foot Notation

Because vehicle position information is only gathered every 60 seconds, the vehicle is only marked as delayed if the time_difference is greater than 90 seconds. This number was chosen because it includes a 60 buffer for the vehicle to pass right after the prior collection and 30 seconds in addition.

3.2 Statistics

Taking a look at the statistics from the delays will give us a good idea as to the shape of the data we are working with and what might be useful direction to look at in feature selection. Some statistics looked at will be if delays vary throughout the day or vary by transit. It will also help with pointing out irregularity in delay calculations so the erroneous values won't be input into the machine learning pipeline.

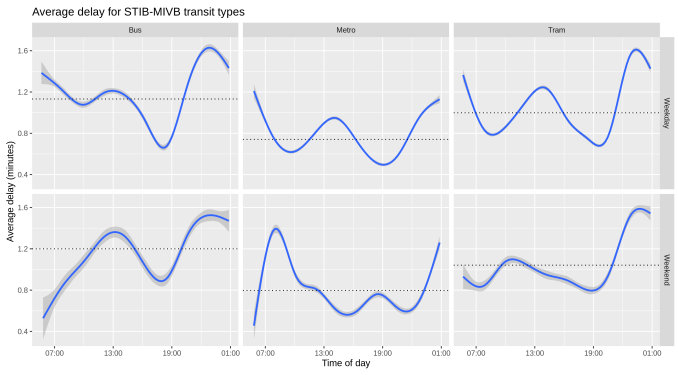
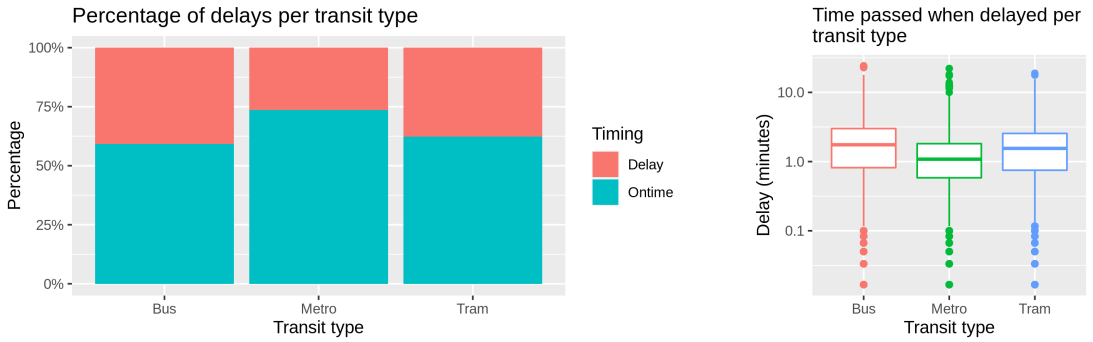


Figure 3.4: STIB-MIVB average delays

In Figure 3.4 above, each of the transit types shows the delay average throughout the

day between weekdays and the weekend. When performing t-tests between the means of the delay times between weekday and weekend on the same transit type, only metro and tram came back having statistically significantly different means between weekday and weekend delays. The buses have no significant mean difference between weekend and weekday. When looking closely at the graph it can be seen that in the mornings on the weekend the buses seems to have a much lower average delay than during the weekdays. When comparing the buses average delays before 13:00, buses are statistically significantly less delayed on average during weekend mornings than during the weekday.

The average arrival time past the scheduled stop for buses is 1.145773 minutes, metros is 0.7547081, and trams is 1.007678. While performing a one-sided ANOVA[22] between all the transit types delays, there was a p-value of $<$ 2e-16 each time signifying that at least one mean is significantly different and we are unable to accept the hypothesis that all transit types experience the same degree of delays. In Figure 3.5 (a) we can see that buses and tram experience proportionally more delays than the metro. Looking at Figure 3.5 (b) you can see that some outliers exist past 10 minutes or greater when normally a transit vehicle should have passed by. These errors are sometimes caused by vehicles appears on the transit map when they are not scheduled to run and missing vehicles from the network leave an open gaps for these erroneous vehicles to pick up a scheduled stop with a larger than possible delay value.



(a) Delay time percentage

(b) Delay time box plot

Figure 3.5: Per transit comparisons

When comparing delays between transit types the metro network has the least amount of delays with 73.5% of scheduled stops having an on time arrival. Buses and trams having a 59.2% and 62.25% respectively on time arrival percentage. Both the transit types which run on tracks, metro and tram, have higher on time arrivals rates where buses have more delays. Maybe this is influenced by the fact that metros are completely separated from outside traffic. There are also only 4 metro lines compared to the 50 bus lines(+11 Noctis night lines) and 17 tram lines. The lower number of lines to share routes with could also contribute. The bus lines probably have the most delays because they have to share the road with cars and other vehicles, introducing an uncertainly in road conditions. The

trams mostly travel on tracks, like the metro, but they also share the road at times with other vehicles. This could explain why they have a delay percentage between the other two.

When training the ML model it will be important to keep in mind from these findings that different transit types have significantly different delays and delay frequencies and that the time of day can greatly impact the delay time as well.

Chapter 4

Delay Classification Model

Using the previously collected and created delay data, it will be used to build features for a delay classification model. In this classification model input data will be used to return a binary value indicating whether a delay will occur or not at a specified transit stop at a scheduled time. The input data will be a feature vector containing delay values, time of day, day of the week and line id. Feature creation and feature importance in modeling building across different transit types is examined. After features are selected model selection will begin.

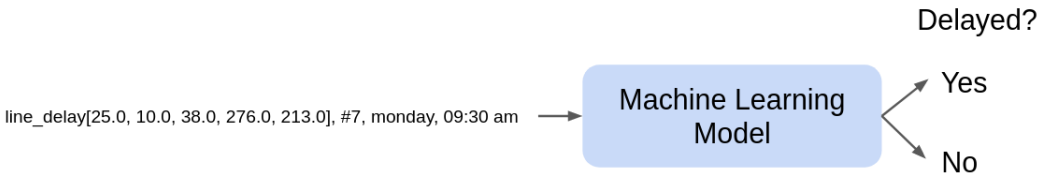


Figure 4.1: Delay classification

4.1 Feature Building

Features will be based off of each scheduled stop and the corresponding delay time and binary delay indicator. The binary delay indicator is a 0 if there is no delay and 1 if the vehicle was delayed arriving at the scheduled stop. Other features are the day of the week and time of day. Since the vehicles are running on a network and could affect each other, additional features to be tested is the average delay on the associated transit line in the 10 minute interval prior, with five 10 minute intervals going back to 50-60 minutes prior to the stops scheduled time; [10-20 minutes prior, 20-30 minutes prior, 30-40 minutes prior, 40-50 minutes prior, 50-60 minutes prior], see Figure 4.1 below.

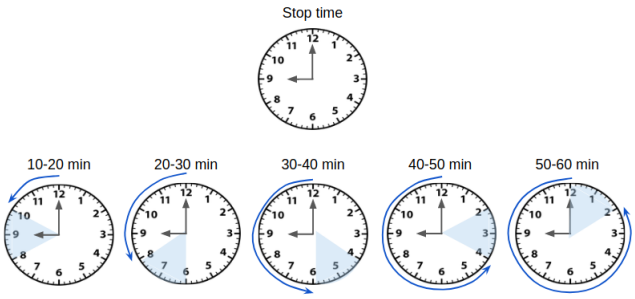


Figure 4.2: Delay time interval calculation

The thought was that if the line is congested and this can be tracked up to an hour prior to the stop we might be able to predict if the delay will occur or not. Below is the metro line 2 with an arrow pointing to the Art-Loi/Kunst-Wet stop. For the previous line delays going back an hour the average delay would be taken over the entire line in 10 minute intervals.

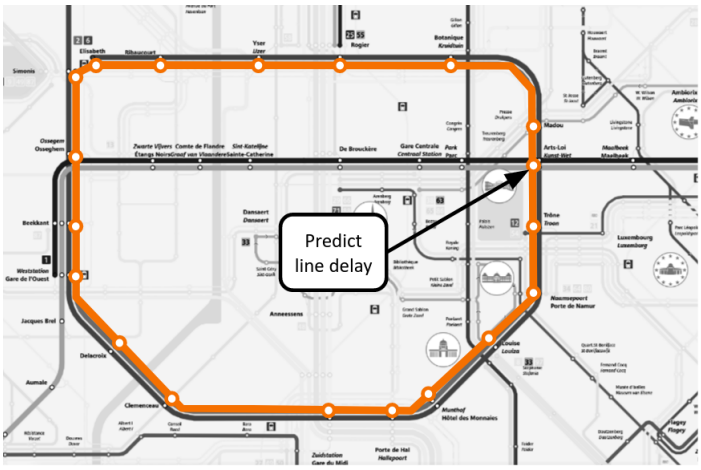


Figure 4.3: Average transit line delay calculation

In addition to having features summarizing the average delay on the line, features summarizing the delay in the geographical range around the stop will be looked at as well. Maybe there is congestion that doesn't affect the whole transit line and only effects the geographical area of where the stop is and the passing lines. For the same line and stop as the previous example here is an example of where average delays will be taken for the Art-Loi/Kunst-Wet. We can see that delays used in this example are those within a local radial distance of Art-Loi/Kunst-Wet and include delays of stops not in metro line 2.

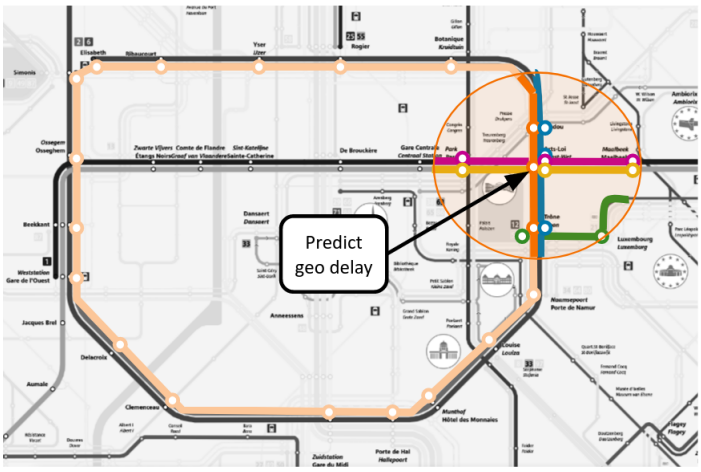


Figure 4.4: Average geographical area delay calculation

4.2 Feature Selection Choices

Selection of a model is done by first looking at what features contribute the most and removing those which don't. Following Occam's razor we want the shortest description of the data as the best model. Feature selection is also good for reducing overfitting of data and having shorter training times. Using backward elimination and k-fold cross-validation different features were tested. During the k-fold cross-validation the k-means models was used as the base testing model before moving onto testing other models with a sub-selection of features.

The first group of features tested during this process are features using only transit line delays, only geographical region delays, or using both. With this subset we are seeing after how far in advance can we predict a binary delay by line and/or geographical delays. Each prior 10 minute interval is an individual value of the average delay for the delay type. When both line and geographical delays are being used they are both used as individual variables. Are line delays or delays in a geographical area more influential to delay prediction?

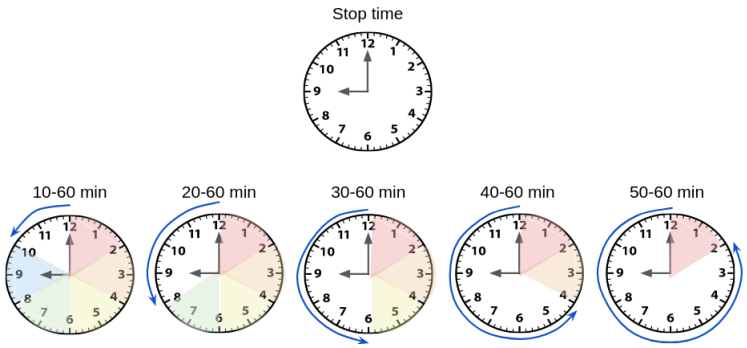


Figure 4.5: Delay interval features

Looking at the Figure 4.5 above, going from left to right, each interval will be testing delay prediction 10, 20, 30, 40, and 50 minutes in advance of the scheduled stop time. Below in the table are what the input features for the feature delay intervals tested would be like. Each 10 minute interval has its own value for the interval type. The 10-60 minute prior interval will be used to predict delays 10 minutes in the future, the 20-60 minute prior interval will be used to predict delays 20 minutes in the future, and so forth.

Below in the table is a complete representation of the combinations of delay grouping method and delay time intervals used.

	Delay grouping method	Delay average interval
(a1)	Line and Geographical	10-20(min),20-30(min),30-40(min),40-50(min),50-60(min)
		10-20(min),20-30(min),30-40(min),40-50(min),50-60(min)
(a2)	Line	10-20(min),20-30(min),30-40(min),40-50(min),50-60(min)
(a3)	Geographical	10-20(min),20-30(min),30-40(min),40-50(min),50-60(min)
(b1)	Line and Geographical	20-30(min),30-40(min),40-50(min),50-60(min)
		20-30(min),30-40(min),40-50(min),50-60(min)
(b2)	Line	20-30(min),30-40(min),40-50(min),50-60(min)
(b3)	Geographical	20-30(min),30-40(min),40-50(min),50-60(min)
(c1)	Line and Geographical	30-40(min),40-50(min),50-60(min)
		30-40(min),40-50(min),50-60(min)
(c2)	Line	30-40(min),40-50(min),50-60(min)
(c3)	Geographical	30-40(min),40-50(min),50-60(min)
(d1)	Line and Geographical	40-50(min),50-60(min),40-50(min),50-60(min)
(d2)	Line	40-50(min),50-60(min)
(d3)	Geographical	50-60(min)
(e1)	Line and Geographical	50-60(min),50-60(min)
(e2)	Line	50-60(min)
(e3)	Geographical	50-60(min)

The next features looked at are line id and day of the week. If the transit line id should

be included, how influential is the line to delay patterns? The days of the workweek and weekend might show variability in delays as well, since peoples' daily activities would normally differ when they are not working on the weekends versus commuting during the workweek. Should binary variable should be used to indicate workweek or weekend instead of separating each day of the week?

Weekday	Monday, Tuesday, Wednesday, Thursday, Friday
Weekday/end binary	0, 1

From what we learned in the statistics section the different transit types have different mean delay times and different rate of delays occurring. For this reason each transit type will have its own model, as each transit type has many different external factors that affect each differently, which cannot be taken into account for. Such as metros being isolated from street traffic and trams which occasionally mix in and out of street traffic but are generally isolated as well.

4.3 Feature Selection

In "Feature Selection" the features will be clustered to allow for timely model creation without loosing information. The performance of a subset of models will be used to determine the most influential attributes of the input feature vector and which features can be removed. The effect of transit type will be analyzed as to which features are the most important to each type.

4.3.1 K-means Clustering

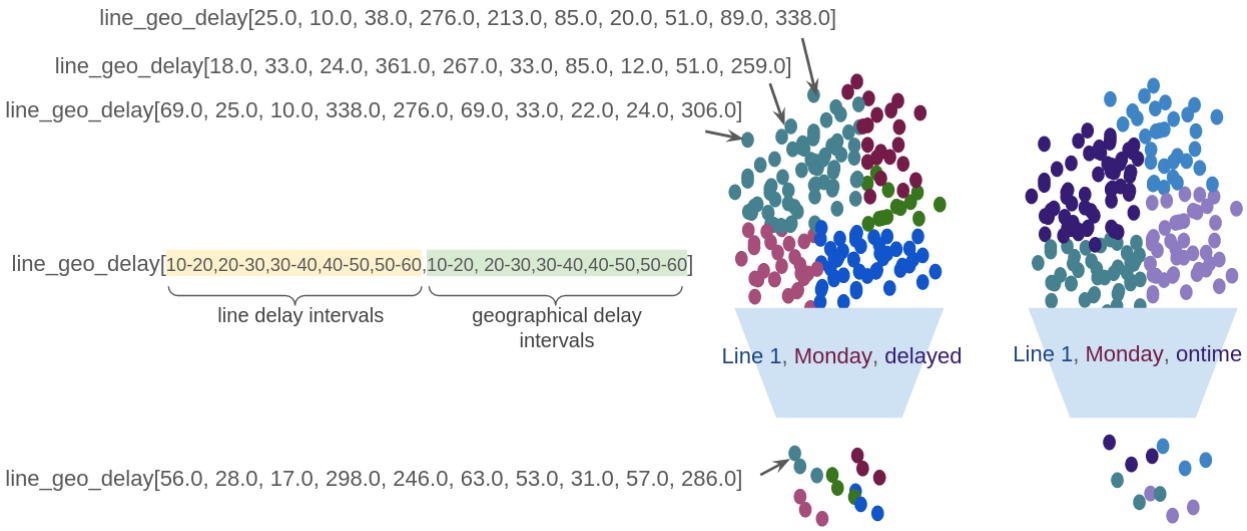


Figure 4.6: Clustering example by features

Because of the large amount of data collected over the 55 day period it is necessary to summarize the data to have a compressed selection small enough for model building. Some linear based machine learning models have no problem handling a quarter of the unclustered input vectors but others require more memory which is a disadvantage. Using all of the data to train a model would be best to get an accurate summary of each feature, but a representative subset of the data can also yield good results. For example, if there is only one Monday represented then it would be a weak representation. Having multiple days of the week represented would be ideal, and being able to use this information by clustering it into a practically sized summary to test on different model types which would otherwise not be able to handle the larger data set will be beneficial.

A clustering algorithm runs on the notion that items which are more related are nearer than items which are less related. Centroid-based clustering(k-means)[7] will contain k clusters that are represented by k vectors of which the closest items will be assigned to and summarized by. These centroids can then be taken and used as compressed features. In k-means clustering the data will be clustered in groups by day, delay binary indicator, and line id.[18] This will give us a group of clusters summarizing the delay variables for each group. K-means was selected because the number of centroids created can be defined and this will help control the size of the summarized data set so the model computing doesn't run into the same size problem this solution is attempting to correct. Another method looked at was mean-shift, this method is very effective at finding groups of high density but the number of centroids found cannot be chosen and the radius of some cluster could potentially end up being too large for what is practical.

Cluster sizes of 30, 50, and 70 were testing in an initial phase to see how small the cluster size used can get without losing information and accurately summarizing the data. As mentioned in section 4.3.1 K-means Clustering, non-delay interval features of the same value were grouped and those values used to form clusters, and whose mean delay interval values will be used as features for training. Tested on RandomForestClassifier, DecisionTreeClassifier, and MLPClassifier the BCR were compared for every model. The BCR is the balanced classification rate. The BCR will be calculated by :

$$BCR = \frac{1}{2} * \left(\frac{tp}{tp + fn} + \frac{tn}{tn + fp} \right)$$

A one sided t-test was performed to see if either model performs better and there was no significant different in any instance. A larger cluster size wasn't used due to memory availability on the available hardware and the high memory complexity. A middle ground of 50 clusters was selected for further use in the feature and model selection.

4.3.2 Feature importance

Starting with the delay intervals used during model building, it will be shown which interval has the most influential role in predicting if a delay will occur. With this information it will be know if any feature variables can be removed from the vector and if different transit vehicles require different features.

The 2 models used during feature selection are DecisionTreeClassifier[37] and RandomForestClassifier[38]. These two models were chosen for classification feature selection because both DecisionTreeClassifier and RandomForestClassifier are tree based and have *feature_ importances_* attributes which return the impurity-based feature importance, also known as Gini importance. This method calculates each feature importance as the total decrease in node impurity weighted by the approximation of the proportion of samples reaching that node.

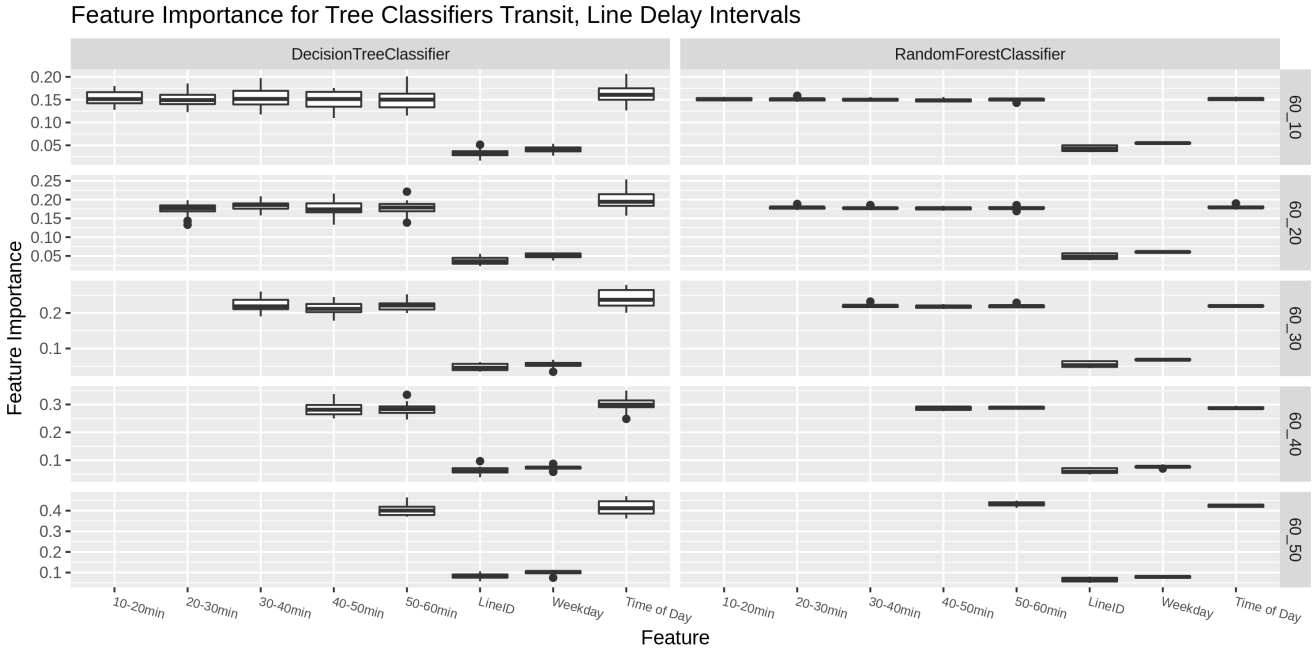


Figure 4.7: Tree Classification Feature Importance

First we will be looking at the tree classifiers built for all time intervals using the average transit line delay. Of the three non-time interval features both the transit line id and the weekday features show the least contribution to the result where as time of day is considered to be more influential. The time of day is ranked as having approximately the same Gini importance as all of the time intervals in each model. An assumption going into this was that the further away the time interval was from the scheduled stop the less importance it would have towards the classification result. These results show that all time intervals have approximately the same feature importance. The results of models built using only geographical delay shows an identical pattern in the results. Impurity-based feature importance, such as Gini importance, for trees are strongly biased and favor high cardinality features, such as the delay average time intervals and the time of day features. All of these features can have a wider range of variables they can be, where as line id and weekday are categorical variables. This could account for the high importance of these features and artificially increased their ranking.

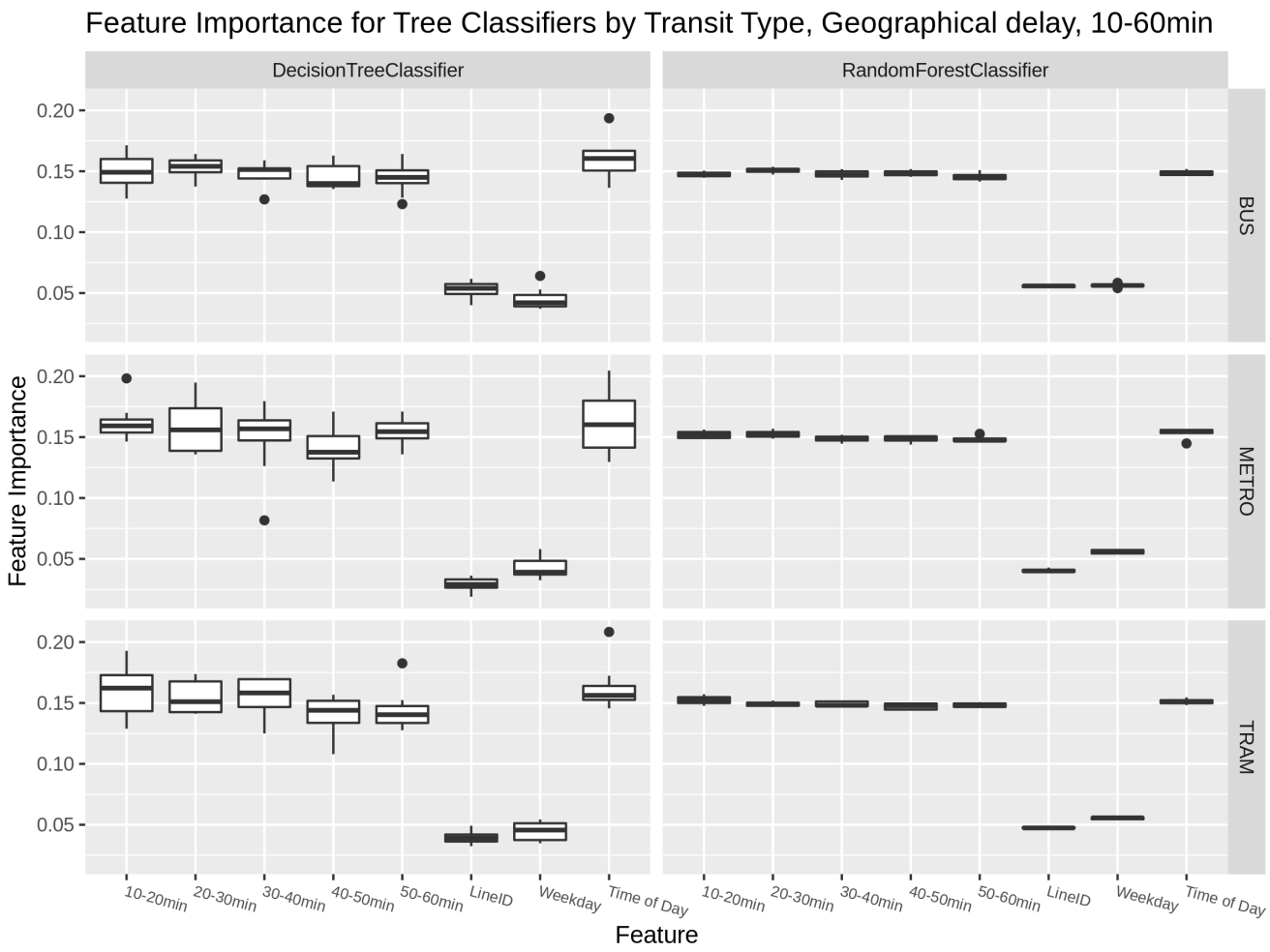


Figure 4.8: Tree Classification by Transit for Feature Importance

When looking at feature importance separated by line delay averages for the time intervals there is no significant difference between time intervals importance. Looking at the above Figure 4.8 you can see that for the DecisionTreeClassifier for only the tram lines the closer the delay interval is to the scheduled stop the larger its importance is in classification. For this transit line the prior assumption of the most recent delays having greater importance has a small amount of support now. An ANOVA test supported that the time intervals have at least one which have a mean that differs. Performing a one sided t-test with the 10-20 minute interval versus the 50-60 minute interval for both tram and metro show that they are statistically different and the 10-20 minute interval has a greater importance than the 50-60 minute interval.

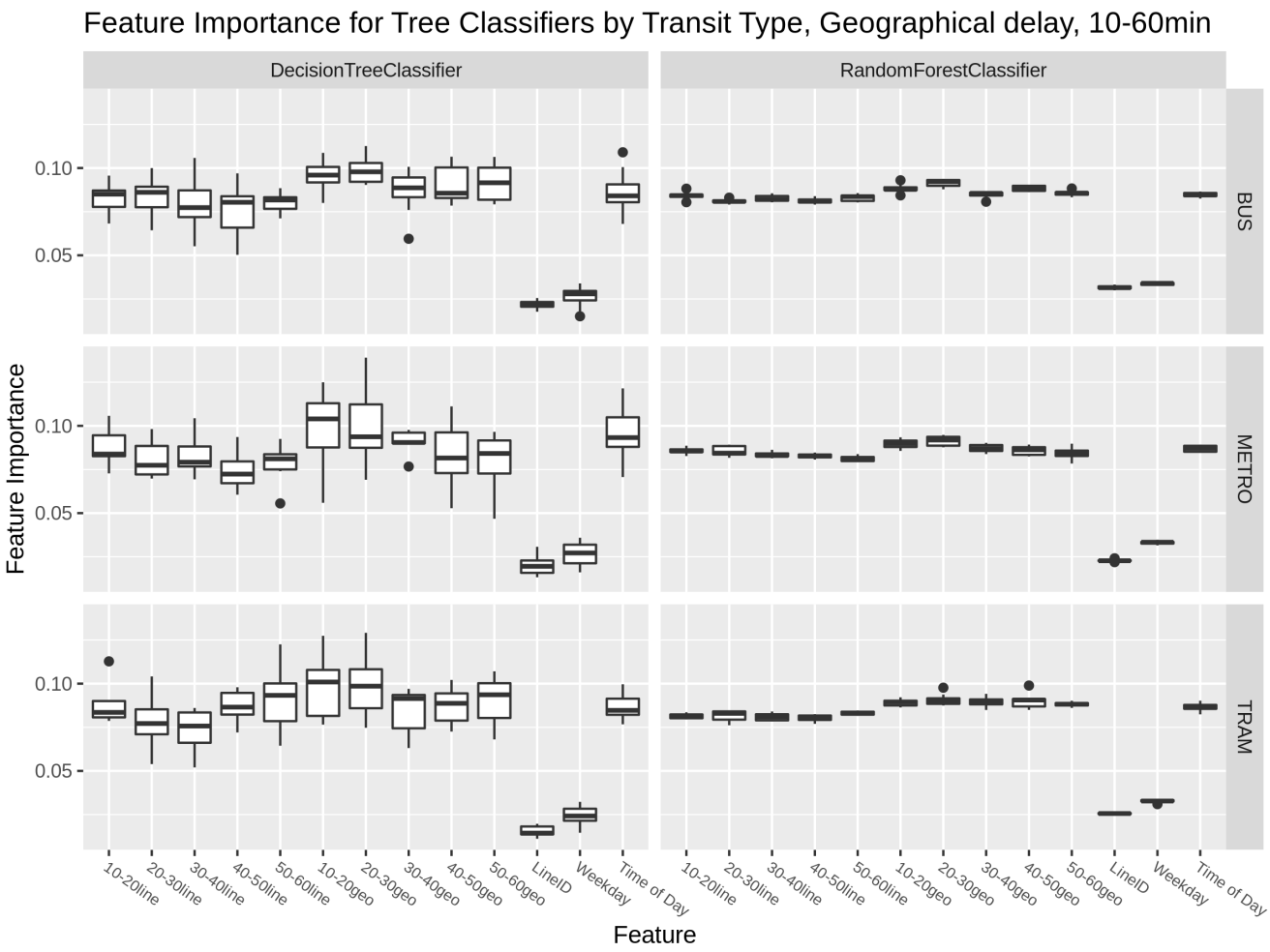


Figure 4.9: Tree Classification by Transit for Feature Importance, transit line and geographical delays

When performing a one sided t-test on RandomForestClassifer models built with both transit line and geographical delays show a statistically significant, with 98% confidence, greater importance for geographical delays in all three transit lines for each pair of corresponding delay time intervals. In DecisionTreeClassifer models the geographical delays are statistically greater in importance, with 95% confidence, than the transit line delays for 2/5 tram and metro delay intervals, and with 97.5% confidence for 4/5 bus delay intervals.

To account for bias by the categorical variables tests were also done with the line id and weekday variables converted to binary features. Below we have an example of the bus lines with binary features for each bus line id and the weekday variable indicating 0 for weekday and 1 for weekend. Converting these features didn't change the importance they previously had. The id feature continues to be ranked the least important in delay

classification. The weekday binary indicator had a greater variance in importance across the multiple models created but remained higher in importance over line ids.

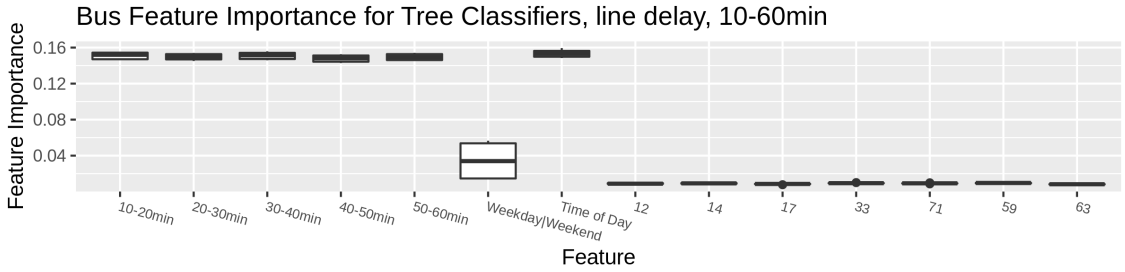


Figure 4.10: Categorical features converted to binary features

Using only tree based feature selection the best features to select so far seem to be the geographical delay intervals and the time of day. The transit line delay intervals don't show that they are statistically greater than their geographical counter parts, though the line delay intervals don't perform significantly worst in all cases either. The line id and weekday feature can continue to be looked at even though their importance is marked as very low because of the inherent biased the tree-based models have towards the type of variable representation.

4.3.3 Transit differences

The look at feature importance gave a glimpse into how transit type might play an important role in the type of feature used during model training. Here I will look into how these models perform using all non-delay interval features. For this we will be looking at the accuracy and BCR, the accuracy will give us an overall idea of how the model works on a normal data set and the BCR will show us how accurate the models are when delay and on-time classifications are give more equal weight to correct for the unbalanced test set.

Models were tested with 4-fold CV. Each fold was created by dividing up the dates where data was collected, randomly shuffling them, and then splitting them into 4 equal groups. ¼ of the dates would be used to train the models and the remaining ¾ would be the testing set.

First taking a look at the accuracy, we can see that using delay intervals from a geographical range alone or in conjunction with line delay intervals was detrimental to the accuracy of the RandomForestClassifier for metros. As the number of delay intervals used goes down the accuracy levels out to random chance. A different trend can be seen for the buses, where the inclusion of geographical delays seems to boost the accuracy slightly, in particular that of bus line 33, otherwise the difference between line and geographical delays has a negligible effect. The trams follow a similar pattern to the metro, though to a less extreme extent. I could be that this is an effect of these two transit lines running on isolated rails separate from outside congestion.

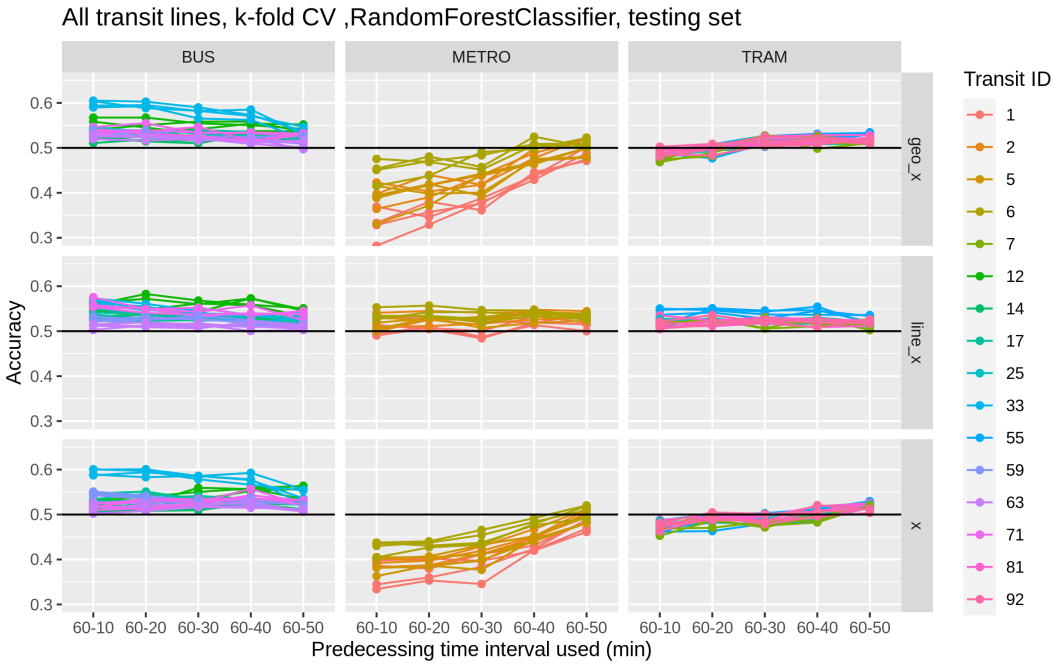


Figure 4.11: RandomForestClassifier by Transit accuracy

The BCR is another good metric to look at since the proportion of delayed versus on-time vehicles is significantly different, this will help make sure the delayed vehicles are not being severly under predicted since they are the minority of cases. Here we can see that the models are all more precise when more time intervals are used. These values can be used to make sure that the models have sensitivity and specificity, in our case it would be easier to always predict on-time and get a high accuracy but the sensitivity and specificity of these results would be very poor especially for delayed transit lines.

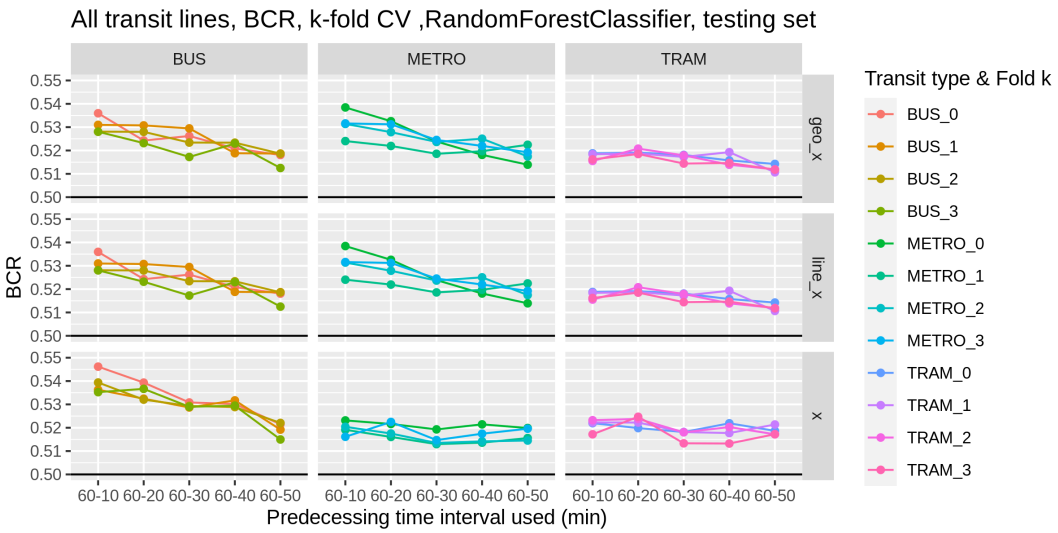


Figure 4.12: RandomForestClassifier by Transit BCR

The accuracy and BCR transit comparisons were repeated on several other models: DecisionTreeClassifier, MLPClassifier, GaussianNB, and SVC. They all followed the same pattern where metro and tram models scored better using line delays without geographical delays and bus models scored very similarly regardless of delay type used. This is interesting to see as the feature importance values found from the RandomForestClassifier models indicate that geographical delays are statistically significantly more important than line delays in determining if a delay occurs. Though importance of a feature during the building of the model doesn't necessarily result in a high accuracy, as we see here with metro and tram lines.

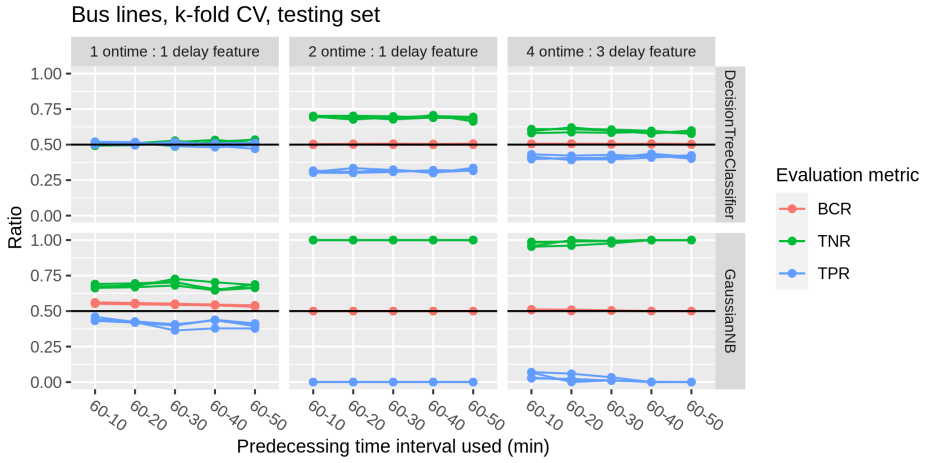
4.3.4 Final feature selection

The final delay interval type selected to be used further are the line delays. These still showed high feature importance and also all transit types were able to have a good accuracy score with this interval type. Weekday, line id, and time of day will be kept as well and line id removed since the inclusion of this features is the one with the least importance by far and contributes the least information to the model being built. It will allow the model to better try to cover all lines without risking being overly specific to certain transit lines.

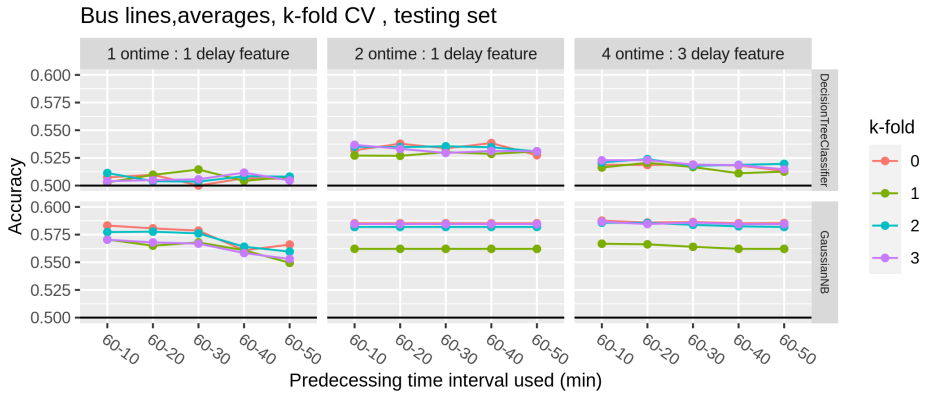
4.4 Model Selection

Model selection was done using features clustered in groups of 50 with only averaged line delay intervals, and continued use of weekdays, time of day, and line id. When selecting a model the most difficult goal was to have one that performed with a high overall accuracy and a high BCR. Models that scored high on the accuracy tended to over estimate the

number of on-time vehicles and would miss many delayed vehicles resulting in a low BCR. Models looked at were the ones available via scikit-learn such as Support Vector Machines, Gaussian processes, Naive Bayes, Decision Trees, and Random Forests.



(a)



(b)

Figure 4.13: GaussianNB and DecisinTreeClassifier ML model

For example in this Gaussian model, three were built using different ratio of on-time and delay features. The models were very sensitive to the number of delayed features used. A true positive is defined as a correctly predicted delay, and a true negative is a correctly predicted on-time vehicle. Predicting on-time for all of the transit vehicles results a higher accuracy throughout, like seen in the *2 on-time : 1 delay feature* model. Though this model never correctly predicts any delayed vehicles at all. The *1 on-time : 1 delay feature* model, which over represents the number of delays, causes a decrease in the TNR in comparison to the other two, but is able to predict with a better frequency the delayed vehicles.

The DecisionTreeClassifier is able to classify delayed vehicles at a higher rate for all the models, but has a lower accuracy since the TNR never gets that high and the classifier seems more balanced when predicting. The TNR and TPR also seem highly correlated to the proportion of delayed examples used; the TNR and TPR ratio for the *2 on-time : 1 delay feature* model is approximately 2:1 and the *1 on-time : 1 delay feature* model is also approximately 1:1.

In all the different model types it was quite difficult to get a high BCR. Part of the difficulty might have come from the fact that a vehicle could follow the pattern of an on-time vehicle and arrive 30 seconds past the max delay allowed to the stop. It could be that small delays would be harder to predict since they might tend to follow patterns of on-time vehicles. Since the majority of delays experienced in this dataset are only minor delays, this might be what is contributing to the difficultly of accurate delay distinction from on-time delays.

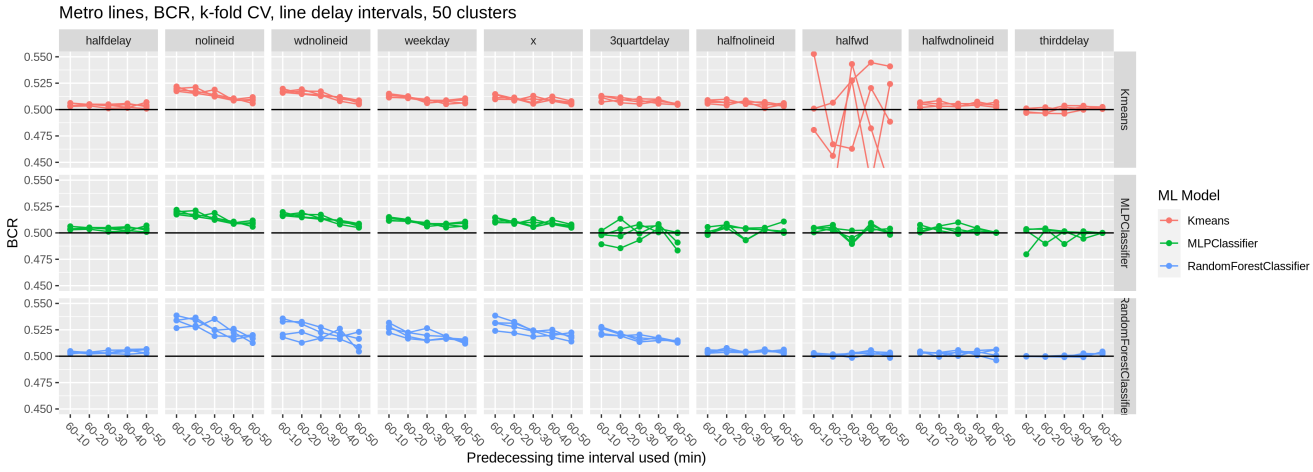


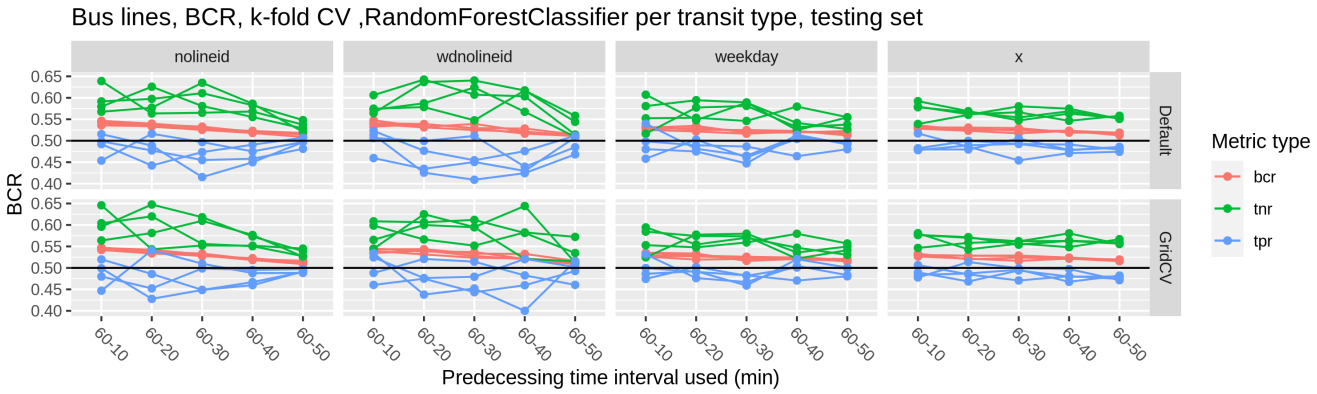
Figure 4.14: Delay classification ML examples

When comparing the GaussianNB and the DecisionTreeClassifier model, the GaussianNB scored a higher BCR on average and performed with a higher accuracy. Other models tested were k-means, MLPClassifier, and RandomForestClassifier. So far the RandomForestClassifier also performs as well as the GaussianNB and can detect the delays at a higher rate than the other models tested.

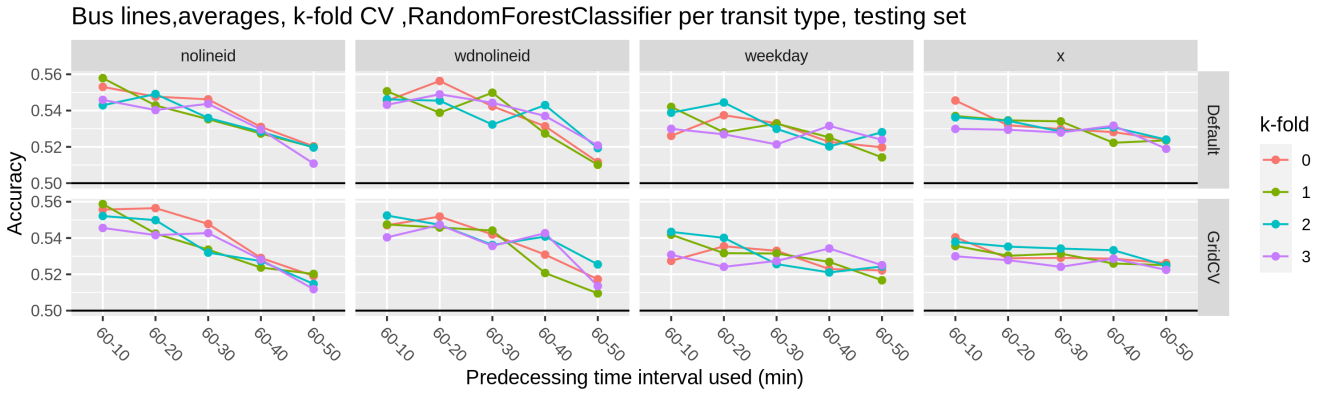
Other adaptions such as removing the transit line id, indicated with "nolineid", and changing the weekday indicator into a binary weekday or weekend indicator, indicated by "weekday" or a "wd", were tested here as well. Removing the use of transit line ids helped slightly in all of the models whereas changing the weekday indicator to a binary one saw no improvement in the models performance. For BCR the RandomForestClassifier has one of the better performances, and of models that have similarly high TPRs this model has a better overall accuracy.

The RandomForestClassifier with line delays, nolineid, weekday, and a combination of the two will be further looked into and have parameters tuned. This model and fea-

ture combinations were selected for further testing because they were the group which had the best BCR and didn't completely miss classifying delayed vehicles. Other models which had a much higher accuracy only scored so much higher because they only predicted all vehicles as on-time and classified only a few hundred or zero delays when there were 10s of thousands. Using the *model_selection.GridSearchCV* [41] method from scikit-learn different model parameters were tested; including *criterion*, *max_features*, *min_samples_split*. The *model_selection.GridSearchCV* method will split the feature vectors into 5-fold cross validation. GridSearch was selected since it would most efficiently allow the testing of many parameters, it also fine tuned the parameters for each time interval set in case the distance in the future a delay is being predicted will affect the best parameters for building a model.



(a)



(b)

Figure 4.15: RandomForestClassifier parameter search with GridSearchCV

Looking at the results from the bus models, the best found model from GridSearchCV and the default RandomForestClassifier have no significant difference in accuracy or BCR. The TPR also did not change significantly in any model variation between the default

and GridSearchCV model.

In the end the best performing features was "wdnolineid", which is the short name for using the weekday and weekend binary features and no transit line id. The best delay intervals to be used were line delays, this was the best choice by a large margin for both metro and trams. The geographical delays gave no significant change to results for buses making line delays an adequate choice for this transit type as well.

When graphing the accuracy of the RandomForestClassifier model, with the feature set "wdnolineid", throughout the day we see that the accuracy is relatively stable. The model starts off at a very high accuracy and drops drastically around the same time morning commuting starts. In the model using the most recent delay intervals, 60-10 minutes prior to the current scheduled stop, the overall accuracy is much higher with less dramatic drops in accuracy. All the time intervals show that in the evenings after 19:00 the model is more accurate at predicting delays, before another small evening slump.

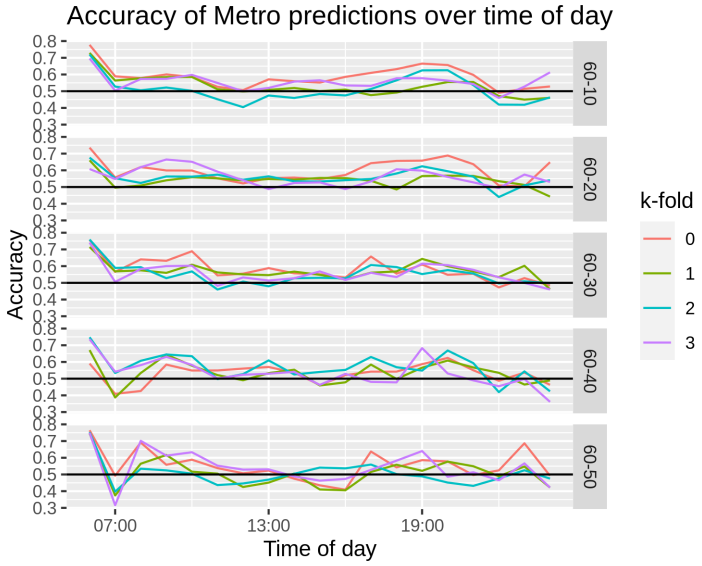


Figure 4.16: Delay classification accuracy over time

The best model was the RandomForestClassifier with default parameters. Other models could have been chosen for their higher accuracy, but this would have lead to an extraordinarily high false negative rate, in some cases not a single delay was accurately predicted at all because every vehicle was classified as on time.

Chapter 5

Mobile Application

In order to bring the previously trained machine learning model to use in the real world a mobile application was created. The goal of this application is to allow users to track STIB-MIVB vehicles in real time and see the predicted delays for future scheduled vehicles. In addition users will be able to view delays statistics and use the app to prepare for potential delays.

5.1 Functionalities

The target audience for this application are passengers of STIB-MIVB in Brussels. For this application to be of interest and useful we must adequately define what functionalities will exist and how they will help the user.

5.1.1 Functional Requirements

FR1 The user is able to view and explore a map of Brussels.

FR2 The user is able to select a STIB-MIVB transit line, and this line and corresponding stops will be displayed on the map.

FR3 The user will be able to view real-time vehicles on the map for selected transit lines.

FR4 The user will be able to click on a stop from the map to view the next coming vehicles for the selected transit line.

FR5 The user will be able to view estimated delays for the future stops of a selected stop from the selected line on the map.

FR6 The user will be able to click on a vehicle and see which line and direction this vehicle is traveling.

FR7 The user will be able to view delay statistics from selected transit lines.

FR8 The remote server will be able to collect and store data from the Vehicle Positions(Real-Time) and GTFS STIB-MIVB API.

FR8 The remote server will be able to calculate transit vehicle delays in real-time.

FR9 The remote server will be able to use a pre-trained machine learning model to predict future delays in real-time.

FR10 The remote server will be able to return vehicle and line information to the mobile application.

5.1.2 Non-Functional Requirements

NFR1 The user has mobile broadband or Wi-Fi connection available on their phones.

NFR2 The user had an Android phone with an OS no older than Android 4.1 Jelly Bean(API level 16).

NFR3 The STIB-MIVB API is running and all real-time vehicle APIs are live and running.

NFR4 The remote server is well documented and expandable for future functionalities.

5.1.3 User interface

A simple design will be used to display the necessary information to the user. Only two tabs will be used which the user will be able to navigate between. Since the main information is based off of transit line and stop selection and the users will most likely be using this app while navigating it would be helpful to have this data on a map. The map will be the main tab and what the user first sees when opening the application. This way the user gets a geographical view of where they are and the line they are interested in, thus allowing them to better envision their trajectory.

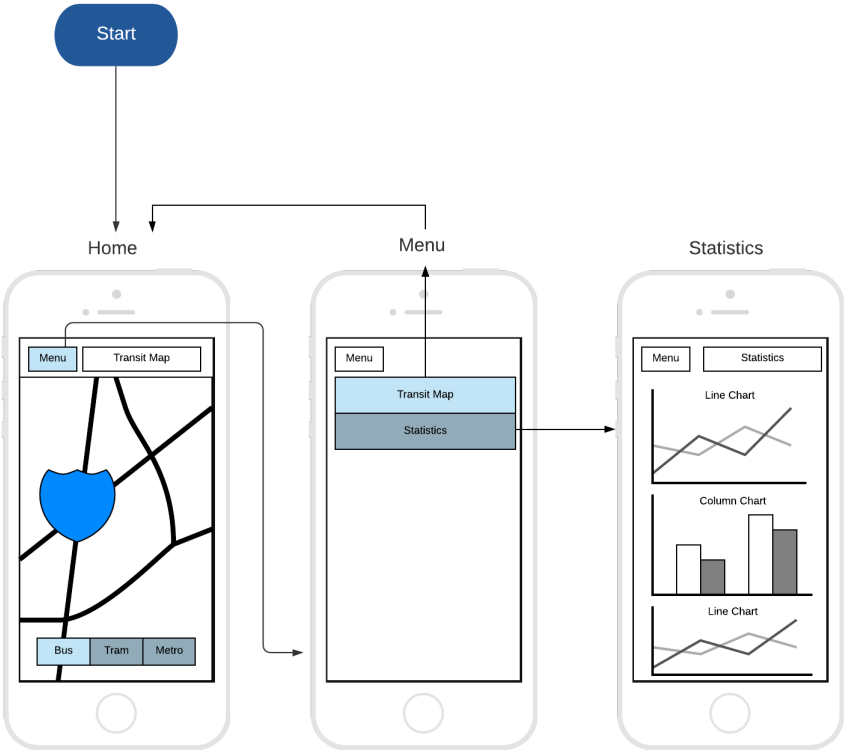


Figure 5.1: Wireframe diagram of main pages

The Figure 5.1 above is the wire frame diagram of what the user will see when they first open the application. It will automatically have opened onto the map page since this will contain the real time line and stop information. From this page the user can click a menu to navigate to the second statistics page or back to the original map page.

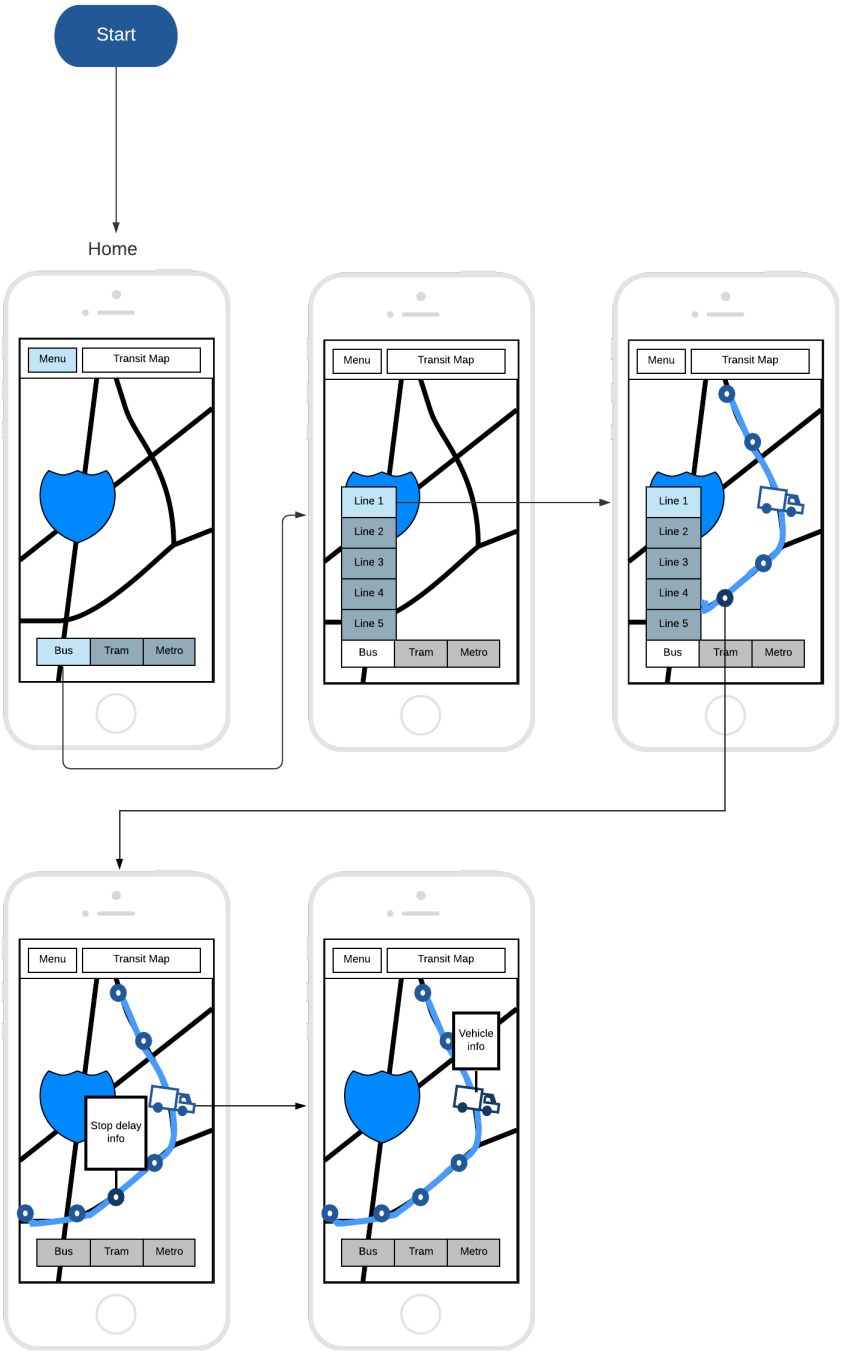


Figure 5.2: Wireframe diagram of map exploration

When the user wants to see delay predictions for future stops they will open the app and start on the main page with the transit navigation map. From here the user will

look down to the bottom of the screen and see three buttons, each labeled for a different transit type: bus, tram, or metro. When the user clicks one of these transit types all of the lines associated to this transit type will appear for the user to select from. When the user selects a transit line from the selected transit type the line will be displayed on the map along with two different types of markers. One marker will mark the vehicle stops which are along the transit line, which the user will be able to click and see the stops scheduled along with delay information. The second will be the vehicles in real-time that are traveling on this line, if clicked they will reiterate the line and direction on the line they are traveling.

5.2 Application Technologies

5.2.1 Android

The mobile application was built for Android mobile operating system in Java[20], using Android Studio 4.0[1]. Android was selected in part because for the past year they are the mobile OS that has the majority market share[32] and since last year has over 2.5 million active users[6]. It is also the OS my mobile phone has and one which I have user experience with. Since my phone runs Android I am able to test locally more easily without using an emulator. The minimum API Level required for the application to run is 16, which corresponds to the Android version 4.1(Jellybean)[19], and the target API level is 29 which corresponds to Android 10[2]. As of May 2020 95.8% of Android users have Jellybean 4.1 or higher[**?**], this allows my application to be usable on the vast majority of Android phones.

5.2.2 Android Studio

Android Studio 4.0 was selected as the IDE used during development. It was developed by both Google and JetBrains; it was based on their IntelliJ IDEA IDE[17], specifically for Android application development. As it is Google's official IDE for Android applications there are also many templates, libraries, resources and tutorials available. There are also many templates to select from in order to get an easy start to development. The two templates which became a starting base for the application were the Navigation Drawer Activity and [3] Google Maps Activity.

5.2.3 Java

Android Studio main languages choices are Kotlin[23] and Java. Kotlin is a relatively new language, first released in 2011, but is already considered by Google to be Androids preferred language for app development.[24] While this may be true and there are noted to be several reason why it should be used over Java, such as eliminating Null references and being able to write shorter code for the same task[27], Java was still chosen. Java is a language that I already have many years experience using and it is the original language of choice for Android development with many more resources and tutorials available.

5.2.4 Maps

For displaying a map in the transit navigation tab a map will be used. For this a library will be needed. The most accessible is the Maps SDK for Android[28]. It automatically handles access to Google Maps servers and allows the addition of markers and polygons to the map. Another option looked at was OpenStreetMaps[33]. This is open source and great if you want to support a collaborative project. For ease of use, Maps SDK for Android for access to Google Maps was selected. It is the map I have the most familiarity using as a user and there are plenty of resources available, for example a Google Maps template in Android Studio. In the end the map is only needed for displaying polygons and markers, so it won't have a large impact on the user experience.
In order to get the access to Google Maps API a key needs to be created on the Google Cloud Platform Console and add it to the *google_ maps_ api.xml* in the Android Studio project.[14] For the features used this will be free of charge.

5.2.5 Graphing

In the statistics section the option of having data displayed in graphs is needed. When looking into the options MPAndroidChart[21] and GraphView[15] were two of the most frequently recommended. Both allow you to use many different types of graphs and are visually appealing. MPAndroidChart was chosen in the end because of a higher popularity in general[29] and since the X-axis on a few of the charts would have to be time, it was the better choice. GraphView was noted as having trouble updating with time as the X-axis.[16]

5.3 Data and Server Technologies

5.3.1 DigitalOcean

DigitalOcean is the remote server which was being used for the initial data collection and since the server is already set up and has a majority of the packages needed installed it will continue to be used for the mobile application. It will be useful to have a single central location collecting data and creating delay predictions. Since collecting the vehicle data can consume a lot of space and prior delays are needed in order to make predictions it is better this is not done on the users mobile phone. This will allow for faster predictions and data manipulation where the only thing the user has to do is to make a selection and the requested data will be copied from the server and sent to the users application.

5.3.2 MongoDB

MongoDB is also used as the infrastructure for data storage and manipulation on the server. The code used during the machine learning portion is also reliant on the use of MongoDB and this code will be recycled and adapted for integration with the application.

5.3.3 Python3

As with the machine learning aspect of the project, data manipulation and collection will be done with Python3. Since the machine learning models were testing, trained, and built using Python3 and associated libraries it is a good idea to keep this the same and continue using the same results for the application. Several libraries are used:

flask[48]	Enables the creation of an endpoint of which the mobile application can make requests to the server for data retrieval.
pymongo	This is used to connect and interact with the MongoDB through python3.
requests	HTTP library which allows the server to make HTTP requests to the STIB-MIVB API for transit data collection and the mobile app uses to send GET requests to the server.
sklearn	Machine learning library used for the initial creation and usage of the trained model used for predicting delays.

5.4 Architecture

As seen above in the below Figure 5.3 there are two main components to the mobile application: the mobile app and the remote server. A server was created on DigitalOcean to run as a remote database and API for the mobile application to retrieve data from. This remote server functions to store and retrieve data from STIB-MIVB which will then be displayed to the user.

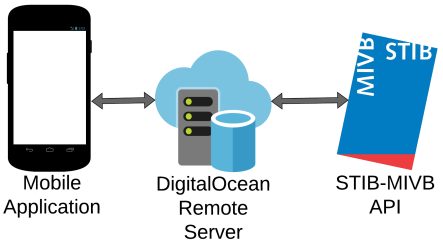
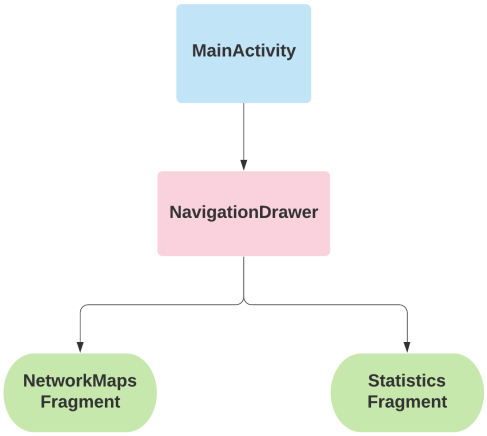


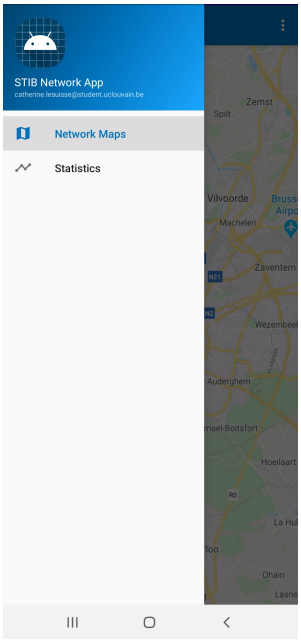
Figure 5.3: STIB-MIVB Mobile Application Architecture

5.4.1 Application structure

The Mobile application has two main pages which will need to be displayed to the user: the transit network map and the statistics page. The base of the application construction is a NavigationDrawerActivity with two fragments: transit Network Map and the Statistics page. The Network Map fragment doubles as the home page as well. See the Figure 5.4 below.



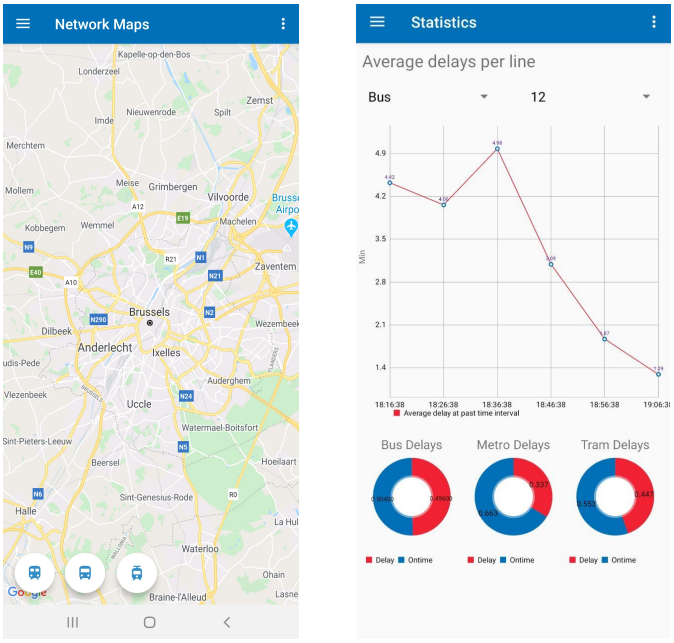
(a) Mobile app activity and fragments



(b) Mobile app navigation drawer

Figure 5.4: Mobile app navigation

The Network Map fragment has three buttons for each of the transit types to allow the user to easily find the line they are looking for. In the statistics fragment the user can scroll down to see multiple graphs displaying real time delay statistics from the STIB-MIVB network. For several of these graphs the user is able to select a transit type and transit line to view specific data for it.



(a) Mobile app Network Map

(b) Mobile app Statistics

Figure 5.5: Mobile app fragments

Mapping in Android

Mapping in accomplished using Maps SDK for Android. This gives use access to Google Maps and all the associated features. To use the gradle needs to be updated to contain the corresponding dependency and an API key linked to the app.

The NetworkMaps XML contains a fragment to display the map on the entire screen:

```
<fragment
    xmlns:android="http://schemas.android.com/apk/res/android"
        xmlns:map="http://schemas.android.com/apk/res-auto"
        xmlns:tools="http://schemas.android.com/tools"
        android:id="@+id/map1"
        android:name="com.google.android.gms.maps.SupportMapFragment"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        tools:context=".MapsActivity" />
```

Listing 5.1: GoogleMaps fragment

The NetworkMaps java class extends Fragment implements OnMapReadyCallback in order to allow addition of polygons and markers to the map. In addition to the basic map and fragment functions: such as onMapReady(), onViewReady(), or onView(). Additional functions like getSTIBPositions() were created to allow calls to the server for data collection.

Graphing in Android

Graphing was done using the MPAndroidChart. This is an open source graphing library, to use the gradle needs to be updated to contain the corresponding dependency. The Statistics XML contains a new fragment for each graph :

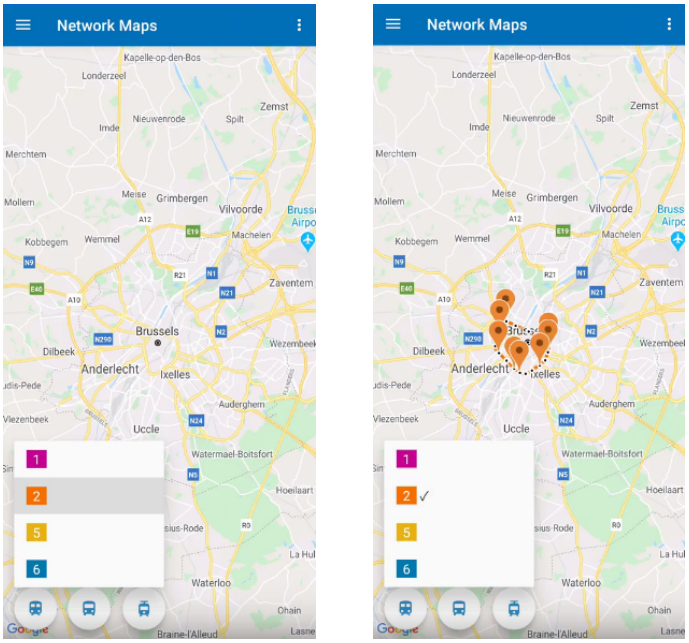
```
<com.github.mikephil.charting.charts.PieChart
        android:id="@+id/piechart_busdelay"
        android:layout_width="130dp"
        android:layout_height="150dp"
        />
```

Listing 5.2: MPAndroid Chart Piechart example

In the Statistics java class a getData() function was created to make requests to the remote server for statistics data to be displayed. The data is added and graphed here in the onCreateView() function where the line chart is updated via a OnItemSelectedListener.

5.4.2 Network Map and interaction with remote server

In the MainActivity when the app is first opened the first thing that is loaded is the transit ids and associated colors, these are received from the remote sever through a GET request. The app then opens up onto the Network Maps fragment. This prompts the app to send a GET request for the lines that are available for transportation on each transit type.



(a) Transit line selection

(b) Transit line selected and displayed on map

Figure 5.6: Mobile app Network Map

In the below Figure 5.7 the sequence diagram describes how the different layers of the application interact with each other and the remote server to complete the task of showing the user the selected transit line on the Network Map. The Network Maps fragment makes a GET request to the remote server for each time a transit line is selected to collect the transit line shape and the real-time vehicle positions. The shape of the line is stored in the app afterwards and if the user de-selected and re-selected the same line it won't have to make a second call to the server. The user is able to select multiple transit lines from different transit types and display them on the map simultaneously.

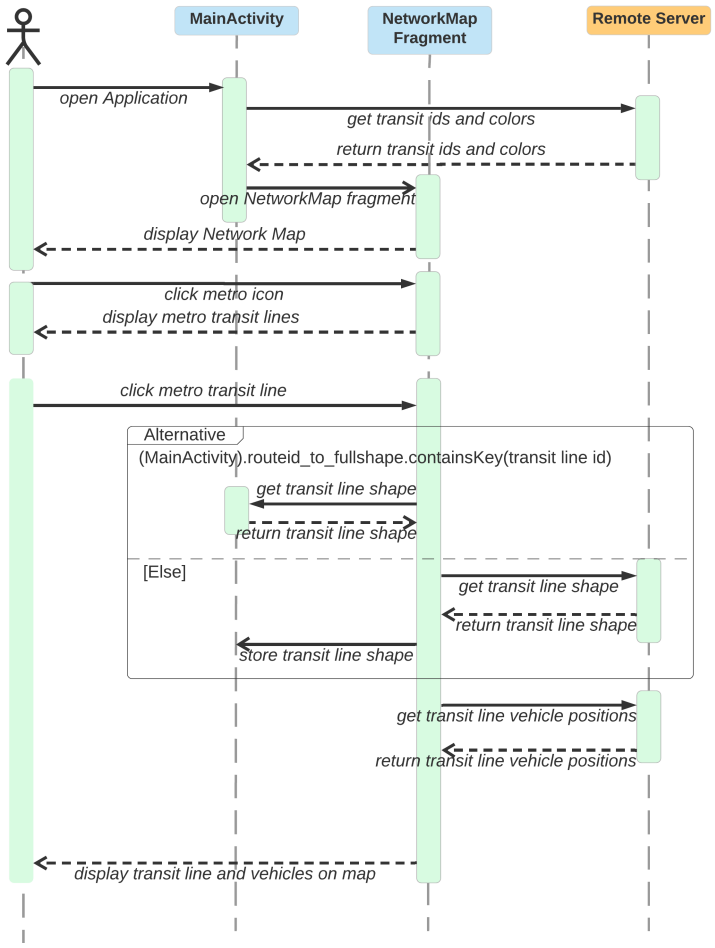
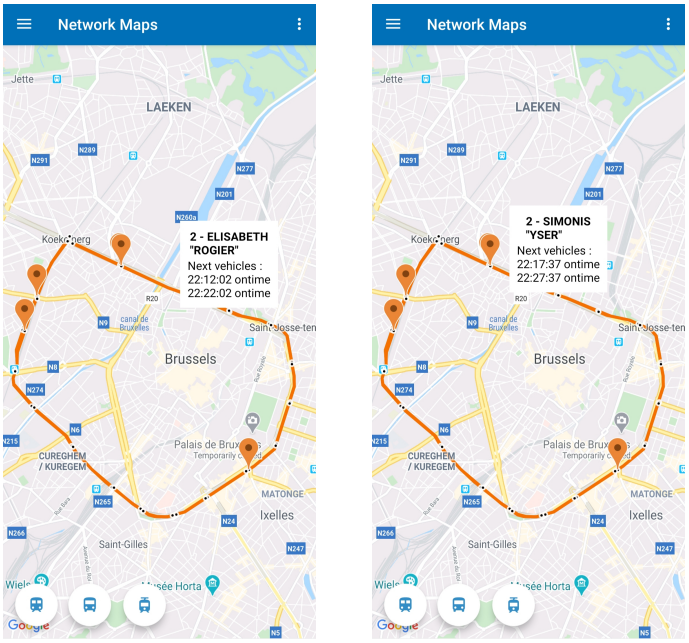


Figure 5.7: Network Map line selection Sequence diagram

When the user clicks on a vehicle the line id and the direction of travel is displayed to the user. When a transit stop along a line is clicked the next scheduled stops in the next 50 minutes are displayed along side their scheduled time and the delay prediction. The delay prediction is indicated by text beside the schedualed time as "delayed" or "on time".



(a) Metro line 2 direction Elizabeth, Rogier

(b) Metro line 2 direction Simonis, stop Yser

Figure 5.8: Mobile app Network Map transit line stop selected

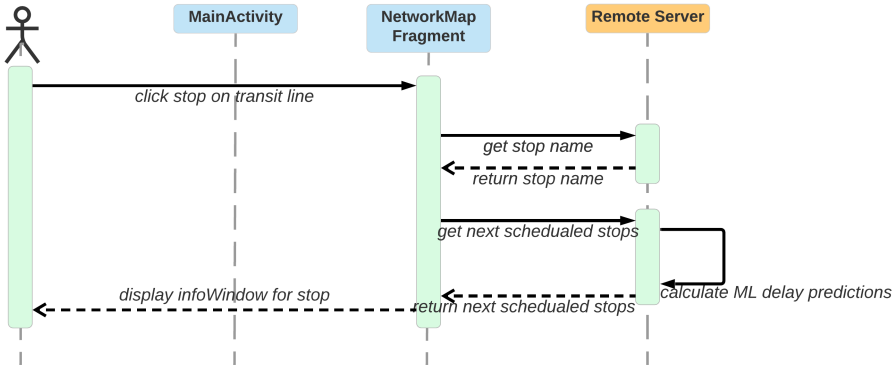
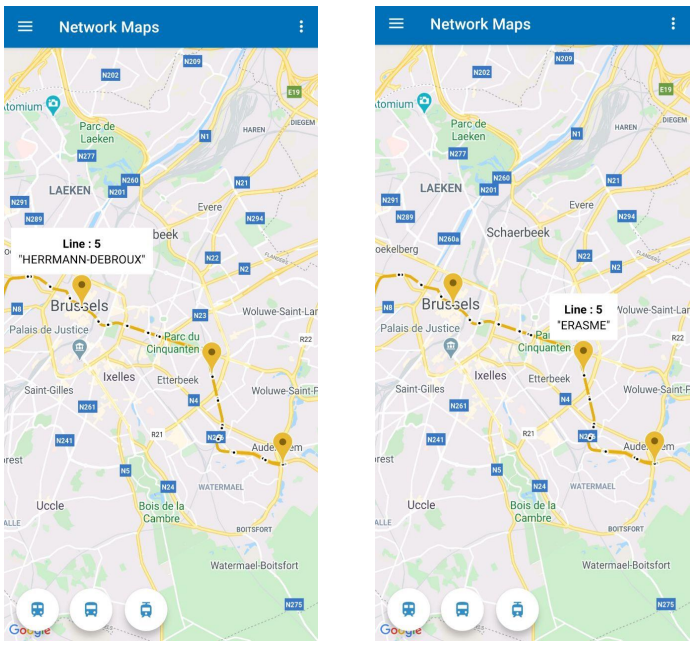


Figure 5.9: Network Map stop selection Sequence diagram

The above sequence diagram shows how the app would interact with the server in order to get the next scheduled stops for the line. The string name would be received first because this isn't stored in the app. It would be too time costly to grab every piece of line information at once when the user doesn't need it at the moment, so objects like names are called in moments like this. Once the name is grabbed it will call for the next scheduled stops along with their predicted delay. The predicted delay is calculated with

the machine learning model previously built in the chapter "Delay Classification Model". All of this information will be formatted in a custom InfoWindowAdapter which is then displayed to the user.



(a) Metro line 5 direction Herrmann-Debroux

(b) Metro line 5 direction Erasme

Figure 5.10: Mobile app Network Map transit vehicle selected

Above is an example of clicking a vehicle on a displayed line. Each vehicle will display the line they are a part of and the direction in which it is traveling. In the example metro line 5 is used. In image (a) the metro is traveling east bound towards Herrmann-Debroux and in image (b) the selected metro vehicle is traveling east bound towards Erasme.

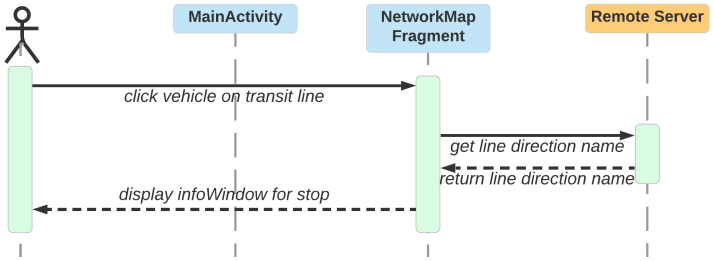


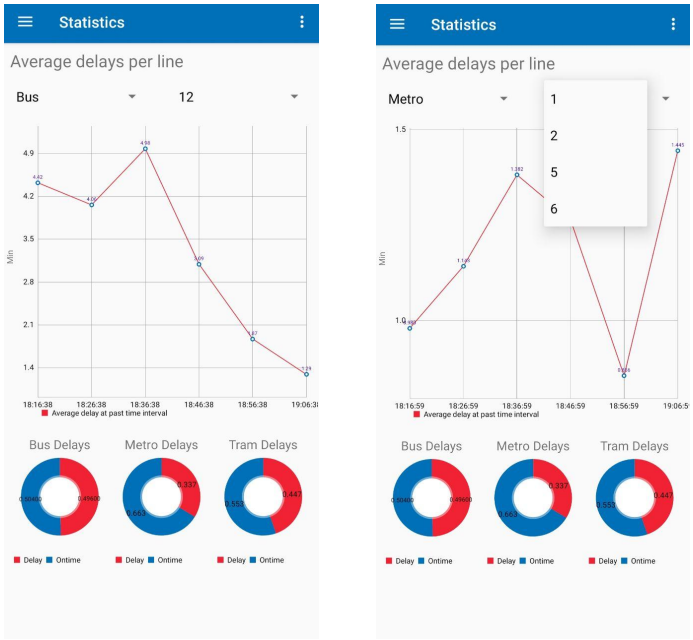
Figure 5.11: Network Map vehicle selection Sequence diagram

The Figure 5.11 above is the sequence diagram which shows how the NetworkMap

fragment interacts with the remote server in order to get the name of the line direction. The line direction is encoded as a transit stop ID and since it is too time and space consuming to store all the stop names, like in stop selection, the stop text names are called on as needed.

5.4.3 Statistics and interaction with remote server

The second fragment of the application is the statistics page. The top graph is a real time delay average for all the different lines in the network. The user can select a transport type and then a line to view how delayed this line is in the last hour. Each tick on the x axis is the time 10 minutes prior from the current time going back one hour. The y axis shows the average delay on the line for that last 10 minute interval. The pie charts below give the ratio of on-time versus delayed stops for each line.



(a) Bus line 12 10 minute interval delay averages

(b) Metro line selection

Figure 5.12: Mobile app Statistics

Below we can see a sequence diagram showing how the Statistics fragment interacts with both the MainActivity and the remote server to display transit statistics to the user. By default the app will automatically display the first line in the list of the first transit type listed. In this case it is bus line 12, as seen above in Figure 5.12 (a). The average delay statistics are requested from the remote server and returned to be displayed to the user. The user could then select a different transit type which prompts the Statistics fragment to get the new transit id list from the MainActivity. Once the new transit type

is selected the user can select a new line to be graphed. This will send another request to the remote server for the corresponding delay averages, and be subsequently displayed to the user.

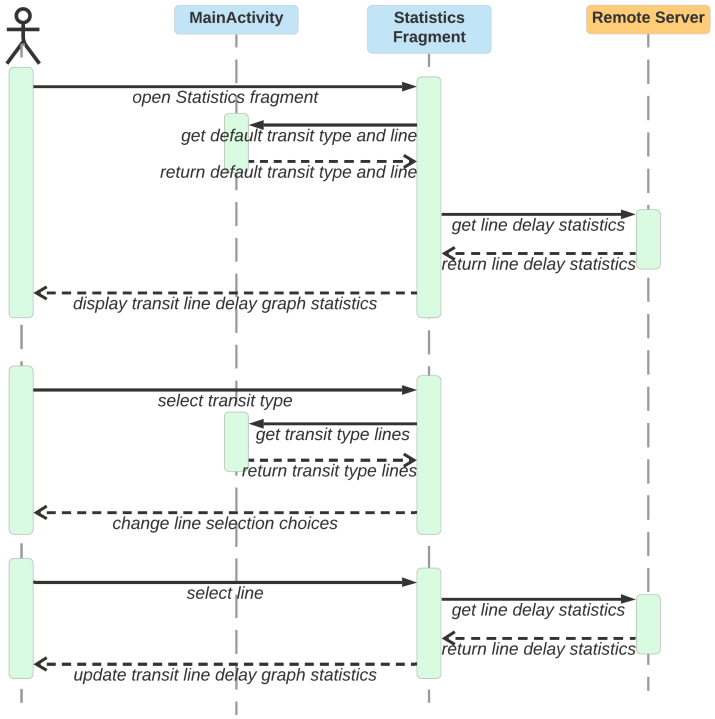


Figure 5.13: Statistics Sequence diagram

5.4.4 Remote server data collection

The server was set up similarly to the one which was used for data collection during the machine model building process, see Chapter 4, and is running on the same DigitalOcean droplet previously used. The *STIB_API.py* was recycled for this purposed with minor alterations to allow the mapping of vehicles to a schedule in real time instead of in data file chunks as it was done post-data collection previously.

Threading

There are four groups of threads running while the server is up. These tasks are separated so delay calculation can take priority and the work needed to keep delay calculation going and creating statistics are done by secondary threads.

Main thread	Collects real time Vehicle Positions from STIB-MIVB and calculates their delays.
Updating access tokens	Updates the access tokens required for API access to STIB-MIVB data, these expire after 3600 seconds and need to be automatically remade in order to have continuous access.
Updating GTFS files	Pulls from the STIB-MIVB a new GTFS zip file everyday and checks if it contains new information and needs to replace the current GTFS data set.
Average delay updating	Updates and creates the new interval of delays statistics every 10 minutes.

There are 5 tokens used for making requests to the STIB-MIVB API. This is because each token has a limited number of calls per minute which can be done before access for the token is temporally frozen. To avoid this there is a rotation of 5 tokens used for all requests, this way the load is spread more equally and there is less risk of getting a request rejected. These tokens are all continuously updated by threads every 3600 seconds, which is the lifetime for the tokens. The thread to start access token updating is the first to get launched. The next thread to get launched is the GTFS file collection, this thread will collected the GTFS zip from STIB-MIVB every morning and check to see if there is any new information and if the server's mongoDB database needs to use the new set. At the same time the *todays_stop_times* collection will be reset with the new day's scheduled stops. The average delay over each transit line is created every 10 minutes and is controlled by a separate thread. The main thread is in charge of collecting the Vehicle Positions for each transit line and calculating their delay and storing this value.

On resources shared by multiple threads, such as the access tokens or delay interval list, locks are used. This helps prevent the GTFS file updater or main thread from using an expired access token and having their process delayed.

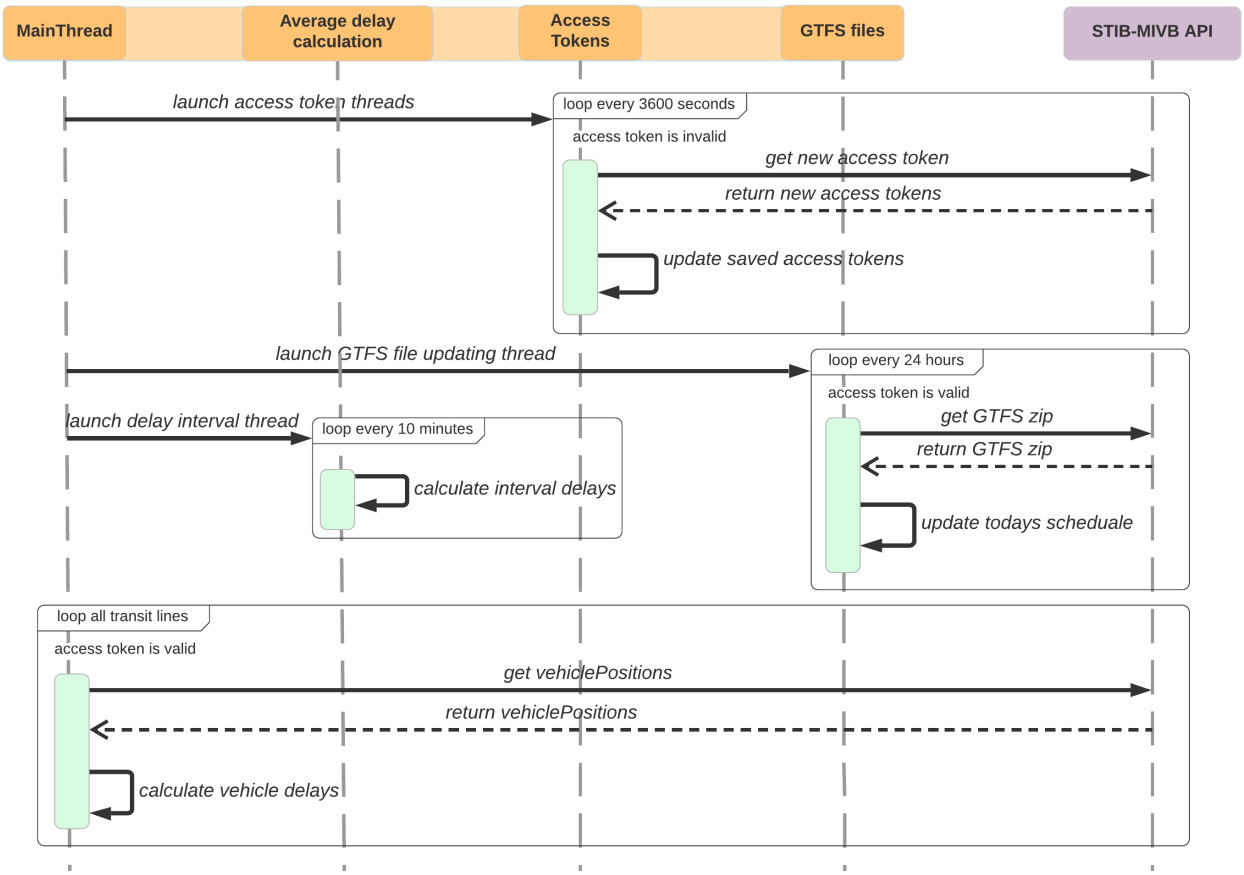


Figure 5.14: Remote server thread sequence diagram

Database

MongoDB is used to store the GTFS files and calculate the delays. The same *STIB_API* script which was used in the machine learning to calculate delays was used here, with minor adjustments made to process mapping vehicles to the schedule one at a time instead of an entire day at once.

Data requests

Data requests are sent to the server via an HTTP request, using Python3 and Flask an endpoint was set up to handle this. When the user clicks on a stop, a request will be sent to the server asking for the next vehicles which are scheduled to arrive. This request will be formatted as below in the mobile application.

```
1 HttpURLConnection con;
2 con = (HttpURLConnection) new
    URL("http://104.248.250.39:5000/return_stopid_to_nextVehicles?stopid="
```

```
    + stopid + "&ids=" + lineid).openConnection();
con.setRequestMethod("GET");
con.setRequestProperty("Accept", "*/*");
con.addRequestProperty("Content-Type", "application/json");
```

Listing 5.3: Java HttpURLConnection GET request example

The endpoint will then route the request to the appropriate method and parse the variables to return the next scheduled stops to the mobile user.

```
@api.route('/return_stopid_to_nextVehicles', methods=['GET'])
def return_stopid_to_nextVehicles():
    stopid = request.args['stopid']
    ID = request.args['ids']
    return endpoint.return_stopid_to_nextVehicles(stopid, ID)
```

Listing 5.4: Flask GET request handling example

Chapter 6

Future Improvements

After the completion of the project I've been able to reflect on what has already been accomplished and what could be changed or added to the project for further exploration. For the machine learning models new features could be looked into and training a regression model. In the mobile application the variable storage could be more efficient and the user-interface more friendly.

6.1 Machine learning model

Another model to create for this project would be to train a regression machine learning model. In the regression model the predicted output would be the number of minutes delayed.

Changing the time cutoff to indicate when a delay has occurred might be interesting to look into. If it would be easier to predict delays when only more significant delays were registered as such the models might be able to predict with a better accuracy and BCR. The current definition was quite strict with a 1.5 minute delay maximum to constitute as on-time. Since many vehicles have only a minute delay, many of these delayed vehicles follow a similar delay pattern as their on time counterparts. This shift would allow more leniency as to what constituted as an on-time vehicle.

It is possible that weather has a significant impact on delays, maybe there are more delays because of cautious driving during rainfall or less delays because of less traffic? Perhaps a sunny day will bring out more people than normal and cause an unusual fluctuation in ridership and thus cause delays?

Looking into delays for academic periods versus academic holiday would be another area to explore. Would students traveling to and from home for classes create a large influx of riders? During the holiday period they will not be attending classes and a significant reduction of riders is expected.

6.2 Mobile application

There are several things which could be done to make the application more aesthetically pleasing and user-friendly. One would be to implement a real cache in the app instead of only storing java variables while the app is running. Storing java variables is a very short term local storage solution as this gets reset whenever the application is reopened. This way the user doesn't have to re-request data if it hasn't expired yet if they close the application and re-open it only a short while later.

Since Brussels is a bilingual city it would be good to have a language feature where the user could change the application languages to French/Dutch. Currently the application is using English on the user-interface with French transit and stop names.

Another point to make mapping of vehicles more fluid on the map would be to utilize the *distanceFromPoint* feature during Vehicle Position collection. This feature tells approximately how far away from the previously passed stop the vehicle is located. It would help for moving a vehicle more smoothly along a line instead of being anchored to transit stop locations only.

Chapter 7

Conclusions

In this thesis I analyzed the STIB-MIVB transit network by focusing on tracking a subset of lines from each of the three transit types: buses, trams and metros. On average I found that metros contained the least delays and buses were delayed at the highest frequency. At the same time metros had the smallest delay average and the buses had the largest delay average. This was most likely influenced by the fact that buses tend to have a larger time gap between vehicle arrivals at a stop for a specific line and that they run on different surface types. Looking at the statistics it was concluded that it would be a wise idea to separate each transit type into its own model since magnitude of delay varies depending on the transit type and each have different variables which affect possibles delays, such as street traffic delaying buses. Street traffic doesn't affect metros and trams to the same degree and we have no way to account for this in the model when distinguishing the transit types.

Two methods were created to utilize delays as features, one using delays only in the same line, and a second using delays of stops in the same geographical range. Once we got to feature selection an initial analysis was done using the feature importance value calculated from the DecisionTreeClassifier and RandomForestClassifier models. This showed us that geographical delays played a slightly larger role in tree creation than the line delay counter parts. It also showed us that line id and weekday were the least important but that time of the day was an important feature when predicting delays.

When moving onto model selection it was seen that the type of delay interval helpful for producing a higher accuracy and BCR was dependant on the type of transit being modeled. Using geographical delays severely hurt the prediction of metro and tram delays to a lesser extent. This is thought to be because these transit types are isolated form other traffic and run on separate tracks, so they are uninfluenced by outside traffic. Unlike buses which in only a few select lines saw a tiny increase in accuracy using geographical delays, but overall the difference between line and geographical for buses was negligible. The final model selected was a RandomForestClassifier using line delays with weekday binary indicator and no line ids. It was difficult to get a high accuracy on the model without compromising the BCR, because a majority of delays follow a similar delay pattern as on time vehicles and a distinction is very difficult to capture.

An Android mobile application was then created to allow the user to view transit vehicles and see what predicted delays are for future scheduled stops. In addition to delay prediction the user is able to use it as a STIB-MIVB map and view multiple lines simultaneously. A remote server was created for remote computation of delays and data collection so results can be ready at any moment when the user makes a request. The user is also able to view simple delays statistics from the past hour for each transit line.

In conclusion a pipeline was created to collect vehicle positions and calculate vehicle delays which were then used to create a classification ML model. The model was utilized in a mobile application for real time use. A remote server was created to collect data and predict delays in real time and communicate this data to the mobile application on the users request.

Bibliography

[1] “Android Studio Release Notes: Android Developers.” Android Developers, 2020, developer.android.com/studio/releases4-0-0.

[2] “Android 10 : Android Developers.” Android Developers, developer.android.com/about/versions/10.

[3] “Add Code from a Template : Android Developers.” Android Developers, developer.android.com/studio/projects/templates.

[4] Amazon Web Services, aws.amazon.com/.

[5] Berger, Annabell, et al. "Stochastic delay prediction in large train networks." 11th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2011.

[6] Brandom, Russell. “There Are Now 2.5 Billion Active Android Devices.” The Verge, The Verge, 7 May 2019, www.theverge.com/2019/5/7/18528297/google-io-2019-android-devices-play-store-total-number-statistic-keynote.

[7] “Clustering Algorithms | Clustering in Machine Learning.” Google, Google, developers.google.com/machine-learning/clustering/clustering-algorithmscentroid-based-clustering.

[8] “collection.” Glossary - MongoDB Manual, docs.mongodb.com/manual/reference/glossary/term-collection.

[9] “Data Access.” Opendata.stib, opendata.stib-mivb.be/store/data. PFD file.

[10] DigitalOcean, www.digitalocean.com/.

[11] “document.” Glossary - MongoDB Manual, docs.mongodb.com/manual/reference/glossary/term-document.

[12] Dignan, Larry. “Top Cloud Providers 2019: AWS, Microsoft Azure, Google Cloud; IBM Makes Hybrid Move; Salesforce Dominates SaaS.” ZDNet, ZDNet, 15 Aug. 2019, www.zdnet.com/article/top-cloud-providers-2019-aws-microsoft-azure-google-cloud-ibm-makes-hybrid-move-salesforce-dominates-saas/.

[13] “GTFS Static Overview | Static Transit | Google Developers.” Google, Google, developers.google.com/transit/gtfs/.

[14] “Get an API Key.” Google, Google, developers.google.com/maps/documentation/android-sdk/get-api-key.

[15] Gehring , Jonas. “jjoe64/GraphView.” GitHub, 5 Apr. 2019, github.com/jjoe64/GraphView/wiki.

[16] “Graphs in Android: AChartEngine vs GraphView.” Stack Overflow, 4 Aug. 2015, stackoverflow.com/questions/31174183/graphs-in-android-achartengine-vs-graphview.

[17] “IntelliJ IDEA: The Java IDE for Professional Developers by JetBrains.” JetBrains, www.jetbrains.com/idea/.

[18] Jain, Anil K. “Data Clustering: 50 Years beyond K-Means.” Pattern Recognition Letters, vol. 31, no. 8, 2010, pp. 651–666., doi:10.1016/j.patrec.2009.09.011.

[19] “Jelly Bean : Android Developers.” Android Developers, developer.android.com/about/versions/jelly-beanandroid-4.1.

[20] Java, 28 May 2019, java.com/en/.

[21] Jahoda, Philipp. “PhilJay/MPAndroidChart.” GitHub, 20 Jan. 2020, github.com/PhilJay/MPAndroidChart/wiki.

[22] Kim, Hae-Young. “Analysis of variance (ANOVA) comparing means of more than two groups.” Restorative dentistry endodontics vol. 39,1 (2014): 74-7. doi:10.5395/rde.2014.39.1.74

[23] “Kotlin Programming Language.” Kotlin, kotlinlang.org/.

[24] Lardinois, Frederic. “Kotlin Is Now Google’s Preferred Language for Android App Development.” TechCrunch, TechCrunch, 7 May 2019, techcrunch.com/2019/05/07/kotlin-is-now-googles-preferred-language-for-android-app-development/?guccounter=1.

[25] Marchant, N. (2016). Prédiction des temps de trajet dans un réseau de bus à l’aide de données historiques. Retrieved from https://github.com/C4ptainCrunch/info-f-308/blob/master/rapport/delay.pdf

[26] Meeûs, Brieuc de. “Activity Report 2018.” STIB-MIVB, 2019, 2018.stib-activityreports.brussels/en.

[27] Misal, Disha. “5 Reasons Why Developers Choose Kotlin Over Java.” Analytics India Magazine, 2 May 2019, analyticsindiamag.com/5-reasons-why-developers-choose-kotlin-over-java/.

[28] “Maps SDK for Android.” Google, Google, June 2020, developers.google.com/maps/documentation/android-sdk/intro.

[29] “MPAndroidChart vs GraphView.” LibHunt, android.libhunt.com/compare-mpandroidchart-vs-graphview.

[30] MySQL, www.mysql.com/.

[31] “Operation Monitoring - Vehicle Position.” Opendata.stib, www.stib-mivb.be/irj/go/km/docs/resource/OpenData/9529ab42-f3fc-4960-bf21-703db6d6cc57.pdf. PDF file.

[32] “Operating System Market Share Worldwide.” StatCounter Global Stats, gs.statcounter.com/os-market-sharemonthly-201906-202006.

[33] OpenStreetMap, www.openstreetmap.org/about.

[34] “PyMongo 3.10.1 Documentation.” PyMongo 3.10.1 Documentation - PyMongo 3.10.1 Documentation, pymongo.readthedocs.io/en/3.10.1/.

[35] “Release History - 2.18.4.” Community Updates - Requests 2.24.0 Documentation, requests.readthedocs.io/en/master/community/updates/.

[36] Stratec, www.stratec.be/.

[37] “Sklearn.tree.DecisionTreeClassifier.” Scikit, scikit-learn.org/stable/modules/generated/sklear n.tree.DecisionTreeClassifier.html.

[38] “Sklearn.ensemble.RandomForestClassifier.” Scikit, scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html.

[39] “Sklearn.svm.SVC.” Scikit, scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html.

[40] “Sklearn.feature_selection.RFE.” Scikit, scikit-learn.org/stable/modules/generated/sklearn.feature_selection.RFE.html.

[41] Sklearn.model_selection.GridSearchCV. (n.d.). Retrieved August 06, 2020, from https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.htmlsklearn.model_selection.GridSearchCV

[42] “The Most Popular Database for Modern Apps.” MongoDB, www.mongodb.com/.

[43] Tharwat, Alaa. “Classification Assessment Methods.” Applied Computing and Informatics, 2018, doi:10.1016/j.aci.2018.08.003.

[44] Triggs, Robert. “Android Version Distribution: Are Google’s Faster Rollout Initiatives Working?” Android Authority, 24 May 2020, www.androidauthority.com/android-version-distribution-748439/.

[45] “Version 0.23.0.” Scikit, 2020, scikit-learn.org/stable/whats_new/v0.23.htmlversion-0-23-0.

[46] Varangaonkar, Amey. “Why MongoDB Is the Most Popular NoSQL Database Today.” Packt Hub, 26 Mar. 2018, hub.packtpub.com/mongodb-popular-nosql-database-today/.

[47] WELCOME TO OUR OPEN DATA PORTAL. 2018, opendata.stib-mivb.be/store/.

[48] “Welcome to Flask.” Welcome to Flask - Flask Documentation (1.1.x) - Version 1.1.1, flask.palletsprojects.com/en/1.1.x/changelog/version-1-1-1.

[49] Yu, Bin, William HK Lam, and Mei Lam Tam. "Bus arrival time prediction at bus stop with multiple routes." Transportation Research Part C: Emerging Technologies 19.6 (2011): 1157-1170.

UNIVERSITÉ CATHOLIQUE DE LOUVAIN
École polytechnique de Louvain
Rue Archimède, 1 bte L6.11.01, 1348 Louvain-la-Neuve, Belgique | **www.uclouvain.be/epl**