Norwegian University
of Life Sciences

Master's Thesis 2021 30 ECTS
Faculty of Science and Technology

Prediction of passenger load on busses in Oslo using data from Automatic Data Collection-systems

Johanne Krokene Jarmund
Masters of Science Environmental Physics and Renewable Energy

Preface

This thesis represents the final work of a 5 year masters program at NMBU (Norges miljø og biovitensaplige universitet or Norwegian institute of life sciences). It is a 30 ECTS credit thesis written in collaboration with the Data Science department at Ruter, the public transport provider in Oslo, Norway.

I would like to give a big thanks to my advisors at NMBU, Associate Professor Kristian Hovde Liland and Associate Professor Oliver Tomic at REALTEK. Both for their excellent advise on the thesis, but also their encouragement while writing a thesis during ”these strange times”.

I would also like to give a big thanks to my colleagues at Ruter, in particular Umair M. Imam and Jostein-André Nordmoen for making the collaboration possible in the first place. In addition I would like to thank Daniel Haugstvedt, Jawad Saleemi and the rest of the Data Science-team for all their input and encouragement. I had a great time working with all of you.

Finally I would like to give a big thanks to all my friends and family for all the support. I promise I will start talking about other things than passenger counts soon.

Oslo, 31st May, 2021

Johanne Krokene Jarmund

Abstract

Public transport is key to reducing the usage of private vehicles, and by extension carbon emission in urban areas. Ruter is responsible planning and coordinating public transport in Oslo. Through different Automatic Data Collection-systems (ADC-systems) they have access to data about the performance of all vehicles in operation.

In this thesis we explore the possibility of using data from Automatic Vehicle Location- and Automatic Passenger Counting-systems in order to predict passenger load on busses in Oslo. Predictions of load can be used by passengers when planning a trip, who may choose a departure where the predicted load is lower. This can serve a dual purpose, giving the passenger a more pleasant trip, but also reducing the pressure on public transport by encouraging a better distribution of the load. Predictions of load can also be used by those monitoring public transport, helping inform decisions when trying to resolve incidents affecting public transport.

Two operation situations are explored in this thesis, one where predictions are only based on plan-data, and one where real-time location-data is included. For the first operation situations the model with best performance yielded a mean absolute error (MAE) in predicted passenger load of 7.10, providing a reasonable prediction of load when no major delays or other factors were affecting the flow of traffic. Models developed for the second operation situation managed to account for differing passenger behaviour caused by deviations in planned trips. The best performing model in this situation had a MAE of 6.26.

ADC-systems for public transport are complex systems with many potential sources of error. Emphasis it therefor put on how to prepare data for analysis. A machine learning method, isolation forest, is used for automatic detection of trips with erroneous data. This method is compared to manual screening based on observed fallacies on the data, with the result that model performance were slightly better when models were trained on data screened using isolation forest.

Contents

List of Tables

List of Figures

Listings

Chapter 1

Introduction

1.1 Background

Private cars represent a large source of pollution in urban areas like Oslo, and with a growing population it is important for public transport to appear attractive to passengers. In may 2020, Oslo adopted a new climate strategy, aiming for the city to reduce its greenhouse gas emission to almost zero by 2030. This strategy states that vehicle-transport shall be reduced by 20% by 2023 and by 1/3 by 2030 [1], and that the preferred modes of passenger transport should be public transport, walking or bicycling. The aim is to reduce greenhouse gas emissions, air pollution and noise.

Prioritizing public transport by making it more affordable and expanding the service is expected to aid in this. In order for public transport to be the first choice for all passengers, the service also needs to be perceived as both reliable and comfortable compared to private vehicles. One of the factors affecting how comfortable travel by public transport is the level of passenger crowding.

1.1.1 Ruter

Ruter is the publicly owned company tasked with planning, coordinating, ordering and marketing the public transport system in and around Oslo [2]. Their aim is to do so in an environmentally friendly fashion, while also providing convenience for the inhabitants of Oslo. According to the yearly report from Ruter in 2020 [3], 240 million trips were made by public transport in 2020. This number was not surprisingly strongly affected by the covid-pandemic, and represented a 40% decrease from the previous year. In 2019 398 million trips were made by public transport in Oslo, after a steady increase over the past years.

Ruter#

Figure 1.1: *Logo for Ruter, the public transport provider in Oslo.*

Ruter stated in 2020 that their long term vision is *bærekraftig bevegelsesfrihet*, roughly translated to sustainable freedom of movement. In the presentation of their vision, ”Målbilde for bærekraftig bevegelsesfrihet” [4], the emphasis is on an inclusive and sustainable approach to public transport. In order to accomplish this, data-driven solutions are highlighted as important tools. That is, solutions where decisions are based on data and not personal experience or intuition. Both *artificial intelligence* and *machine learning* is expected to change the way public transport operate in the future.

The desire for data-driven solutions is fueled by large amounts of data becoming available. By using data gathered through user-interactions, such as ticket purchases or route-planning, a more personalized experience can be tailored for the end user. There is also a desire to use existing data-sources in new, novel ways. In particular the existing system for monitoring traffic in real time, such as APC- and AVL-systems discussed in the next section, are of interest. This has the potential to allow more rapid adjustments to situations in real time, and may even help better prepare for situations in the future.

1.1.2 APC- and AVL-systems

In public transport it is important to know how vehicles, and by extension passengers, are moving through the system. To this end, systems for *Automatic Data Collection (ADC)* are installed on all vehicles in operation. ADC-systems can encompass many different technologies, for example *Automatic Passenger Counting (APC)* for keeping track on how many passenger there are aboard a vehicle. Vehicles can also be equipped with *Automatic Vehicle Location-systems (AVL)*, providing real-time information about the location of the vehicle when it is in operation. By combining data collected through both the APC- and AVL-systems, it is possible to gain detailed information about the passenger-flow across the city.

Automatic Fare Collection (AFC) can also be an important part of the data collection system. Different cities have different methods of fare collection. In Oslo the primary method for ticket purchases are through the mobile app *RuterBillett*. Alternatively passengers can use smart cards loaded with either a ticket or credits that allow you to pay as you go. Compared to other fare-collection systems these alternatives provides little information about the intended travel-pattern of a passenger. When purchasing a ticket in-app, the passenger is only required to state what zone she/he is traveling to and from. When a smart card is loaded with a single ticket the passenger needs to scan it in order to activate it before boarding, but is not required to scan it again when alighting. A periodic ticket, such as a 7-, 30- and 365-day tickets is only required to be activated once. Other public transport systems require passengers to scan their ticket both when boarding and alighting the vehicle, or when entering or leaving a station. This allows AFC in those cities to serve much the same function as APC does in Oslo, but with additional information about the origin and destination of individual trips.

While the primary function for ADC-systems on public transport is monitoring, data from AVL- and APC-systems can be used to answer a variety of questions, and solve a variety of problems. If a costumer complains that a bus was late, historical AVL-data can be used to confirm or refute the claim. Aggregated historical APC-data can also show if there is a need for more departures on a given line, as well as changes to timetables caused by a high percentage of late buses. Real-time AVL-data is distributed to signs on bus-stops and is continuously updated on mobile applications. This flexibility and versatility makes ADC-data a valuable recourse, with new applications continuously being developed.

1.1.3 State of the art

As the proportion of vehicles with automatic data collection-systems (ADC) increases, the possibility of more advanced analysis emerges. This has resulted in numerous publications utilising data from ADC-systems in recent years, in particular data from automatic fare collection (AFC) and automatic passenger counting (APC). There are various avenues of interest being explored, from ways of improving estimated time of arrival to methods for automatically changing schedules in real-time. ADC-systems are present on all forms of public transport (trains, metro, tram and bus), and there are numerous publications centered around each. In this thesis we primarily focus on passenger-flow on buses, and methods for predicting the future passenger load. This has also been the subject of many publications.

In a review titled *A Comprehensive Comparative Analysis of the Basic Theory of the Short Term Bus Passenger Flow Prediction* [5], the authors review more than 20 recent publications on the subject. They make the distinction between long-term and short-term passenger-flow prediction, the former mainly aiding in the development of time-tables, while the latter can help with monitoring. As timetables usually are fixed they will not be able to accommodate short-term changes in passenger-flow resulting form a change in weather, special events and traffic accidents. Therefore one can not expect them to be very accurate. Short-term passenger-flow predictions on the other hand needs to be more accurate in order to be useful.

The authors of the review identify three main methods used for predicting passenger-flow; linear, nonlinear and combined methods. The data from both the APC- and AVL- system can be thought of as a time-series, linking an event (such as the boarding/alighting of a number of passenger, or the arrival of a vehicle at a stop) to a timestamp. This lends the data well to linear methods of time-series analysis and linear regression. A downside to these methods is that external factors, such as weather and traffic, are not so easily implemented. In order to alleviate this, several publications are using nonlinear methods such as *Artificial Neural Networks* and *Support Vector machines*, and others a combination of the two.

In general, publications on short-term bus passenger load predictions seams to vary on several fronts. The method (linear, nonlinear, or combination) may vary, but also the object of prediction. Are you looking at the number of passengers on a specif bus line? Maybe the number of passengers boarding or alighting at a specific stop throughout the day? Or maybe the number off passengers traveling through a public transport hub? The data source and the data structure may also vary, if it is from APC- or AFC-systems (or a combination of the two), or even security cameras installed at bus-stops or on vehicles. Different cities also have differing population sizes and public transport coverage. Therefore it might be difficult to know in advance what methods are best suited for a given situation.

1.2 Problem Statement

The purpose of this thesis is to examine whether methods from machine learning can be applied to APC- and AVL-data from Ruter in order to predict future passenger-load.

Particular emphasis will be put on outlier detection. This is because data form ADC-systems have many possible sources of error, and successful predictions require these errors to be identified and removed. The ensemble algorithm isolation forest will be used for automatic outlier detection, and its performance will be compared to manual screening.

We will focus on two cases, representing two different operation situations:

- **Case 1:** Analysis of historical AVL-data and historical APC-data.
- **Case 2:** Analysis of real-time AVL-data and historical APC-data.

The primary way these cases differ is the availability and latency of data from both APC- and AVL-systems. Further description of these cases and how they will be treated is provided in Section 1.2.1.

As the target, passenger load, can be thought of as a continuous variable (even though it is discrete (count)), regression analysis suits this problem well. Introduction to regression for predicting continuous variables will be presented in chapter 2, and a variety of models, from linear regression to XGBoost, are included.

The two cases will in turn be evaluated with two different end users in mind: Passengers using public transport, and those monitoring the traffic. These two groups of end users have different needs and requirements, and we aim to highlight how these can be met. Further description of the end users will be provided in Section 1.2.2.

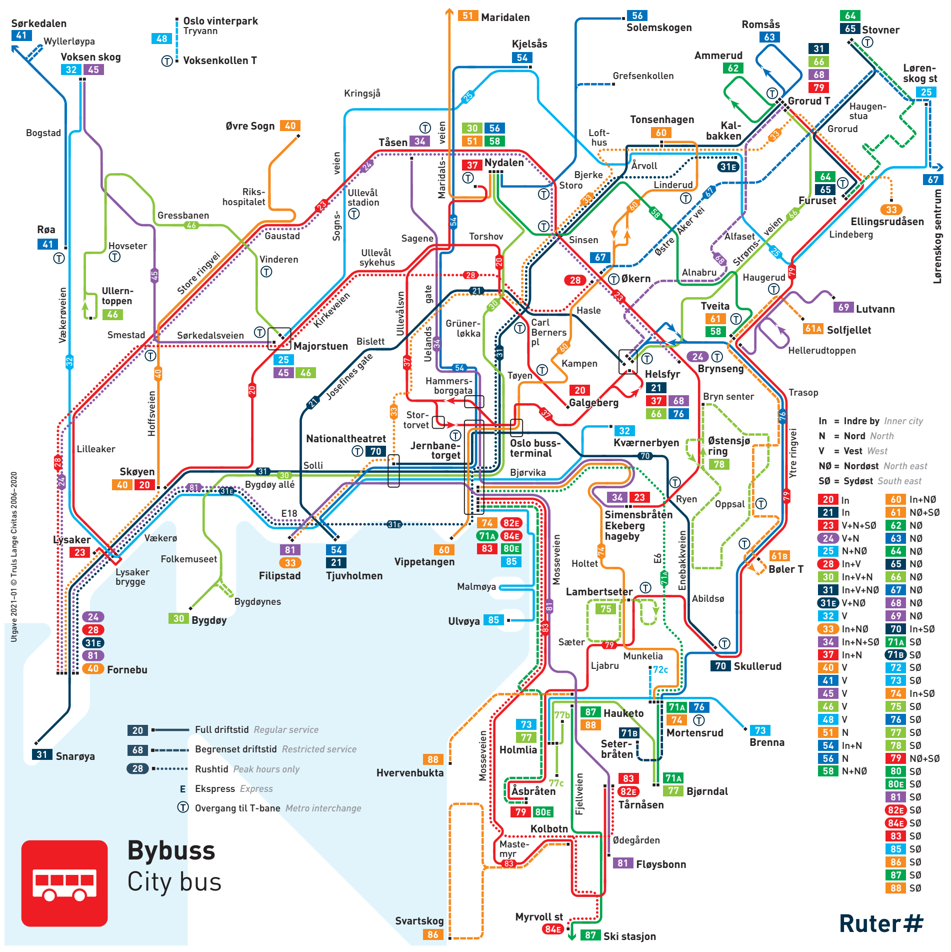


Figure 1.2: Map of all bus lines in Oslo

The analysis will be done using APC- and AVL-data provided by Ruter for the inner-city bus lines 20, 31 and 37 from August to December 2019. This dataset will hereby be referred to as *historical data*. In addition to historical data, a dataset containing all planned trips for the given lines in the given time period will be included. This last data-set will hereby be referred to as *plan data*. Features in both the historical data and plan data can be found in Section 3.3. Figure 1.2 show the routes of these lines along with all other bus lines in Oslo.

We have chosen not use data collected after February 2020 as these are highly affected by Covid-19.

1.2.1 Operation situations

At the time of writing, there are two different platforms for which APC and AVL are operational; SIS and TaaS. The oldest buses are operated by SIS, a service provided by IniT (Innovations in transport). These buses are equipped with both AVL and APC equipment. TaaS is a platform developed by Ruter, and is intended to be the primary platform for both APC and AVL in the future.

The main difference between the two platforms is the delay by which data is received. While AVL-data from SIS is available in real-time, APC-data is only available after 3 days. TaaS on the other hand, reports both AVL- and APC-data in real time.

The common goal for all cases examined in this thesis is to be able to say something about what the passenger-load will be like at some point in the future. This means that our target in all cases are the passenger load. The available features, that is the columns in our input data, will vary form case to case.

Case 1

Case 1 represents the most simple operation situation. Here the only predictors are the ones that are available in plan data. As plan data is available in advance, this enables predictions of load to also be made in advance. This could be made accessible alongside planned departures, informing passengers of how high the load is expected to be up to several days in advance.

Case 1 is set up to emulate the 3 day delay that effect data from SIS-vehicles. Even though Taas-data is available in real-time it will be treated as if it were effected by the same latency. In this case we only utilize APC-data in the final model, but AVL-data is used in the preprocessing-stage.

Case 2

Case 2 expands upon case 1 by taking advantage of real-time AVL-data and combining it with historical APC-data. Here APC-data is affected by the same 3 day-delay as in case 1, but features based on AVL-data is added. As the model is dependent on location data acquired in real-time, predictions may not be made in advance.

1.2.2 End users

Predictions of load could be beneficial for many users, both internally at Ruter and externally. For this thesis we have chosen to focus on two groups of end users: Passengers using public transport, and those monitoring the traffic.

Passengers

Passengers using public transport may alter their travel plans if predictions of load were available. If the route-planner app could indicate how many passengers there usually are on a given departure, new passengers may alter their plans in advance, preferring departures that are less crowded. If predictions of load could also be available in real time at bus stops, passengers may choose to wait for the next vehicle if the first is predicted to be overcrowded. This may result in better distribution of load. Providing this type of information to passengers may therefor be beneficial both for passengers and for Ruter.

IOSS

Within Ruter the department of IOSS (Informasjons- og samordningssentral) is tasked with monitoring and directing traffic in real time. Together with the operators they handle incidents that affect public transport. In order to do this they monitor data from all vehicles around the city equipped with Automatic Data Collection-systems. Vehicles transmitting location-data can be visualized on a map, making it possible to see if there any areas with greater delays. The newer TaaS-vehicles are also transmitting their passenger load in real-time. As these only constitute a proportion of the fleet they don't give a full picture.

If a prediction of real-time load on all vehicles in operation could be available alongside real-time location, IOSS may better be able to prioritize. An example of this could be to better alleviate overcrowding during rush-hour. IOSS have some extra buses at their disposal, ready to be dispatched to areas where bus drivers report a surplus of passengers. The hope is that APC- and AVL-data can be used to predict where the need for these extra buses will arise. If these predictions are deemed precise and timely enough, they can be used by IOSS to dispatch buses in time to prevent congestion from occurring. Thus providing a more seamless experience for passengers.

1.3 Structure of thesis

The thesis will go through all steps required to go from raw data to a final model:

- In chapter 2, *theory*, an overall introduction to the field of Machine learning is provided, before an introduction to regression analysis, with an emphasis on linear regression and decision tree regression. Ensemble algorithms are covered, including regression forests, XGBoost and isolation forest, as well as necessary concepts such as bagging and boosting. We cover how the training of models are performed, and evaluation of model performance is also included.
- Chapter 3, *materials*, covers in detail how the data is collected. An introduction to the systems facilitating the collection of location- and passenger-data is provided, with a focus on the sensors used for counting the passengers. Known errors associated with both the systems and the sensors is covered. Lastly an overview of all original features in the dataset is presented.
- In chapter 4, *methods*, the precise methods used for the analysis is outlined. Methods used for data preprocessing are covered before an explanation of how passenger load is calculated. There is a particular focus on outlier detection, both by a machine learning and by manual screening. A brief explanation of exploratory analysis is also included, before the construction of the two different datasets are performed. An outline of how training, validation and testing is also provided.
- Chapter 5, *results*, includes all findings from the exploratory analysis and the preprocessing of the data. The performance of outlier detection by isolation forest is covered, and comparisons are manual screening are made. Finally, the result of the different regression models are covered for both cases.
- Chapter 6, *Discussion*, is an in-depth discussion on how machine learning can be applied on AVL- and APC-data. We cover advantages and disadvantages for the different models presented in chapter 5, and focus on how different models can meet the demand of the two end users described in 1.2.2. Finally, recommendations on how similar solutions could be implemented on other modes of public transport is provided.

Chapter 2

Theory

In this chapter the relevant theory for this thesis is presented. First an overall introduction to the field of Machine learning is provided in Section 2.1. This Section is intended as a short introduction to machine learning and its history, with the aim to set the tone for further theory, as well as informing further discussion in chapter 6. An introduction to regression analysis is provided in Section 2.2, before individual algorithms, including regression trees, are presented in Section 2.2.2 and Section 2.2.3. Ensemble models are introduced in Section 2.3, and the two models random forest and XGBoost are covered in depth in Section 2.3.1 and 2.3.3. Section 2.3.4 introduces isolation forests, another ensemble model used for detecting outliers in the data. Finally, we cover how the training, validation and testing of models are performed in Section 2.5.

2.1 Introduction to Machine Learning

2.1.1 Short history

Machine learning is a sub-field of artificial intelligence centered around algorithms that allow insight to be gained from data. Even though machine learning has a reputation of being quite ”modern”, the history of the field can be traced back to the 1940s. 1943 saw the publication of the first artificial neuron by McCulloch and Pitts [6]. Their work, titled *A Logical Calculus of the Ideas Immanent in Nervous Activity*, presented model of a simplified neuron consisting of a logic gate with multiple inputs and binary outputs. This model was expanded on in 1957 by Frank Rosenblatt when he introduced *the Perceptron* [7].

The neuron introduced by McCulloch and Pitts could make decisions based on an input signal $x_1, x_2, ..., x_n$, by assessing the outcome of the linear function $f(\mathbf{x}, \mathbf{w}) = x_1w_1+x_2w_2+...+x_nw_n$. In order for this model to work, the weights $(w_1, w_2, ..., w_n)$

needed to be set correctly. With the perceptron, Rosenblatt introduced a way for the model to learn the correct weights in order to produce the desired output. Together with *Adaline* (adaptive linear element), a similar model to the perceptron with continuous output, the foundation for the field of machine learning was set.

The field has continued to get inspiration from the field of Neuroscience. Maybe most notably the idea of networks. *Artificial Neural Networs* (ANN), a model (vaguely) inspired by networks of neuron in the human brain, has gained a lot of attraction due to its performance on a variety problems.

2.1.2 Types of machine learning

Methods of machine learning are often presented as belonging to one of tree groups; **supervised learning**, **unsupervised learning**, and **reinforcement learning** [8].

In **supervised learning** the model is trained on a labeled set of training data and its performance is tested by comparing the model predictions on unseen test data. What makes this type of learning ”supervised” is the existence of labels representing the desired output for each sample. Depending on the type of output, whether it is categorical or continuous, supervised learning can be divided into two groups; *classification* or *regression*. Classification is the process of assigning a discrete class label to a new input based on past observations, whereas regression assigns an output on a continuous scale based on a learned relationship between a set of features [8].

Unlike supervised leaning, **unsupervised learning** is performed without predefined targets. We do not know the ”right answer”. Despite this, unsupervised learning can be used to find meaningful information from the data. A common method for unsupervised learning is clustering, which is an exploratory technique used to organize observations into subgroups [8]. Another useful class of methods from the realm of unsupervised learning is *dimensionality reduction*. Dimensionality reduction allows large datasets with many features to be compressed down to a smaller number of features, while both retaining most of the relevant information and possibly removing some unwanted noise. This may be useful if a dataset is very large, but can also be used for feature extraction.

The last type of machine learning, **reinforcement learning**, represents still another approach of teaching machines. Instead of predefined targets, a reward signal is defined in order to give the model feedback as it interacts with its environment. Through repeated interactions, and subsequent feedback from the reward signal, the model tries to maximise reward.

2.1.3 Essential terminology

Before we go into further details we want to introduce some of the terminology that is going to be used in this chapter. We will try to stick to these terms throughout the thesis, but some exceptions might occur.

The term **model** usually refers to a mathematical function mapping an observation (row), $\mathbf{x_i}$, to an output y_i;

$$y_i = f(\mathbf{w}, \mathbf{x}_i). \tag{2.1}$$

Here $\mathbf{w}$ represents a **parameter**, which is the undetermined part of a *model*. That is the part that needs to be learned from the data. For specific models alternative terms as *weights* and *coefficients* are used instead of parameter. **Hyperparameters** are external variables needed for some models. These are not inferred or updated through training, but their values may affect the model greatly, and special considerations should be made to set them correctly. The tuning of these hyperparameters is called **hyperparameter tuning**.

Training is the process of finding the best *parameters* to fit to our training data. An **objective function** is used to quantify how well a given *parameter* fit the data.

$$obj(\mathbf{w}) = L(\mathbf{w}) + \Omega(\mathbf{w}) \tag{2.2}$$

The objective function usually consist of two parts: the **loss function**, L, and the **regularization term**, Ω. The job of the loss function is to measure how well the model's predictions fit the target. The terms *cost function* and *loss function* are often used interchangeably. In short one can think of the loss function as the error of a single training example ($L(f(x_i, w_i), y_i)$), and the cost function as a measure of the error on the whole training set ($\sum_{i=1}^{n} L(f(x_i, w_i), y_i)$) [8].

The regularization term adds a penalty to the cost function that penalises complexity and in this way can prevent **overfitting**. Overfitting occurs when the model is too closely fitted to the training data, and results in poor predictions on unseen data because it is not able to generalize well. The presence of both terms in the objective function helps balance **bias** and **variance** in our final model.

2.2 Regression

As mentioned in Section 2.1.2, regression is a form of supervised learning where the goal is to predict a continuous output. This is done by trying to find a relationship between the available features and the target. There are several different approaches to establish this relationship. We start by introducing *linear regression* in Section 2.2.1. This is the simplest form of regression model, aimed at determining the linear relationship between the input variables and the target. Linear regression falls short where the relationship between features and targets are non-linear. A method used for establishing such non-linear relationships are *regression trees*, presented in Section 2.2.3. There are also other regression models developed to mitigate shortcomings in simple linear regression, such as sensitivity to multicollinearity, Section 2.2.2 gives a short introduction to some of these.

2.2.1 Linear regression

The basis for linear regression is the assumption that there is a linear relationship between input and output, or features and targets. If you only had one feature, x, and one target, y, a linear relationship could be expressed as

$$y = w_0 + w_1 x, \tag{2.3}$$

with w_0 representing the intercept (the point where the line crosses the y-axis), and w_1 the weight coefficient for the feature x. w_0 is also referred to as the *bias*. It is easy to visualize this graphically as well, as a straight line in the x-y plane.

With two features, x_1 and x_2, a graphical presentation would still be possible, now as a plane. Visualization becomes difficult when more than two features are present. Even though visualization is hard, linear relationships can still be generalized for multiple features, called a *multiple linear model*. A linear relationship with m features can be expressed as

$$y = w_0 + w_1 x_1 + w_2 x_2 + ... + w_m x_m. \tag{2.4}$$

If we define $x_0 = 1$, this can be changed to

$$y = w_0 x_0 + w_1 x_1 + w_2 x_2 + ... + w_m x_m = \sum_{j=0}^{m} w_j x_j = \mathbf{w}^{\mathbf{T}} \mathbf{x} \tag{2.5}$$

because $w_0 x_0 = w_0$ [8].

Linear *regression* is the process of defining the weights that go into the linear model, that is finding the best fitting line through the training data. In order to find the "best fit" we need to find a measure of how well our linear model fits the data. This is the job of the *loss function* we introduced in Section 2.1.3. There are several functions that can be used for this, one of the more popular ones being *ordinary least squares*. This method adopts the squared distance between observed points and the regression line as a measure of the error,

$$L(f(\mathbf{x}_i, \mathbf{w})) = (y_i - f(\mathbf{x}_i, \mathbf{w}))^2. \tag{2.6}$$

Lets say we have a linear model, like equation 2.5, with n training examples. The sum of squared error (SSE) between the predicted values $\hat{\mathbf{y}}$ and the observed values $\mathbf{y}$ for all n examples can be expressed by the cost function

$$\epsilon(\mathbf{w}) = \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 = \sum_{i=1}^{n} (y_i - f(\mathbf{x}_i, \mathbf{w}))^2. \tag{2.7}$$

It is this error, ϵ, we wish to minimize, and this is done by updating the weights $\mathbf{w}$. But how do we go about updating them?

Gradient descent

Gradient descent is a type of optimizing algorithm that can be used for finding the best weights to satisfy any predefined objective function. The objective function can for example be a cost function, as is the case for our sum of squared error where our goal is to minimize the SSE. The process of gradient descent can be thought of as climbing down a hill attempting to find the lowest point [8]. For every step taken, the direction of the slope, and its steepness, is evaluated. This informs in what direction the next step should be made, and how big the next step should be.

We define this weight change $\Delta\mathbf{w}$ as

$$\Delta\mathbf{w} = -\eta\nabla\epsilon(\mathbf{w}), \tag{2.8}$$

where η is the learning rate and $\nabla\epsilon(\mathbf{w})$ the gradient of the cost function. The negative sign thus pushes $\mathbf{w}$ in the opposite direction of its gradient. The learning rate, η, is a hyperparameter that effects how big each step is. If it is too small the algorithm may need many iterations, or epochs, to find the best fit. If on the other hand the learning rate is set to high it might overshoot the global minimum. It is therefore important to set reasonable values for hyperparameters like this, and it might be beneficial to try out a range of different values for them.

Feature scaling

When preprocessing data for analysis it is common to perform some sort of feature scaling. If our data contain features with differing ranges, (f.ex [0, 1], [-1000, 1000] or [0, ∞]), the performance of objective functions may suffer. This can in turn lead to updates in weights greatly favoring certain variables. In order to prevent this the features can be scaled, or *normalized*, so that the contribution of each feature is the same.

There are several different methods used for normalization. **Min-max normalization** is one such method used for scaling features down to the same range. The value $\mathbf{x}_i$ contained in vector $\mathbf{x}$ can be scaled to the range $[0, 1]$ using the formula

$$\mathbf{x}'_i = \frac{\mathbf{x}_i - min(\mathbf{x})}{max(\mathbf{x}) - min(\mathbf{x})}, \tag{2.9}$$

or any predefined range $[a, b]$ by the formula

$$\mathbf{x}'_i = a + \frac{(\mathbf{x}_i - min(\mathbf{x}))(b - a)}{max(\mathbf{x}) - min(\mathbf{x})}. \tag{2.10}$$

Another commonly used method of normalization is **standardization**. For a feature $\mathbf{x}$, the standardized transformation $\mathbf{x}'$ can be found by

$$\mathbf{x}' = \frac{\mathbf{x} - \mu}{\sigma}. \tag{2.11}$$

Here μ is the mean of the feature $\mathbf{x}$, and σ is its standard deviation. An advantage to this method is that the standardized feature $\mathbf{x}'$ has zero-mean ($\mu(\mathbf{x}) = 0$) and unit variance ($\sigma(\mathbf{x}) = 1$). This can be an advantage for many optimization algorithms, including gradient descent.

There exist many different packages in Python to perform feature scaling. Scikit learn's [9] preprocessing-module has methods for both min-max scaling and standardization, called *MinMaxScaler* and *StandardScaler* respectively. Some models requiring normalization of features, such as Linear regression, comes with the option to normalize the data directly. For scikit-learns *LinearRegression* this is done by setting the option `normalize=True`. Note that this performs normalization and not standardization.

Multicollinearity

While the aim of linear regression is to determine the linear relationship between input features and a target, the method is sensitive when there exists a linear relationship between input features. When one predictor in our input data is linearly related to another predictor we call it collinearity or multicollinearity. The former is when two predictors are linearly related, and the latter when more than two are linearly related.

2.2.2 Other regression-models

In Section 2.1.3 we said that an objective function usually consist of two parts; the loss function and the regularization function. This section introduces some linear models that employ regularization to prevent overfitting and make models more robust.

All hyperparameters presented in the following sections are set using cross-validation in combinations with either grid-search or randomized search. Further description of how hyperparameter-tuning performed is included in Sections 2.5.

Ridge regression

Ridge regression is a method intended to alleviate the problem of multicollinearity by introducing a penalty on the size of the weights. In short the squared sum of the weights are added to the cost function resulting in

$$\epsilon(\mathbf{w}) = \sum_{i=1}^{n}(y_i - \hat{y}_i)^2 + \alpha||\mathbf{w}||_2^2, \tag{2.12}$$

where the hyperparameter $\alpha \geq 0$ determines how big this penalty is [8]. The regularization term $\Omega(\mathbf{w}) = \alpha||\mathbf{w}||_2^2$ is known as L2-regularization, where

$$\alpha||\mathbf{w}||_2^2 = \sum_{j=1}^{m} w_j^2. \tag{2.13}$$

By adding this regularization term to the cost function it ensures that the weights are also kept low when the cost function is minimized.

Lasso regression

Lasso regression also employs a penalty on the size of the weights in a model, as did ridge regression, but instead of limiting the square of the weights the absolute value is used instead. That means that Lasso regression employs L1-regularization. The cost function for for lasso-regression is

$$\epsilon(\mathbf{w}) = \sum_{i=1}^{n}(y_i - \hat{y}_i)^2 + \alpha||\mathbf{w}||_1, \tag{2.14}$$

where

$$\alpha||\mathbf{w}||_1 = \sum_{j=1}^{m} |w_j|. \tag{2.15}$$

ElasticNet regression

ElasticNet is a model made to make benefit of both L1 and L2 regularised regression, resulting in the objective function

$$\epsilon(\mathbf{w}) = \sum_{i=1}^{n}(y_i - \hat{y}_i)^2 + \alpha||\mathbf{w}||_2^2 + \alpha||\mathbf{w}||_1. \tag{2.16}$$

Principle component regression

In Section 2.1.2 the subject of unsupervised learning was briefly introduced, and it was mentioned that one of its applications was for dimensionality reduction. One such unsupervised learning method is called *principal component analysis* or PCA.

In short PCA is a method of dimensionality reduction where the aim is to identify the directions of maximum variance in the dataset and project them down to a lower dimensional space [8]. While the original data may have features with high correlation, the resulting principal components will be orthogonal, thus eliminating the problem of multicollinearity. Another result of PCA is that the number of features needed to represent the data may be greatly reduced as the principal components will have descending variance. Feature scaling (2.2.1) can be performed before PCA as it will affect the directions of the principle components.

A principle component regressor through scikit-learns pipeline-function. This regressor first performs standardization using the `StandardScaler` before feature reduction is done by principal component analysis. Then the `LinearRegressor` is used to fit a linear model on a certain number of principle components. The optimal number of principle components to use are found through grid-search and cross validation.

2.2.3 Decision tree regression

All previously discussed models assume a linear relationship between predictors and target, but this may not be the case. Decision tree regression is a regression algorithm that have the capability of finding relationships that are not linear. It is built on decision trees, a method most commonly used for classification. The approach of a decision tree is for the model to learn a series of ”questions” to ask the data. This allows the model to infer the target based on how new data ”answer” these questions.

A decision tree start with a single node containing all datapoints. From here the parent node can be iteratively split into child nodes based on decision rules until all leaf nodes are pure. The concept of **Information gain** is used to quantify what decision rules are most informative.

Information gain and impurity measure

In Section 2.2.1 we introduced *gradient descent* as an optimizing function to find the best fit for our regression line to the target. The goal of a decision tree model is to find the best decision rules to split *nodes* on. In order to do this we define an objective function; maximise the *information gain* at each split [8].

The information gain is defined as

$$IG(D_p, f) = I(D_p) - \sum_{j=1}^{m} \frac{N_j}{N_p} I(D_j) \tag{2.17}$$

where f is the feature where the split is performed, D_p is the dataset of the parent and D_j is the dataset of the child node. N_p and N_j are the number of training examples in the parent and the child nodes respectively, and I is an impurity measure quantifying how similar values within one node are to each other. In short Information gain is a measure of how impure the parent node is compared to its child nodes. When the impurities of the child nodes are low compared to the parent, the information gain is high.

When decision trees are used for classification *Gini impurity* is often used as the impurity measure for categorical outputs. When using decision trees for regression, as so called *regression trees*, we need to measure the impurity of a continuous variable within each node. There are many error measures that can be used for this. Scikit learn's Decision tree regressor offers among other *mean squared error* (MSE) and *mean absolute error* (MAE).

In order to maximize information gain, MSE can be utilized as impurity measure by finding the intra-node MSE,

$$I(t) = MSE(t) = \frac{1}{N_t} \sum_{i \in D_t} (y_i - \hat{y}_t)^2, \tag{2.18}$$

where

$$\hat{y}_t = \frac{1}{N_t} \sum_{i \in D_t} y_i. \tag{2.19}$$

N_t is the number of training examples in node t, D_t is the dataset at node t. The prediction, $\hat{y}_t$ is the mean within the node, while y_i is the true value. Thus the result of minimizing MSE is that intra-node variance is reduced [8].

Preventing overfitting

Like many other algorithms, decision trees are prone to overfitting and therefore not generalizing well to unseen data. This is especially true when the number of features are high [10]. There are a couple of ways to handle this problem. *Pruning* is one such method where you reduce the depth of decision trees. Another is to set a minimum number of samples per split or at the outermost nodes of the tree, referred to as the *leaf*.

In Scikit learn's decision tree regressor the parameter `max_depth` indicates how deep the tree should be. The parameter `min_samples_split` regulates how many samples there need to be in a node in order for a new split to be made, and `min_samples_leaf` regulates the minimum number of samples in a leaf-node.

2.3 Ensemble models

This section expands on the theory presented in Section 2.2 for linear regression and regression trees by introducing *random forest regression* and the concepts of *bagging* in Section 2.3.1. random forest regression can be thought of as a predecessor to the popular model *XGBoost* which stands for *extreme gradient boosting*. It uses *boosting*, a concept that will be explored further in Section 2.3.2. Another ensemble model, Isolation forest, used for outlier and anomaly detection, is introduced in Section 2.3.4.

2.3.1 Random forest regression

Random forest regression is a form of **ensemble model**. An ensemble is a collection of machine learning models that are combined to work together. A random forest regressor contains, as the name suggest, several regression trees. In the context of ensemble models, the individual models contained in the ensemble are referred to as *base learners*. In the case of a random forest regression the base learner is a regression tree. The main benefit of ensemble models is that the combined model is less prone to overfitting than individual base learners.

In ensemble models, the final prediction is based on input from all the base learners. For a random forest classifier, an ensemble of decision trees used for classification, *majority voting* can be used to decide the final prediction. Here all base models get a vote, and the classification with the most votes is set as the final class label. For random forest regressors the mean of the predictions of all base models can be used as the final prediction. Alternatively one could weight them according to how well they have previously performed, giving well-performing base models greater sway when voting.

Bagging

When implementing an ensemble model, such as a random forest regressor, it would not be very effective if all the base learners were identical. In order to prevent this *bootstrapping*, taking a sample with replacement, can be used. In the context of machine learning this entails selecting a subset of k features from n available features in the dataset and using them to train a base model. Doing this multiple times and training many models with individual selections of k bootstrapped features is called **bagging**, or **bootstrap aggregation** [11].

2.3.2 Boosting

Bagging is a great way to reduce the variance in the final model, making it better equipped to make predictions on unseen data, but it requires all individual base models to be trained from scratch. Each base model is only concerned with itself, and not of the model trained before it. In contrast, **boosting** is a method that allows new base models to learn from the mistakes of previous base models.

The idea behind *boosting* is to make an ensemble of *weak learners* and letting them learn from each others mistakes, thus becoming *strong learners*[8]. A weak learner is a simple base model which is only required to perform slightly better than random guessing, while a strong learner is a model that performs relatively well.

A general boosting-procedure for classification can be summarized as follows [8]:

1. Draw a subset d_1 without replacement from the set D containing all training examples. Train a weak learner C_1 on the subset d_1.

2. Draw a new subset d_2 without replacement from the set D, and add 50% of the training examples that C_1 misclassified. Train a new weak learner C_2 on subset d_2.

3. Construct a third subset, d_3, containing all training examples for which the weak learners C_1 and C_2 give differing classification. Train a third weak learner C_3 on subset d_3.

4. Combine the predictions of C_1, C_2 and C_3 through majority voting.

In this general procedure models are trained on a subset of the data, but there are some boosting-procedures where that is not the case. AdaBoost is one such procedure where base learners are trained on all samples of the dataset, before weights are assigned each training sample after each iterations based on the mistakes made by the previous base learner [8].

Gradient boosting

In Section 2.2.1 we introduced the optimization algorithm *gradient descent*. This algorithm updates the weights of a model according to the direction and the steepness of the gradient, thus "descending the gradient" and making it smaller. The decrease in the gradient is here caused by changes of the weights, but for *gradient boosting* the decrease is caused by training a new base model based on the gradient of the previous model.

Gradient boosting requires a weak learner and loss function that is to be minimized. The first could be a decision tree and the latter a differentiable function, f.ex mean square error. Whereas AdaBoost make predictions based on majority voting, the predictions from a gradient boosting ensemble is made by adding the the predictions of each base learner together. The first weak learner is trained on the original dataset, and each additional tree is trained on the residuals of the previous tree. This means that even though each additional tree only makes predictions slightly better than pure chance, the gradient of the loss function is still descended.

Even though the constituent parts of a gradient boosted ensemble are weak learners, the ensemble as a whole is still prone to overfitting. In order to prevent overfitting, a few different precautions can be made: For each additional base learner it is possible to scale down its contribution, thus slowing down the rate of learning. This is referred to as the learning rate. Another option is to penalize large leaf weights through regularization. Both L1- and L2-regularization (introduced in Sections **??** and 2.2.2) can be used for this purpose.

2.3.3 XGBoost

As stated in the beginning of this section the name *XGBoost* stand for ”extreme gradient boosting”. We now know what gradient boosting is, but why is XGBoost ”extreme”?

XGBoost is an ensemble model following the principle of gradient boosting. Its performance is often credited to the way it uses regularization to prevent overfitting. For a tree $f(\mathbf{x})$ the regularization term is defined as:

$$\Omega(f) = \gamma T + \frac{1}{2}\lambda \sum_{j=1} T w_j^2, \tag{2.20}$$

where T is the number of leaves in the tree, and w_j is the weight associated with leaf j. Through the parameters γ and λ, both the number of leaves in each tree and the values for the weights on the leaves, are regularized [12].

Even though XGBoost owes a lot of its success to regularization, the ”extreme” in ”extreme gradient boosting” refers to the engineering goal of creating an extremely efficient algorithm, as stated by the original creator of XGBoost [13]. Because of its efficiency XGBoost is well suited for large datasets.

2.3.4 Isolation forest

The presence of outliers in our data can greatly effect the performance of models, especially linear models [8]. Isolation forest is an ensemble model introduced in 2009 in a paper of the same name [14] and implemented as part of Scikit learn’s ensemble-module. Isolation forest can be used for outlier detection and work by growing trees that are divided at randomly chosen split values for a randomly chosen feature. This recursive partitioning results in a tree structure where the length of the path between the base of the tree and each ”leaf” (terminating node) is the same as the number of splits required to isolate a sample.

Outliers can be detected from isolation trees because the number of splits required to isolate an outlier is typically less than the mean number of splits for the whole tree. By training an ensemble of isolation trees the resulting forest provides a robust way to detect outliers in multivariate data.

By default isolation forests train one base learner on 256 randomly drawn samples from 1 randomly drawn feature in the dataset. If there are less than 256 samples in the dataset, all samples are used. This means that the ensemble consists of base learners that are ”experts” one feature each. Because the number of samples each base learner is trained on is so low, the model scales well to larger datasets [14].

2.4 Missing data

In the book *Statistical analysis with missing data*, Little and Rubin defines missing data as ”unobserved values that would be meaningful for analysis if observed; in other words, a missing value hides a meaningful value” [15]. They argue that if this definition applies to you data you should consider ”imputing” the data you are missing. If it does not apply imputation makes little sense. But how do you know whether a value you are missing would be valuable to you or not?

Little and Rubin describe two characteristics to distinguish between types of missing data, pattern and mechanism. Examples of the former can be whether there is univariate or multivariate non-response, meaning if the missing values are present in one or more features. The latter is concerned with what caused the missingness, and in particular if the missingness is related to the missing value itself.

They distinguish between three mechanisms for missing data, first introduced by Rubin in 1976 [16]:

- **Missing Completely at Random (MCAR)**
  Data is MCAR if the probability of missingness is the same for all samples.
- **Missing at Random (MAR)**
  Data is MAR if the probability of a value being missing is the same within defined groups of the observed data.
- **Missing Not at Random (MNAR)**
  Data is MNAR if the probability for a sample to be missing is dependent on its own value or some other unknown factors.

2.5 Training, validation and testing

2.5.1 Training- and test-set

In order to test our models the original dataset is split into two parts, one for training and one for testing. This is done to make sure that our model is not overfitted and to ensure that it performs well on unseen data.

2.5.2 Cross-validation

Cross-validation is a method used for evaluating model performance as well as aid in tuning hyperparameters in our models. One common variant of cross-validation is called k-fold cross-validation. Here the training data is split into *k* folds (without replacements). The model is then trained on the data in *k-1* folds, and tested on the last fold. This is repeated *k* times, resulting in *k* models and *k* model evaluations [8].

Because the data used in this thesis can be thought of as a time series we need to account for that when using cross-validation. This can be done by preserving the order of entries when making the folds. There are several ways to do this, but two popular approaches are moving- and expanding-window cross-validation. For moving window the first model is trained on the first fold and validated on the second, the second model is trained on the second fold and validated on the third. So on and so forth.

For expanding window cross-validation the first model is also trained on the first fold, while the second is used for validation. But then the next model is trained on the first *and* second fold, and validated by the third. That means that the size of the validation set remains the same, but that the size of the training data increases with each fold. Figure 2.1 illustrates the relationship between training- and test-data as well as the folds used for expanding window cross-validation.

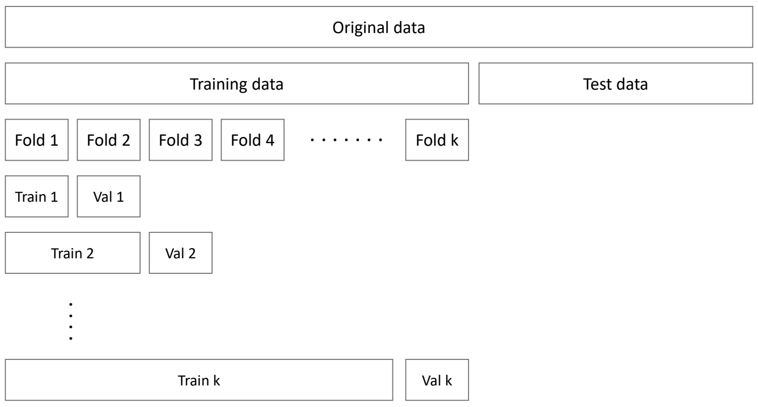


Figure 2.1: *Partitioning of data into training- and test-data, before cross-validation for data with time series characteristics*

2.5.3 Hyperparameter-tuning

Grid search method used for hyperparameter-tuning where different combinations of hyperparameters are tested in order to find the combination that yields the best performance. Scikit learn has a few different methods for grid-search and the ones used for this thesis are `GridSearchCV` and `RandomizedSearchCV`.

Both these method requires a dictionary to be defined, containing a hyper-parameter and its potential values as key-value pairs. For example the depth of a decision tree regressor can be defined as `'max_depth': [3, 6, 9, 12, 15, 18, 21]`, and the minimum number of samples in a leaf as `'min_samples_leaf': [10, 20, 30, 40, 50]`. `GridSearchCV` performs an exhaustive search, meaning that all possible combinations of hyperparameters are tested. In this case that would be $7 \times 5 = 35$ combinations. If a model has many hyper-parameters requiring tuning, the total number of combinations can become high, thus taking a lot of time. In order to reduce the time needed for tuning, `RandomizedSearchCV` train a fixed number of models based on a random selection of hyperparameters.

Both `GridSearchCV` and `RandomizedSearchCV` use cross-validation in order validate the performance of each hyperparameter combination.

2.5.4 Evaluation metrics

In order to evaluate models we need a metric to quantify the performance of the model. In earlier sections both mean square error (MSE) and mean absolute error (MAE) were used as cost functions. These error metrics can also be used to compare regression models to each other after the models are trained.

As evaluation metrics, the main difference between MSE and MAE is how much they are affected by large differences between true and predicted values. While MAE is a linear score, increasing linearly with larger differences, MSE grows exponentially. Thus, MSE penalizes larger differences more than MAE.

One benefit of MAE as an error metric is that it has the same unit as the target, thus making it more intuitive to interpret. Because of the squared term this is not the case for MSE. By taking the root of the MSE, resulting in root mean squared error (RMSE), we get an error metric with the same unit as the target.

Chapter 3

Materials

This chapter covers in detail how the raw data is collected on vehicles and processed into the data used in this thesis. In Section 3.1 the methods for data collection are outlined, both for location data and passenger data. Section 3.2 goes over known errors associated with both the ADC-system and the sensors. Finally, in Section 3.3 all original features in both the historical data and plan data are presented.

3.1 Data collection

On board every vehicle is a computer, often referred to as the copilot, connecting the vehicle to the APC- and AVL-system. While in operation every vehicle is transmitting information allowing them to be monitored in real-time. This information is used directly by operators and IOSS to direct the traffic. At the start of a trip the driver signs on to a unique trip-id, allowing correct route information to be displayed on screens on/inside the vehicle. Since the vehicle is also transmitting its location it is possible both to monitor the progression of the trip, and publish real-time information about arrival times on digital signs and on mobile applications.

The copilot is also connected to sensors mounted on each door of the vehicle. These sensors allows for the counting of every passenger boarding or alighting the vehicle. As there is one sensor for every door on the vehicle the total number of sensors varies with the type of vehicle. Some smaller buses may only have a few doors, requiring only a few sensors. While other, larger vehicles require more.

Data transmitted by the copilot is stored and transformed for future use on many different levels of granularity. Even though vehicles are transmitting near continuous data, they need to be aggregated in order to be useful for analysis. In this thesis we primarily use historical data on trip-level. By that we mean location- and passenger- data connected to unique trip-ids, where each entry represents the stopping of a vehicle at a bus-stop where passengers are allowed to board and alight. This requires the driver to have signed on to the trip correctly in order for the data to be assigned correctly.

3.1.1 Location data

Location-data is collected through GPS on every vehicle. The location of a vehicle can be communicated in different formats, coordinates and location-names (stop names). Coordinates, given in longitudes and latitudes, can be informative for real-time surveillance of vehicles, and can provide precise measurements of location (assuming accurate GPS-sensors). Stop-names can on the other hand be more descriptive as it reduces the complexity of the data, especially when preparing data for analysis.

The dataset used for this thesis includes coordinates in longitudes and latitudes in addition to stop-names. Models are trained using latitudes and longitudes, but stop-names will be used during preprocessing and for visualizing data.

3.1.2 Passenger data

Passenger counts are facilitated by sensors mounted on every door of each bus. By counting all passengers boarding and alighting the vehicle, one can calculate the load on the buss by summing over the number of boarding and alighting passengers. The number of boarding and alighting passengers are stored, but no calculation of load is made internally on the sensor unit.

There are currently two types of sensors installed on vehicles in Oslo; Dilax IRS-320 and Init Iris Matrix [17, 18]. Both are infrared sensors detecting passing passengers when a matrix of emitted infrared beams are disturbed. Depending on the order the beams are disturbed in (front to back or back to front), passengers are registered as either boarding or alighting.

Because it is likely that the sensors have differing properties, including accuracy, it would have been useful to know what type of sensors are installed on what vehicles. Unfortunately we are not aware of this information existing. We therefore make the assumption that all measured counts from the different source-systems are made by the same sensors.

When APC-data is displayed on trip-level, as is used for this thesis, only one measurement is provided for the number of boarding and alighting passengers. That means that counts from the individual sensors are added together and only one number reported for each stop the vehicle makes.

3.1.3 Plan data

The dataset referred to as *plan data* is a table containing all planned trips. This dataset is set up by route-planners internally at Ruter, and is published in advance of all trips. Versions of plan data is available in tables at bus-stops, route-planners online or in-app. Plan-data is also distributed to vehicles through the copilots, allowing drivers to sign on to the different trips as described in Section 3.1. Section 3.3 covers details about what is included in the plan data, and Table 3.2 in the same section lists all features present.

3.2 Known errors

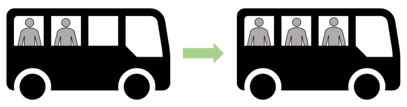
ADC-systems on public transport are complex systems collecting data from several different sensors and other equipment. This also means that there are many potential sources of error that can affect the final dataset, and some of these sources of errors are discussed in this section.

One of the main challenges when validating data from ADC-systems is how to go about establishing ground truth. Most studies looking into the performance of APC-sensors establish the accuracy of sensors by comparing them to counts made by checkers manually counting passenger on all doors in a vehicle. This method assumes that the human checkers count every boarding and alighting correctly, which on a crowded vehicle may not be the case.

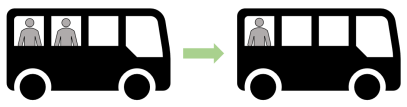
Systematic over- or under-counting have been observed in earlier studies, as well as systematic discrepancies between accuracy of counts of boarding and alighting passengers. A study in Italy looked at 950 counting events on a busy bus line, comparing manual counts to counts made by similar infrared sensors as those used by Ruter. It showed that the sensor tended to under-count both boarding and alighting passengers, and that the count for alighting passengers tended to be more accurate that for boarding passengers [19].

3.2.1 Sensor accuracy

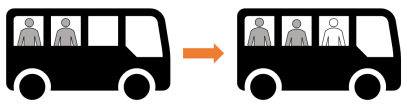
We can imagine six unique scenarios of counting as illustrated in Figure 3.1. The first two (3.1a and 3.1b) being true positives, that is a passenger has boarded or alighted and has been counted correctly. False positives would mean that no passengers has boarded, but a passenger was counted (Figure 3.1c), or no passengers has alighted, but a passenger was counted (Figure 3.1d). The first would result in the accumulated load being overestimated, while the second would lead to underestimating of the load. The last scenarios to take into consideration are false negatives. A false negative for a boarding passenger (Figure 3.1e) would mean that a passenger boarded but was not counted, resulting in an underestimation of the load. While a false negative for an alighting passenger (Figure 3.1f) would lead to



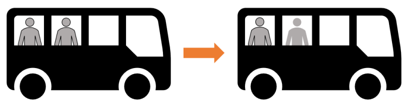
(a) True positive: *Boarding passenger counted by the sensor.*



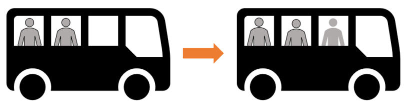
(b) True positive: *Alighting passenger counted by the sensor.*



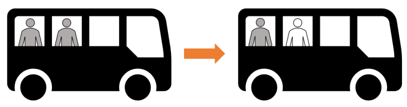
(c) False positive: *No boarding passenger, but counted by the sensor.*



(d) False positive: *No alighting passenger, but counted by the sensor.*



(e) False negative: *Boarding passenger, but not counted by the sensor.*



(f) False negative: *Alighting passenger, but not counted by the sensor.*

Figure 3.1: *Scenarios of counting.*

overestimation.

In order to evaluate the accuracy of the sensors we could take advantage of the difference in boarding and alighting passengers on a vehicle. During the course of an operation-day we know that the same number of people board a vehicle as alight from it. One can make the same assumption for individual trips provided that passengers were not able to stay aboard the vehicle from one trip to the next. This can be expressed as

$$\sum_{i=1}^{n} b_{i,act} = \sum_{i=1}^{n} a_{i,act}, \tag{3.1}$$

where $b_{i,act}$ is the number of actual boarding passengers at stop i, and $a_{i,act}$ is the actual number of alighting passengers at stop i.

The inaccuracies outlined in Figure 3.1 can be expressed as

$$b_{measured} = b_{act} + b_{false\,positive} - b_{false\,negative} \tag{3.2}$$

for the boarding passengers, and

$$a_{measured} = a_{act} + a_{false\,positive} - a_{false\,negative} \tag{3.3}$$

for alighting passengers.

Based on these equation it is not possible to get a complete picture of how big the error is, as we have 2 equations with 4 unknowns. We can however get an indication of the minimum value for the error.

For every vehicle on every operation day we can calculate

$$\sum_{i=1}^{n} b_{measured} - \sum_{i=1}^{n} a_{measured} = \sum_{i=1}^{n} (b_{act} + b_{fp} - b_{fn})_i - \sum_{i=1}^{n} (a_{act} + a_{fp} - a_{fn})_i \tag{3.4}$$

which due to equation 3.1 can be abbreviated to

$$\sum_{i=1}^{n} b_{i,measured} - \sum_{i=1}^{n} a_{i,measured} = \sum_{i=1}^{n} (b_{fp} - b_{fn})_i - \sum_{i=1}^{n} (a_{fp} - a_{fn})_i \tag{3.5}$$

It will not be possible for us to separate all these different counting scenarios, and a thorough assessment of the accuracy and precision of the sensors is beyond the scope of this thesis. Outlier detection is used to identify trips where either AVL- or APC-data is anomalous. In depth description about how outlier detection is performed is provided in Section 4.7.

3.2.2 Installation and maintenance

How the sensors are installed and maintained has the possibility of greatly affecting the accuracy of the data. Each sensor has specific requirements for installation, and if these are not met the sensor cannot be expected to provide the stated level of accuracy. The accuracy of individual sensors may be reduced if installation is not done correctly, leading them to under- or over-count the number of passengers.

Regular maintenance is also required in order that the sensors work as intended. This may include regular cleaning of the sensors, monitoring of their performance and reconfiguration of faulty equipment. If this is not done, the accuracy may again be lowered. In worst case, sensors might stop working all together.

As mentioned in Section 3.1.2, the sum of measurements from all sensors are reported as the number of boarding and alighting passenger. That means that detecting whether individual sensors are faulty is not possible when viewing data on trip-level, but an observed discrepancy between the number of boarding and alighting passengers on a vehicle may indicate that one or more sensors are faulty. As discussed in the previous section, sensors like these are not without errors, so a certain discrepancy is to be expected. Still, if the discrepancy is large, faulty sensors are a probable cause.

3.2.3 Sign on

In order for data to be associated with a given trip the bus-driver need to sign on to the trip. This is done through the on-board co-pilot, and enables real-time location data to be distributed and stop-announcements during the trip. The driver is incentivized to sign in correctly, but errors in sign-on still occur. This can lead to inaccurate counts or missing counts.

One recurrent issue is that a vehicle is signed on to a trip that were scheduled earlier in the day, or a trip from the previous operation day. This is apparent in the dataset when the delay of the vehicle (that is the difference between the feature *act_arr* and *plan_arr*) is from several hours to a day long. This needs to be screened for, and an appropriate limit for accepted delay needs to be set. See Section **??** for more information.

Even though the driver is signed on to the correct trip, inaccuracies can still occur. This is usually related to the start or the end of a trip. If passengers start boarding the vehicle before the driver has time to sign in, they will not be accounted for. The same can happen for passengers alighting at the last stop if the driver signs out too fast. If the buss is left signed on for too long, passengers boarding for the next trips might also be assigned the wrong trip.

In order to alleviate this, corrections should be made for passengers boarding on the last stop and alighting on the first. This requires identifying the last stop on the previous trip and first stop on the subsequent trip for every vehicle, and modifying their passenger-counts. One must be wary of vehicles having extended breaks during the course of an operation day as these should not be modified. This has proven difficult, and we have not succeeded in implementing a correction for this in the processing of the data.

3.2.4 Passenger behaviour

Passenger behaviour may also affect the accuracy of the passenger data. If two passengers are moving close to each other, the sensor might not register them as two different passengers, but count them as one. One might also imagine that a big backpack, or a suitcase, might be counted as an extra passenger.

The differing behaviour while boarding and alighting might also affect the accuracy of the data. From personal experience passengers tend to move closer together while boarding, trying to get onto the vehicle as fast as possible, while staying more calm when alighting. This might result in an under-counting of boarding passengers, while more accurate measurements of alighting passengers.

During rush-hour, when vehicles are crowded, passengers near the door may need to exit the vehicle in order to let other passengers of. Hopefully these passengers are counted correctly, both while alighting and re-boarding the vehicle, but as this behaviour increases the number of measurements made by the sensors, it may also increase the overall error.

3.3 Dataset

The variables described in Table 3.1 are included in the raw data. This dataset is used in conjunction with Table 3.2 to account for all planned trips.

In order to match AVL- and APC-data to plan data we assume that the combination `op_day`, `line_no`, `route_direction`, `plan_start`, `plan_end`, `plan_arr`, and `stop_name` is unique. This is because we assume that a vehicle may stop several times at one stop, but not twice at the same time. This allows us to link actual events to planned events. This method allows a bus line to circle back on itself, and while this is not the case for any of the lines used in this thesis, some trips have repeat stops.

In its original state all timestamps are given as an integer indicating the number of seconds from midnight. How these are treated will be covered in Section 4.6.1.

Variable	**Type**	**Description**
op_day	datetime	Operation day
line_no	int	Unique code of buss-route
vehicle	str	Unique vehicle identifier
route_direction	int	Flag to indicate direction of route
stop_name	str	Unique identifier of stop
stop_idx	int	Index of stop on the specific route
plan_start	int	Planned start of trip
act_start	int	Real start of trip
plan_arr	int	Scheduled arrival at stop
act_arr	int	Actual arrival at stop
plan_dep	int	Scheduled departure at stop
act_dep	int	Actual departure from stop
plan_end	int	Planned end of trip
act_end	int	Real end of trip
alighting	int	Nr of passengers alighting
boarding	int	Nr of passengers boarding

Table 3.1: *Table of variables in the historical dataset from AVL- and APC-systems*

Variable	**Type**	**Description**
op_day	datetime	Operation day
line_no	int	Unique code of buss-route
route_direction	int	Flag to indicate direction of route
stop_name	str	Unique identifier of stop
stop_idx	int	Index of stop on the specific route
plan_start	int	Scheduled start of trip
plan_end	int	Scheduled start of trip
plan_arr	int	Scheduled arrival at stop
plan_dep	int	Scheduled arrival at stop

Table 3.2: *Table of variables in the plan dataset.*

Chapter 4

Methods

This chapter covers the methodology used for this thesis. We start by how data selection was done in Section 4.2 and how initial preprocessing was performed on the raw dataset in Section 4.3. In Section 4.5 we cover in detail how we calculated the load based on the number of boarding and alighting passengers. This was be used as the target in our models. Further in Section 4.6 we describe how additional features were added to the datasets. In chapter 3 errors related to passenger counting was presented. Section 4.7 covers a proposed method for detecting trips where errors occurred, as well as manual screening methods used to verify the precision of this method. Description of the methods used for exploratory analysis is covered in Section 4.8, before the creation of two different datasets for the two different cases is covered in Section 4.9. Illustration 4.1 shows and overview of the methods used in this thesis.

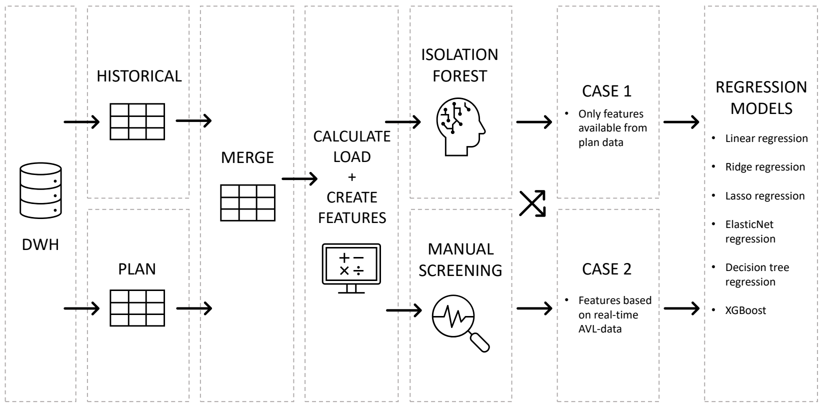


Figure 4.1: *Overview of methods used in the thesis.*

4.1 Software

An anaconda-environment with Python version 3.9.5 was used for this thesis. Numpy version 1.20.2 [20], pyodbc version 4.0.30, pandas version 1.2.4 [21], scikit-learn version 0.24.2 [9] and seaborn version 0.11.1 were utilized for analysis.

4.2 Data collection

Historical data from AVL- and APC- systems was accessed through an internal SQL-server. There were two separate tables, one with only data from SIS-vehicles, and one with data from both SIS- and Taas-vehicles. A table containing all plan data was also available on the same server. For a given bus line and time-period, a query was run to access data from the two historical-tables as well as the plan-table.

Data was gathered directly from the SQL-server into python using the python-package pyodbc, where it was stored as a pandas dataframe. From there column-names were changed to a standardized format, and data-types reformatted. In the original dataset time was recorded as seconds from midnight, these could be changed into Timestamp-objects as needed. Some data-wrangling was performed to change longer string-entries into shorter integer-entries. This mainly concerned stop-names, which sometimes included additional information specifying a platform at larger bus-stops.

Copies of the raw data, both historical- and plan-data, were stored as .pkl-files (pickle-files) before further preprocessing was done.

4.3 Preprocessing

Data rarely exist in a format and structure that can be fed directly to a machine learning model. Optimal performance can also not be expected when training models on raw, unprocessed data. Preprocessing can therefore be thought of as the process of building good training datasets [8].

4.3.1 Identifying individual trips

As mentioned in the previous section, two separate tables for historical data was accessed; one with only SIS-data, and one with SIS- and Taas-data. After collection it became apparent that key columns in the data from Taas-vehicles were missing. Because of this the identification of individual trips needed to be performed in two stages:

SIS

To aid in preprocessing, and in order to merge historical and plan data a new column, `trip_id`, was created in each dataset. This `trip_id` was made as shown in Listing 4.1 by combing five columns (`line_no`, `line_direction`, `plan_start`, `plan_end` and `op_day`) in order to create a unique identifier for every trip. The same method was also used for historical data from SIS-vehicles.

```
plan['trip_id']=(plan['line_no'].astype(str) + '_' +
                 plan['line_direction'].astype(str) + '_' +
                 plan['plan_start'].astype(str) + '_' +
                 plan['plan_end'].astype(str) + '_' +
                 plan['op_day'].dt.strftime('%Y-%m-%d'))
```

Listing 4.1: *Creating trip_id*

From the historical data it became apparent that some `trip_id`s had registrations from more than one vehicle. In order to examine this, an additional column was created called `vehicle_trip` as shown in listing 4.2.

```
historical['vehicle_trip'] = (historical['vehicle']
                               .astype(str)) + '_' +
                              (historical['trip_id']
                               .astype(str))
```

Listing 4.2: *Creating vehicle_trip*

Taas

Taas-data did not include the features `plan_end` and `act_end`. Because of this it was not possible to create the same trip-id for taas-trips as for SIS-trips. In order to merge taas-trips with planned trips a shortened version of trip-id were constructed. This feature, aptly named `trip_id_short`, were constructed similarly to 4.1, but without including `plan_end`. This same feature was added to plan data.

4.3.2 Removing duplicates

Before we started any preprocessing we checked for duplicates in the raw data. Listing 4.3 shows how this is done in pandas, and what subset is assumed to be unique.

```
plan = plan.drop_duplicates(subset = ['op_day',
                                       'line_no',
                                       'line_direction',
                                       'stop_name',
                                       'plan_start',
                                                'plan_end'
                                                'plan_arr'])
```

Listing 4.3: *Removing duplicates in plan data*

In order to account for two vehicles being signed on to the same trip, `vehicle` was added to the subset when detecting duplicates in the historical data.

4.3.3 Merging plan- and historical data

Before plan- and historical- data were merged, some changes were made in order to make them compatible. This included changing some stop-names as these seemed to vary a bit between the two datasets, and also within datasets.

For whatever reason `route_direction` in plan data was indicated by 0 and 1, while it in the historical data was indicated by 1 and 2. In order to make them compatible `route_direction` in historical data was changed to 0 and 1.

SIS

First, plan and historical data from SIS-vehicles were merged using the pandas-function `merge` as shown in listing listing 4.4.

```
data = plan.merge(historical,
                  on = ['trip_id', 'stop_name'],
                  how = 'left')
```

Listing 4.4: *Merging historical and plan data*

By setting the parameter `how='left'` all entries from plan is kept, even if there is no data in historical for it to mach on. Because some `trip_id`s have more than one vehicle related to it, the resulting dataframe is expected to be a bit larger than plan.

Taas

In order to merge Taas-trips with plan data we identify all trips with Taas-data. This was done by subtracting the set of all unique values in `trip_id_short` in the dataset containing only SIS-trips from the set of all unique `trip_id_short` in the dataset with data from both Taas- and SIS-vehicles. Data related to these short trip-ids should contain NaN-values for all APC- and AVL-features in the merged dataset created in the previous section. Rows containing any of the short trip-ids identified is removed from the previously merged dataset.

Two new dataframes containing plan-data and historical-data for trips covered by Taas-data is created. These are merged using the same method as Listing 4.4, but with `trip_id_short` instead of `trip_id`. Finally, the merged dataset with Taas-data is concatenated with the earlier merged dataset.

4.3.4 Filling in data where vehicle passed a stop

The historical data only contain entries when a vehicle stopped at a bus-stop, meaning that no entry exist if the vehicle drove past a stop. After merging, stops where the vehicle did not register data will have missing values in other columns as well. These include `vehicle`, `act_start`, `act_arr`, `act_dep`, `act_end`, as well as location in latitude and longitude. Because `vehicle`, `act_start` and `act_end` are global for each trip, they can be filled inn by backward-filling. That means filling in NaN values in these columns with values in subsequent rows if available.

We chose to only use backward-filling and not forward-filling because some trips lack data only at the end of the trip. When using forward-fill we observed a strange pattern of static load at the tail end of these trips. By that we mean several stops at the end of trips were the load was unchanging on an otherwise normal trip.

The values for `act_arr` and `act_dep` are more difficult to fill. In order to approximate what these values actually would have been we can calculate the delay in arrival and departure at all stops where `act_arr` exists and use backwards-fill to fill in rows here these delays are missing. For each stop where `act_arr` and `act_dep` is missing they can be approximated by replacing them with the sum of `plan_arr` and `delay`.

Location

In order to fill in location in latitude and longitude where they were missing we identify the mean values for both `latitude` and `longitude` for all stops in our dataset. We also calculated the standard deviation for the locations in order make sure that there were not too much variety in their values. These mean-location values were substituted where location-data was missing.

4.4 Identifying route id

The raw data included variables for both `line_no` and `route_direction`, but during the time-period several permutations of the order of stops were observed within the same `line_no-route_direction` combination. Based on plan-data we extracted the order of `stop_names` within each `trip_id` and created a unique `route_id` for each unique combination of stops.

4.5 Calculating load

Raw data from the APC-sensors only contain the number of passengers boarding or alighting the vehicle. The accumulated load needed to be calculated. This load is what was used as the target for all models in this thesis. We are basing this on the methods outlined in the report "Using Archived AVL-APC Data to Improve Transit Performance and Management" published by the Transit Cooperative Research Program (TCRP) in 2006 [22].

We differentiate between three types of load:

1. *Arriving load:* The number of passengers on the vehicle when it is arriving at a stop.
2. *Through load:* the number of passengers on the vehicle after alighting passengers has left, but before new passengers has boarded
3. *Departing load:* The number of passengers on the vehicle when it is leaving the stop.

That means that *departing load* on the first stop is identical to *arriving load* on the next stop. We make the distinction between these types of load when performing the calculation, and aim to use the terminology in future discussion. If no indication for type of load is provided one can assume it is *departing load.*

In order to calculate load we made the following assumptions:

1. *Arriving load* at first stop and *departing load* at last stop is zero:

$$L_{0,arrival} = 0 \tag{4.1}$$

$$L_{n,departure} = 0 \tag{4.2}$$

2. *Through load* at any stop can never be less than zero:

$$L_{through} \geq 0 \tag{4.3}$$

3. *Departing load* at stop i equals *arriving load* at stop i+1:

$$L_{i,departure} = L_{i+1,arrival} \tag{4.4}$$

Load was calculated on trip level by grouping the data by vehicle and `trip_id` and applying the function outlined in Listing 4.5. In order to adhere to constraint 2, we did not allow the load to be bellow 0. If while taking the accumulated sum the load was bellow 0, it was set to 0 before further summation was performed. This is expected to have skewed the data to some extent, as will be discussed further in chapter 6.

```python
def calculate_load(boarding, alighting):
    load = 0
    calculated_load = []
    for boarding, alighting in zip(boarding, alighting):
        load -= alighting
        if load <= 0:
            load = 0
        load += boarding
        calculated_load.append(load)
    return calculated_load
```

Listing 4.5: *Function used for calculation load on a trip given ordered lists of boarding and alighting passengers. The function returns a new ordered list containing the calculated load.*

4.6 Feature engineering

With the goal of improving the performance of our models, new features were extracted from the raw data. In the following sections we provide a brief explanation of how and why new features were extracted from both APC- and AVL-data.

We distinguish between features that are engineered based on data within one trip, and features engineered based on data from previous trips. These are referred to intra-trip features and cross-trip features respectively.

4.6.1 Intra-trip features

Day characteristics

Traffic vary from day to day, and differ between weekdays and weekends. To that end we used the variable `op\_day` in order to extract information about any specific date. We created a column containing a number of 1-7 depending on what day it is (Monday = 1, Sunday = 7). We also created a flag for whether it is a weekend or not (weekday = 0, weekend = 1). Columns were also created for month of the year (0-11) and day of the year (0-364).

The dataset used in this thesis overlapped with the last days of the summer holiday for public schools in Oslo. Fall break also fell within the time-constraints for this thesis. Columns were created for both, where `fall_break = 1` and `summer_break = 1` indicated the different holidays.

Time characteristics

As mentioned in Section 3.3 all timestamps were originally given as seconds from midnight (int). For ease of computation these were kept as integers until conversion was needed for either further feature extraction or visualization. This is because analysis with datetime-objects are computationally exhaustive and scikit-learn does not support datetime-objects as input in their models.

Similar to the reasoning for extracting day characteristics we may expect traffic to vary over the course of a day. To this end we used the timedelta-function in pandas to convert the time-columns into a timedelta-object, before adding them to the DateTime-object for `op_day`. This was done to account for the fact that the original time-columns sometimes contains values larger than 86 400, which is the total number of seconds in a day. This is the case for trips starting before midnight and ending after midnight. From the DateTime-object we extracted hour and minutes past the hour.

On weekdays the passenger-load is expected to be larger during rush-hour. As part of the exploratory analysis the hours of the day where the sum of boarding passenger was largest was identified, and stops made in those hours were defined as rush-hour. The identification of these hours is included in the Results (Section 5.3). A new column, `rush_hour`, was added for these hours.

Delay of bus

It is only to be expected that the number of passengers boarding at a given station varies with how much the buss is delayed. If the buss is several minutes late we can expect more passengers to be waiting. On the other hand, if the buss arrives before the assigned time the number of boarding passengers might be less then expected.

We found the current delay of a vehicle by comparing planned arrival time with actual arrival time. This was done directly by subtracting `act_arr` from `plan_arr`. Delay for both start and departure were calculated the same way. Delay at the end of the trip was not included as this feature, naturally, never will be available in real-time.

Dwell-time

One might imagine that the amount of time a vehicle spends on a stop is related to how many people are alighting and boarding it. If there are a lot of people boarding and/or alighting, the vehicle may spend longer time at the stop. Similarly, it may not need to dwell for a long time if there are few (or none) passengers boarding or alighting and if a vehicle passed a stop altogether, dwell would be zero. If the bus is ahead of schedule it may also need to dwell for an extended period of time.

In order to represent this in the dataset, the feature act_arr was subtracted from act_dep. This was stored as a new feature *dwell*, containing integers representing the number of seconds the vehicle had spent at the given stop.

Accumulated dwell-time

In the same vain as the previous feature, the total accumulated time the vehicle has been dwelling at stops may be related to the load on the vehicle. If there is a lot of traffic on a given departure, the vehicle may have spent more time dwelling. If, on the other hand, passenger demand is so low that the vehicle passed several stops, this could be reflected in lower accumulated dwell-time.

By taking the accumulated sum of the feature dwell, we get a new feature. This is named accumulated_dwell. If we observe a large variation in dwell at the first stop of a line, we may have to correct for it by setting it equal to zero. This is because the vehicle may spend a significant amount of time at the first stop waiting for the trip to depart.

4.6.2 Cross-trip features

One may expect that any trip could be affected by the previous trip on the same route. If the previous trip was delayed there may be more passengers on that trip, and fewer on the next. This could potentially lead to bus-bunching, where a greater number of passengers on the delayed trip lead to greater delays, but subsequent trips run ahead of schedule because of reduced dwell-time caused by fewer passengers. If all trips are delayed, there may be less crowding than expected because the spacing between trips are preserved even though the trips are delayed.

During preprocessing the trip-id of the previous trip was identified. This was done by grouping trip-ids by operation day and route-id, before shifting all values one row down. After having identified the previous trip, all AVL-features from the previous trips were merged with the original data. The prefix ”previous_” are added to these features.

Time since previous buss

By subtracting previous_act_arr from act_arr we get a measurement of the time since a vehicle last passed the stop. This variable is called *headway*. Since we are subtracting an integer from an integer, our resulting column *headway* will also contain integer. If either previous_act_arr or act_arr are NaN's this subtraction would result in NaN.

We also find the difference between previous_plan_arr and plan_arr, which we call planned_headway.

Delay of previous buss and relative delay

We also identified the relative delay between the current and the previous trip. That is a measure for whether the current trip has caught up with or fallen behind the previous one.

We started by identifying the delay of the previous trip when it passed a given stop and storing it as a new variable `previous_delay`. By subtracting `delay` from `previous_delay` we got an integer which is positive if the current trip is less delayed than the past, and negative if it is more delayed.

4.7 Outlier detection

There is uncertainty related to all types of sensor data, and some amount of error is to be expected. In Section 3.2.2 we mentioned that improper installation or lack of maintenance can lead to faulty sensors. The behaviour of passengers and errors in sign-in can also affect data quality as discussed in Sections 3.2.4 and 3.2.3. This is on top of the fact that the sensors may have a tendency to over- or under-count the number of boarding and/or alighting passengers in the first place (3.2.1).

In order to remove erroneous data we needed to identify trips with faulty sensors or other issues, and separate them from those with a "normal" amounts of error. In Section 2.3.4 we introduced the ensemble model isolation forest which we used to identify the trips with erroneous data. The implementation of isolation forest to detect invalid trips is covered 4.7.1.

In order to evaluate the efficacy of outlier detection by isolation forest we compared it to manual screening. The different screening-methods used are presented in Section 4.7.2. This screening was based on observed anomalies in the data and can not be expected to be comprehensive of all possible errors.

Creating dataset for outlier detection

The purpose of this outlier detection was to identify *trips* with invalid data. To this end we aggregated and summarized data based on vehicle and `trip_id` before applying the isolation forest-model.

Variables that are constant for each trip, such as `line_no`, `route_direction`, `route_id`, `plan_start`, `act_start`, `plan_end` and `act_end`, were implemented directly. Delay in both start and end were included, and total trip-time was calculated by subtracting `act_start` from `act_end`. The number of datapoints related to each trips was measured using the pandas-function `.count()`.

Based on boarding and alighting the following features were added: The sum of boarding-passenger, the sum of alighting passenger and the difference in the sum of boarding and alighting passengers. For both boarding and alighting we also made columns for mean, median, and max. The last four aggregations were also performed on `departing_load`. Variables that are not constant, such as `delay_arr` and `dwell_time`, can also be aggregated using mean, median, min and max.

Lastly, a column is added to indicate how many times during a trip the number of boarding passengers were 0. The same is done for the number of alighting passengers. These are called `zero_count_boarding` and `zero_count_alighting` respectively.

4.7.1 Isolation forest

As mentioned in Section 2.3.4, isolation forest is an ensemble method used to detect outliers by measuring the number of random splits needed to isolate an entry in the dataset. Isolation forest is an unsupervised learner, meaning that there is no target or "known answer". The datset created in the preceding section is fed directly to the isolation forest model and both fitting and prediction is done through the `fitpredict`-method.

The implementation of isolation forest as part of scikit-learn's ensemble-module has a number of parameters that can be set by the user, most notably n_estimators and contamination. The first is the number of base estimator trained for the ensemble, and by default this value is 100. Contamination indicates the proportion of outliers in the dataset. If it f.ex is set to 0.1, 10% of entries will be removed. By default this value is set to 'auto' meaning that the level of contamination is determined as in the paper where the model was introduced [14].

Another parameter for isolation forest is `n_jobs`. This allows parallel fitting and prediction of multiple isolation trees. By setting `n_jobs=-1` all available processors will be used, thus greatly reducing the time needed for fitting and prediction.

The output of the isolation forest-model is a new column in our dataset indicating whether the trip is classified as an outlier or not. The value -1 indicates that the trip is an outlier, while 1 indicates that it is not.

4.7.2 Manual screening

From initial inspection of the historical data it became apparent that some screening of the data was needed. This was mainly related to two issues we have observed in the historical data. The first was that some of the passenger-data seems to have been assigned the value 0 instead of NaN-values (*Not a Number*). This was screened for on trip-level. We have not been able to figure out how or why this issue arises. The second is related to possible errors in sign-on discussed in 3.2.3, where both passenger- and location-data can be linked to the wrong trip as a result. This was typically recognizable when the start of the trip was delayed by an unlikely amount. The typical amount here is a couple of hours up to one day.

This screening was performed using the same dataset as created in Section 4.7, and the result of each screening was stored in the same dataframe for comparison.

The following criteria were used to classify a trip as an outlier when manually screening data:

- **Defect sensor:** A trip is set to *defect sensors* if the absolute difference divided by the mean of boarding and alighting passenger is greater than 0.2.
- **Nonreporting trip:** A trip is set to be *nonreporting* is either the sum of boarding passenger *or* the sum of alighting passengers are 0.
- **Lowreporting trip:** A trip is set to be *lowreporting* if there are less than 3 passenger boarding or alighting during the whole trip.
- **High zero percentage:** A trip is set to have *high zero percentage* if more than 80% of stops made have zero boarding or alighting passengers.
- **Low/high observation count:** A trip is set to have *low observation count* if there are less than X stops where the vehicle collected data, and *high observation count* if there are more than 60 stops where the vehicle collected data.
- **Large departing load:** A trip is set to have *large departing load* if there are more than X passenger at any stops during the trip.
- **Large delay start:** A trip is set to have *large delay start* if it is more than 15 minutes delayed at the start of the trip.
- **Large delay end:** A trip is set to have *large delay end* if it is more than 20 minutes delayed at the start of the trip.

4.8 Exploratory analysis

4.8.1 Assessing coverage of plan data

During the course of preprocessing, load-calculation, feature engineering, and particularly outlier detection, we will be assessing how well the data cover planned trips. We distinguish between the tree different lines and the route-direction in order to make sure that none of the methods are affecting one line or one route-direction to a greater extent.

4.8.2 Evaluating missing data

The historical data is expected to contain missing values, and by examining the proportion and distribution of missing data, we can make an assessment about how they should be treated in the analysis. If the missing values are deemed to have the characteristics *missing at random* or *missing completely at random*, the missing values may be discarded altogether. If on the other hand the data is deemed to be *missing not at random*, other provisions have to be made to treat them correctly.

We distinguish between two different circumstances of the missingness of our data. The first are missing observations on trips where there otherwise is data, indicating that the bus simply skipped a stop. These are assumed to be missing at random as they may be dependent on features in our dataset, such as `stop_name` and `day_of_week`, but are not dependant on load.

The other circumstance where missing data may appear in our dataset are cases when all observations of a trip are missing. This would indicate that the vehicle has faulty equipment and we have no way directly inferring what the unobserved measurements were.

As the purpose of this thesis primarily is to predict the relative load for each trip, that is how full a vehicle is compared to another, we may afford ourselves some leeway here. By removing or imputing missing data where the mechanism is not random, the characteristics of the data may change. This may be detrimental if the purpose of the analysis is to report *exactly* how may passengers there are aboard a given vehicle. For our purpose we may make the assumption that one vehicle is fuller than another if its reported number of passengers are lager, even though the exact values are not correct.

4.9 Constructing data-sets

In all cases the target is the column `departing_load`. This is the number of passengers aboard the vehicle when it leaves the station, as introduced in Section 4.5, and is calculated using the method from the same section.

This section covers what features are included in the data-sets for the four different cases. An overview of the features present in the different cases are presented in Table 4.1a.

Case 1

The dataset for case 1 contain only the features from plan-data. Extraction of day-characteristics based on `op_day` is included, as well as time-characteristics based on `plan_arr`. Table 4.1a and 4.1b indicate which features are included.

Variable	Case **1**	Case **2**
op_day		
line_no	✓	✓
route_direction	✓	✓
stop_name		
stop_idx	✓	✓
plan_start	✓	✓
plan_arr	✓	✓
plan_dep	✓	✓
plan_end	✓	✓
vehicle		
act_start		✓
act_arr		✓
act_dep		✓
act_end		
longitude	✓	✓
latitude	✓	✓
alighting		
boarding		
trip_id		
route_id	✓	✓
departing_load	✓	✓

(a) Primary features.

Variable	Case **1**	Case **2**
day_of_week	✓	✓
day_of_year	✓	✓
month	✓	✓
weekend	✓	✓
summer_break	✓	✓
fall_break	✓	✓
hour	✓	✓
minute	✓	✓
rush_hour	✓	✓
source_system	✓	✓
delay_start		✓
delay_arr		✓
delay_dep		✓
delay_end		
previous_trip_id		
prev_plan_start	✓	✓
prev_plan_arr	✓	✓
prev_plan_dep	✓	✓
prev_plan_end	✓	✓
prev_act_arr		✓
prev_delay_arr		✓
prev_delay_start		✓
rel_delay_arr		✓
rel_delay_start		✓
headway		✓
planned_headway	✓	✓
dwell_time		✓
acc_dwell_time		✓

(b) Engineered features.

Table 4.1: *Features included in datasets for case 1 and case 2.*

Case 2

Case 2 includes features based on AVL as well as those used in case 1. This includes intra-trip calculations such as delay, dwell-time and accumulated dwell-time. It also includes AVL-features from the preceding trip such as relative delay and headway. Table 4.1a and 4.1b indicates what features are included for case 2 as well.

4.10 Training, testing and validation

The original datasets contained 3 236 083 historical entries and 3 635 174 plan entries collected from August 1. to December 1. 2019. In order to train and test our models datasets for both case 1 and case 2 are split into two dataset:

1. **Training:** August 1. - November 1.
2. **Test:** November 02. - December 1.

The training-data is used for training and validating models, while the test data is set aside and only used for final testing and comparison of models.

4.10.1 Evaluating outlier detection method

Two version of training data are created: One where outliers detected by the isolation forest is removed, and one where outliers detected by manual screening is removed. All models are trained on both these datasets, resulting in two models of the same type. Details about the amount of outliers removed with each method is provided in Section 5.4.

In order to compare the performance of the two outlier detection-methods the two models are testes using one common test set. In this test set all trips identified as outliers by either isolation forest or manual screening are removed.

4.10.2 Cross-validation

Because the data can be thought of as a time-series, the order of data needs to be preserved. In order to evaluate models and perform we use the folds presented in Table 4.2.

Fold	**Training**		**Validation**	
	From	To	From	To
1	01.08	27.08	31.08	06.09
2	01.08	03.09	07.09	13.09
3	01.08	10.09	14.09	20.09
4	01.08	17.09	21.09	27.09
5	01.08	24.09	28.09	04.10
6	01.08	01.10	05.10	11.10
7	01.08	08.10	12.10	18.10
8	01.08	15.10	19.10	25.10
9	01.08	22.10	26.10	01.11

Table 4.2: *Training and validation folds*

Chapter 5

Results

This chapter covers all results that were obtained from following the methodology outlined in chapter 4. Further discussion of the results will be provided in chapter 6. Section 5.1 covers how data was collected. Preprocessing is done in Section 5.2, before Exploratory analysis is performed in Section 5.3. In Section 5.5 the results for all models in the different cases are presented.

5.1 Data collection

Historical data and plan data is collected with the constraints:

```
start = 2019.08.01
stop = 2019.12.01
lines = [20, 31, 37]
```

This results in two data-frames with 3 236 083 historical entries and 3 635 174 plan entries. By only looking at the size of the dataframe, the historical data covers about 89.02% of plan data. This means that we are missing data from 10.98% of planned stops, either because whole trips are lacking data or because a bus skipped a stop.

5.2 Preprocessing

5.2.1 Examination of raw data

Duplicates are removed as described in Section 4.3.2. This results in 0 entries being removed from the historical data, and 252 from plan data.

Initial examination of the raw historical data shows that 2.93% of entries have missing values for boarding and alighting. This suggests faulty APC-equipment. Similarly 2.52% of entries have missing values in latitude and longitude, suggesting faulty GPS.

Out of a total of 207 vehicles SIS-vehicles, missing APC-values affect 13 vehicles. 6 of these vehicles are only reporting missing values, while the remaining 7 are only reporting a fraction missing values. In addition there are 54 Taas-vehicles, none of which have any missing data. As expected, raw plan-data does not include any missing values.

5.2.2 Merging historical and plan data

In total there are 122 441 unique trip_ids in plan data, and 118 359 in historical data. This suggest a 96.67% coverage. Overall, 7.8% of trips are made by Taas-vehicles. Table 5.1 shows the number of unique trips in the historical data for each line, and what percentage of those trips were made by Taas-vehicles.

Line no	**Nr trips**	**Percentage Taas-data**
20	37 839	2.64%
31	39 418	12.65%
37	41 102	8.11%
Total	118 359	7.80%

Table 5.1: *Percentage of data from each line generated by Taas-vehicles.*

Comparing historical data with plan data we also see that there are stops where the vehicle did not register data. This suggest that the vehicle drove by the stop without stopping to let passengers board or alight. In our data there are 20 985 planned stops that are not in the historical data (excluding trip_id where no data is registered). Boarding and alighting passengers at these stops are set to 0.

Historical data and plan data are merged, and missing values are filled in where vehicles passed a stop. Detailed explanations of how this is done can be found in Sections 4.3.3 and 4.3.4 respectively. Load is calculated using the method presented in Section 4.5, and features are added using the methods outlined in Section 4.6.

5.3 Exploratory analysis

Figure 5.1 shows the mean sum of boarding passengers for the different bus lines, differentiated by weekday and weekend. The translucent bands surrounding the line-plots represent the confidence interval (CI), which in this case is the standard deviation. Broader bands therefore means that there is a larger standard deviation between the sums of boarding passengers during that hour. We observe larger CI during the weekend, especially for line 20 and 37. During weekdays the CI is also slightly larger around the times commonly thought of as rush-hour.

Based on the peaks in the line-plots in Figure 5.1 the feature `rush_hour` is set to 1 for the hours 7, 8, 15 and 16.

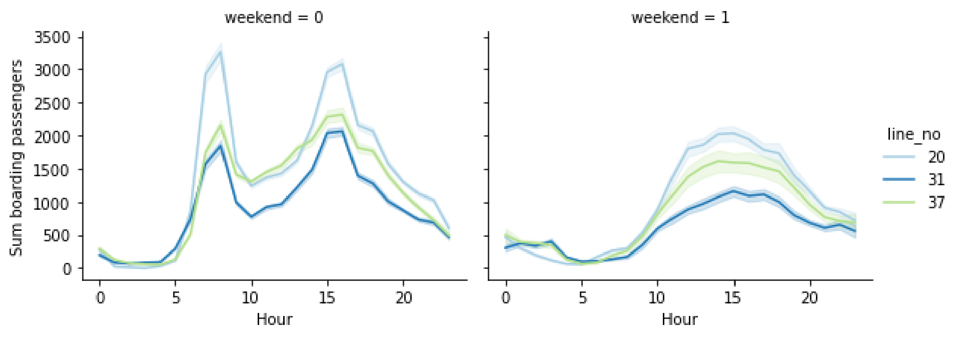


Figure 5.1: *Line-plot showing the mean sum of boarding passengers for every hour differentiated by line-number. Results for weekdays and weekends are shown side by side.*

5.3.1 Correlation matrix

Figure 5.2 shows correlation heatmaps for the features included in case 1 and case 2. As expected there is high correlation between planned time-variables, such as `plan_start` and `plan_arr`, and also a high correlation between time-variables of current and previous trip. The correlation matrix for case 2 show that real-time variables also are highly correlated to plan-variables as seen with the correlation between `act_arr` and `plan_arr`.

Features that are extracted from either operation day or time naturally have higher correlation to each other or the features they are based on. For example is `hour` correlated to `plan_arr` and `weekend` correlated to `day_of_week`. We also observe greater correlation between `line_no` and `route_id`, and between `stop_idx` and `accumulated_dwell_time`, witch both are to be expected. By looking at the correlating matrix for case 1 there are no features that are especially correlated to `departing load`, our target. For case 2 the features `delay_start`, `delay_arr` and `dwell_time` have a slightly higher correlating with our target.

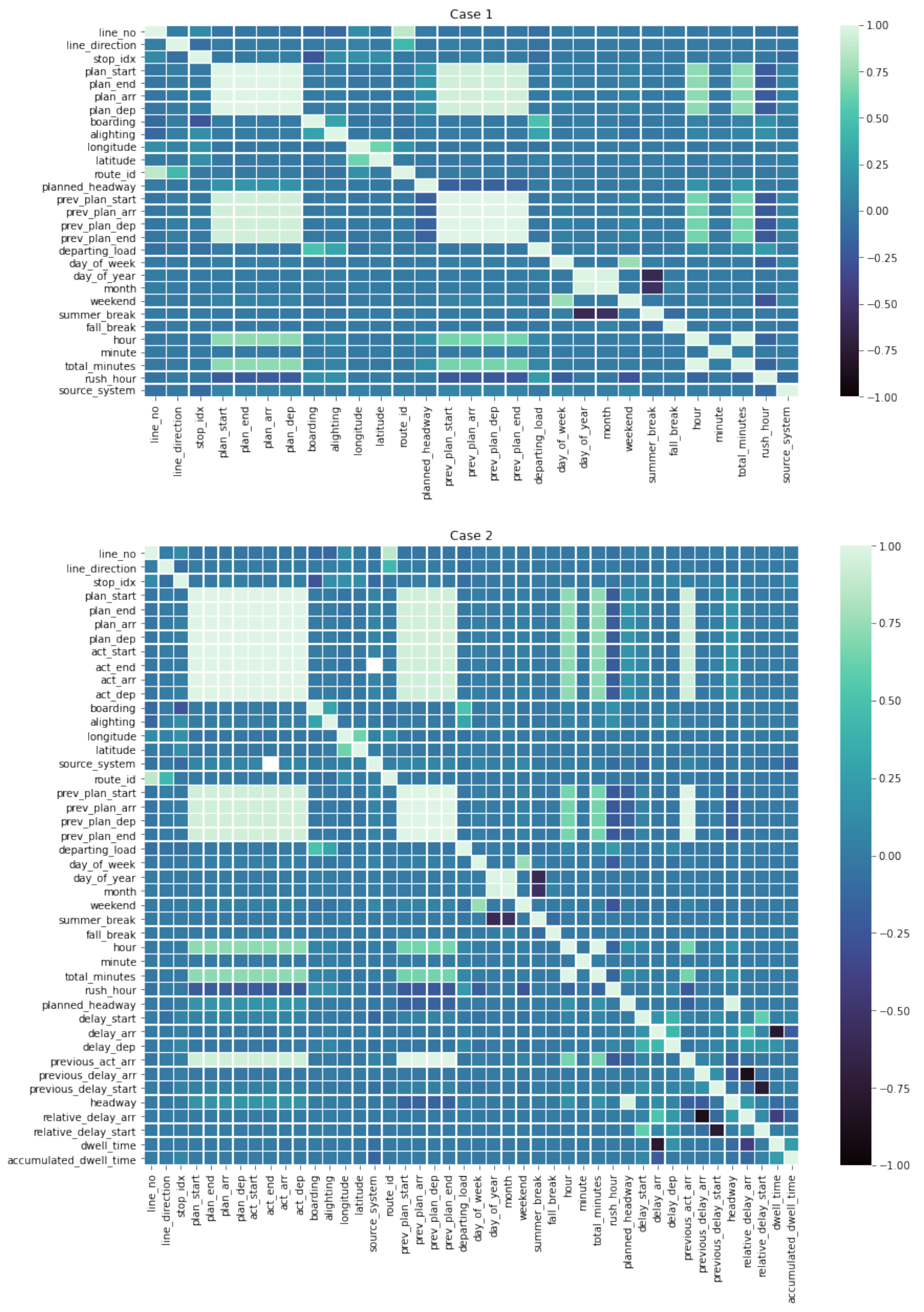


Figure 5.2: *Correlation heatmap illustrating the correlation between the features in the datasets for case 1 and case 2.*

5.3.2 Evaluating missing data

The three heatmaps in the first column of Figure 5.3 illustrates the trip-coverage in the raw data for all three bus lines. Looking at this we can assume that APC-data is Missing at Random (MAR). This is due to the fact that even though missingness is related to individual vehicles (and therefore not Missing Completely at Random (MCAR)), the chances of an individual vehicle having defect APC-equipment is assumed to be random. Due to this assumption we can take greater liberties in how we treat the missing values, and this allows us to remove trips with missing data and not impute the missing values.

We cannot assume data is missing at random if the chances of a vehicle having defect equipment was dependent on how many passengers were aboard the vehicle. The second column of heatmaps in Figure 5.3 gives us cause for concern. In these heatmaps all data from trips classified as non-reporting are removed. Here we observe a distinct pattern of a higher percentage of missing data during rush-hour for lines 20 and 37. Because the passenger-loads during rush-hour typically are greater this may indicate that the chances of missingness is connected to our target, the passenger load. If this is the case, the removal of data from trips with missing data may skew our predictions and result in poor predictions for trips during rush-hour.

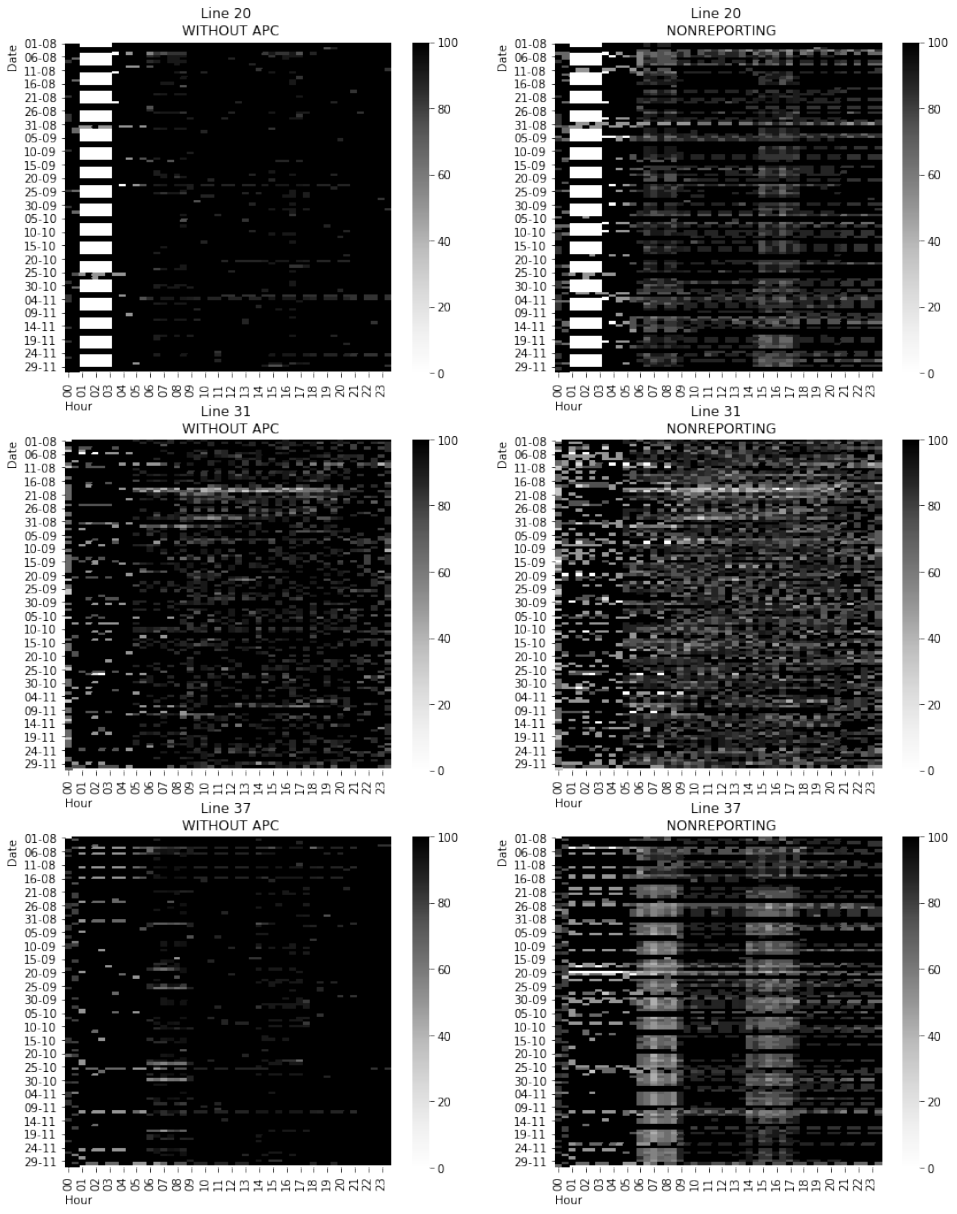


Figure 5.3: *Heatmap illustrating trip-coverage for trips where the historical data have APC-records, compared to trip-coverage when data from non-reporting vehicles are removed. The x and y-axis represents hour of the day and date of operation respectively. The color is determined by the percentage of trips during one hour with passenger data.*

5.4 Outlier detection

Outlier detection is performed after preprocessing and load-calculation, but before (or rather parallel to) feature engineering. The isolation forest-model is trained in order to detect outliers, and manual screening is performed as described in Section 4.7.2.

In order to find the ideal number of base estimators for the isolation forest, a list of increasing number of estimators are tested with the aim of finding where the model stabilizes. Figure 5.4 illustrates the result of this testing. The number of outlying trips (vehicle-trips) detected by the isolation forest is 7671, while manual screening yields 21 482 outliers.

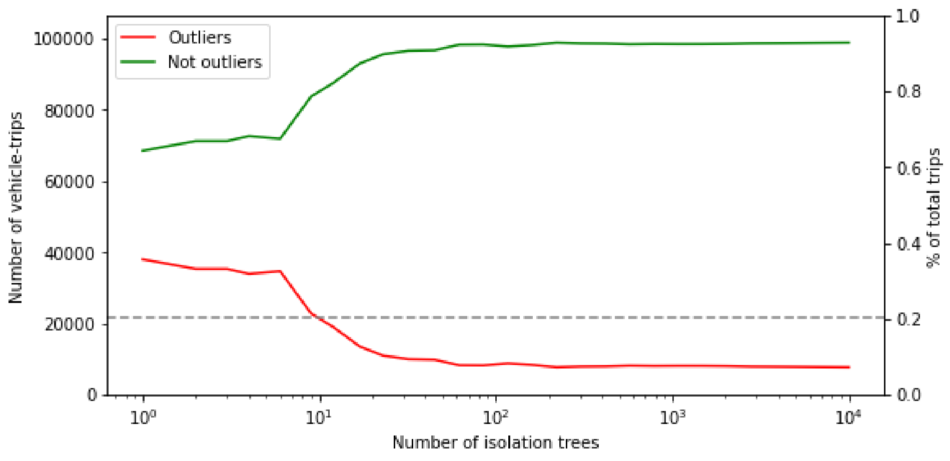


Figure 5.4: *Plot over the number of outliers detected by isolation forests of increasing size. The gray dashed line represent the number of outliers detected by manual screening.*

5.4.1 Comparing isolation forest and manual screening

The overlap between outliers detected by the final isolation forest model and manual screening is illustrated in Figure 5.5.

There is partial overlap between trips classified as outliers by the isolation forest and via manual screening, but the majority trips classified as outliers by manual screening is not corroborated by the isolation forest model. In Figure 5.6 the result of individual manual screening conditions are compared to that of the isolation forest. We observe that some of the screening conditions are to a larger extent included among the isolation forest outliers. This include nonreporting trips (where sum of either boarding or alighting is zero), trips with large departing load, and trips with large delay at end of trip. Most striking is the lack of overlap with trips classified as having defect sensors.

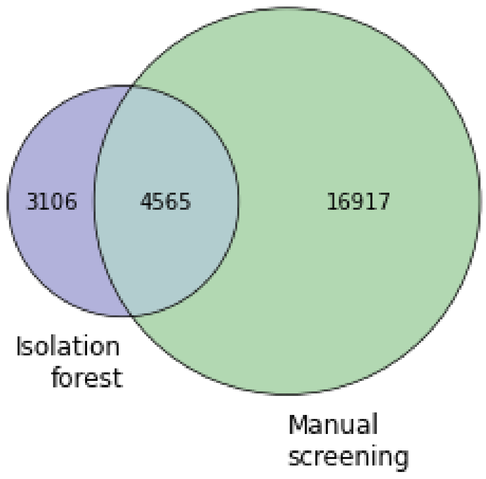


Figure 5.5: *Venn diagram illustrating the overlap between the outliers detected by isolation and manual screening.*

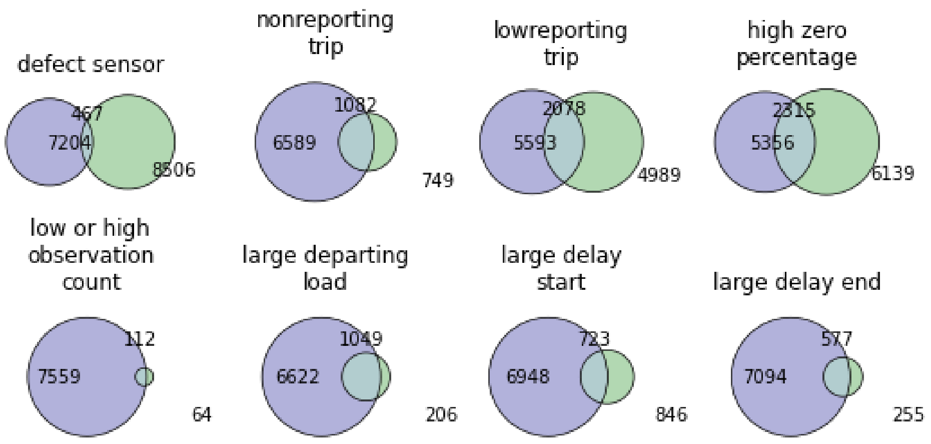


Figure 5.6: *Venn diagram illustrating the overlap between the outliers detected by isolation and manual screening. The blue circles to the left represent outliers detected by isolation forest, while the green cirle represent outliers detected by the different manual screening critera.*

5.4.2 Trip-coverage after outlier-detection

Table 5.2 compares how many vehicle trips are classified as outliers by the isolation forest model and by manual screening for the three bus-lines and their respective line-directions. We observe that the portion of outliers vary between the different bus lines/direction for both outlier detection methods. There is also a difference between the amount of outliers detected for the two different source-systems, as seen in Table 5.3, where the isolation forest removed far more Tass-trips than manual screening. This table also shows what proportion of trips are classified as nonreporting, and we observe that over half of Taas-trips are classified as nonreporting.

Line no	**Line direction**	**Isolation forest**	**Manual screening**
20	0	7.87%	5.60%
	1	7.13%	8.46%
31	0	23.84%	19.61%
	1	29.68%	38.34%
37	0	9.99%	44.49%
	1	4.87%	4.10%
Total		13.90	20.10

Table 5.2: *Portion of vehicle-trips classified as outliers by isolation forest and manual screening for both directions on all three bus lines.*

Source system	**Nonreporting**	**Isolation forest**	**Manual screening**
APT3	55.64%	26.27%	5.39%
SIS	6.48%	11.38%	19.16%

Table 5.3: *Portion of data from the two source-systems, SIS and Taas, classified as either nonreporting, or as outliers by isolation forest and manual screening.*

Trip coverage after removing outliers detected by isolation forest and manual screening can both be seen in Figure 5.7. There is a visible pattern of outliers detected during rush-hour on weekdays, especially by the isolation forest. This may be problematic, especially if trips are classified as outliers solely because of higher passenger load during rush-hour.

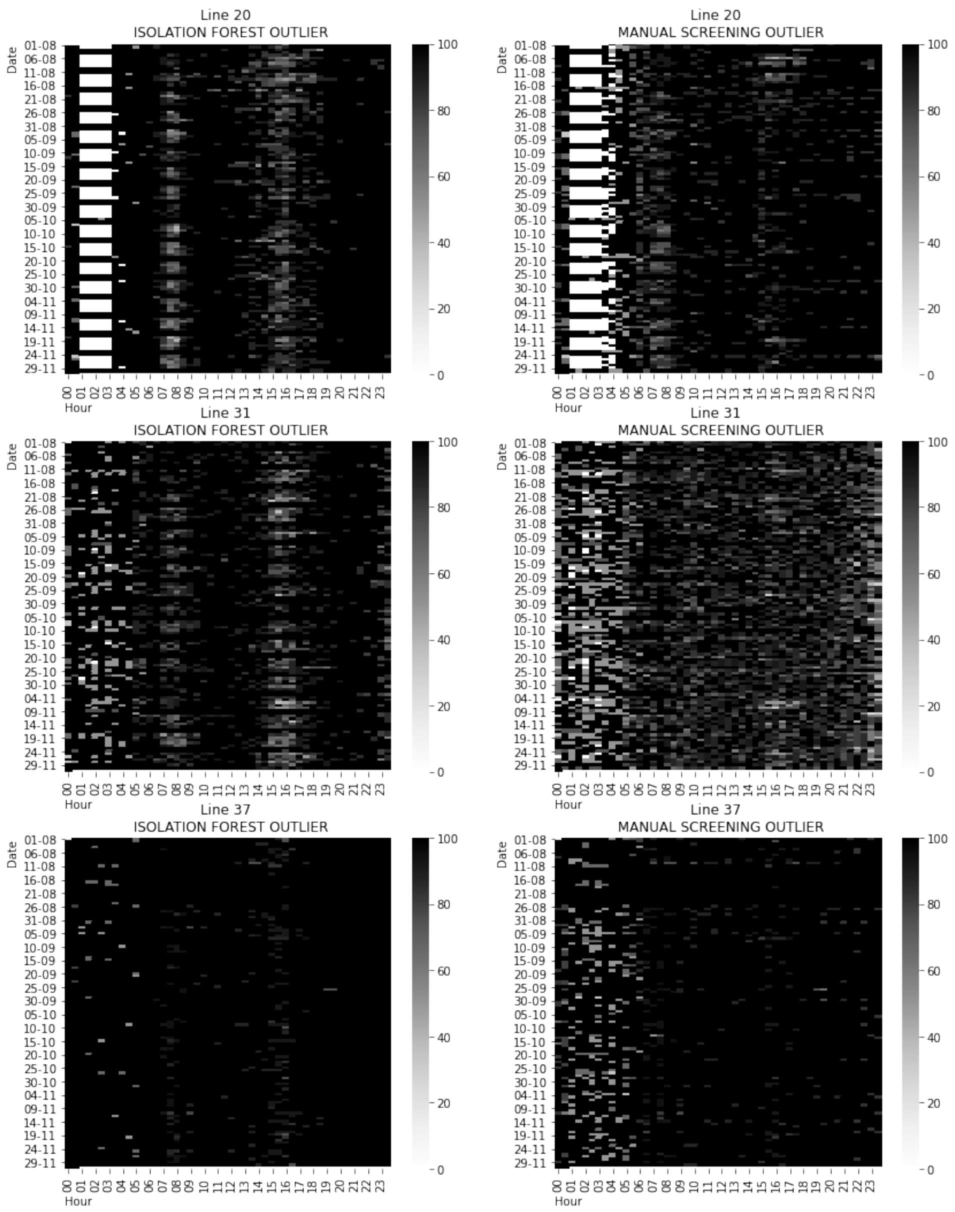


Figure 5.7: *Heatmap illustrating percentage trip-coverage for trips approved by the isolation forest-model compared to those approved by manual screening for all operation-days in the dataset.*

5.5 Models

Training and validation is performed as described in section 2.5. Figure 5.8 compares the performance of all models. The plot shows box-plot of mean absolute error calculated for every unique trip-id in the test-set. While in Figure 5.9, violin plots for the difference between the predicted load and actual load are shown. We observe a distinct difference in the shape of the violin plots for the linear models (linear, ridge, lasso and ElasticNet) and the violin plots for the decision tree regressor and XGBoost-regressor.

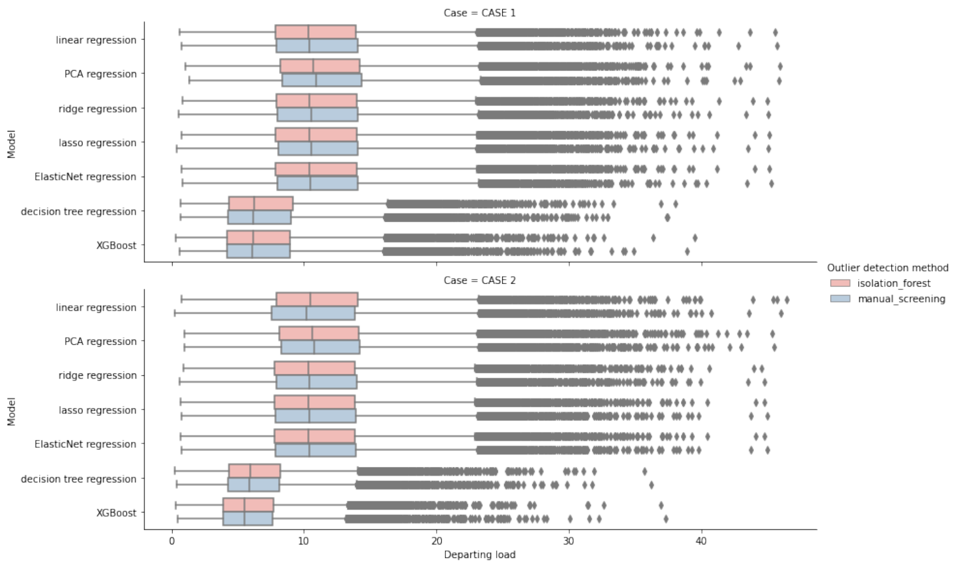


Figure 5.8: *Box-plots comparing distribution of the error-metric mean absolute error (MAE) calculated for every unique trip-ids in the test-data.*

5.5.1 Linear regressors

Through cross-validation, the hyperparameters listed in Table 5.4 were found to lead to best performance. The mean absolute error (MAE) and root mean square error (RMSE) listed are from the models trained on data where isolation forest was used for outlier detection. Note that `alpha=0` was selected as best parameter for lasso-regression, and that `l1_ratio=0` was selected for ElasticNet. This means that L1-regularization did not help improve model performance in any of the cases.

We observe similar performance by the linear regressor, PCR, ridge regressor, lasso regressor and elastic-net regressor, all yielding a mean absolute error slightly above 11 for the departing load. There is little difference between the performance of these regressors for case 1 and case 2. Across the board, models trained for case

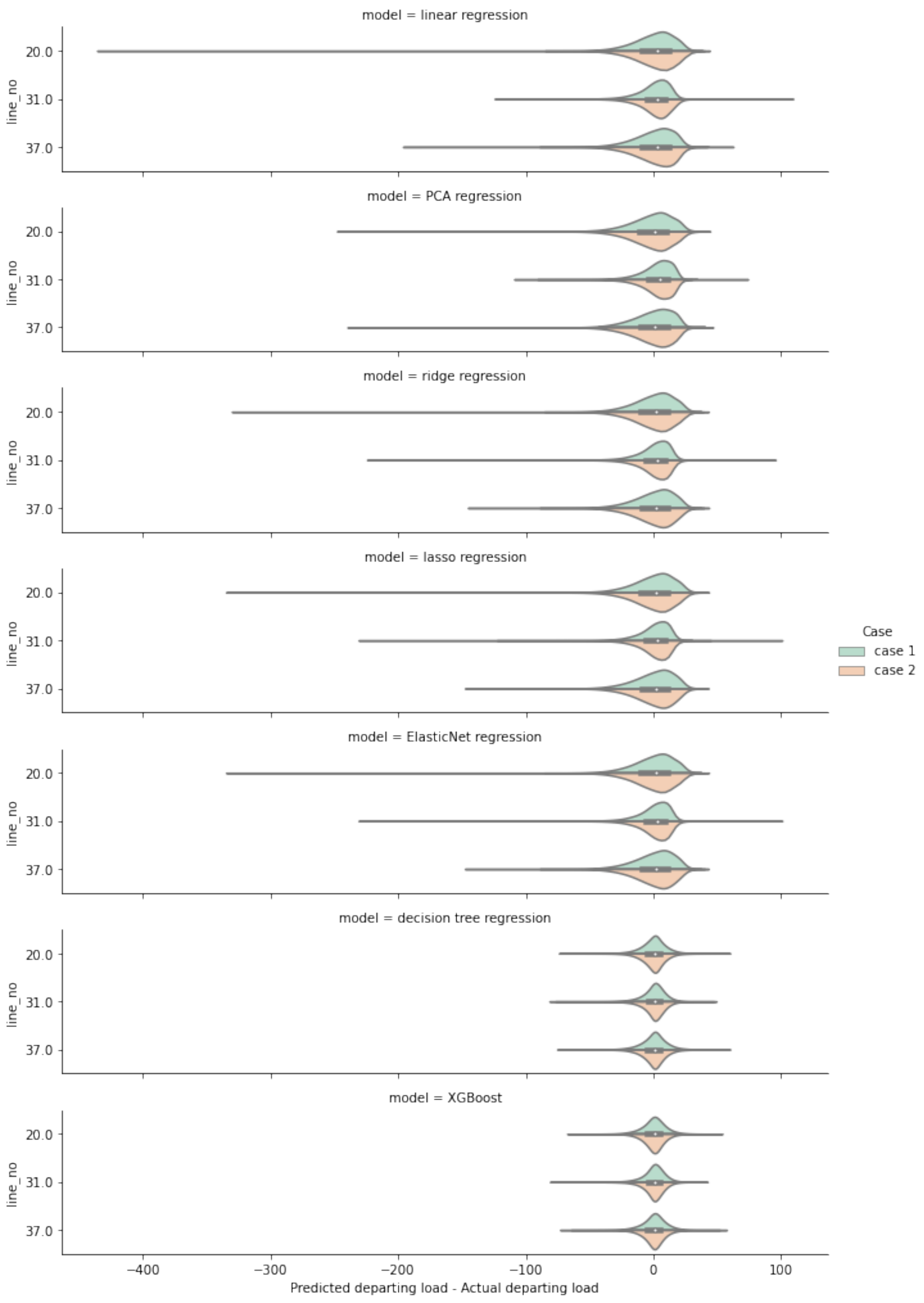


Figure 5.9: *Violin plots comparing distribution of the difference between the predicted departing load and the actual departing load for all models.*

Model	**Parameter**	**Value**		**Case 1**		**Case 2**	
		Case 1	Case 2	MAE	RMSE	MAE	RMSE
Linear reg				11.26	14.41	11.35	14.42
PCR	n_components	18	23	11.70	14.88	11.62	14.80
Ridge reg	alpha	0.02	0.01	11.32	14.48	11.20	14.37
Lasso reg	alpha	0.0	0.0	11.30	14.47	11.19	14.35
ElasticNet	alpha l1_ratio	0.0 0.97	0.0 0.57	11.30	14.47	11.19	14.35

Table 5.4: *Hyperparameters used for the linear models along with mean absolute error (MAE) and root mean squared error (RMSE) based on predictions on the test data.*

Model	**Parameter**	**Value**		**Case 1**		**Case 2**	
		Case 1	Case 2	MAE	RMSE	MAE	RMSE
Decision tree regression	max_depth min_samples_leaf	27 1.18e-4	22 8.59e-5	7.22	9.87	6.71	9.19
XGBoost	num_boost_rounds eta max_depth	305 0.18 6	499 0.12 6	7.10	9.69	6.26	8.51

Table 5.5: *Hyperparameters used for the linear models along with mean absolute error (MAE) and root mean squared error (RMSE) based on predictions on the test data.*

2 improved MAE by around 0.1 to 0.3 compared to models trained for case 1. We also observe little difference between performance when outliers are detected by isolation forest and manual screening.

5.5.2 Decision trees and XGBoost

Looking at the results in Figure 5.8, both the decision tree regressor and XGBoost-model stand out. Table 5.5 shows what hyperparameters were found to yield the best performance. This table also show that both mean absolute error and root mean squared error is significantly lower for decision tree regression and XGBoost compared to the linear models. This is true for both case 1 and case 2. Figure 5.10 shows violin-plots for the difference between predicted load and actual load, same as Figure 5.9, but only for decision tree regression and XGBoost.

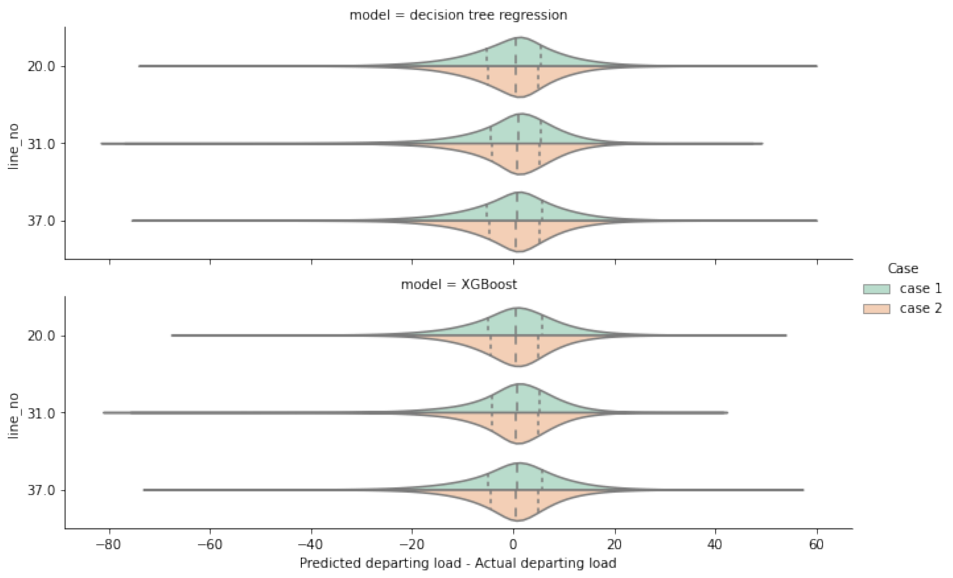


Figure 5.10: *Violin plots comparing distribution of the difference between the predicted departing load and the actual departing load for decision tree regression- and XGBoost-model. The two colors differentiate the results for case 1 and case 2.*

5.5.3 Best model

For both case 1 and case 2 XGBoost stand out as the best performing model,

In order to evaluate how the model perform during the course of an operation day we calculate the mean absolute error within each trip in the test set. These are aggregated based on start-time of the trip, and grouped into segments of 30 minutes. Figure 5.11 shows a heatmap with the maximum mean absolute error measured within each segment for our best performing model, XGBoost. The first columns shows the result for case 1 for the three bus lines, and the second columns shows the same for case 2. From this plot we see that MAE is larger during rush-hour on weekdays.

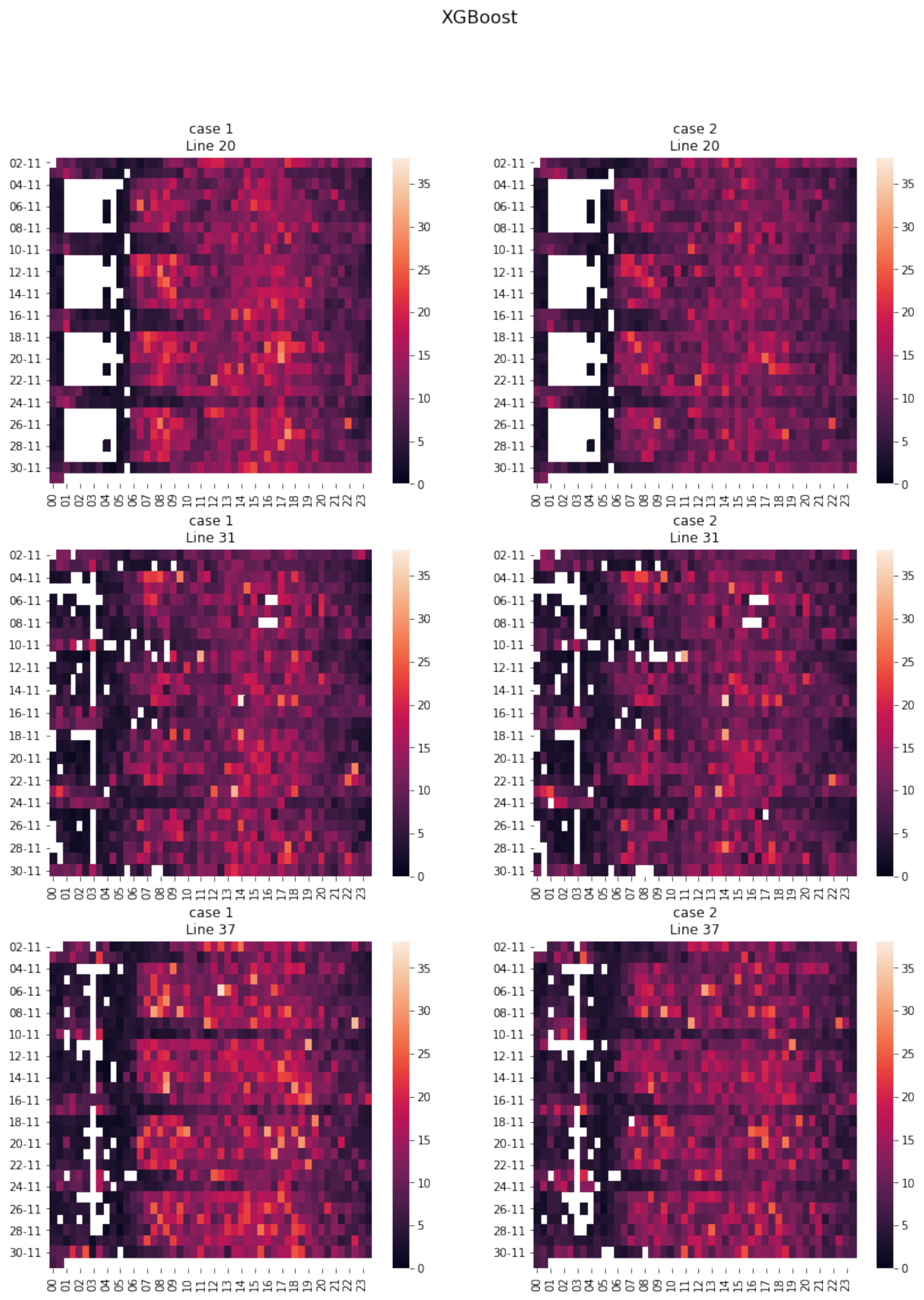


Figure 5.11: *heatmap illustration the maximum intra-trip MAE when trips are grouped into segments of 30 minutes.*

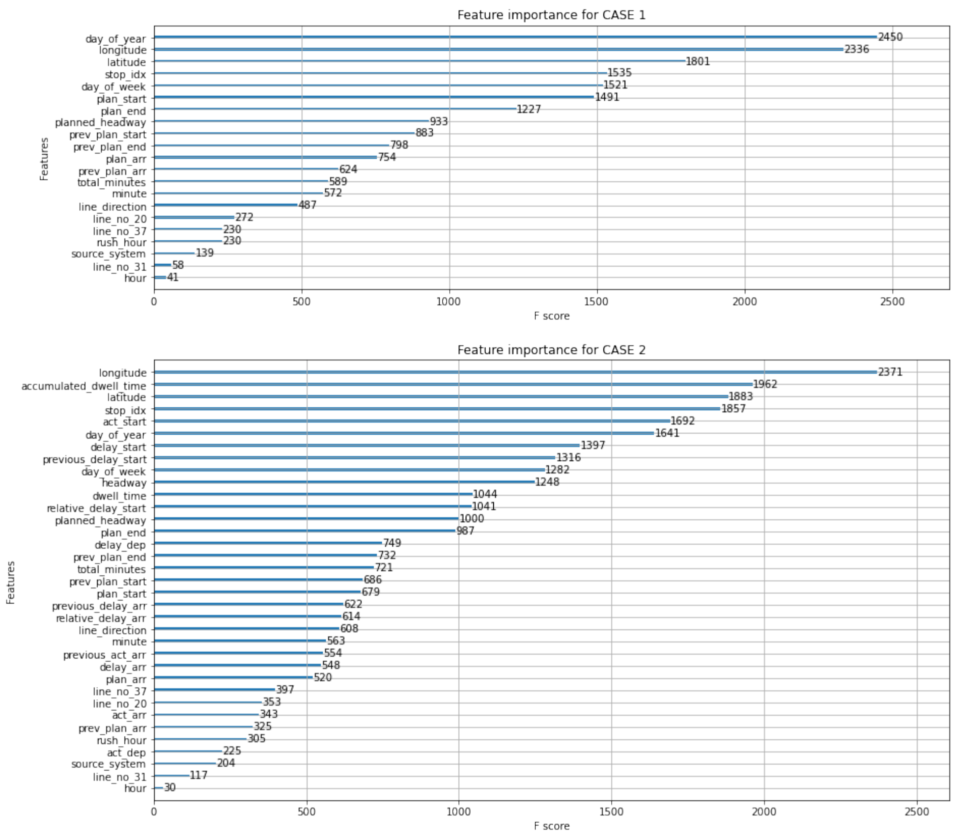


Figure 5.12: *Feature importance for the XGBoost models trained for case 1 and case 2.*

Feature importance

Looking at the plot of feature importance for the XGBoost-model in Figure 5.12 we observe that latitude, longitude and route-id constitute the top three features for both case 1 and case 2. Stop-index and day of week are also important feature for both cases. For case 1 we see that plan_start is the time-feature with highest importance, and likewise act_arr is for case 2. There is high correlation between all original time-features (plan_start, plan_arr, act_start, act_arr), which may explain why other time-features have not gained a higher score.

Out of the engineered features, headway (both planned and actual) and dwell stand out as successful. Particularly, accumulated dwell is the 4th most important feature for XGBoost in case 2, indicated that the total time spent by a vehicle at stops is a good predictor for load.

Chapter 6

Discussion

6.1 Data

6.1.1 Dataset used for the thesis

The bus lines and time-period examined in this thesis were selected in an arbitrary fashion and steps could have been made to make the selection more representative. All buss-lines examined in this thesis are frequented by many passengers, but we have no basis to compare their passenger demand to other bus lines in Oslo. We observed variation between the models' performance on the three different lines examined in this thesis, but do not necessarily have good explanations for these differences.

One may expect the methods proposed here would perform better on less frequented bus lines, particularly if the bus line is less prone to delays due to traffic. Similarly one may expect greater errors if the methods were applied to trips with greater passenger-flow and/or passing through more congested areas of the city. The methods proposed are intended to be universal, meaning that they can be applied to other bus lines with only minor adjustments. Short of doing this and observing the models performance, it is difficult to know the models accuracy for any bus line other then the three used in this thesis.

While collecting data from the internal server we chose to discard some columns deemed unnecessary at the time. One of these were a column assigning a unique stop-id to all stops in the dataset. This was discarded in favor of stop-name as stop-name was thought to be adequate. Later it became apparent that this stop-id would have been useful when matching plan- and historical data. In absence of stop-id some changes were made to stop-names in order to merge the datasets. This may have introduced some uncertainty. Another column containing a more precise description of the line-number was also discarded while initially collecting data. This line description may have helped differentiating between different routs, especially for line 31 which has two distinct routes, in addition to express-routes during rush-hour.

6.1.2 Data from Taas vehicles

Originally, two datasets were collected for historical data, one containing only data from SIS vehicles and one containing data from both Taas- and SIS-vehicles. The latter was intended at the main dataset, but unfortunately this dataset did not include `plan_end`. This meant that the trip-id needed to combine historical and plan data could not be made using this dataset. Additional preprocessing-steps needed to be performed in order for Taas-data to included in the final dataset.

Taas-vehicles were only responsible for 7.8 % of the data in this period, in the beginning we therefore decided to discard this data. Because the proportion of Taas-data was larger for line 31, this choice resulted in a greater proportion of data for this line not being used for analysis. After examining the trip-coverage it also became apparent that Taas-vehicles were frequently in operation during rush-hour. This meant that we discarded a high proportion of data during rush-hour by choosing to drop Taas-data. In the future one should make sure that the column plan_end is included, or find an alternative way to identify individual trips.

Based on a cursory comparison of APC-data from Taas and SIS vehicles, Taas-vehicles may be more skewed towards over-counting boarding passengers and/or under-counting alighting passengers. In order to account for differences in sensor-accuracy, the feature `source_system`, indicating if a vehicle is a SIS- or a Taas-vehicle, was kept when training models.

Looking at a feature-importance plot from the XGBoost models for both case 1 and case 2 we see that source-system is among the features plotted. While not among the features with highest importance (rather among the bottom three), this indicates that measured passenger load may be somewhat related to what source-system the vehicle has. We may assume that no passengers are influenced by the type of vehicle driven, meaning that passenger load in reality should be the same on a given route weather the source-system is SIS or Taas. Even a low feature importance for source-system could indicate that sensor accuracy for both SIS and Taas-vehicle should be further evaluated.

The incorporation of Taas-data represented somewhat of a challenge during this thesis, but it is worth pointing out that this source-system was quite new at the time when the data was collected. It is therefore likely that the data quality from these vehicles have improved since, and that incorporation of Taas-data will be much easier now.

6.1.3 Representing location

Model training was at first performed only using stop-names and not latitudes and longitudes. This required one-hot-encoding in order to be applicable for the regression models. As there were over 100 unique stop-names this method was very memory-intensive as it required one extra column for every stop-name. If more bus lines had been implemented into this model, the number of columns needed to represent bus-stops would have become even larger. Alternatively, separate models would need to be made for every new bus line. When data was onehot-encoded, linear regression yielded a MAE around 9.8 (compared to 11.3 when stop-names where not onhot-ecoded), and similar results were observed for ridge and lasso regression, but because of the larger size of the dataset cross-validation became too time consuming.

In order to reduce the number of columns in the dataset, latitude and longitude were tested out in place of stop-names. While not investigated in detailed in the thesis, this change did not result in poorer performance of the decision tree regressor and XGBoost. On the contrary, model performance seemed to improve. This may be due to the fact that this allows the different bus lines to learn from each other because latitudes and longitudes are universal. This is unlike stop-names which are mostly specific to the individual lines safe for a few common stops.

6.2 Outlier detection

Outlier detection by isolation forest proved useful in detecting trips where APC-equipment was faulty and/or other errors occurred. While the difference in performance for models were never greatly affected by either of the outlier detection methods, data filtered by isolation forest performed slightly better across the board. Especially decision tree regression and XGBoost had fewer extreme values for MAE and RMSE for individual trips when data was filtered by isolation forest.

Looking at plot of trip coverage in Figure 5.7 it is apparent that the isolation forest model removed data around rush hour. This might be because many of the features in the dataset generated for trips-screening were centered around passenger load. Both maximum departing load, and the maximum number of both boarding and alighting passengers were included, which might explain why these trips were singled out.

The main benefit of using a model such as isolation forest for outlier detection is that there is no need to manually identify and correct for all scenarios that can lead to errors in passenger counts. In this thesis we covered a few scenarios that can affect the quality of passenger counts, but in order for manual screening to be comprehensive, many other would likely need to be identified. For all the bus lines examined in this thesis the model identified a reasonable amount outliers, but it is difficult to extrapolate this to other bus lines. Further work is needed to establish whether this model is suited for other bus lines. The choice of isolation forest as outlier detection method was also somewhat arbitrary, mainly motivated by the fact that isolation forest were known to be efficient on larger datasets. There are several other outlier detection methods that may be even better suited, and further exploration might be beneficial.

6.3 Models

We observed little difference between the overall performance of the four linear regression models. Looking at the distribution of MAE for all trips in the test-data, we could see that the regularization in ridge-, lasso- and ElasticNet-regression resulted in fewer extreme values, but there was otherwise little difference. There was also little to no difference between the performance of these to models on the datasets from case 1 and case 2. This may be caused by a high level of correlation among variables in both datasets. If time permitted, more effort could have been put into feature selection and extraction to improve the performance of these models.

For both case 1 and case 2 the best performing models were decision tree regression and XGBoost. Regression forest did not end up being included because training of this model turned out to be too time-consuming. Due to long training times, hyperparameter tuning for XGBoost were also somewhat halted. Focus was mainly centered around tuning the hyperparameter `eta`, and finding the ideal number of boost-rounds based on the best.

Hyperparameter tuning of the decision tree regressor were also limited by long training-times, and focus was put on `min_samples_leaf` indicating the minimum samples needed to form a leaf in the decision tree. Because expanding window cross-validation was used, it was decided to set this parameter using a float-number indicating the fraction of the dataset to be used for this parameter. The alternative would be to set a fixed number, but this option was not tested during cross-validation.

For both decision tree regression and XGBoost there are other hyperparameters for which tuning could have improved performance. We were able to test several regression models in this thesis, but there are still many other that might be able to yield even better results.

6.4 End users

The aim of the thesis was to explore whether data from automatic data collection-systems (ADC) on public transport in Oslo could be used to predict passenger load. The two cases presented have shown adequate predictions can be made when only plan-variables are available, and that performance can further be improved by including real-time data from automatic vehicle location-systems (AVL).

6.4.1 Passengers

The best performing model for case 1 yielded a MAE of 7.10 in departing load. Because the variable used for predictions are plan-variables, they are all available in advance. Because of this, predictions with this level of accuracy could be made available to passengers in the route-planning app. This would allow predicted passenger load to inform the travel-planning of passengers. From Figure 5.9 we see that the decision tree regression-model and XGBoost-model made reasonable prediction for the majority of stops and, unlike some of the linear models, there were also not much difference between the performance among the three bus lines.

While the model trained for case 2 performed slightly better, it is unlikely to be of benefit to passengers because it requires real-time location data in order to make load predictions. At best, predictions of load could be made available at the same time as the bus arrives at a stop. In that case a potential passenger would be better served looking through the windows of an approaching vehicle and making a personal assessment of the crowding level.

Speaking as a frequent passenger of public transport, the availability of passenger load prediction could be useful, but has its limitations. Most passengers know what passenger load to expect on the routes they frequently travel, and may extrapolate their experience to other lines or travel-times. Still, if on desires to avoid departures with high passenger loads, load predictions like these could be beneficial.

For this thesis the choice was made to not use data past February 2020 because the passenger load was greatly affected by the outbreak of the covid-pandemic. Nevertheless, it is apparent that load predictions could be especially useful during a pandemic when passengers are encouraged not to travel during peak hours. Because of reduced number of passengers, resulting in lower passenger load, it is possible that the models could perform even better. Another unexpected side effect of the pandemic could also be that passenger are keeping more distance from each other, resulting in better conditions for the passenger counting sensors.

6.4.2 IOSS

As mentioned in section 1.2.2, IOSS is the department at Ruter responsible for resolving situations affecting the flow of public transport. While the models proposed in this thesis give decent predictions of passenger load, it is difficult to say whether or not the predictions are good enough to be used for monitoring. The best performing model for case 2 had a MAE of 6.26 and RMSE of 8.51, both a fair bit lower than the similar model for case 1 (with a MAE of 7.10 and RMSE of 9.69). In addition to yielding better load-predictions overall, the lower RMSE also indicates that there are fewer cases where the difference between the true load and the predicted load is large. Thus, this model seem reasonably well equipped at accounting for external situation.

As mentioned in Section 1.2.1, the new Taas-system facilitates real-time APC-data in addition to AVL-data. With a larger number of Taas-vehicles continuously reporting their passenger load, the need for the form of model proposed with case 2 diminishes. But there may still be a need for outlier detection to identify vehicles with defect sensors. Correct implementations of load-calculations are also needed in order to report correct data.

6.5 Further work

Automatic data collection-systems are present on all forms of public transport, meaning that similar predictions of load are not only possible for other bus lines, bu also for trams, the underground, and even boats. While the ADC-systems may be similar, there may be other differences that needs to be taken into account.

Based on the experience from prediction load on busses, it seems reasonable that these methods could be applied to trams. While the maximum number of passengers on most trams is larger than that on busses, there are still similarities in their passenger-demand. Trams may even have the added benefit of being more constrained in their routes, meaning that there is less difference in stop-combinations within one line. Predictions of load on the underground may on the other hand be more of a challenge as the vehicles are larger, resulting in larger passenger-volumes and more doors. With more doors the chances of any of their APC-sensors being defect increases.

Bibliography

[1] Climate strategy for oslo towards 2030. [Online]. Available: https://www.klimaoslo.no/wp-content/uploads/sites/88/2020/09/Klimastrategi2030-Kortversjon-ENG_2608_enkeltside.pdf

[2] "About us." [Online]. Available: https://ruter.no/en/about-ruter/about-us/

[3] Årsrapport 2020. [Online]. Available: https://aarsrapport2020.ruter.no/no/

[4] Målbilde for bærekraftig bevegelsesfrihet. [Online]. Available: https://ruter.no/globalassets/dokumenter/ruterrapporter/malbilde-barekraftig-bevegelsesfrihet-2020.pdf

[5] H. Zhai, L. Cui, Y. Nie, X. Xu, and W. Zhang, "A comprehensive comparative analysis of the basic theory of the short term bus passenger flow prediction," *Symmetry*, vol. 10, p. 369, 08 2018.

[6] W. Mcculloch and W. Pitts, "A logical calculus of ideas immanent in nervous activity," *Bulletin of Mathematical Biophysics*, vol. 5, pp. 127–147, 1943.

[7] F. Rosenblatt, "The perceptron: A probabilistic model for information storage and organization in the brain." *Psychological Review*, vol. 65, no. 6, pp. 386–408, 1958. [Online]. Available: http://dx.doi.org/10.1037/h0042519

[8] S. Raschka and V. Mirjalili, *Python Machine Learning: Machine Learning and Deep Learning with Python, scikit-learn, and TensorFlow 2, 3rd Edition*. Packt Publishing, 2019.

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

[10] "1.10. decision trees." [Online]. Available: https://scikit-learn.org/stable/modules/tree.html

[11] C. Wade and K. Glynn, *Hands-On Gradient Boosting with XGBoost and*

Scikit-learn: Perform Accessible Machine Learning and Extreme Gradient Boosting with Python. Packt Publishing, Limited, 2020.

[12] "Introduction to boosted trees." [Online]. Available: https://xgboost.readthedocs.io/en/latest/tutorials/model.html

[13] T. Chen and C. Guestrin, "XGBoost: A scalable tree boosting system," in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '16. New York, NY, USA: ACM, 2016, pp. 785–794. [Online]. Available: http://doi.acm.org/10.1145/2939672.2939785

[14] F. T. Liu, K. M. Ting, and Z. Zhou, "Isolation forest," in *2008 Eighth IEEE International Conference on Data Mining*, 2008, pp. 413–422.

[15] R. Little and D. Rubin, *Statistical Analysis with Missing Data*, ser. Wiley Series in Probability and Statistics. Wiley, 2019.

[16] D. B. Rubin, "Inference and missing data," *Biometrika*, vol. 63, no. 3, pp. 581–592, 12 1976. [Online]. Available: https://doi.org/10.1093/biomet/63.3.581

[17] "Internal documentation for dilax irs-320."

[18] "Internal documentation for init iris matrix."

[19] A. Olivo, G. Maternini, and B. Barabino, "Empirical study on the accuracy and precision of automatic passenger counting in european bus services," *The Open Transportation Journal*, vol. 13, pp. 250–260, 12 2019.

[20] C. R. Harris, K. J. Millman, S. J. van der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith, R. Kern, M. Picus, S. Hoyer, M. H. van Kerkwijk, M. Brett, A. Haldane, J. F. del R'ıo, M. Wiebe, P. Peterson, P. G'erard-Marchant, K. Sheppard, T. Reddy, W. Weckesser, H. Abbasi, C. Gohlke, and T. E. Oliphant, "Array programming with NumPy," *Nature*, vol. 585, no. 7825, pp. 357–362, Sep. 2020. [Online]. Available: https://doi.org/10.1038/s41586-020-2649-2

[21] T. pandas development team, "pandas-dev/pandas: Pandas," Feb. 2020. [Online]. Available: https://doi.org/10.5281/zenodo.3509134

[22] P. Furth, B. Hemily, T. Muller, and J. Strathman, "Using archived avl-apc data to improve transit performance and management," 01 2006.