UNIVERSITY OF BERGEN

Department of Information Science and Media Studies

MASTERS THESIS

Travel time prediction using historical big open data

Author: Marius Falch Lillevik

Supervisor: Andreas Lothe Opdahl

November 30, 2019

"People like us, who believe in physics, know that the distinction between past, present and future is only a stubbornly persistent illusion."

- Albert Einstein

Abstract

The need for travel time estimations and prediction for both transit companies and travelers are increasing. Intelligent transportation systems are often plagued by a shortage of data sources to properly assess the traffic situation. This thesis propose an approach to improve the reliability of travel time predictions through the creation of a combined model that relies on traffic estimations from both buses and cars. We found that the use of multiple sources of traffic data can improve the accuracy and reliability of travel time estimations and prediction where one of the initial datasets suffer from data sparcity.

Acknowledgment

I would like to thank the fellow students at the studyroom for their nice conversations and support through tough times.

I would also like to thank my family and friends for their comfort and moral support towards me throughout the writing of this thesis.

My supervisor Andreas Opdahl has provided great support and has guided me through the project steps and the writing of this thesis.

M.L.

Contents

List of Figures

List of Tables

List of source codes

List of abbreviations

SQL Structured Query Language

CQL Cassandra Query Language

TTP Travel Time Prediction

API Application Programming Interface

GCP Google Cloud Platform

ML Machine Learning

XML Extensible Markup Language

JSON Javascript Object Notation

Chapter 1

Introduction

We live in a world were data are generated from a large number of sources, and it is really easy to collect and store such data. However, the true value of data is not related to the data itself, but with the algorithms that are capable of processing such data in a tolerable elapse of time, and to extract valuable knowledge from it [30]. Finding new and efficient uses for data is one of the main challenges in the technology industry today. With the emergence of smart cities, many different data sources have been made available for a wide variety of applications. The common technique for handling multiple data sources is data fusion, where it improves data output quality or extracts knowledge from the raw data [16]. These sources of data have introduced new ways of thinking about data and have changed the way we work.

1.1 Motivation

Even though the amount of available data today is growing at an increasing rate, much of the relevant data is owned by commercial companies who often have the intention of gaining profit from its data and do not usually give it away for free. It is hard to compete with these data companies in terms of data size and quality. Open data is an alternative to proprietary data that can be used freely without any form of restrictions. The open data are often collected opportunistically, and do not always answer the most

societally important questions, which often forces data users to use proprietary data. The large volume and availability of open data can be used alone or in combination with the relevance and quality of proprietary data to benefit the society through the development of new and innovative solutions.

1.2 Travel time prediction

City-wide travel time prediction in real-time is an important enabler for efficient use of the road network. It can be used in information for travelers to enable more efficient routing of individual vehicles as well as decision support for traffic management applications such as directed information campaigns or incident management [3].

It is impossible to know the future traffic state due to unforeseen circumstances that may occur, such as accidents or other events that slow down traffic and create congestion. However, estimates and predictions of travel time based on previous historical traffic data can be very useful. In this thesis we are investigating how transportation methods such as cars and buses overlap and if we can benefit from combining bus and car data into a unified dataset. It is interesting to find out to what degree we are able to predict future travel times and traffic flow by using machine learning on a combined dataset of bus and car data. We are looking into traffic datasets that are about the same domain but with different different qualities. We are specifically looking into the connection between commercial car travel time estimations available from Google and are combining these with bus travel time data that is openly available from Entur.

The data available from Google is limited in terms of time due to limitations in request frequency, but has unlimited available paths. The bus data from Entur has the opposite problem as it is not limited in terms of time, but has a limited amount of paths. We are therefore proposing an approach for dealing with data sparsity of car travel time data in urban areas where car and bus routes overlap. We want to improve accuracy and reliability of car travel times by training a combined machine learning model on bus and car data using a path based approach. We believe that this can discover the connection between the datasets and improve the accuracy and reliability of car travel

times without being dependant on Google as the only data provider. Making use of both proprietary and open data can get the best of both worlds and improves the quality of data.

1.3 Research questions

We have stated the following research questions to answer throughout this thesis.

1. **RQ 1:** Can combining multiple sources of different traffic data predict accurate travel times?
2. **RQ 2:** Can we improve the accuracy of predictions by combining a low density dataset with a dataset of higher density from the same domain?
3. **RQ 3:** Can we gain useful knowledge from the connection between these different datasets?

1.4 Contributions

With this thesis we want to contribute with a general approach to discovering coherence between similar datasets within the same domain where data sparsity may be a concern. We want to contribute with additional knowledge to not only the transportation domain, but the general field of data mining and machine learning.

1.5 Outline

This thesis has been structured into 7 chapters with the following content.

Chapter 1: Introduction

This chapter presents a general overview of the field of big data in the context the transportation domain as well as the motivation for this research and the research questions to be answered in this thesis.

Chapter 2: Background

This chapter goes into the relevant theoretical topics for this thesis, related work on travel time predictions and the tools used for data processing and model training.

Chapter 3: Methods and workflow

This chapter describes the methods used in this thesis and the workflow used during development.

Chapter 4: Data collection and transformation

This chapter goes into the dataflow from harvesting to pre-processing and transformations. Further, the data is explored through a visual overview.

Chapter 5: Modelling

This chapter describes in details how our models are trained, and we go through the different model training iterations.

Chapter 6: Evaluation

This chapter gives an overview of the results and evaluates the performance of the different machine learning models.

Chapter 7: Conclusions

This chapter discusses the findings, concludes the thesis and proposes suggestions for further work.

Chapter 2

Background

We will now look at some theory behind the field of big data and data mining as well as some related research within the transportation domain and compare some of the known problems to the problems we are facing in this thesis. In addition to that we will look at some of the tools and technologies used throughout the thesis and theory behind machine learning.

2.1 Big data

The term "BIG DATA" has become increasingly popular and is frequently used in both academia and the technology industry in general. However, the definition of the term is often shrouded by many vague concepts. Big data is often referred to in the context of data aggregation, processing and increased value of analysis as well as the increasing impact it has on society today.

When defining big data, "size" is what first comes to mind considering the word "big". However, the true definition is not only determined by the size of the data. The term is often defined using something called the 3 V's which each describe important parts of what are considered characteristics of big data. These where first defined in [19] as Volume, Variety, and Velocity in a form of 3D data using a cube as an example. The three V's have since emerged as a common framework to describe big data. The 3 V's

of big data can be described as below.

- **Volume** - The volume characteristic of big data is concerned with the size of the data; often measured in the form of bytes. How big it has to be to be categorized is often relative to the context. A survey conducted by IBM in mid-2012 revealed that just over half of the 1144 respondents considered datasets over one terabyte to be big data [10].
- **Variety** - Variety refers to the structural heterogeneity in a dataset. Technological advances allow firms to use various types of structured, semi-structured, and unstructured data.
- **Velocity** - Velocity refers to the rate at which data are generated and the speed at which it should be analyzed and acted upon. The proliferation of digital devices such as smartphones and sensors has led to an unprecedented rate of data creation and is driving a growing need for real-time analytics and evidence-based planning [10].

In the context of big data for emergency management we have also seen some new emerging characteristics of big data such as veracity, validity and visualizations [2]. However, these are more relevant in the context of textual data such as twitter messages or news articles.

In the transportation domain, Big Data has the potential to improve the safety and sustainability of transportation systems. Many cities have installed monitoring equipment, such as cameras, roadside sensors, and wireless sensor networks, to observe traffic conditions and promote traffic safety [26].

2.2 Open data

The Open Knowledge Foundation defines open data as the freedom to use, reuse, and redistribute without restrictions beyond a requirement for attribution and share-alike.

Any further restrictions make an item closed knowledge [24]. With the increase of data in various forms, many organization are often limited as to how much they are able to do with it. Therefore, some organization decide to open their data to the public so they can use this data to create new software which in turn can provide new solutions that would likely have not been created otherwise. In addition to software and applications, science is built on data: its collection, analysis, publication, reanalysis, critique, and reuse. Making data publicly available improve the possibilities of science and can allow for new experiments that would not have been possible otherwise.

Limitations to open data

In general there is a cultural reluctance to publish data openly, for a multiple of reasons-from researchers' fear about releasing data "into the wild" where they lack control over its usage to a lack of incentive or credit for doing so [24]. Many businesses today are solely relying on income from selling data or analysis services performed using it and may therefore reject making it public due to a high chance of profit loss as well as privacy laws.

2.3 Travel time prediction

There has been done a series of studies with focus on traffic flow estimation and prediction in the past. However, most of the recent studies have primarily been focusing on travel time estimation and prediction for highway traffic and not as much on suburban main road networks. This section will look into some of the most recent approaches to travel time estimation and prediction.

2.3.1 Adaptive smoothing method

The adaptive smoothing method is a two-dimensional spatio-temporal interpolation algorithm to estimate the speed attribute i.e, a continuous function of the local average

speed $V(t,x)$ in terms of time t and location x. The ASM method was proposed in [38] via the isotropic kernel is now widely used in traffic state estimation problems.

2.3.2 Highway travel time prediction

An approach to travel time prediction conducted in Taiwan [36], developed 2 models to predict freeway travel time through analysis of big data collected from the Taiwan Highway Electronic Toll Collection System. The goal for their system was to provide drivers with accurate travel time predictions in response to real-time traffic data. Their travel time prediction models were established based on historical freeway data: one-destination travel time prediction (OTTP) and adaptive travel time prognosis (ATTP). Their OTTP module was developed for use under normal traffic conditions, meaning, it did not account for unexpected traffic congestion or accidents. This OTTP model would not be accurate enough for real-time travel time predictions as it was only based on historical data, it would not be accommodating dynamic and abrupt changes in freeway traffic conditions. Therefore, in order to enhance the accuracy and adaptability they developed the second adaptive model which would remedy the shortcomings of the OTTP model at a cost of reduced accuracy [36]. Their ATTP model was trained incrementally in real-time and updated the predictions of the OTTP model if there was a significant difference between the predictions of the 2 models.

2.3.3 Urban travel time using heterogeneous data

Most existing studies of travel time prediction have primarily focused on travel time estimation for freeways. Since urban expressways play an important role in alleviating congestion and connecting the road network as backbones. A reaserch focused primarily on estimating and predicting states of urban expressways. Traffic flow estimation and prediction for urban expressways have some challenging aspects that require further investigations [5].

They developed an adaptive rolling smoothing (ARS) approach in an attempt to im-

prove urban expressway traffic state estimation and prediction based on the heterogeneous data. The ARS optimization mechanism was developed to dynamically evaluate algorithms' performance based on historical data in the regression horizon which were based on much of the similar aspects of the adaptive smoothing method used in [38]. Heterogeneous data was used instead of a single data source to reconstruct spatio-temporal speed profiles, based on which the future traffic states can be predicted by their advanced ASR approach. By applying the ASR approach in practice it managed to outperform the global optimization algorithm for estimation and prediction, and both algorithms lead to better results than applying the default non adaptive parameters.

2.3.4 Floating car data

The domain of intelligent transportation systems are plagued by a shortage of data sources that adequately assess traffic situations. Typically, to provide routing and navigation solutions map attributes in the form of static weights as derived from road categories and speed limits used for road networks. With the availability of cheap positioning technology and positional tracking in management applications, vehicle tracking data becomes an important component when creating tools or applications for traffic assessment and prediction [28]. Floating car data (FCD) is location data collected from cars using GPS and motion based technologies and is often referred to as probe data. FCD is a commonly used method of data collection used for travel time prediction [28].

A recent study working with floating car data collected from probes dealt with the problem of data sparsity. They had collected probe data from around 15 taxi cars in urban areas of Schezhen China. Compared to other similar studies basing their data on FCD using probes in cars have often had around 1000 probes collecting data simultaneously such that the traffic speed on a subset of the roads are observed by at least one vehicle within a short time window and does therefore not suffer from the problem of data sparsity. In [20] they proposed a solution to the difficult problem of travel time prediction when having a small amount of concurrently active GPS-floating cars on the road network. Their problem consisted of 2 main challenges where data sparsity is the most

obvious one and the second one is the large variance in travel time observations of the same path. Their solution finds shared pathlets(sub-paths) which are overlapping on important areas where the chance of congestion is high. They then identified the current congestion patters from the relevant pathlets and inferred the travel time of the full paths. This resulted in findings which improved accuracy compared to baseline approach of just using historical data as well as state-of-the art travel time prediction methods that uses both historical data and real-time data[22].

2.4 Languages and frameworks

The main programming language used in this thesis is Python 3. Python was chosen mainly because of its increasing popularity in use within the academic field and because there is a large variety of high quality machine learning frameworks and libraries available for Python. Sci-kit learn is a popular framework used for pre-processing data, model training and generating predictions. Initially sci-kit learn in combination with some other statistical libraries, was the library that we were going to look into due to it's simplicity, ease of use and it's ability to process data fast in memory. However, due to large amounts of data that would not normally fit inside memory on a single regular machine, we decided to use a big data framework called Spark which would handle large quantities of data much better and is able to scale much better. Spark is described further in section 2.4.2.

2.4.1 Python

Python is a programming language that lets you work more quickly and integrate your systems more effectively. Python is developed under an OSI-approved open source license, making it freely usable and distributable, even for commercial use. The Python Package Index (PyPI) hosts thousands of third-party modules for Python. Both Python's standard library and the community-contributed modules allow for many possibilities[29].

2.4.2 Apache spark

Spark was initially created as part of a research project conducted at the UC Berkeley AMPLab in 2009, and was later published open sources in early 2010. Some of the ideas behind the spark system were then presented in various research papers of the years after that, where the original framework was proposed in [40]. After Spark was released, Spark then grew into a much broader developer community, and was moved to the Apache Software Foundation in 2013. Today, the Spark project is further being developed collaboratively by a community of developers from several hundreds of organizations around the world and has a seemingly bright future as it is being used by more and more organizations as times goes on.

Sparks' functionality

Spark is generally a unified analytics engine for large-scale data processing [40] that utilizes in-memory cluster computing to process and analyze big data fast and with the possibility of using it interactively. Although many of existing similar frameworks have numerous abstractions for solving cluster computational problems, they lacked abstractions for leveraging distributed memory [36]. One of the main issues when dealing with cluster computing is to handle the distribution of data across multiple machines in a fault tolerant way. Spark solves this by creating a data structure called RDD (Resilient Distributed Dataset) which is a fault-tolerant abstraction for in-memory cluster computing.

In a formal context, and RDD is a read-only, partitioned collection of records of an unknown type. RDDs can only be created through deterministic operations on either data in stable storage or other RDDs. These operations are called transformations in order to differentiate them from other operations on RDDs. Some examples of transformations include map, filter and join; where an RDD is transformed and a new RDD is returned with different values. The RDDs does not need to be materialized at all times. Instead, an RDD has enough information about how it was derived from other datasets (its lineage) to compute it's partitions from the data in stable storage. This is

a very powerful property; in essence, a program cannot reference an RDD that it cannot reconstruct after a failure. Users have the options of controlling aspects of RDDs, namely the persistence and partitioning. Users have the ability to indicate which RDDs they want to reuse and choose a storage strategy for them (e.g., in-memory storage). The user can also ask that an RDD's elements is partitioned across machines based on a key in each record. This is useful for placement optimizations, such as ensuring that two datasets that will be joined together are hash-partitioned in the same way [40].

2.4.3 The Spark MLlib pipeline

The Spark MLlib library standardizes APIs for machine learning algorithms to make it easier to combine multiple algorithms into a single pipeline, or workflow. This section covers the key concepts introduced by the Pipelines API, where the pipeline concept is mostly inspired by the scikit-learn project [27].

1. **Dataframe**: This ML API uses DataFrame from Spark SQL as an ML dataset, which can hold a variety of data types. E.g., a DataFrame could have different columns storing text, feature vectors, true labels, and predictions. Dataframes have become a common general purpose use in Spark and are used as an abstraction layer on top of the original RDD datastructure.

2. **Transformer**: A Transformer is an algorithm which can transform one DataFrame into another DataFrame. E.g., an ML model is a Transformer which transforms a DataFrame with features into a DataFrame with predictions.

3. **Estimator**: An Estimator is an algorithm which can be fit on a DataFrame to produce a Transformer. E.g., a learning algorithm is an Estimator which trains on a DataFrame and produces a model.

4. **Pipeline**: A Pipeline chains multiple Transformers and Estimators together to specify an ML workflow. These are executed in order to secure a better structure and re-usability with different algorithms.

2.5 Machine learning

Machine learning is an actively evolving branch of computational algorithms that are designed to be able to emulate the capabilities of human intelligence by learning from the surrounding environment. Machine learning algorithms are considered the main drive in the new era of the so-called big data where we have to deal with problems such as data deluge. When dealing with data deluge, it calls for new automated methods of data analysis, which is what machine learning provides.

Machine learning can be defined as a set of methods that can automatically detect patterns in data, and use the uncovered patterns to predict future data, or to perform other kinds of decision making under uncertainty [11, 31]. Different machine learning techniques have been applied in diverse fields of ranging from pattern recognition, computer vision, spacecraft engineering, finance, entertainment and computational biology to biomedical and medical applications [25]. This chapter will look into some of the different methods of machine learning and how they can be used in practice.

2.5.1 Problem types

We can usually divide machine learning into two main types. These are supervised and unsupervised machine learning. There exits other problem types called semi-supervised learning and reinforcement learning, but these are not as common as supervised and unsupervised learning and covers different problem areas.

2.5.2 Supervised learning

Supervised learning can be considered a predictive learning approach and is the form of machine learning that is most widely used in practice. For supervised learning the goal is to learn a mapping from inputs x to outputs y, given a labeled set of inputs-output pairs $D = \{x_i, y_i\} \sum_{i=1}^{N}$. In this example D is called the training set, and N is the number of entries used for training.

The input data used for predictive behavior is denoted as x_i in the example and is typically represented as vectors which are often referred to as features or attributes. The output variable y_i, which is frequently called the label, is categorical when we are dealing with a classification problem also known as pattern recognition, and when y_i is continuous, we have what is considered a regression problem[31].

Traditionally people refer to regression, classification and structured output problems as supervised learning. Density estimation in support of other tasks is usually considered unsupervised learning [11].

Classification

The goal when solving a classification problem is to take a mapping containing inputs X and convert these to output Y, where $Y \in 1, ..., C$ where C is denoted as the numbers of distinct classes or types to predict. Classification problems can generally be split further into three categories called binary, multi-class and multi-label classification. We consider a classification problem binary if it has only 2 classes(yes or no) and a multi-class problem when we have more than two classes. If we are trying to classify multiple labels we have what is called multi-label classification and expect multiple outputs predicting different properties of the same object. Classification is typically used in cases such as text categorizing, image recognition and medical diagnosis prediction [31].

Regression

A regression problem is essentially the same as a classification problem with the exception of it's label or output variable. Regression problems are often used when estimating some value over time or in the future. Some examples of real-world regression problems are predicting short term stock prices, predicting age of a viewer on YouTube®, predicting temperature at a location based on multiple factors such as weather data, time, door sensors, etc [31], and predicting travel time based on spatial and temporal aspects.

Unsupervised learning

The second main type of machine learning is the descriptive or unsupervised learning approach. Unsupervised learning is only given inputs as such $D = \{(x_i)\}\sum_{i=1}^{N}$, and has a goal of finding interesting patterns in the data as opposed to making predictions. This may be useful when looking for new ways in which we can use data for and may in some cases be referred to as knowledge discovery [31].

2.5.3 Model generalization

The main objective when training a machine learning model is to be able to use it to generate accurate predictions when being introduced to new data that has not been previously used to train the model. Being able to adapt to new data is often referred to as generalization. A common way to evaluate or measure how well a model performs is to use a hold out test or validation set and make predictions on these and compare the predictions with the actual observed values in order to measure the error rate of the model. In order to achieve generalization capabilities in a model, it is necessary for the training error to be as small as possible, and that the gap between training and test error should be narrow. There are two central challenges that we have to deal with in machine learning in the context of generalization which are the issues of over and under fitting the model [11].

Challenges of over- and underfitting

If we have found a predictor whose performance on the training set is excellent, yet its performance on the true "world" is very poor. This phenomenon is called overfitting. Intuitively, overfitting occurs when a model fits the training data "too well" [33]. Underfitting occurs when the gap between the test data and the model is too wide which results in a high error rate.

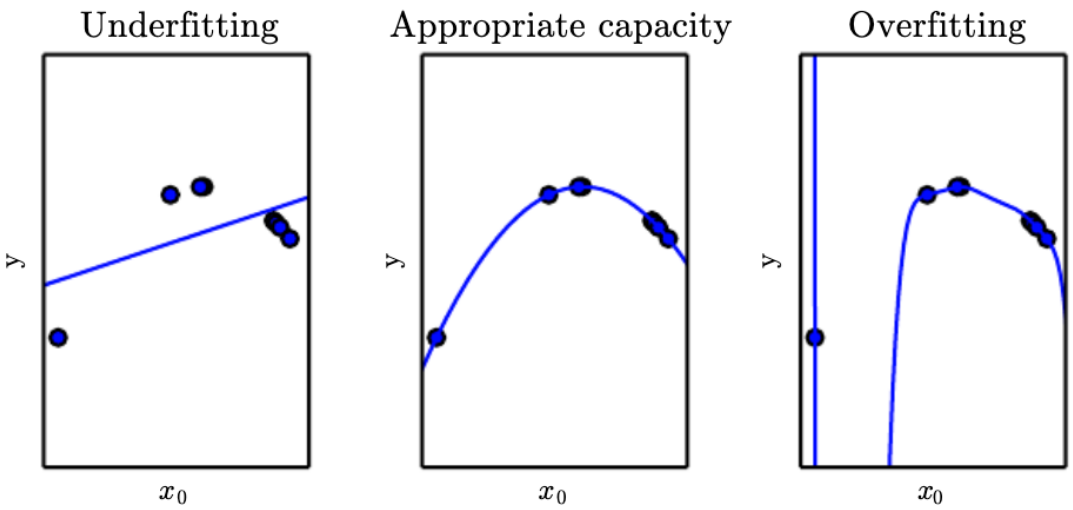


Figure 2.1: Visualisation of over and under-fitting [11]

.

Over and underfitting essentially means that the model either performs too well on the training data or does not perform well at all. The model is therefore not well generalized.

2.5.4 Cross validation

Often we use about 80% of the data for the training set, and 20% for the validation set. If the number of training cases is small, this technique runs into problems, because the model won't have enough data to train on, and we won't have enough data to make a reliable estimate of the future performance. A simple but popular solution to this is to use cross validation. The idea is simple: we split the training data into K folds; then, for each fold $k \in 1, ..., K$, we train on all the folds but the k'th, in a round-robin fashion. We then compute the error averaged over all the folds, and use this as a proxy for the test error. (Note that each point gets predicted only once, although it will be used for training $K-1$ times.) It is common to use K = 5; this is called 5-fold CV [31].

We have used cross validation to train our models for hyper parameter tuning. This selects the best parameters we can use in order to achieve best performance for our model.

2.5.5 Model performance evaluation

When evaluation the performance of machine learning models we need to compare the predictions with the test data and see how close we have managed to get to actual observed data. The test data is not included in the training of the models and is therefore unbiased. By using the approach for testing our model we are able to see how well it adapts to new data.

When evaluating regression performance, one of the most commonly used methods is the Root Mean Square Error or Mean Square Error. If a vector of N predictions generated from a sample of n data points on all variables, and Y is the vector of observed values of the variable being predicted, with $\hat{Y}_i$ with Y_i being the predicted values, then the within-sample MSE of the predictor is computed as below.

$$RMSE = \sqrt{\frac{\sum_{i=0}^{N-1}(\mathbf{y}_i - \hat{\mathbf{y}}_i)^2}{N}} \qquad MSE = \frac{\sum_{i=0}^{N-1}(\mathbf{y}_i - \hat{\mathbf{y}}_i)^2}{N}$$

MSE is the average of the squared error that is used as the loss function for least squares regression while RMSE is the square root of MSE. Both the root mean square error (RMSE) and the mean absolute error (MAE) are regularly employed in model evaluation studies [4]. RMSE is not ambiguous in its meaning, and it is more appropriate to use than the MAE when model errors follow a normal distribution [4]. In addition, RMSE satisfies the triangle inequality required for a distance function metric. Giving higher weighting to the unfavorable conditions, the RMSE usually is better at revealing model performance differences. A combination of metrics, including but certainly not limited to RMSEs and MAEs, are often required to assess model performance[4].

In addition to RMSE and MSEs, we have calculated the R2 rating which estimates coefficient of determination using the following formula.

$$R^2 = 1 - \frac{MSE}{\mathrm{VAR}(\mathbf{y}) \cdot (N-1)} = 1 - \frac{\sum_{i=0}^{N-1}(\mathbf{y}_i - \hat{\mathbf{y}}_i)^2}{\sum_{i=0}^{N-1}(\mathbf{y}_i - \bar{\mathbf{y}})^2}$$

In our case the r2 rating gives us the same results as calculating NRMSE (Normalised

RMSE).

2.5.6 Travel time prediction

Predicting travel time can be very useful for both travelers and transit companies in order to prevent spending too much time in traffic congestion and it allows better planning for routes so it can be avoided. In a machine learning context, the value of the travel time is usually measured in seconds and is considered a continuous output variable. This means that we are working with a regression problem as previously stated in section 2.5.2. Most current distributed systems that rely on travel time estimation and prediction have shown best results using models or ensembles where Random Forest has given the best results for long-term travel time prediction[23]. Some of the most recent research have shown great results using multi-layer perceptron algorithms, also referred to as neural networks, or deep learning depending on the amount of layers [21, 39]. Throughout this thesis we are looking at linear regression and ensemble/tree methods for regression analysis, which will be described in more detail in chapter 5 where we show how our models are implemented using the Spark ML pipeline.

2.5.7 Random Forests ensemble

Random Forest is a tree-based ensemble with each tree depending on a collection of random variables. Random Forests can be used for either a categorical response variable, normally referred to as "classification," or a continuous response, referred to as "regression". Similarly, the predictor variables can be either categorical or continuous. From a computational standpoint, Random Forests are appealing because they are relatively fast to train and to predict, they depend only on one or two tuning parameters, they have a built in estimate of generalization error, can be used directly for high-dimensional problems and they can easily be implemented in parallel [6].

Often, trees are deliberately grown larger than necessary and “pruned” back to prevent overfitting [6].

2.6 Summary

We have through this chapter seen some of the methods that have been used for travel time estimation and prediction. The current methods used which are considered state-of-the art for travel time prediction are based on the use of a combination of both historical data and real-time prediction.

Similarly to [20], we are proposing an approach for dealing with data sparsity of car travel time data in urban areas. We want to attempt to improve predictions on a small dataset of car data by combining it with a larger dataset of bus data with overlapping paths that we believe can discover the coherence between the datasets and improve the accuracy. We will now look at the methods used to complete this research.

Chapter 3

Method and workflow

This chapter describes the different methods used to complete this thesis and the steps taken to ensure a proper execution of the research, as well as an overview of the workflow throughout the project.

3.1 Design Science Research

Most of the research conducted today in academia are not applied or is unknown by the professional practitioners, either in businesses or large organizations. Scientist have reported that their work are rarely applied in practice. However, in order for research to be respected scientifically and reliable, it is important that it does not only concern relevance, but also rigor; it should be presented early on, throughout the evaluation and the end results. Using research methods allows for better rigor when doing investigative work. It is also important that the choice of research methods is aligned with the nature of the problem in which one wants to study [8].

In order to make the gap between theory of studies aimed at organizations smaller, it is possible to apply design science or design science research to produce useful knowledge and conduct studies within the area [32].

3.1.1 What is design science research?

Design Science Research (DSR), which is also known as Constructive Research, is considered to be a method that considers devising artifacts that serve human purposes [8]. It is a form of knowledge production that is about development of new and innovative constructions that intends to solve real world problems while simultaneously making a prescriptive scientific contribution to the common knowledge base. It is important that the outcome of such a research is an artifact developed with an intention of solving a domain specific problem, also known as solution concept, which must be assessed against criteria of value or utility. Design Science Research has got a significant increase in its interest in fields such as information systems, business management, and management accounting; this due to lack of practical relevance and scientific knowledge being produced [8].

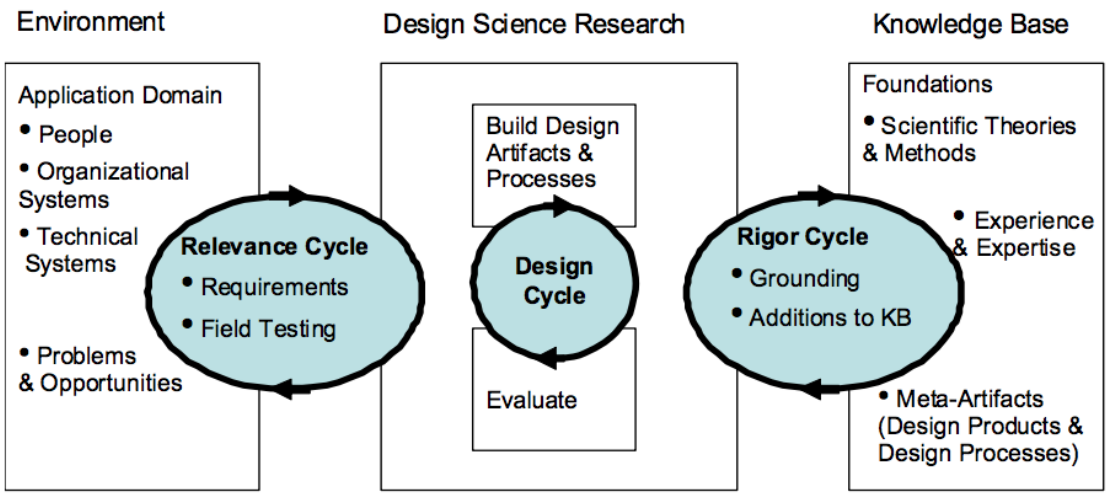


Figure 3.1: Design Science Research Cycles[14]

There exists quite a few models of Design Science Research. One that is commonly used is the three cycles view of design science research. Hevner separates design science research into three separate rotations. These cycles consists of the relevance cycle, the rigor cycle, and the design cycle. Following this model should ensure good quality work with good rigor.

The Relevance Cycle

Design science research concerns the creation of new and innovative artifacts and the processes which we use to develop these artifacts and achieve great results. The relevance cycle is the first part of the design science research cycles and good design science research often begins here by establishing the domain in focus and connecting it to a real-world organization and explain how the artifact can improve the current situations by adding improvements; either to existing functionality or by creating something completely new [14]. It is also important to mention the people involved in the project, their organizational systems and the technical systems. Discussing some possible problems can make it clearer as to what may need some work and then what possible opportunities that are available to handle the problems in the best way possible.

The output after applying design science research must be returned back into the environment in order to be studied and evaluated within the application domain in focus.

The Rigor Cycle

The knowledge base that design science is built on consists of theories and engineering methods that function as a base for rigorous design science research. In addition to that, the knowledge base also contains two different types of additional knowledge [14]:

1. Experiences and expertise that defines "state of the art" in the application domain of the research.
2. The already existing artifact and processes that can be found in the application domain.

Therefore, by applying the previous knowledge in the development of new artifacts; we can achieve good rigor while also making sure that we are innovating and not creating

something that already exists. The new knowledge that the new artifact creation creates can then naturally be added to the knowledge base and provide further knowledge for other artifact creations.

Not everyone agrees with this. It is argued in [14] whether a design theory should be an essential part of the design science rigor or not. He states that some scientific articles have been rejected because of the lack of a scientific theory and believes that it is unrealistic that all design research should be grounded on design theories and that it could even be harmful to the field when good research is denied because of lack of a theoretic method. It would be much better to base the creative design, but rather use the theories as the basis of creative ideas[14].

The Design Cycle

The design cycles is the last of the three cycles, and also the most work intensive. The design phase is where everything is implemented in practice. The cycles iterates over the requirements and theories from the relevance cycles, while using the theories chosen from the rigor cycle. Different artifact design alternatives are designed and evaluated and feedback is generated to refine the design further. These cycles continue until a satisfactory result has been achieved, and not until then can one evaluate the resulting artifact [14].

3.1.2 Applying DSR

Design Science Research will be used to make sure that it follows the necessary steps to achieve good rigor. The relevance cycle for this project will consider big data domain with focus on public transit data and travel time predictions.

Through the rigor phase we will be choosing a theoretical approach suited for the project and use this together with the requirements when designing the artifacts in the design cycle. The method chosen specifically for this project is KDD method, which is based on an original model created in 1996 [9] , due to a great increase in the amounts

of data collected. The KDD method is further described in the next section, and also how it will be integrated into this project.

As mentioned, the design cycle is where the majority of the time will be spent. Through this phase we will be developing the artifacts. These will primarily consist of a model which takes in pre-processed data from the database we have stored data in. The development of the models is executed through as many iterations as needed in order to achieve the best results. When the final artifact design has been completed, it can be evaluated based on the error rates we receive from the models. The evaluation of our models throughout the different iterations are described and evaluation in chapter 6.

3.2 Knowledge Discovery in Databases

Over a wide variety of different fields, large amounts of data are being collected and accumulated at an increasingly fast pace. It is therefore a need for a new generation of computational theories and tools to assist human beings in extracting the useful information, namely the knowledge, from these growing volumes of data stored in databases. These theories and tools are the subject of the emerging field of knowledge discovery in databases (KDD) [9].

If we look at it from an abstract level, the KDD field concerns the development of methods and techniques for making sense of the available data. The basic problem considered by the KDD process is low-level data (that can sometimes be too large to understand and get through easily) into other forms that might be compact or manageable (for example, a short), more abstract (for example, a descriptive approximation or model of the process that generated the data), or even more useful (for example, a predictive model for estimating the value of future events or cases). The core of the process is the application of specific data-mining methods used for pattern discovery and knowledge extraction [9].

There is an urgent need for a new generation of computational theories and tools to assist humans in extracting useful information (knowledge) from the rapidly growing

volumes of digital data - Fayyad, Piatetsky-Shapiro, and Smyth

The aforementioned quote is from the old article that created the first KDD theory which has a significant amount of citations through the last few decades. Although the term KDD might not be used frequently today, it still has a significance when it comes to explaining the field of data mining. Most of the concepts are still valid, although we might have some more knowledge within certain areas. In figure 3.2 shown below, we can see each step the KDD model goes through from start to end and what the different processes produce.

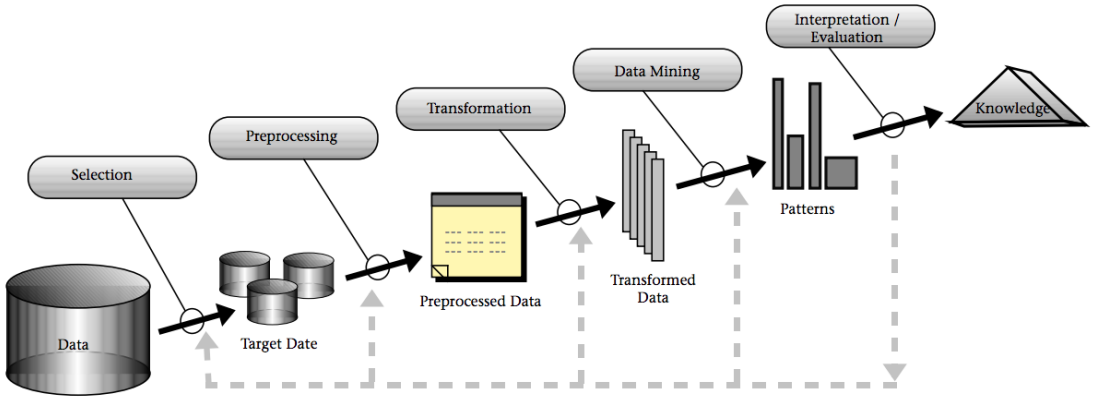


Figure 3.2: An overview of the steps that compose the KDD process [9]

The model in figure 3.2 was created a long time ago, however, the steps that it goes through are mostly the same today and the abstract level of the model makes it possible to use it for this project as well. How we will proceed through the different steps of this model is described in section 3.3 and further discussed in section 7.2 for of the thesis.

3.3 Work flow

The work flow defines a sequence of logical steps that a study will go through in order to reach the goals a researcher has created for the study. It is therefore important that the work method has a rigid structure and is followed accordingly in order to make

sure of the study's reputability [8]. Now, considering that this study will use the KDD research method; these steps are, as displayed in figure 3.2:

1. Domain orientation - Throughout this phase, we want to develop an understanding of the application domain and figure out what data is available to work with. As for this study we are going too look into the transportation domain, with a majority of data coming from public transport as well as a smaller dataset of car travel times.

2. Data selection - Through this phase we will be selecting a dataset to work with, or a subset of values which we are interested in doing discoveries. The specific data set that we are going to analyze the bus and car data as described in chapter 4.

3. Pre-processing - This phase handles cleaning and pre-processing the data. This includes basic operations like removing unnecessary noise of appropriate, collecting the necessary information in order to model or account for noise, deciding on strategies for handling missing data fields, and accounting for time-sequel.

4. Transformation - Here we want to find useful features to represent the data depending on the goal of the task we want to achieve. Using dimensional reduction or transformation methods, the number of variables considered can be reduced or a constant representations for the data can be found.

5. Choosing approach - The fifth step of the method is where goals of the KDD process (step 1) are matched against a particular data-mining method. Some examples can be summarizing, classification regression, clustering and more. We are using machine learning and regressing analysis as our approach for data mining.

6. Model selection - Here we explain how we are going to choose the different machine learning algorithms suited for the task and find out which parameters might be fitting and match this with the overall criteria the KDD process. For example the end user might be more interested in understanding how the model looks rather than its underlying predictive capabilities. We are testing several different ML algorithms too see which ones gives us the best results.

7. Model training - The seventh step is to run the ML algorithms and generate results

for us to compare and analyze. These results should contain patterns that we want to analyze in step 8. This process searches for patterns of interest in a particular representational form or a set of such representations, including classification rules or trees, regression, and clustering.

8. Analysis - This step uses the results gathered from step 7 and manually analyze the results in order to detect patterns. Patters found can be used to run new iterations using the previous steps until we have achieved some substantial new knowledge of the data. This is probably the most important and time consuming part of the KDD process.

9. Distribute knowledge - Once all of the preceding steps have been completed and we have gained new knowledge from the data, we have too choose what we want to do with the data. It can for instance be used directly, implementing it into already existing systems or just document it into a knowledge base where it can stay or be further explored at a later time.

Overall the process mentioned in the 9 steps above from the Knowledge Discovery in Databases [9] are almost the same as how the model was originally proposed. However, instead of using data mining which can be quite general, we have have chosen to use supervised machine learning algorithms instead. Although, one might argue that machine learning is a subset of data mining.

3.4 Software development methodology

Having a good workflow is important when working on a sizable and challenging project. In order to maintain control while at the same time completing the different steps in iterations, software development methodologies can help achieve this. This is a solo project, and there is therefore not a need for task distribution among team members. The methodology that seems suitable for this project is the Kanban methodology.

The word Kanban comes from the japanese language and has the meaning "card or signboard, a verbal instruction, a light, a flag or even a hand signal and is based on

a pull system. In the 'state of agile', the use of Kanban has increased from 31% - 39% just within 2015 and 39% - 50% in 2016 [1]. That is a significant increase and makes it clear that the method has become increasingly popular. Software engineers have been plagued by several problems like (1) lack of reliability, (2) poor response to change(lack of flexibility), (3) limited agility and (4) excessive costs. The Kanban method was created with the intention to solve these problems, by allowing developers to adapt quickly to change increase quality, reduce waste and improve predictability[1].

Kanban is defined by as a set of concepts, principles, practices, techniques, and tools for managing the product development process with an emphasis on the continual delivery of value to customers, while promoting ongoing learning and continuous improvements. We can define Kanban in software process as a pull system with WIP limits and visualized by the Kanban board. Kanban is used as a workflow management method especially suitable for managing continuous software engineering work[1].

The main purpose for Kanban is that is its ability to visualize the workflow through the use of a Kanban board. The Kanban board consists of a wall full of notes stating ideas, processes, what has been done and what is being done as well as a backlog of tasks that has already been finished. The board can be physical, but does not have to be. Trello[37], is a great way of easily managing the current workflow with minimum effort. Trello is an online and real-time planning board which allows for multiple editors simultanously and makes it a great tool for teams of any sizes. Trello is used as a Kanban board during the development of this project.

Chapter 4

Data collection and transformation

This chapter gives an overview the different datasets that have been collected for use in this thesis. We elaborate on where the data comes from, how and why it has been used as data sources for this project. Example of contents of each dataset is described in further detail with examples of some data from each set. We will also go through the steps of pre-processing and transformation of the datasets.

4.1 Available open datasets

We will now be going through step 2 of the workflow which concerns the selection of specific data sources and time period.

After spending some time searching for interesting open datasets online that could prove useful, we found a Norwegian organization called Entur [7]. Entur offers multiple open APIs that are free of charge for anyone to use. The data that we considered to be potentially useful as historical data was Enturs real-time timetable data. The real-time timetable was being served through a distributed system called SIRI(Service Interface for Real-time Information). For this thesis we have selected data from a time period ranging from 12. August 2019 until 12. November 2019.

4.1.1 Entur real-time data

The Entur organization started creating a centralised system for all of the collective transportation methods available in Norway. This made it possible for us to collect this data free of charge without any limits. This project has been focusing on data from around the Bergen area and therefore chose to collect all of the data from Skyss [35], which is the company responsible for most of the buses and railways available in and around Bergen. The SIRI system that Entur use for serving real-time data, have 3 api endpoints which the users can send requests data from.

1. Estimated timetables (ET)

   The estimated timetables is the main source of data and contains data about the current traffic situation through journey estimations

2. Situation exchange (SX)

   The situation exchange contains data about any know traffic situations that may interfere with estimated travel times

3. Vehicle monitoring (VM)

   The vehicle monitoring gives us the current positions of vehicles. This however was unavailable for Skyss during the writing of this project.

The real-time data from the SIRI system does not give us too much information on its own and needs to be connected with static data in order to get a full overview of which routes go where, their names and so on.

General Transit Feed Standard

GTFS is a transit standard created by Google, to represent an overview of all the different stops, timetables, routes, lines and services available for an organization offering transport services, like Skyss. Google's intention when creating the General Transit Feed Specification was to collect as many transit companies as possible to one data format, in order to make it easier to get an overview of all the different data sources(GTFS

feeds) without having to handle many different formats for each company. A general format also allows for easier processing and querying of larger quantities and sets of data. Using the GTFS system for creation of new services allows for great flexibility as it can easily be reused for other transport organizations using the same standard. Entur is delivering their data through both GTFS and Netex formats. GTFS is split into a static component that contains schedule, fare, and geographic transit information and a real-time component that contains arrival predictions, vehicle positions and service advisories[12].

Estimated Time tables

Entur offers a variety of different datasets from most of the public transit companies around Norway. The main dataset which we found interesting and possibly useful for historical time series analysis was the Estimated Timetable(ET) real-time data. We found it useful as it would give use a constant data flow of estimated travel time data from busses with little to no temporal limitations.

The ET data is a part of a CEN (Comité Européen de Normalisation) standard system for real-time information called SIRI (Service Interface for Realtime Information and is used several other places around Europe, with Germany being one of the initial users of the standard system [34]. The ET real-time data is both served through a frequent get-request based manner or by subscribing to updates by giving a callback URL where the data is sent to whenever it is updated. For this project we used a simple request based approach and collected data every minute, using cron, to get all of the updated changes from minute to minute and store these on disk.

The ET consists of a list of journeys with information about its current state at the time it was harvested. Each journey contains a list of estimated calls which describes each of the stops using information about when it is scheduled to be at each stop and when it is expected. The expected arrival or departure times are estimations based on based the current bus position and speed, and therefore varies depending on traffic.

The SIRI system also has Vehicle Monitoring or VM, which gives deeper insights into

```
<Siri xmlns="http://www.siri.org.uk/siri" version="2.0">
  <ServiceDelivery>
    <EstimatedTimetableDelivery version="2.0">
      <EstimatedJourneyVersionFrame>
        <EstimatedVehicleJourney>
          <LineRef>SKY:Line:1</LineRef>
          <DirectionRef>10</DirectionRef>
          <DatedVehicleJourneyRef>6647090</DatedVehicleJourneyRef>
          <PublishedLineName>Linje 1</PublishedLineName>
          <DirectionName>Utgående</DirectionName>
          <DestinationRef>NSR:Quay:51855</DestinationRef>
          <DestinationName>Flesland</DestinationName>
          <Monitored>true</Monitored>
          <PredictionInaccurate>false</PredictionInaccurate>
          <EstimatedCalls>
            <EstimatedCall>
              <StopPointRef>NSR:Quay:51867</StopPointRef>
              <StopPointName>Birklandskiftet</StopPointName>
              <Cancellation>false</Cancellation>
              <AimedArrivalTime>2018-04-30T00:10:09+02:00</AimedArrivalTime>
              <ExpectedArrivalTime>2018-04-30T00:09:27+02:00</ExpectedArrivalTime>
              <AimedDepartureTime>2018-04-30T00:10:29+02:00</AimedDepartureTime>
              <ExpectedDepartureTime>2018-04-30T00:09:47+02:00</ExpectedDepartureTime>
              <DeparturePlatformName>Birkelandskiftet</DeparturePlatformName>
            </EstimatedCall>
            ...
          </EstimatedCalls>
        </EstimatedVehicleJourney>
        ...
      </EstimatedJourneyVersionFrame>
    </EstimatedTimetableDelivery>
  </ServiceDelivery>
</Siri>
```

Listing 4.1: An example response from SIRI Estimated Timetables.

the positions of the vehicles, but this is not currently public for Skyss through Entur. The SIRI CEN standard serves data using XML and their main format. Listing 4.1 shows a short XML example of how the raw ET data looks when it is initially collected. A typical response consists of around 20 journeys which each on their own has around 20 - 40 estimated calls(stops) that is contained within the journey object every time. Since this is collected every minute it quickly builds up to become quite large quantities of data and can be a challenge to handle without the proper tools to do the job.

4.1.2 Google distance matrix

Google offers a large variety of API services through their Google Cloud Platform(GCP). Some of the data they offer to the public are APIs such as maps, routing, places, directions and distances. The distance matrix API gives us an estimated travel time for a path based on an origin address or coordinates `A`, to another destination address `B`. The information we get from Google's distance matrix API gives us an estimate of the distance in meters, travel time, travel time in the current traffic and more about each path. As previously mentioned, we have chosen to focus on data around Bergen, and have targeted a specific area from the senter of Bergen going north towards Sandviken and Åsane.

In order to get a higher spread of different paths and locations throughout the day, we selected a set of different paths starting from the center of Bergen and going out towards the northern part of Bergen. This was mainly because this part is quite busy when it comes to bus traffic and is not affected by the railway (Bybanen). By affected we mean that all travelers who travel in this direction will currently either have to travel by bus or by driving. The numbers in figure 4.1 each represents a point which distances have been estimated for, going in directions in and out of Bergen. All of the distances are calculated from Point 1 on and to the other points on the map so that they all are somewhat connected to the same main road where most if not all cars are driving.

The data from the Google distance matrix API seem to work well in combination with the data collected from entur.org due to its similarity. However, google no longer offers data for free and have started with much stricter rate limiting than what they previously had available. Google still gives every registered user a fixed number(200$ as of September 2019) of credits every month that they can use however needed and will have to pay for each request thereon after, which can we quite expensive in the long run. We have still been able to collect a significant amount of data over time that is enough to get a decent representation of each path.

Due to the rate limitation for the GCP, we had to limit the number of requests sent daily. In order to do this and still get some relevant results we selected certain periods from

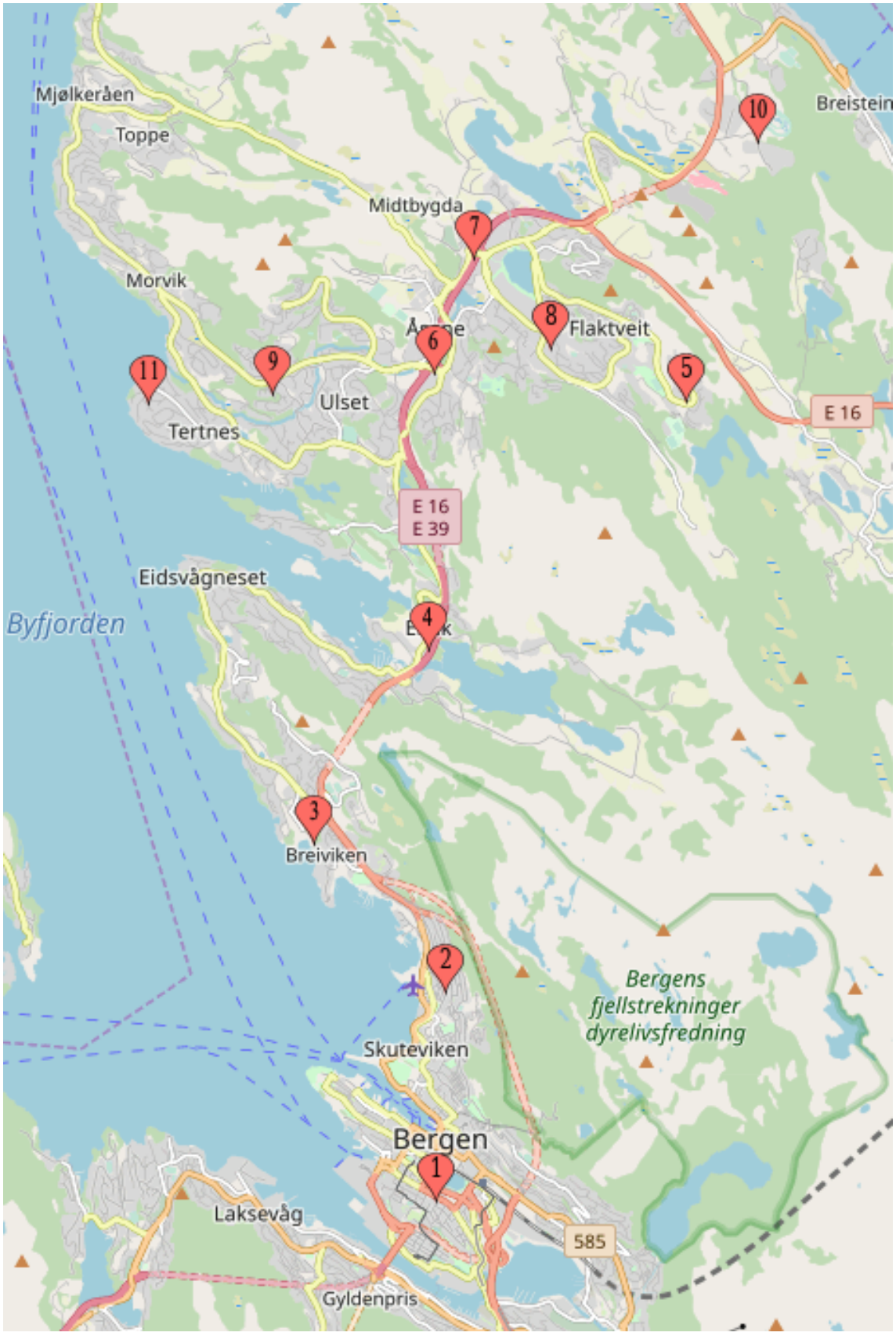


Figure 4.1: The selected distances harvested from the google distance matrix api

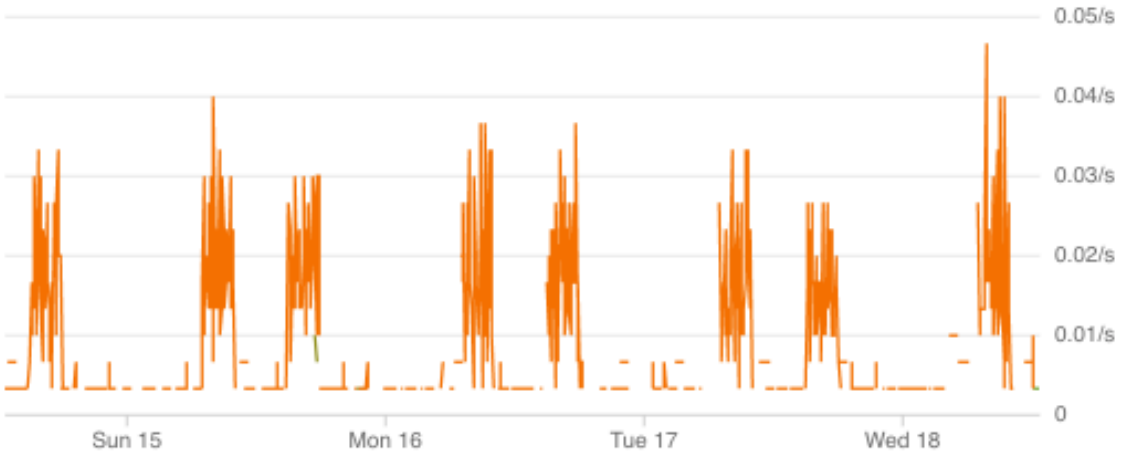


Figure 4.2: Google distance matrix request frequency

each day and had a much higher request frequency from these periods than for the rest of the day. As can be seen in figure 4.2, the main periods the data were harvested from was during the rush-hour traffic when most people are going to and from work, which is between 6-10 each morning and 15-19 in the afternoon. The times for when this was harvested was selected randomly for each day with the most requests sent during rush hour and the rest spread around the rest of the day. This way we are able to get a better spread of travel times from different times on the day which gives us a more accurate result when used for training to a much higher variance. Having some degree of variance is an important factor when training a machine learning model so its ability create a more generalized model is higher.

As we can see in code example 4.2, the google distance matrix API returns a JSON containing information about the path requested. The API can be used to request multiple paths by adding more origins and destinations, but in our case we decided to separate all of the requests into single paths in order to get a better spread of harvest times.

The response we get from the Google distance matrix api contains two different values for travel time. The first value called `duration` is an estimate of the travel time based on historical averages alone and does not account for any current traffic or anomalies. The second value called `duration_in_traffic` is an estimate based on both historical data and information about the current traffic. This means that the second estimate has a much higher variance and is likely be more coherent with the bus data we have

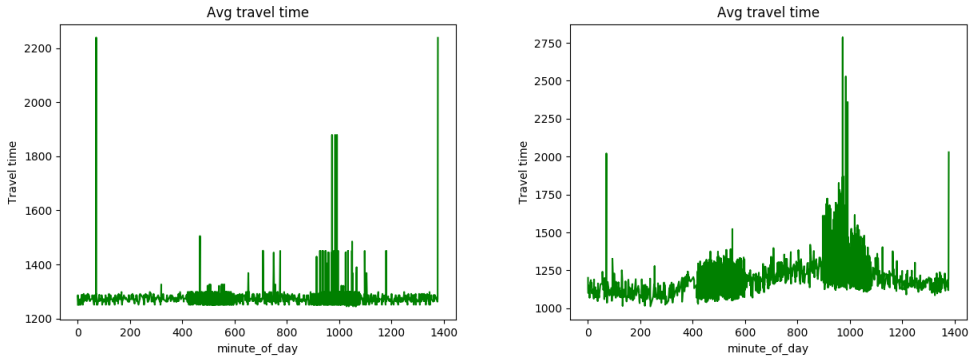
```
{
   "destination_addresses":[
      "Åsamyrane 250, 5131 Bergen, Norge"
   ],
   "origin_addresses":[
      "Christies gate, 5016 Bergen, Norge"
   ],
   "rows":[
      {
         "elements":[
            {
               "distance":{
                  "text":"11,5 km",
                  "value":11452
               },
               "duration":{
                  "text":"17 min",
                  "value":1006
               },
               "duration_in_traffic":{
                  "text":"14 min",
                  "value":850
               },
               "status":"OK"
            }
         ]
      }
   ],
   "status":"OK"
}
```


Listing 4.2: Example response from the Google distance matrix API.

collected.

The distance matrix API have a few required parameters that needs to added to each request. The only required parameters are the origins and destinations, however, in order to receive information about `duration_in_traffic` we also need to apply the travel mode and departure time for the travel distance. The only travel mode that includes traffic information is the driving mode.

Another parameter `traffic_model`, specifies the assumptions to use when calculating time in traffic. This setting affects the value returned in the `duration_in_traffic` field in the response. The *traffic_model* parameter may only be specified for requests where the travel mode is driving, and where the request includes a `departure_time` [13]. We included this for every request.



(a) `duration` value (b) `duration_in_traffic` value

Figure 4.3: An illustration of the visual difference between the different values returned from the Google distance matrix API

If we take a look at some historical Google data from august 12 2019 and 2 months ahead we can see that there is a much higher variance when taking in the other factor of traffic situation. Although this may improve the accuracy of predictions it may also be harder to use for further predictions due to lack of variance. The `duration_in_traffic` value should be the best estimate of travel time given what is known about both historical traffic conditions and live traffic. Live traffic becomes more important the closer the departure_time is to now [13]. Our models will therefore be focusing on predicting the duration_in_traffic value from Google, as this is the travel time which is most similar to the bus travel times collected from Entur.

By comparing both the data example from Entur in listing 4.1 and from google in listing 4.2 we can see that they are quite dissimilar in the sense that we have bus data for each stop and car data for a specific path. Therefore, in order to be able to connect these together we need to transform them to better match each other. We are describing the transformation process further in section 4.3.

4.2 Data harvesting

The data harvesting process is split into 3 steps for each of the aforementioned datasets. Step 1: downloading raw data(XML and JSON) and then directly store this on disk. Step 2: extract the relevant information and remove duplicates where possible. Step 3: in-

sert the data into a database where it is stored for later use in analysis. By separating these steps it allows for a more fault tolerant harvesting architecture and it makes it easier to debug through separate logs when or where something fails.

4.2.1 Cassandra

Cassandra is a distributed storage system for managing very large amounts of structured data spread out across many commodity servers, while providing highly available service with no single point of failure. Cassandra aims to run on top of an infrastructure of hundreds of nodes (possibly spread across different data centers). At this scale, small and large components fail continuously. The way Cassandra manages the persistent state in the face of these failures drives the reliability and scalability of the software systems relying on this service. While in many ways Cassandra resembles a database and shares many design and implementation strategies therewith, Cassandra does not support a full relational data model; instead, it provides clients with a simple data model that supports dynamic control over data layout and format. The Cassandra system was designed to run on cheap commodity hardware and handle high write throughput while not sacrificing read efficiency [18]. The benefits of using a Cassandra database comes with limitations in terms of querying the data. Reads and writes towards the database are limited to only querying on predefined indexes in order benefit from the super fast read and write speeds that Cassandra makes possible. Cassandra uses a query language called CQL (Cassandra Query Language), which at first sight might seem very similar to SQL (Structured Query Language) in many ways, but it is not the same. CQL lacks much of the functionality that SQL has available, with relational joins being one of the major differences. We also had some difficulties performing windowed queries on the database to it key constraints limiting allowed queries. By windowed queries we mean in this context a query that returns data that falls within a given windows based on a start and end value where the value typically is a timestamp or a value with high variance.

Due to the aforementioned limitations that Cassandra comes with, it turned out not to

be the optimal solution for time series analysis. In order to be able to handle the data it was important to choose frameworks that was able to choose process data efficiently at scale. We therefore started looking at Apache-spark as a tool for both pre-processing and training a model on the data.

4.2.2 Cron Scheduling

The Cron daemon, crond, packaged with most Linux distributions, controls scheduling of regularly occurring jobs. When started upon entry into multi-user mode, crond scans the directories `/var/spool/cron/crontabs` and `/etc/cron.d` and the file `/etc/crontab` for work to do. crond then awakens every minute, performs the work its record of jobs says it should do at that time, mails the output (by default) to the owning user, then sleeps until the beginning of the next minute [15]). We used Cron for scheduling when the harvesters should run. Using Cron allows us to schedule tasks down to the minute and worked perfectly well for this case of scheduling data harvesting scripts run times. We scheduled our tasks, fir both the Entur and the google API to run every minute. However, as mentioned and illustrated in figure 4.2, a separate script was written in order to select random times for when to send requests to the google distance matrix API. This script was checked every minute, but was only sending API requests when the current time existed in a file containing randomly generated timestamps every day.

4.2.3 Architecture

The general system architecture consists of a harvesting server, a database server and a modelling server. This section describes the dataflow from data harvesting to model training and predictions.

Harvesting server

The harvesting server is responsible for collecting raw data from both Entur real-time data and Google distance matrix API. All of the raw data is then stored as files on the

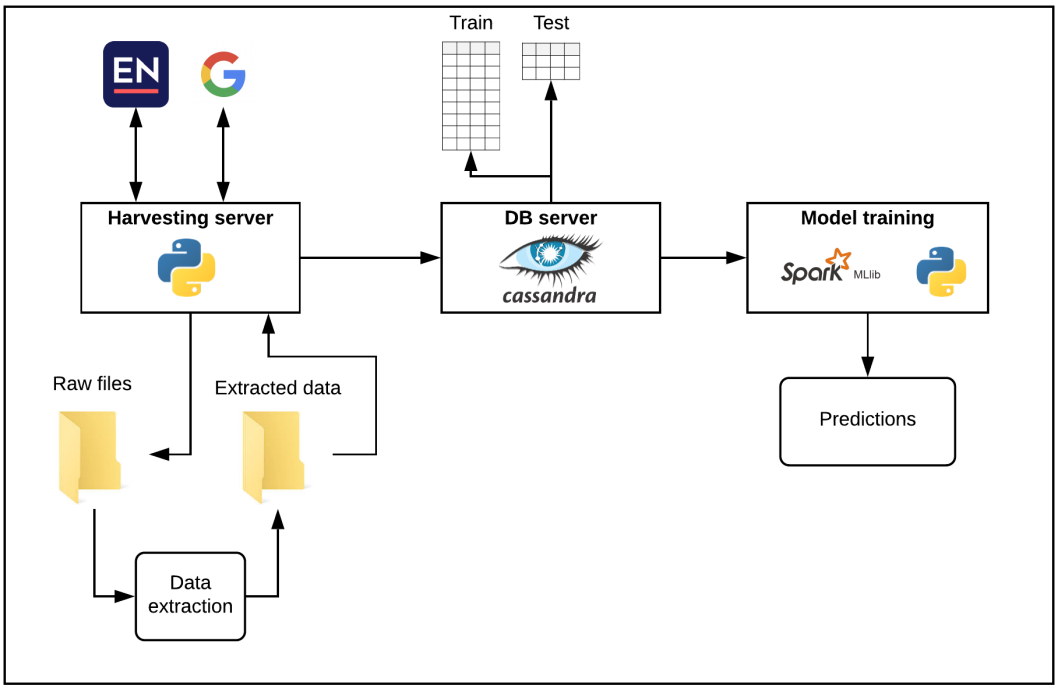


Figure 4.4: An overview of the dataflow from harvesting to predictions

harvesting server as xml and json in a folder separate folders named data. The raw files are then processed to extract the relevant data and convert it into tabular format before moving the files to a folder for extracted data and sending the extracted structured data to the database server.

Database server

The database server serves only one purpose, which is to store store and pass along data to to a spark cluster. As can be seen in figure 4.4, the data is stored into separate tables as it is being harvested to ensure a consistent and even training and testing set along the way.

Spark cluster

The last part of the system consists of a spark cluster which downloads all of the training and test data for both the Entur and Google dataset and then trains 3 different

models on the data and generates travel time predictions. This is described further in chapter 5.

4.3 Data structure and pre-processing

The following section will go through step 3 of our workflow which is about the pre-processing the data files before we continue with model training. We will have a look at how we have used spark to process and transform the raw data so it can be used to train models on the different datasets that were mentioned in section 4.1 and create predictions based on these.

Pre-processing

Many factors affect the success of Machine Learning (ML) on a given task. The representation and quality of the instance data is first and foremost. If there is much irrelevant and redundant information present or noisy and unreliable data, then knowledge discovery during the training phase is more difficult. It is well known that data preparation and filtering steps take considerable amount of processing time in ML problems. Data pre-processing includes data cleaning, normalization, transformation, feature extraction and selection, etc. The product of data pre-processing is the final training set. It would be nice if a single sequence of data pre-processing algorithms had the best performance for each data set but this is not the case [17]. This section describes the steps taken in order to pre-process the data described in chapter 4 and how the data is prepared for training.

Extracting raw data

Before beginning working with the data, it needs to be in a format that can be easily read by the libraries and frameworks. When dealing with machine learning algorithms it is common practice to work with data in tabular or structured format, often referred

to as table format or tables. All of the data harvested for this project was initially just stored as as files in a mostly flat folder structure. These files were all named with a timestamp in order to have a reference of when each record was collected.

In order to be able to use the raw data for training, it was important to extract the data and insert it into a more machine learning friendly format. Most of the data was therefore inserted into a Cassandra database designed for handling big data. Cassandra was described in section 4.2.1. However, even tho Cassandra has great write and read speeds it turned out that its query language CQL was not flexible enough to select the data efficiently due to constraint limitations on the database. In order to handle this, we decided to export all of the data stored in Cassandra and use Spark as a data processing engine to handle the big amounts of data that had to be dealt with. An export to csv format of all the Entur data from the `estimated_call` table came out at around 120gb of data for around a year of harvesting and inserting to Cassandra. Considering the data was getting so big and was no longer able to to fit into memory, was one of the main reasons why we ended up using Spark to process the data and Spark MLlib to train models on the data. Initially we wanted to use libraries such as Pandas and scikit learn, however, these would not have been able to handle this as efficient as Spark potentially can do with its power of scalable in-memory cluster computing.

Data structure

In order to convert the raw XML and JSON data structures harvested from Entur and Google, we needed to first create data schemas that could be used to describe the different parts of the data. The following tables describes the table structures the Entur data was separated into.

Entur estimated timetables schemas

The raw xml data contained within the estimated timetable data shown in example 4.1 were separated into 2 tables to get the `estimated_vehicle_journey` records into one table and the `estimated_call` records into another table which creates a normalized

Entur data table schemas

Table 4.1: Estimated vehicle journey

Column	Data type
journey_ref	ascii
harvested_at	timestamp
block_ref	smallint
cancellation	boolean
destination_name	ascii
destination_ref	ascii
direction_name	text
direction_ref	ascii
is_complete_stop_sequence	boolean
line_ref	ascii
monitored	boolean
operator_ref	tinyint
prediction_inaccurate	boolean
published_line_name	ascii
vehicle_mode	ascii

Table 4.2: Estimated call

Column	Data type
dated_vehicle_journey_ref	ascii
stop_point_ref	ascii
harvested_at	timestamp
aimed_arrival_time	timestamp
aimed_departure_time	timestamp
cancellation	boolean
departure_boarding_activity	ascii
departure_arrival_activity	ascii
expected_arrival_time	timestamp
expected_departure_time	timestamp
prediction_innacurate	boolean
stop_order	smallint
stop_point_name	text

table structure. Table 4.1 contains the columns and its respective datatypes of the Entur estimated vehicle journeys, table 4.2 contains the columns and datatypes for an Entur estimated call. Of these 2 tables, the Entur estimated call is the largest and contains most of the data.

Google distance matrix schema

We managed to contain the information from the google distance matrix api within one table as it was not necessary to normalize these any further. The table schema for the data shown in example 4.2 is displayed in table 4.3 below.

4.4 Data transformations

The following sections concerns step 4 of our workflow which is about the transformations of our data. We can see that the data structures for the google distance matrix data and the Entur estimated timetable data is quite different from each other. There-

Column	Data type
id	bigint
harvested_at	timestamp
origin_address	text
destination_address	text
duration_seconds	bigint
duration_minutes	ascii
distance_meters	bigint
distance_km	ascii
duration_in_traffic_seconds	bigint
duration_in_traffic_minutes	ascii

Table 4.3: Google distance matrix schema

fore, we need to transform the Entur data to match a similar structure where a single record represents the travel time of a path. A path in this context is considered as the geographical length between 2 points, not using straight or direct lines, but following the roads.

Selecting subset of transit lines

Performaing transformations can be both processing and memory heavy. Therefore, it is important to select only a subset of the journeys before performing large data transformations.

```
# Join condition based on journey_ref, harvested_at and direction_ref
cond = ( (calls.dated_vehicle_journey_ref == journeys.journey_ref) \
         & (calls.harvested_at == journeys.harvested_at) \
         & (calls.direction_ref == journeys.direction_ref))
df = calls.join(journeys, on=cond, how="left_outer")

# Filter out the relevant lines
line_refs = ["SKY:Line:3", "SKY:Line:4", "SKY:Line:5",
             "SKY:Line:6", "SKY:Line:83"]
df = df.filter(df.line_ref.isin(line_refs))
```

Listing 4.3: Selecting a subset containing the relevant distances

The selected paths collected through the google distance matrix API all travel through the same main road in order to get to their destination. Therefore, in order to overlap with the same paths, we need to select only the bus lines that overlap with the

paths shown in figure 4.1. We have shown how this is done using spark in example 4.3 shown above. There we see that we need to join the `estimated_call` table together with the `estimated_vehicle_journey` table. In Spark, a table is represented using a data structure called a dataframe. Dataframes in spark are based on underlying RDDs which was the original data structure Spark was created with, but was later moved in a direction where dataframes became the common way to structure data.

Estimated timetable transformations

As stated earlier in section 4.1.1, each of the estimated timetable vehicle journeys contains information about each stop from its origin stop to its destination stop. Therefore, in order to get the duration or travel time of a distance, we need to get the time of arrival at the last stop minus the time of departure from the first stop. In order to do this with the new table structure, we need to connect each of the journeys to its estimated calls and then collect the information from the first and last stops.

By using some of the functionalty that spark has to offer, we were able to perform some transformation relatively easy to a dataframe using the following steps:

1. Transform the dataframe into an rdd
2. Map the rows of the rdd into key value pairs where the key consists of tuple containing `journey_ref`, `harvested_at` and `direction_ref`, and the values as a list containing a single stop.
3. Reduce the rows to contain all the estimated travel times for each journey
4. Map each journey to a single row with travel time
5. Convert back to a dataframe.

This results in a transformed and significantly smaller dataframe that now matches the google distance matrix schema(4.3) to a much higher degree. We are now able to continue onto training a model that uses both datasets with overlapping data. After

```
def map_journey(row):
    # Sort on stop order to make sure first is first and last is last elements
    stops = sorted(row[1], key=lambda  k: k["stop_order"])
    start_point = stops[0]
    end_point = stops[-1]
    travel_time = (end_point["expected_arrival_time"]
                   - start_point["expected_departure_time"]).total_seconds()
    return (row[0][0], row[0][1], start_point["stop_point_ref"],
            end_point["stop_point_ref"], travel_time, len(stops),
             end_point["direction_ref"], end_point["line_ref"])


df = df.rdd.map(lambda row: ((row.harvested_at, row.dated_vehicle_journey_ref, row.direction_ref),
                                [
                                    {
                                        "stop_point_ref": row.stop_point_ref,
                                        "expected_departure_time": row.expected_departure_time,
                                        "expected_arrival_time": row.expected_arrival_time,
                                        "direction_ref": row.direction_ref,
                                        "stop_order": row.stop_order,
                                        "line_ref": row.line_ref
                                    }
                                ]
                               )) \
            .reduceByKey(lambda  x, y: x + y) \
            .map(map_journey)
```

Listing 4.4: Example of the transformation of estimated call structure into a full distance record with travel time estimates

performing the this transformation, we end up with a new dataframe schema that is described in listing 4.5 using a StructType containing each of the new columns and their datatypes.

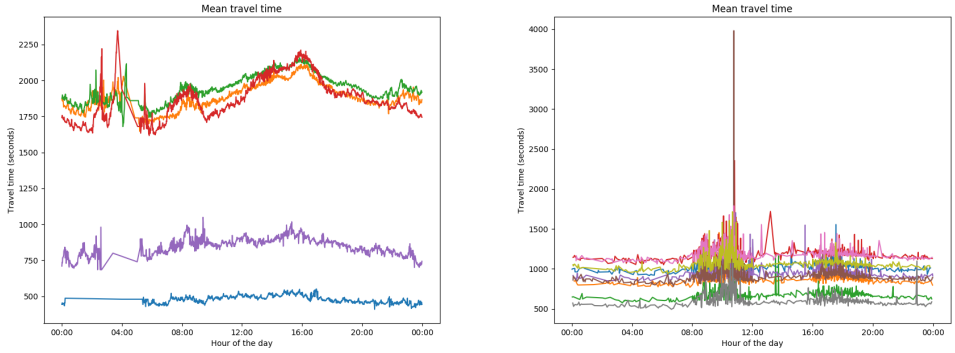
This data schema was then transformed further to match the unified which we will get to in section 5.1. The path ref was generated by combining both `start_ref` and `end_ref` into a combined `path_ref` as stated in the unified schema and then dropped. The `is_actual_time` attribute is generated based on whether the last stop's expected arrival time is before the time of harvest, meaning its the closest we get to an actual time and not a prediction from the SIRI system based on vehicle positioning.

```
estimated_journey_travel_time = StructType([
    StructField("harvested_at", TimestampType(), True),
    StructField("is_actual_time", BooleanType(), True),
    StructField("journey_ref", StringType(), True),
    StructField("path_ref", StringType(), True),
    StructField("start_point_name", StringType()),
    StructField("end_point_name", StringType()),
    StructField("travel_time", FloatType(), True),
    StructField("direction_ref", IntegerType(), True),
    StructField("line_ref", StringType(), True),
    StructField('distance_meters', FloatType(), True)
])
```

Listing 4.5: The resulting dataframe schema of the transformations.

4.5 Data exploration

We will now look at a few visualisations of the data collected and used to train the different models as described in chapter 5. The following figures gives an overview of the travel time for all the car paths and all the bus paths for both inbound and outbound direction using an aggregation of the data to calculated travel time grouped by minute of the day for a 3 month period from 12. august 2019 to 12 november 2019.

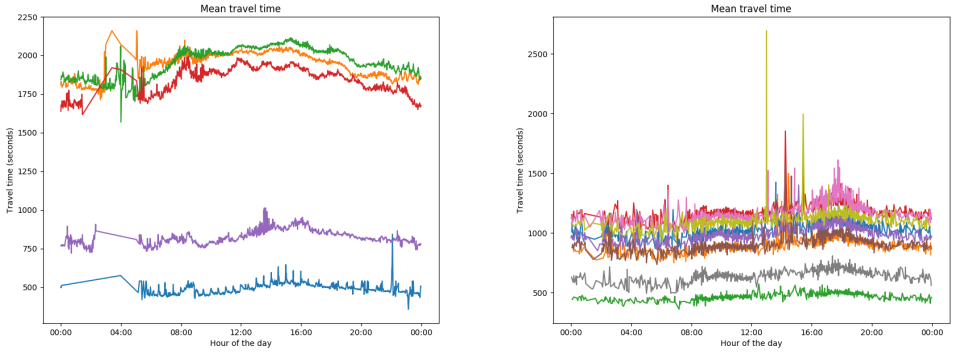


(a) Paths for Entur

(b) Paths for Google

Figure 4.5: All inbound paths

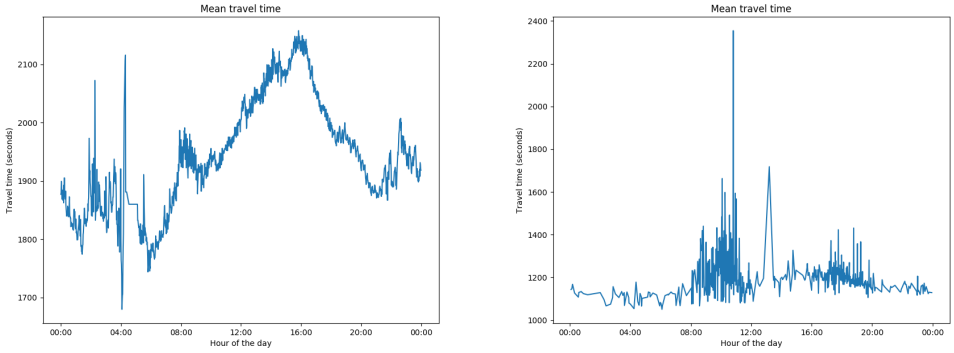
As we can see in the graphs in figure 4.6, there is quite a difference in travel time between the different datasets. There are likely several reasons for why this is happening. We will discuss what we think may be the main issues when using bus data to predict car travel times later on in chapter 7.



(a) Paths for Entur (b) Paths for Google

Figure 4.6: All outbound paths

If we look specifically at bus line 4 and the car path that is tightly overlapping from origin to destination, there are some interesting patterns that might be useful when used to train predictive models.

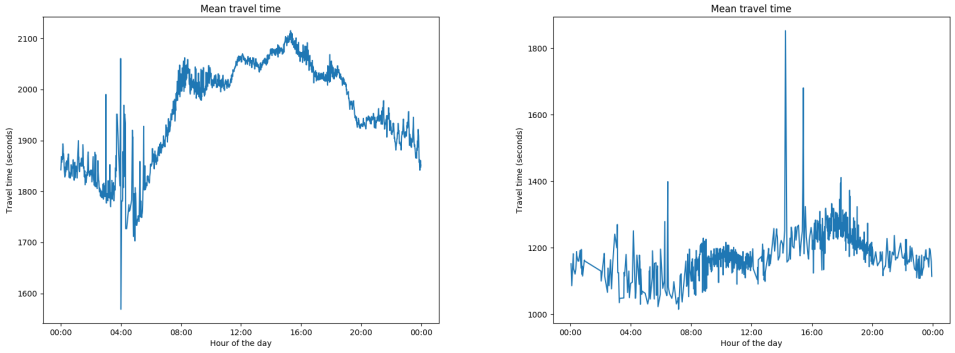


(a) Bus 4 (b) Car path overlapping

Figure 4.7: Inbound paths

The main distinction between the bus and car data is the difference in travel time. It appears that the buses use a significantly longer time to travel the same distance. The second thing is that the buses seem to be less affected by morning traffic when going inbound direction. However, this might be due to lack of car data. There are likely several different reasons for why this may be occurring in which we will discuss further in the discussion path of the thesis.

Despite the aforementioned differences between the bus and car data, there still seems to be a similar peak during rush hours in the morning and afternoon which we hope



(a) Bus line 4 (b) Car path overlapping

Figure 4.8: Outbound paths

will be discovered when training a combined model.

Chapter 5

Modelling

We are used machine learning as our approach to pattern recognition as part of step 5 of our workflow stated in section 3.3. This chapter will go through the steps 6 and 7 concerning model selection and training. We will go through the steps taken in order create future travel time predictions based on the data and its transformations described in chapters 4. We will look into which methods and algorithms has been used and how we have trained models to allow for future predictions based on an input of a timestamp in the future and a given travel path.

5.1 Feature selection

Before we can train a model we need to select a subset of features from a dataset we want to use for training a model. Machine learning algorithms automatically extract knowledge from machine-readable information. Unfortunately, their success is usually dependant on the quality of the data that they operate on [17], and it is therefore highly important to select the most relevant features of the dataset.

Feature subset selection is the process of identifying and removing as many irrelevant and redundant features as possible. This reduces the dimensionality of the data and enables learning algorithms to operate faster and more effectively [17]. Features can generally be characterized as:

- Relevant: These features have an influence on the output and their role can not be assumed by the rest
- Irrelevant: Irrelevant features are defined as those features not having any influence on the output, and whose values are generated at random for each example.
- Redundant: There exists a redundancy whenever a feature can take the role of another.

Removing irrelevant and redundant features leaves us with only the relevant features and will return the best results when training a model.

5.2 Model input data

This sections looks into the input data used to train the machine learning models. We will look at the different the datasets both combined and separately.

5.2.1 Unified training schema

As stated in research questions 1.1, one of the main questions we are trying to answer is whether we are able to improve the accuracy of travel time prediction by combining datasets about the same domain but different in terms of size and quality, where the label we want to predict lies within a dataset of lower density, which in this case is the Google distance matrix data. In order to do this, our approach for this has been to keep the Google data structure pretty much as it is, while transforming the Entur bus data to match the same structure.

The unified schema as shown in table 5.1 contains the most relevant features shared between both datasets. We have split the `harvested_at` columns into 3 separate columns `day_of_year`, `day_of_week` and `second_of_ day` in order to distinguish between them when training a model. The `travel_time` column consists of a floating point number representing the amount of seconds it takes for a vehicle from the start of a path to

Column	Datatype
day_of_year	Integer
day_of_week	Integer
second_of_day	Integer
path_ref	String
direction_ref	Integer
vehicle_type	string

Table 5.1: A unified data structure used for combining Google and Entur data.

its end. Each path is categorized using a `path_ref`, which is a string made of an origin location and a destination location and serves as a unique identifier for each path. The `direction_ref` tells us which direction the path is going, it is either inbound or outbound from the main sentrum stop and is presented through an integer with 1 for outbound and 2 for inbound.

5.2.2 Google input data

The google input data is quite straight forward with the exception of the choice between `travel_time` column. As can be seen in table 4.3, there are 2 estimates for travel_time, namely `duration_in_traffic` and `duration`. We have chosen to use the duration_traffic estimate as our target label for predictions due to its increased coherence with the Entur bus data.

	summary	path_ref	travel_time	distance_meters
0	count	41498	41498	41498
1	mean	None	935.693	10944.824
2	stddev	None	235.072	3806.406
3	min	Christies gate, ...	360.0	2478.0
4	max	Åsane Storsenter...	4211.0	38897.0

Table 5.2: Google car data statistics

5.2.3 Entur input data

Earlier in section 4.4 we transformed the Entur data from estimated calls and journeys into just estimated journeys. The results of this transformation is shown in the table

below. The Entur data after the initial transformations has been further transformed to match the unified schema perfectly. We have replaced the `harvested_at` feature from the schema in code listing 4.5 with the same temporal features as contained in the unified schema.

	summary	path_ref	travel_time	distance_meters
0	count	1823564	1823564	1823564
1	mean	None	1781.198	20854.720
2	stddev	None	450.261	5879.932
3	min	NSR:Quay:530...	241.0	2985.651
4	max	NSR:Quay:537...	5925.0	24688.715

Table 5.3: Entur bus data statistics

5.2.4 Combined input data

The combined input model is trained using a combination of both the bus data from the real-time Entur API and the car data collected from the google distance matrix API together. We are combining these by transforming each of the aforementioned datasets into the unified training schema as seen in table 5.1. These datasets are simply combined using a union in spark. What we are hoping to achieve when doing this is that the error rate of the combined model is, in an optimal situation, lower than the one we get from training a model on the Google data alone.

5.2.5 Training implementation

As mentioned, we have used the spark ML library to train our models. The spark ML library has allowed for great flexibility and generalization which has made it possible to test different models using the same function that takes in an algorithm in the form of a dictionary containing metadata about its' parameters and map parameters used for cross validation using 5 fold for hyper-parameter tuning. The following code example contains the generalized function used to train a model, also referred to as a transformer.

```
def train_model(df, model_meta, feature_cols):
    # Setup regression algorithm based on input
    regressor = model_meta['algo'](**model_meta['kwargs'])

    if model_meta['mapParams']:
        paramGrid = ParamGridBuilder()
        for key, param in model_meta['mapParams'].items():
            paramGrid.addGrid(getattr(regressor, key), param)
        paramGrid = paramGrid.build()
        # Run cross-validation, and choose the best set of parameters.
        model = CrossValidator(estimator=regressor,
                               estimatorParamMaps=paramGrid,
                               evaluator=RegressionEvaluator(),
                               numFolds=5)  # 5+ folds is considered common practice
        model = model.fit(df)
        bestModel = model.bestModel
        model = bestModel.stages[-1]
    else:
        model = regressor.fit(df)
    return model
```

Listing 5.1: Generalized training function

The input to our training function is a pre-indexed dataframe containing a "features" column which holds a vector containing all the features where strings have been indexed as numbers and each feature has been categorized as either a categorical or continuous variables. In addition to the features input we also have the label column which is the desired output of our model e.g. the travel time in seconds.

5.3 Iteration 1 - Unified schema

To test our hypothesis we have trained 3 different models which are based on 3 different datasets using the same unified schema as stated in table 5.1. These 3 models are trained using 3 different input datasets. The first model is trained using the car travel time data from google alone, then the second model is trained using only Entur bus travel time data and finally a combined model that use the data from both Google and Entur.

All of following models discussed below are based on data within a time period from 12 August 2019 to 12 November 2019. This period was selected as Skyss, the bus com-

pany, had started their new routes for the Autumn at that day. This way we could avoid having possible changes in travel times because of route changes.

The models trained in this thesis are based on a batch learning approach where we are using historical real-time data. In iteration 1 we have trained 3 models based on the initial unified schema as described in section 5.1. All of the complete result tables are listed in the appendix.

5.3.1 Google car data

All of the data collected from within our target period sums up to a total count of 34144 rows for our Google training set and a total count of 7354 rows held out for testing. We have stored the training data separately with 80% for training and 20% for testing in order to evaluate model performance.

The point of training a model on the Google car data alone is to see how these results compare to a combined model trained on both car travel times and bus travel times with overlapping paths. The first model trained using the historical car data from our selected period showed decent results as shown in section 6.1.

5.3.2 Entur bus data

Similar to the Google Car dataset, the Entur bus dataset is also split randomly into separate training and test sets with a distribution of 80% for training and 20% for testing. We have a significantly larger dataset for bus travel time compared to car travel time. The total amount of training entries comes to a count of 1454870 and the total amount of testing entries comes to a total of 368694 for the time period from 12 August to 12 November 2019.

The bus data from Entur is the largest dataset of the two and was intended to be used in attempt at improving the accuracy of car travel times by combining it with the car travel data. However, to see how a model trained on the Entur bus data performs at predicting bus travel times, the second model is trained on just bus data from Entur

alone. By training a model on just the bus data we can get an overview of how well the model performs at predicting itself, and whether or not it may be useful for predicting car travel data.

5.3.3 Combined model

The third model is the combined model which is a combination of both the Entur bus travel times and the Google car travel times. The intention of this model is to see if we are able to better predict car travel times by combining the two different datasets. The way this has been done is through a union between the two datasets and a model have been trained on all the data. The total count of entries in the combined used for training is therefore a total of 1489014 and the test set is the same as for our car travel times model, meaning we have a small test set of 7354 entries.

We found that the predictions were performing slightly better when using random forest regression. However, the predictions had potential improvements and we decided to generate a new feature column that could improve the accuracy as measured by the RMSE results.

As previously mentioned in section 4.1.2 and shown in figure 4.3b, the Google model takes into consideration the current traffic based on the time of query execution. The models trained so far have had no indication of traffic intensity. In the second iteration we train a model for that predicts traffic flow estimation or intensity to improve the accuracy of our initial models.

5.4 Iteration 2 - Traffic intensity

We will now describe how we have trained a model that predicts traffic intensity or based on a combined model where the output is an estimated number under or over 1.0 where lower means low flow and higher means good flow.

5.4.1 Calculating averages

To get an estimate of the current traffic intensity we calculated an average of traffic flow for each of the unique paths in our dataset and grouped them using the following steps.

1. Group all the entries based on path reference, minute of week and perform an aggregate action to calculate the average travel time for that path in time. This gives us base in which we use for calculating traffic flow.
2. Join the original dataframe with the group to get both the actual travel time and the average travel time for that minute
3. Calculate the traffic flow/intensity by dividing the actual travel time with the average travel time which returns a floating point estimate.

Using the aforementioned steps have given us en estimate of the traffic flow based on historical data which has been used to train a traffic flow model. In order to predict future traffic flow we trained a model using the estimated traffic flow calculation.

5.4.2 Intensity model

We train an intensity model using the input columns as described in table 5.4, and use the predicted traffic flow as the training label. The data fed into the model is a combined set using both car data and bus data.

Column	**Datatype**
day_of_year	Integer
day_of_week	Integer
second_of_day	Integer
path_ref	String
direction_ref	Integer
vehicle_type	string
travel_time	float

Table 5.4: Input data for traffic intensity model that outputs a traffic flow prediction

When applying the transformation of our unified schema we hope to achieve a better correlation between the datasets and improve the accuracy of our models. The actual results of the aforementioned models are evaluated in the next chapter.

Chapter 6

Evaluation of results

This chapter evaluates our results as stated in step 8 of our KDD workflow. We will look at the results we have managed to achieve while training our 3 machine learning models in both first and second iteration. We will look at the statistical results in table form as well as visualisations of our observed test compared to the predicted output of the models.

We have primarily measured the accuracy of our models using the RMSE metric as described in section 2.5.5 as well as an R2 rating for coherence determination which is similar to a normalized version of the RMSE. Buses and cars have dissimilar travel times for the same paths and the RMSE can therefore not be used as the only metric when comparing the results for the models trained alone. The R2 coherence score is a way to moralise the RMSE. All of the best results for the models trained are displayed in table 6.1 below.

Model	mae	mse	r2	rmse
Iteration 1 - alone without intensity				
Entur	79.573	14802.640	0.927	121.666
Google	38.619	4638.130	0.915	68.103
Combined	36.572	3960.605	0.927	62.933
Iteration 2 - with added intensity				
Entur	59.624	8931.502	0.970	94.506
Google	24.689	2586.978	0.952	50.862
Combined	24.403	2491.484	0.954	49.914

Table 6.1: A summary of the best results for models trained in both iteration 1 and 2. The complete result overview can be seen in appendix A.1

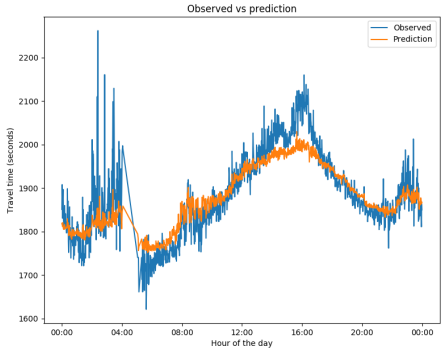
6.1 Analysis

We will now look a bit deeper into the performance of the models to see where we have managed to improve the models using visualisations of the observed test data along with the predictions. Considering all of the test data is shuffled randomly and the collection of car data is mainly within rush hour traffic we have grouped the graph data by minute and calculated the average travel time. We are comparing each of the models based on iteration and using predictions for the path between Torget and Flaktveit which is the same path as represented in section 4.5, overlapping with bus 4.

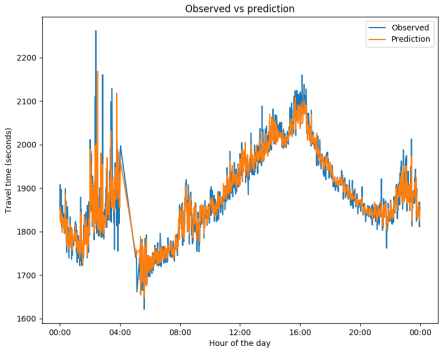
Entur models

The Entur models are the best performing models of the ones we have trained; which is likely due to the size of the dataset. In the first iteration Entur model can see that the prediction have a tendency to predict a lower travel time when the actual observed travel time is high and a higher value when the prediction is low.

After applying the traffic flow feature in the second iteration, it adapts much better to the observations compared to iteration 1.



(a) Entur bus model trained alone

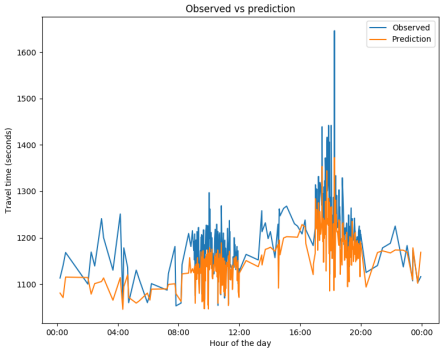


(b) Entur bus model with traffic flow

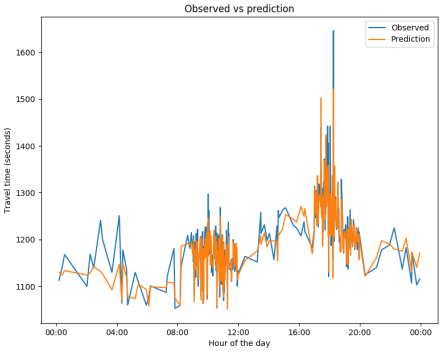
Figure 6.1: Entur model predictions for both iteration 1 and iteration 2

Google models

The predictions we managed to get from the Google model alone were good considering the significantly smaller dataset, however, there were still space for improvement. The initial model trained using the Google car data alone managed to get a best score of 68.102 RMSE using the Random Forest ML algorithm as seen in table 6.1.



(a) Google car data model trained alone



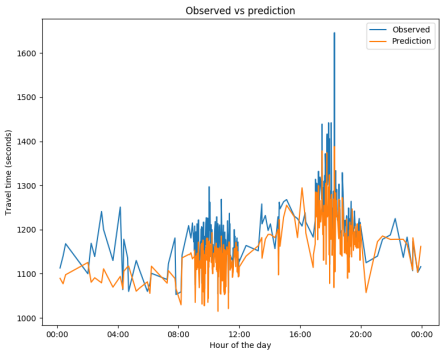
(b) Google car data model with traffic flow

Figure 6.2: Google model predictions for both iteration 1 and iteration 2

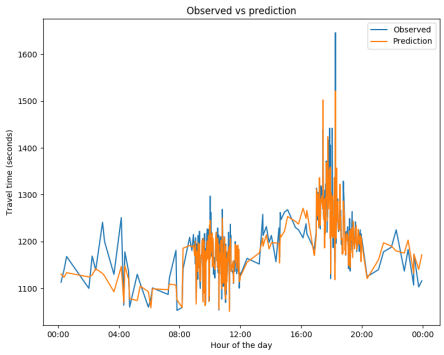
When adding the new traffic flow feature to the initial google model we see that the predictions are closer to the observed data during high intensity periods and according to the results in our results table, it gets a better RMSE score of 50.862. The models appears to be adapting well throughout the day, although the visualisations might not be the best ways to measure considering the grouped Google data is uneven in terms of density. Meaning there may be multiple observations in the same minute.

Combined models

The combined model compared to the google model has an increased performance when comparing RMSE. Distinguishing between the differences in performance through the models we can see from the results in table 6.1 that the first iteration combined model has an improvement as opposed to the Google model trained on car data alone.



(a) Combined model trained alone



(b) Combined model with traffic flow

In the second iteration combined model the prediction accuracy improved performance of the combined model, however, we see that the difference compared to the Google model trained with traffic flow alone (fig. 6.2b) is smaller and that the traffic flow feature has become a large factor in estimating. It is worth noting that the traffic flow model used to predict the traffic flow feature has been trained on both the Google car data and the Entur bus data and the coherence may therefore already be in the added traffic flow feature and therefore show a smaller difference when used on the combined model.

6.2 Summary

We have seen that the first iteration traffic models tended to either overestimate or underestimate the travel time based on how far it is from the mean. This means that the paths that were shorter had a tendency of getting predictions of a higher duration than observed and quite the opposite happened with the longer paths. Adding a new feature for traffic flow allowed the models to predict travel time more accurately without being

affected as much by the general mean travel time overall.

Chapter 7

Conclusions, Discussion, and Recommendations for Further Work

This chapter We will now discuss some of most interesting findings and conclude the thesis as well as propose some suggestions for further work.

7.1 Discussion

I this section we discuss the findings and share the knowledge we have discovered as stated in the final stage of our KDD workflow.

7.1.1 Research questions

RQ 1: Can combining multiple sources of different transit data predict accurate travel times?
Through the training and evaluation of the combined models we have found that the predictions from the models are quite accurate and may be useful in a real-time scenario.

RQ 2: Can we improve the accuracy of predictions by combining a low density dataset

with a dataset of higher density from the same domain?
When comparing the results from the combined models with the models trained alone, we see an increase in accuracy. However, to get the best results we could not solely rely on the combination of the data alone, but through feature engineering of a mostly common factor of traffic flow.

RQ 3: Can we gain useful knowledge from the connection between these different datasets?
We believe that the connection between these different datasets have shown that we can combine similar datasets to improve accuracy and reliability of predictions, not just within the transit domain, but as a general concept where there may be some coherence between datasets of different sizes.

7.1.2 Approach drawbacks

The models we have trained so far have succeeded in general and appears to be quite accurate according to the tests we have completed. There are however a few drawbacks to the path approach. All the models are heavily dependant on the path reference when estimating the future travel time and the distinction between the different paths appeared to be too dependant on the path reference.

7.1.3 Variable differences

The main difference between bus travel times and car travel times is the duration. There are several reasons that cause this; where stopping at bus stops is the main reason for the longer travel time. Time used at bus stops, however, is not time affected by traffic considering the bus is not moving. Public events or happenings in the city center often causes more people to take the bus and may in return increase the waiting times, without actually delaying the flow of car traffic.

Some patterns that appear in the bus data are the high travel times at night time for some buses. Most buses do not travel at night, but there are some exceptions during

the weekends which we can see in our data. These night buses are likely slowed down by a large number of passengers who want to get the last bus home from a night out.

Carpool lanes for buses is another factor that may impact the travel times of buses, or at least the real-time flow information. Carpool lanes allow buses to skip traffic in a separate lane and may therefore in some cases be providing inaccurate information when used for car travel time prediction.

7.2 Summary and Conclusions

Through this thesis we have looked at the possibilities of traffic estimation and prediction for car travel data using a combination of both bus and car data. We have collected bus travel times from the organization Entur as well as car travel times from the Google cloud platform API. The collected data has been processed to match a unified schema that is similar for both the bus data from Entur and the car data from Google before being used for model training. We have trained three different models in 2 iterations with the use of a smaller dataset of car data and a larger dataset of bus data. Using a path based approach, we found that combining bus and car data improves the prediction of car travel times by using a model trained on a dataset containing a combination of both car and bus historical travel times. We can therefore conclude that we have successfully shown that the connections between different datasets about the same domain may improve accuracy and reliability of predictions in general, and not just within the transit domain.

7.3 Recommendations for Further Work

The focus in this thesis has been on travel time predictions for car data. However, there are several other areas in which we could have used the data at hand. Looking at the statistics of both the bus and the car data, there exists some clear anomalies with unknown origins. Using the data we have looked at in this thesis it would likely be pos-

sible to label these anomalies and train a binary classification model that can detect anomalies in real-time. This however, would require a real-time system where our traffic flow model is trained incrementally to continuously provide accurate traffic flow predictions in real-time and not based on historical data alone. Spark streaming provides APIs for training of models and predictions on demand that could be useful when working with real-time data.

Another problem that was not possible to address in our case is the issue of speed and speed limits. We were unable to estimate speed due to these paths having different speed limits along the path. By splitting paths into tightly connected subsection based on speed limits and traffic flow it could possibly be easier to estimate speed for an entire path. Having knowledge of speed limits would also possibly allow for scaling of bus travel times down to car travel times to further improve the accuracy and reliability of predictions. The nature of the car travel time data we have collected did not allow us to create sections since the only indications we had was a travel time from origin to destination.

References

[1] Muhammad Ovals Ahmad et al. “Kanban in software engineering: A systematic mapping study”. In: *JOURNAL OF SYSTEMS AND SOFTWARE* 137 (Mar. 2018), pp. 96–113. ISSN: 0164-1212. DOI: 10.1016/j.jss.2017.11.045.

[2] Carlos Castillo. *Big crisis data: social media in disasters and time-critical situations*. Cambridge University Press, 2016.

[3] Matej Cebecauer. *Short-Term Traffic Prediction in Large-Scale Urban Networks*. 2019.

[4] T. Chai and R. R. Draxler. “Root mean square error (RMSE) or mean absolute error (MAE)? -Arguments against avoiding RMSE in the literature”. In: *Geoscientific Model Development* 7.3 (2014), pp. 1247–1250. ISSN: 19919603. DOI: 10.5194/gmd-7-1247-2014.

[5] Xiqun Chen et al. “Data for Traffic State Estimation and Prediction”. In: 20.4 (2019), pp. 1247–1258. DOI: 10.1109/TITS.2018.2847024.

[6] Adele Cutler, D Richard Cutler, and John R Stevens. “Random Forests”. In: *Ensemble Machine Learning: Methods and Applications*. Ed. by Cha Zhang and Yunqian Ma. Boston, MA: Springer US, 2012, pp. 157–175. ISBN: 978-1-4419-9326-7. DOI: 10.1007/978-1-4419-9326-7{_}5. URL: https://doi.org/10.1007/978-1-4419-9326-7_5.

[7] *Development portal - Entur*. Apr. 2018. URL: http://www.entur.org/dev.

[8] Aline Dresch, Daniel Pacheco Lacerda, and Jos Antnio Valle Antunes. *Design Science Research: A Method for Science and Technology Advancement*. Springer Publishing Company, Incorporated, 2014. ISBN: 9783319073736.

[9] Usama Fayyad, Gregory Piatetsky-Shapiro, and Padhraic Smyth. “From data mining to knowledge discovery in databases”. In: *AI magazine* 17.3 (1996), p. 37.

[10] Amir Gandomi and Murtaza Haider. “Beyond the hype: Big data concepts, methods, and analytics”. In: *International Journal of Information Management* 35.2 (2015), pp. 137–144. ISSN: 02684012. DOI: 10.1016/j.ijinfomgt.2014.10.007. URL: http://dx.doi.org/10.1016/j.ijinfomgt.2014.10.007.

[11] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016.

[12] Google. *GTFS Static Overview*. Apr. 2019. URL: https://developers.google.com/transit/gtfs/.

[13] Google. “The Google Distance Matrix API”. In: 2014.04/29 (2014). URL: https://developers.google.com/maps/documentation/distancematrix/.

[14] Alan R Hevner. “A Three Cycle View of Design Science Research”. In: *A Scandinavian Journal of Information Systems* (2007).

[15] Michael S Keller. “Take Command: Cron: Job Scheduler”. In: *Linux J.* 1999.65es (Sept. 1999). ISSN: 1075-3583. URL: http://dl.acm.org/citation.cfm?id=327966.327981.

[16] X Kong et al. “Mobile Crowdsourcing in Smart Cities: Technologies, Applications, and Future Challenges”. In: *IEEE Internet of Things Journal* 6.5 (Oct. 2019), pp. 8095–8113. ISSN: 2372-2541. DOI: 10.1109/JIOT.2019.2921879.

[17] S B Kotsiantis and D Kanellopoulos. “Data preprocessing for supervised leaning”. In: *International Journal of . . .* 1.2 (2006), pp. 1–7. ISSN: 1306-4428. DOI: 10.1080/02331931003692557. URL: http://www.google.com/search?client=safari&rls=en&q=Data+Preprocessing+for+Supervised+Leaning&ie=UTF-8&oe=UTF-8%5Cnpapers2://publication/uuid/AA4424CD-8BE0-43AB-838B-8BBBDE502355.

[18] Avinash Lakshman and Prashant Malik. “Cassandra - A decentralized structured storage system”. In: *Operating Systems Review (ACM)*. Vol. 44. 2. Apr. 2010, pp. 35–40. DOI: 10.1145/1773912.1773922.

[19] Doug Laney. “3D Data Management: Controlling Data Volume, Velocity, and Variety.” In: *Application Delivery Strategies* (2001). ISSN: 09505849. DOI: 10.1016/j.infsof.2008.09.005. arXiv: 0402594v3 [cond-mat].

[20] Yang Li et al. “Urban Travel Time Prediction using a Small Number of GPS Floating Cars”. In: *GIS: Proceedings of the ACM International Symposium on Advances in Geographic Information Systems* 2017-Novem (2017). DOI: 10.1145/3139958.3139971.

[21] Yisheng Lv et al. “Traffic Flow Prediction With Big Data : A Deep Learning Approach”. In: *IEEE Transactions on Intelligent Transportation Systems* 16.2 (2015), pp. 865–873. DOI: 10.1109/TITS.2014.2345663.

[22] Chip M Lynch et al. “Prediction of lung cancer patient survival via supervised machine learning classification techniques”. In: *International Journal of Medical Informatics* 108 (2017), pp. 1–8. ISSN: 1386-5056. DOI: https://doi.org/10.1016/j.ijmedinf.2017.09.013. URL: http://www.sciencedirect.com/science/article/pii/S1386505617302368.

[23] Joo Mendes-Moreira et al. “Comparing state-of-the-art regression methods for long term travel time prediction”. In: *Intelligent Data Analysis* 16.3 (2012), pp. 427–449. ISSN: 1088467X. DOI: 10.3233/IDA-2012-0532.

[24] Jennifer C. Molloy. “The open knowledge foundation: Open data means better science”. In: *PLoS Biology* 9.12 (2011), pp. 1–4. ISSN: 15449173. DOI: 10.1371/journal.pbio.1001195. URL: https://doi.org/10.1371/journal.pbio.1001195.

[25] Issam El Naqa and Martin J Murphy. “Machine Learning in Radiation Oncology”. In: *Machine Learning in Radiation Oncology* (2015), pp. 3–11. DOI: 10.1007/978-3-319-18305-3.

[26] Alex Neilson et al. “Systematic Review of the Literature on Big Data in the Transportation Domain: Concepts and Applications”. In: *Big Data Research* 17 (2019), pp. 35–44. ISSN: 22145796. DOI: 10.1016/j.bdr.2019.03.001. URL: https://doi.org/10.1016/j.bdr.2019.03.001.

[27] Apache software Organization. “Spark MLLib Pipeline”. In: (2019). URL: https://spark.apache.org/docs/latest/ml-pipeline.html.

[28] Dieter Pfoser, Sotiris Brakatsoulas, and Petra Brosch. “Dynamic Travel Time Provision for Road Networks”. In: c (2008), pp. 8–11.

[29] Python Language Reference. “Python Software Foundation”. In: *Python Language Reference, version 3.6.1* (2010), Version 3.03., http://www.python.org. URL: https://www.python.org/.

[30] Sergio Ramirez-Gallego et al. “Big Data: Tutorial and guidelines on information and process fusion for analytics algorithms with MapReduce”. In: *INFORMATION FUSION* 42 (July 2018), pp. 51–61. ISSN: 1566-2535. DOI: 10.1016/j.inffus.2017.10.001.

[31] Christian Robert. “ Machine Learning, a Probabilistic Perspective ”. In: *CHANCE* (2014). ISSN: 0933-2480. DOI: 10.1080/09332480.2014.914768.

[32] A Georges L Romme. “Making a Difference: Organization as Design”. In: (Oct. 2003). URL: https://pubsonline.informs.org/doi/abs/10.1287/orsc.14.5.558.16769.

[33] Shai Shalev-Shwartz and Shai Ben-David. *Understanding machine learning: From theory to algorithms*. Vol. 9781107057. 2013, pp. 1–397. ISBN: 9781107298019. DOI: 10.1017/CBO9781107298019.

[34] *SIRI standard - Transmodel*. 2019. URL: http://www.transmodel-cen.eu/standards/siri/.

[35] *Startsiden - Skyss*. Apr. 2018. URL: https://www.skyss.no.

[36] Shu-kai S Fan Chuan-jun Su, Han-tang Nien Pei-fang Tsai, and Chen-yang Cheng. “Using machine learning and big data approaches to predict travel time based on historical and real-time data from Taiwan electronic toll collection”. In: (2018), pp. 5707–5718. DOI: 10.1007/s00500-017-2610-y.

[37] The Trello website. “Trello”. In: (2019).

[38] Martin Treiber and Dirk Helbing. “Reconstructing the Spatio-Temporal Traffic Dynamics from Stationary Detector Data”. In: *Cooper@tive Tr@nsport@tion Dyn@mics* 1.3 (2002), pp. 1–24.

[39] Yuan Wang et al. “Enhancing transportation systems via deep learning : A survey”. In: *Transportation Research Part C* 99.October 2018 (2019), pp. 144–163. ISSN: 0968-090X. DOI: 10.1016/j.trc.2018.12.004. URL: https://doi.org/10.1016/j.trc.2018.12.004.

[40] Matei Zaharia et al. “Resilient Distributed Datasets: A Fault-tolerant Abstraction for In-memory Cluster Computing”. In: *Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation*. NSDI’12. Berkeley, CA, USA: USENIX Association, 2012, p. 2. URL: http://dl.acm.org/citation.cfm?id=2228298.2228301.

Appendix A

Model results

Result metrics for all the models trained

A.1 Results from first iteration

A.1.1 Entur bus data

Model	mae	mse	r2	rmse
Linear regression	102.914	22521.453	0.889	150.071
Decision tree regressor	90.364	19368.352	0.904	139.170
Gradient-boosted trees	79.394	15181.146	0.925	123.211
Random forest	79.573	14802.640	0.927	121.666

Table A.1: Results after training a model on just Entur bus journeys

A.1.2 Google car data

Model	mae	mse	r2	rmse
Decision tree regressor	51.125	7928.833	0.855	89.043
Linear regression	50.804	7829.787	0.857	88.486
Gradient-boosted trees	40.886	4979.630	0.909	70.566
Random forest	38.619	4638.130	0.915	68.103

Table A.2: Results after training a model on just Google car travel times

A.1.3 Combined model

Model	mae	mse	r2	rmse
Gradient-boosted trees	86.935	13602.309	0.752	116.628
Decision tree regressor	62.213	9938.545	0.818	99.692
Linear regression	53.431	8548.153	0.844	92.456
Random forest	36.572	3960.605	0.927	62.933

Table A.3: Results for the combined model

Feature importances

Model	path_ref	vehicle_type	day_week	day_year	direction_ref	second_day
Google	0.8775	0.0	0.0351	0.0240	0.0082	0.055
Entur	0.9549	0.0	0.0061	0.0062	0.0018	0.0307
Combined	0.9262	0.0314	0.005	0.0060	0.0017	0.0285

Table A.4: Feature importances first iteration

A.2 Results from second iteration

A.2.1 Entur model trained with traffic intensity

Model	mae	mse	r2	rmse
Linear regression	110.567	25673.125	0.916	160.228
Decision tree regressor	96.139	18827.025	0.938	137.211
Gradient-boosted trees	87.647	17969.176	0.941	134.049
Random forest	59.624	8931.502	0.970	94.506

Table A.5: Entur model with traffic intensity

A.2.2 Google model trained with traffic intensity

Model	mae	mse	r2	rmse
Linear regression	52.074	8489.323	0.845	92.137
Decision tree regressor	50.389	6901.904	0.874	83.077
Gradient-boosted trees	38.267	5620.726	0.897	74.971
Random forest	24.689	2586.978	0.952	50.862

Table A.6: Google model with flow

A.2.3 Combined model trained with traffic intensity

Model	mae	mse	r2	rmse
Gradient-boosted trees	75.869	11846.109	0.784	108.839
Decision tree regressor	69.434	11349.034	0.793	106.531
Linear regression	54.839	8820.190	0.839	93.915
Random forest	24.403	2491.484	0.954	49.914

Table A.7: Combined model with flow

A.2.4 Feature importances the 3 models with flow

Model	path_ref	vehicle_type	day_week	day_year	direction_ref	second_day	flow
Google	0.835	0.0	0.032	0.022	0.006	0.051	0.051
Entur	0.578	0.0	0.006	0.008	0.325	0.028	0.053
Combined	0.877	0.027	0.007	0.006	0.001	0.035	0.043

Table A.8: Feature importances for the flow models

Appendix B

Source code

B.1 Source code

The source code for the project will be available on Github as of January 2020 using the following link: https://github.com/Lillevik/MasterProject