Emil Øien Lunde, Thomas Wolff

Travel Time Prediction: a comparison study on a common data set

Master's thesis, spring 2015

Artificial Intelligence Group
Department of Computer and Information Science
Faculty of Information Technology, Mathematics and Electrical Engineering

Abstract

Intelligent transportation systems are solutions using information and communication technology in the transportation sector. These systems may contribute to increased traffic efficiency and safety, and are becoming an increasingly larger part of the modern society. Accurate and reliable information is crucial for these types of systems, which is why conducting research in this area is important.

Systems for predicting travel time are examples of intelligent transportation systems, and a great variety of approaches to travel time prediction are reported in the literature. Ensemble learning methods are used to increase prediction accuracy, whilst online learning methods are used to adapt to changes in traffic conditions. Determining which ensemble learning approaches, and which online learning approaches produce highest prediction accuracy is difficult since the reported methods are tested on different data sets. Consequently a fair comparison can not be made.

This work investigates the use of the ensemble learning approaches bagging, boosting, lasso, and a fuzzy rule based system on a common data set. The online learning methods online-delayed extended Kalman filter and local online kernel ridge regression are also compared using a common data set.

The results indicate that...

Preface

This work was conducted as part of the authors' Master's degree at NTNU. It is an extension of the work done in Øien Lunde and Wolff [2014]. Sections 1.1, 2.1 and 2.2 in this paper are repeated from that report.

Emil Øien Lunde, Thomas Wolff
Trondheim, May 13, 2015

Acknowledgements

We would like to thank...

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Section 1.1 introduces the concept of intelligent transportation systems and conceptual approaches to travel time prediction. Note that Section 1.1 is repeated from Øien Lunde and Wolff [2014, p. 1-4]. Section 1.2 describes the goal of this study, and states the research questions this work sets out to answer. Section 1.3 describes the research method employed and Section 1.4 summarizes this work's contributions. Finally, Section 1.5 describes the structure of the rest of the paper.

1.1 Background and Motivation

Section 1.1.1 gives an overview of intelligent transportation systems. Then, Section 1.1.2 introduces conceptual approaches to estimating or predicting traffic variables.

1.1.1 Intelligent Transportation Systems

Intelligent transportation systems is a term describing systems and services in the transportation sector using information and communication technology [Norwegian Public Roads Administration, 2007]. Intelligent transportation systems are often highly complex, consisting of multiple levels of hardware and software each responsible for performing different tasks. To collect road data, multiple sources such as cameras, loop detectors, and GPS devices are used. Since road networks span thousands of square kilometers, the distributed systems responsible for collecting and transmitting this data need to be capable of dealing with events such as faulty detectors and lost connections between transmission devices. Additionally, data is collected continuously, producing large amounts of raw data. The systems responsible for processing this data need to be very efficient to keep up

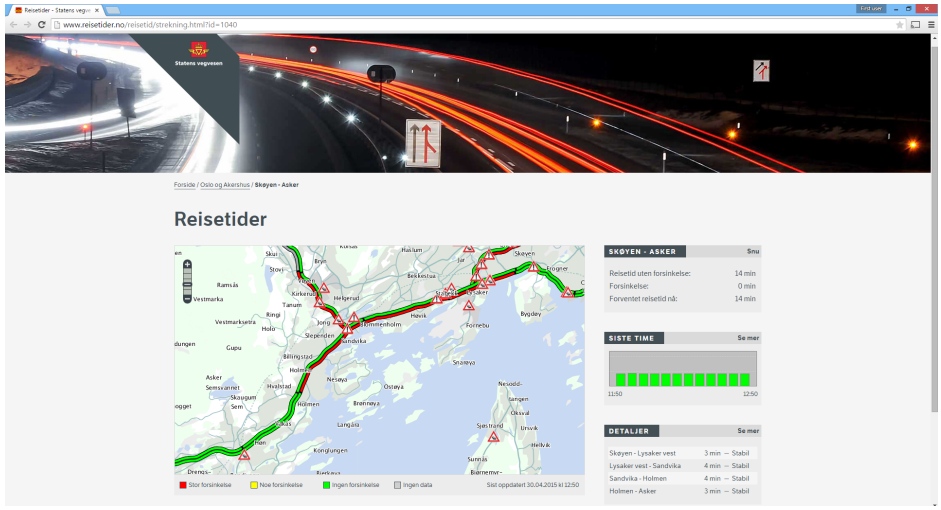

Figure 1.1: Screenshot from `www.reisetider.no`[2]

with the incoming data streams whilst being able to extract useful information from big data sets. Finally, the extracted information needs to be distributed to the end users, such as traffic control centers or website users, requiring good quality of service in terms of availability.

The Norwegian Public Roads Administration's system for providing drivers with travel time information is an example of an intelligent transportation system. The Norwegian Public Roads Administration collects actual travel times from automatic vehicle identification systems on selected roads. This data needs to be processed in order to remove incorrect travel times, for example caused by vehicles stopping in between two detectors. Then, based on the observed data, the system is able to compute expected travel times. This procedure is described in more detail in Section 2.1.5. The travel time information is finally made available to drivers through a website[1], illustrated in Figure 1.1. The roads illustrated on the website are colored in green, yellow or red indicating whether there is no delay, some delay or a large delay on the corresponding road, respectively. The map also displays notifications concerning special circumstances that may affect the traffic, such as a traffic accident or road construction The same type of information, e.g. delays and driving conditions, may also be distributed through electronic road signs as illustrated in Figure 1.2.

[1] `www.reisetider.no`
[2] From `http://www.reisetider.no/reisetid/strekning.html?id=1040`, April 30, 2015
[3] Source: Norwegian Public Roads Administration's archive

Figure 1.2: Example of an electronic road sign[3]

Aside from the challenges involving a highly complex distributed system, what makes intelligent transportation systems interesting is their environmental and economical benefits. Additionally, such systems can potentially increase traffic safety. Commision of the European Communities [2001] stated that intelligent transportation systems have the potential to reduce travel times by up to 20%, whilst network capacity can be increased by $5 - 10\%$. If this is achieved, CO_2 emissions and transportation costs can be significantly reduced. The impact of intelligent transportation systems on traffic safety has been estimated to reduce rear-end collisions by $10 - 15\%$ because of advanced information and control strategies. Furthermore, automatic incident detection systems, making managing emergency situations easier, have increased survival rates when accidents occur. These are all promising properties, but developing systems delivering reliable information such that these results are achieved is not trivial.

In terms of increasing traffic efficiency, providing drivers with expected travel times is a common approach. Travel time is a variable most commuters can relate to. Based on the travel time, it is easy to deduct what the current traffic situation is. This makes it a suitable variable to use in traffic information systems. A survey, reported in Chung et al. [2004], suggests that 78% of drivers are willing to change their departure time or take a different route if they are informed about delays. This indicates that providing commuters with traffic information is a useful tool for efficiently distributing traffic in the road network. Another area

where travel time information is useful is trip planning, which requires predicting what travel times will be in the future. Transportation companies rely on this information to optimize their routes, potentially reducing costs and increasing efficiency.

1.1.2 Estimating or Predicting Traffic Variables

The traffic domain is highly complex as it is affected by a great variety of factors. These include aspects like human behaviour and interaction, how signalized intersections are programmed, and weather conditions. Many of these factors are non-deterministic. Consider the example of cars driving on a road section with a signalized intersection. The fact that the amount of time spent waiting for a green light is different for individual vehicles may lead to different travel times, even though the traffic density is constant. Also, other factors like human interactions and traffic accidents are difficult to predict. The relationships between these factors and how they affect traffic conditions are highly complex, making it difficult to incorporate all of them into a single model. Analytical models, like macroscopic traffic flow theory models, are therefore limited in their capability of making accurate predictions about traffic conditions.

Another approach to predicting traffic conditions is by using data driven approaches. The benefit one can draw from these methods is that they do not focus on using the semantics of the data in order to express relationships between the variables in the data set. Instead, data driven methods attempt to discover patterns in the data by looking at many examples. This makes them suitable for modelling complex domains, such as the traffic domain. However, this black box behaviour limits the possibility of generating a model which is easy for humans to interpret.

Machine learning is an example of a data driven method which estimates the true function that maps inputs to outputs in the underlying domain. In the case of traffic, this can be to find a function that captures the relationship between previous and future travel times.

Researchers are constantly attempting to find ways of improving the prediction accuracy of machine learning models. Russell and Norvig [2010] state that one possible approach is to use ensemble learning. In short, ensemble learning combines predictions from several models to make a single prediction. This technique is described in more detail in Section 2.1.2.

In order to further increase prediction accuracy, one can imagine including more data sources like accidents, incidents, weather data, GPS data from taxis, and bluetooth devices in cars. This may contribute with valuable information that makes data driven methods incorporate new relationships in their models.

Traditionally, machine learning techniques used to predict travel times have

been based on using historical data. Over time, as changes in the underlying data set occur, their models become outdated and their prediction accuracies deteriorate. Online learning methods are able to incorporate the most recent data in their models. This makes them very adaptable, a desirable property for systems deployed in the real-world. Online learning is described in more detail in Section 2.1.3.[4]

A structured literature review investigating in what degree ensemble learning and online learning have been employed in the traffic domain is presented in Øien Lunde and Wolff [2014]. The authors report that various studies investigating ensemble learning and online learning in the traffic domain have been conducted. The results of these studies demonstrate that ensemble learning provides higher prediction accuracy compared to its baseline methods. Additionally, online methods have proven to be better at adapting to changes in the underlying data set. However, the structured literature review did not reveal any studies that compare the proposed techniques on a common data set. This makes it difficult to determine which methods that perform best, and in what situations.

In order to gain knowledge as to how the different techniques compare to each other, this work sets out to conduct experiments that measure the methods' performance on a common data set. The focus in this is work on comparing the methods from an artificial intelligence research perspective. However, the results may also be used to to give guidelines to the Norwegian Public Roads Administration as to which technique(s) may be integrated in their own systems.

1.2 Goals and Research Questions

Goal To find the best ensemble and online learning technique(s), among the ones reported in Øien Lunde and Wolff [2014], for predicting travel times for a given road section.

Research Question 1 Given a set of baseline methods, which ensemble learning technique yields the best prediction accuracy?

Research Question 2 Which online learning technique yields the best prediction accuracy?

[4]Thus far, the text in this section has been repeated from Øien Lunde and Wolff [2014, p. 1-4] with some smaller modifications. The text that follows in the rest of this section is to be considered new and not a part of Øien Lunde and Wolff [2014].

1.3 Research Method

This work employs a comparison study methodology in an attempt to reach the research goal described in Section 1.2. In a comparison study, the performance of one or more methods is assessed through the means of some performance measure. The performance measure is used as a basis for comparison of the methods. However, more importantly, the performance measure can be used to compare the performance of the methods with an existing standard that solves the same problem [Cohen and Howe, 1988]. This way, one can identify whether or not the proposed methods provide any improvements to the existing solutions. In our comparison study, we design several experiments, each suited to produce results that may give answers to one or more of the research questions stated in 1.2. By examining the results of the experiments, and evaluating the performance of the different approaches, we hope to answer these research questions.

1.4 Contributions

- State our contributions here

1.5 Thesis Structure

Chapter 2 gives an introduction to important concepts used throughout this paper and presents the motivation for selecting the methods investigated in this study. Chapter 3 describes the methods that are evaluated in more detail. Chapter 4 describes the experiments conducted in this study and presents the results. An evaluation and discussion of the results are presented in Chapter 5.

Chapter 2

Background Theory and Motivation

This chapter covers relevant background theory and presents the literature review. In Section 2.1, key terminology used throughout this paper is described. Section 2.2 describes how the literature review was conducted. Note that sections 2.1 and 2.2 are repeated from Øien Lunde and Wolff [2014, p. 7-10] and Øien Lunde and Wolff [2014, p. 10-12], respectively. Section 2.3 presents the findings of the literature review, and motivates the use of the techniques investigated in this research.

2.1 Background Theory

In this section, relevant concepts for predicting travel times are introduced. Machine Learning is described in Section 2.1.1. Next, Sections 2.1.2 and 2.1.3 describe two important machine learning concepts referred to in this study, namely ensemble learning and online learning, respectively. Section 2.1.4 covers traffic data collection and describes some important variables used in traffic modeling. The solution for travel time estimation in use at the Norwegian Public Roads Administration as of 2015 is covered in Section 2.1.5.

2.1.1 Machine Learning

Machine learning is a term used in computer science covering methods for learning computers different tasks by investigating data. Mitchell [1997] defines machine learning as:

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

Given many data points of the form $(\mathbf{x}, y)$, where $\mathbf{x}$ is a input vector of features and y is the corresponding output value, the goal of machine learning is to estimate the true function $F(\mathbf{x})$ that maps input vectors $\mathbf{x}$ to output values y. This estimated function $M(\mathbf{x})$ is called a model of the true function. Given a new input vector $\mathbf{x}_{new}$, whose output value is not known, this model can be used to predict the output $y_{\mathbf{x}_{new}}$.

2.1.2 Ensemble Learning

Basic machine learning methods, like Artificial Neural Network (ANN), k-Nearest Neighbors (k-NN) and Support Vector Machine (SVM), use one single model to make predictions. Single model learners have proven to make accurate predictions, but even higher accuracy can still be achieved. The idea behind ensemble learning is to combine several hypotheses in to one prediction. Consider an ensemble of five classification hypotheses, where majority voting is used. For the ensemble's hypothesis to be incorrect, at least three of those five hypotheses have to be incorrect [Russell and Norvig, 2010]. This is the motivation behind the approach. An ensemble of learners that collectively make a prediction is more likely to be correct, especially if the learners in the ensemble have different bias with regards to the training data. By having a low correlation between the errors, it is less likely that all of the learners are mistaken at the same time. The ensemble is able to mask the weakness of each individual learner, thus increasing prediction accuracy.

2.1.3 Online Learning

Online learning is a machine learning approach where the underlying model is trained incrementally on the most recent data. As soon as new data becomes available, the error of the model is computed by comparing its output with the actual values. This way the model can be adjusted so that the prediction error is reduced. By constantly adjusting the model it is able to adapt to changes in the data set that happen over time. In the case of travel time prediction, the model is updated as soon as new travel time information becomes available. This allows the model to adapt to changes in traffic conditions such as new lanes being added to a road section causing a decrease in congestion, or tolls on one road leading to a change in traffic volume.

2.1.4 Traffic Domain

This section introduces the most common variables used to express the state of traffic and the approaches used to measure these variables. The section is based on Haugen and Aakre [2014].

Travel Time

Travel time is defined as the time it takes to drive from point a to point b. It can be expressed as

$$t = t_b - t_a \tag{2.1}$$

where t is travel time, t_a is the point in time where the vehicle passed point a, and t_b is the point in time where the vehicle passed point b.

Traffic Flow

Traffic flow is defined as the number of cars passing through a certain point on the road per unit time. It can be expressed as

$$q = \frac{n}{T} \tag{2.2}$$

where q is traffic flow, n is the number of cars, and T is time. Traffic flow is commonly given in vehicles per hour.

Density

Density is defined as the number of cars per unit length. It can be expressed as

$$k = \frac{n}{l} \tag{2.3}$$

where k is density, n is the number of cars and l is length. Density is commonly given in vehicles per km.

Collecting Traffic Data

There are several approaches to collecting traffic data. This section covers the most commonly used approaches.

Inductive loop detectors are loops of electrical wire integrated in the road. Electrical current runs through the wire, creating a magnetic field. When a car passes through the loop, a controller can register this because the car changes

the magnetic field. Inductive loop detectors can measure several variables like traffic flow and car length. Since inductive loop detectors are installed in pairs, a few meters apart from each other, speed can also be measured.

Piezoelectric wires are another approach to collecting traffic data. They are also integrated in the road, but run across the road. These wires emit an electrical signal when compressed. This way a controller can register when cars run over it. Piezoelectric wires can be used to measure variables such as traffic flow, speed, the number of axles on a car and the weight per axle.

Radars are also used to collect traffic data. They can measure traffic flow, speed, and classify cars by their lengths.

Although their main task is to validate the payment of cars driving on toll roads, AutoPass tags can be used to collect travel time data as well. AutoPass tags are installed in many cars, and sits in the top of the windshield. These tags enable identification of individual vehicles, such that travel times between two measurement points can be registered.

2.1.5 Current Solution at the Norwegian Public Roads Administration

The Norwegian Public Roads Administration (NPRA) are responsible for collecting traffic data on roads in Norway. They use detectors in the road, like loop detectors and piezoelectric cables, to detect passing vehicles. Additionally, many cars in Norway are equipped with electronic devices, called AutoPASS tags, used for automatic toll payments. Measurements from AutoPASS tags are what the NPRA use today to estimate the current travel times on road sections in Norway. The current solution provides commuters with up-to-date travel time information based on travel times collected from the last five minutes. These measurements are used as input to the algorithm described in Wahl and Haugen [2005]. In this algorithm, the individual travel times are placed in groups of 1 minute intervals, e.g. the travel times 25:00 and 25:59 are placed in the 25 minute group. The estimated travel time is calculated based on the interval with the most registered travel times and its neighboring intervals. A weighted average of the travel times of these intervals is the final estimated travel time.

2.2 Structured Literature Review Protocol

This section gives an overview of the structured literature review process employed in this study. This process follows the guidelines reported in Kofod-Petersen [2014]. A more extensive and detailed description of the process can be found in Appendix A.

To find relevant literature, a search string that covers the main aspects of this research is developed. This search string is used to retrieve articles from two search engines: IEEEXplore[1] and Engineering Village[2]. As most of the retrieved literature is not relevant to this study, a screening process to filter out the irrelevant literature is employed. This is done in a top-down approach, first filtering the articles by title, second by abstract, and finally by full text.

The first two filtering steps are used to ensure that the articles included further in the screening process are in the right domain, and that the work exhibits solutions that are relevant to this study. To decide which articles are relevant, inclusion criteria regarding the contents of the titles and abstracts are developed. An example of an inclusion criterion is: "The title indicates that the article predicts, estimates or models road traffic variables, e.g. traffic flow, speed, congestion or travel time". This criterion ensures that articles regarding estimation or prediction of traffic variables are included, as they are of high relevance to this study. Another example of an inclusion criterion is: "The article describes a solution that can easily be extended or adapted to fit our research". By employing this inclusion criterion, articles that present solutions that can be adapted to our problem are included. After evaluating each article based on every criteria for title and abstract, the most irrelevant literature is filtered out.

The last step in the screening process is to assess the quality of the work presented in the different articles. During the quality assessment, questions such as "Is the method/algorithm thoroughly explained?" and "Does the test evidence support the findings presented?" have to be answered to identify the good research. The articles are given a score in the range from 0 to 1 for each quality assessment question. To ensure a certain quality, only articles with a total score above a set threshold is included in the state of the art review.

2.3 Motivation

The structured literature review presented in Øien Lunde and Wolff [2014] reveals several ensemble learning techniques employed in the traffic domain, ranging from simple linear combinations to more complicated non-linear schemes. It also describes several online learning techniques used for making predictions in the traffic domain. This section presents the key findings in the structured literature review, and motivates the methods investigated further in this research.

[1] http://ieeexplore.ieee.org/search/advsearch.jsp?expression-builder
[2] http://www.engineeringvillage.com/search/expert.url?CID=expertsearch

2.3.1 Inclusion criteria

To ensure that there is enough time to thoroughly evaluate the methods, only a small selection of the techniques described in Øien Lunde and Wolff [2014] can be included in the experiments in this study. Therefore a set of inclusion criteria are developed to narrow down the number of methods being tested. These criteria are:

Inclusion Criterion 1 The paper employs an ensemble learning technique or an online learning technique

and

Inclusion Criterion 2 The technique described in the paper is available through an open source implementation or is sufficiently explained in order to be implemented

and

Inclusion Criterion 3 The paper focuses on improving prediction accuracy

Since this research sets out to compare ensemble learning techniques as well as online learning techniques, it is essential that the proposed approach in a paper falls into one of these two categories. This is reflected by Inclusion Criterion 1. Furthermore, as this research has time restrictions, it is an absolute necessity that the methods being tested do not require much implementation, as the main focus in this work is on comparing methods, not spending time implementing them. Inclusion Criterion 2 ensures that only papers presenting methods requiring a minimum of implementation is included. As several papers presented in Øien Lunde and Wolff [2014] focused on other aspects of travel time prediction than prediction accuracy, like computational cost, Inclusion Criterion 3 is present to ensure that the main focus of the proposed method is increasing prediction accuracy.

Of all the papers resulting from the structured literature review in Øien Lunde and Wolff [2014] only those employing ensemble learning or online learning techniques are candidates for the experiments in this study due to Inclusion Criterion 1. Three of these candidate techniques did not satisfy all the criteria. Zhu and Shen [2012] describes a technique that makes use of the notion of traffic regimes. As the authors do not give an accurate definition of different traffic regimes the paper does not satisfy Inclusion Criterion 2, and is therefore not included. van Hinsbergen et al. [2009] is not included as the paper focuses on reducing computational cost, and thus does not satisfy Inclusion Criterion 3. Although Lu [2012] satisfies all the inclusion criteria, it is not included in the experiments. The proposed method's results are compared to a primitive baseline, which makes it

difficult to determine whether the method is good or not, and therefore it is not justifiable to spend time implementing it.

2.3.2 Methods

The motivation for including the methods used in the experiments in this study is explained in the following sections.

Bagging

Sun [2009] investigates whether or not using a multitask learner will improve upon using a simple task learner when predicting traffic flow. Additionally, the authors look at the effect of using an ensemble of multitask learners compared to just a single multitask learner. In contrast to single task learners, which predict one variable at a time, multitask learners make predictions for several variables simultaneously. The motivation for using multitask learners is that they may incorporate more information during training because the different learning tasks share some common properties. The method they use for combining predictions from their ensemble of multitask learners is called bagging, which is described in more detail in Section 3.1.1. The results from the experiments conducted in Sun [2009] illustrate that bagging may improve prediction accuracy compared to using just one model to make predictions.

Boosting

All though not present in any of the papers reported in the structured literature review in Øien Lunde and Wolff [2014], boosting is included in this study. Boosting is one of the most common ensemble learning techniques [Russell and Norvig, 2010] and is hence included due to completeness. Section 3.1.2 explains boosting in more detail.

Lasso

Bagging combines its baseline models simply by calculating an average. This means that the baselines are combined without regards to their performance. Another approach is to assign a weight to each model in the ensemble, representing its contribution to the ensemble's output. The idea behind using a weighted average is that the weights can be calculated based on each models' prediction error, thus allowing the ensemble to adjust for errors in each model. This can help improve the overall performance of the ensemble, e.g. by making the best performing models more prominent.

In Li et al. [2014] three different weighting schemes are explored, namely least squares, ridge regression and lasso. Their results indicate that the weighting schemes increase prediction accuracy compared to the best performing baseline model, but it is difficult to determine which weighting scheme is best. Additionally, they conclude that using a limited number of baseline models generally improves prediction accuracy. The strength of lasso compared to the other weighting schemes is that it can automatically perform model selection (by assigning zero weight to some models), thus outputting a weighted average of only the best performing models. This feature, i.e. being able to automatically select the appropriate baseline models, makes lasso an interesting approach. The weighting schemes tested are not compared to a simple (non-weighted) average or to other ensemble approaches. Thus, although demonstrating the effectiveness of combining predictions with a weighted average, the research does not indicate whether this approach is preferable to any other ensemble approach.

Fuzzy Rule Bases Systems

Stathopoulos et al. [2008] argues that the relationships between the baseline models in an ensemble are not necessarily linear, which is why a Fuzzy Rule Based System (FRBS) capable of representing non-linear relationships between baseline models is proposed in Stathopoulos et al. [2008].

In general a FRBS works by explicitly defining a set of rules that convert inputs to an output. Since these rules are explicitly defined, one is able to incorporate expert knowledge and represent relationships between models that are impossible to represent by linearly combining them. FRBSs are explained in more detail in Section 3.1.4.

The contribution of Stathopoulos et al. [2008] is somewhat similar to Li et al. [2014]. The experimental results demonstrate that the ensemble outperforms the best performing baseline in terms of prediction accuracy, but none of their experiments compare the proposed method to another ensemble approach. Because this approach exhibits some interesting features, more specifically its ability to represent non-linear relationships and to incorporate expert knowledge, it is included in the comparison study.

Extended Kalman Filter

So far, the methods described focus on how to combine predictions from an ensemble of models into one, hopefully more accurate, prediction. The other branch of methods that the structured literature review in Øien Lunde and Wolff [2014] reveals, are methods capable of updating its underlying model each time new observations arrive. These methods are often called online methods or incremental methods because they learn from one observation at a time and continually

update their model during the prediction phase. In contrast, offline methods first learn, then they predict.

Van Lint [2008] presents an online method for predicting travel time using the Extended Kalman Filter (EKF) approach for training the weights of an ANN. Van Lint makes the assumption that the weights in the ANN does a random walk along some high-dimensional path, and that the ANN does a non-linear observation of those weights when making a prediction. The EKF works in an iterative fashion, updating its state vector and variance estimate each time step. This makes it suitable for doing online learning. A more detailed description of the method can be found in Section 3.2.1.

The results reported in Van Lint [2008] demonstrate that the online EKF algorithm does not perform as good as an offline trained model. However, it has the benefit of being online, which makes it more attractive for use in a real time system. Offline methods are limited to the data it is exposed to during training. This makes it hard for offline methods to predict outcomes for situations it has never seen before. Online methods, on the other hand, updates its model continuously as new observations arrive. This way it can adapt its model to the changes in the underlying data that happen over time. This leads to a more adaptive model that does not get outdated if there is a change in the underlying data.

Local Online Kernel Ridge Regression

In Haworth et al. [2014] a novel online learning approach for travel time prediction called Local Online Kernel Ridge Regression (LOKRR) is presented. The approach is based on creating multiple kernels, each corresponding to a specific time interval of the day, using ridge regression to predict travel times based on historical data. One of the most significant drawbacks with using a single kernel to represent the entire data set is that the amount of information it can incorporate is limited due to computational complexity. This makes it difficult to capture for example seasonal variations in the traffic data. LOKRR aims to deal with this limitation. Since each kernel in LOKRR only needs to incorporate data from a certain time of day, more historical data can be used in each kernel. This makes it easier to detect cyclic patterns, such as rush hours, in the data. Additionally, the parameters of each kernel can be tuned individually. This enables each kernel to adapt to its underlying data distribution more effectively, thus increasing prediction accuracy. Another important feature in LOKRR is that it uses a sliding window approach to update the kernels with the most recent data, enabling it to adapt to changes in the data. The advantages of LOKRR are demonstrated in the experiments presented in Haworth et al. [2014]. It outperforms a support vector machine and an artificial neural network for certain prediction horizons in addition to providing better prediction accuracy during non-recurrent congestion

scenarios.

Chapter 3

Models and Architecture

To give a theoretical backdrop for the rest of this study, this chapter explains the methods used in this comparison study in more detail. Section 3.1 explains the ensemble learning techniques bagging, boosting, lasso, and fuzzy rule based system. Section 3.2 explains the online extended Kalman filter, and local online kernel ridge regression.

3.1 Ensemble Learning

This section describes the different ensemble learning approaches being used in this study. All these approaches have in common that they combine the predictions of a set of baseline members. Figure 3.1 illustrates this process where the same data set is available to all K baselines, and K individual predictions are generated from these baselines. The ensemble learning method combines these predictions in some way into a final prediction.

3.1.1 Bagging

Bagging is an ensemble learning technique introduced in Breiman [1996]. All though the technique is actually more concerned with how the models in the ensemble are trained than how the models are combined, it is considered an ensemble learning algorithm. During the bagging training phase, the K different models are trained on a subset of the original training data. The subsets are sampled uniformly with replacement from the training data. When making a prediction from the ensemble, the mean of the predictions from the individual members of the ensemble is used for regression problems, and majority voting is used for classification problems.

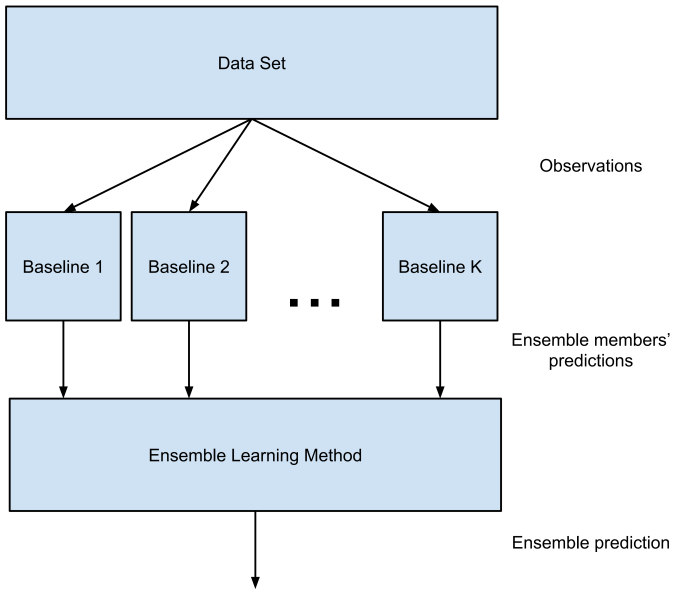

Figure 3.1: Ensemble Learning Process

It is important to note that the members of a bagging ensemble are all of the same type, e.g. all models are artificial neural networks, or all models are k-nearest neighbors. Breiman [1996] explains that bagging may work well for ensembles of machine learning methods that are unstable. However, one has little to gain from using bagging when the ensemble consists of machine learning methods that are stable. A machine learning method is referred to as stable if a change in the training set makes little to no change in the learned model. However, a machine learning method is referred to as unstable if a change in the training set makes large changes in the learned model.

3.1.2 Boosting

Boosting is one of the most common ensemble learning techniques [Russell and Norvig, 2010]. Like bagging, boosting re-samples the training data to construct an ensemble of models. However, the sampling technique in boosting is different from the one used in bagging. In boosting, each training example is given a weight. This weight corresponds to the importance of the example, and is initialized to 1 for all examples, meaning that each example is equally important. The training examples for the first model in the ensemble are drawn from this uniform

distribution.

During training, the first model will correctly classify some examples, while other examples will be misclassified. For regression problems, there will be a resulting prediction error in contrast to the correct-incorrect case for classification. Before constructing the next model in the ensemble, the weights for each example in the training data are updated, increasing the weights of the misclassified examples and reducing the weights of the correctly classified examples. For regression problems, the weights are updated according to how large the prediction error is compared to the largest prediction error among all examples [Drucker, 1997]. This way, the misclassified examples are given higher importance, and have a higher probability of being sampled by the next model in the ensemble. The idea is that the models are getting better at classifying these examples, and that the ensemble is collectively predicting more accurately.

This iterative process continues until K models are constructed, where K is a parameter to the boosting algorithm. When making predictions, a weighted majority vote from the models in the ensemble is used. The weight of each model is proportional to the number of correctly classified examples it has predicted during training, or how low the prediction error is for regression problems.

3.1.3 Lasso Ensemble

Both bagging and boosting include all models in the ensemble when making the final collective prediction. The motivation behind using lasso is to only include the best performing models.

The experimental setup described in Li et al. [2014] is having sensors along a road, registering traffic data at N locations for T time steps. This data forms a matrix $\mathbf{F} \in \mathbb{R}^{N \times T}$ of recent traffic observations. Consider an ensemble of K models, whose task is to make predictions about traffic variables at these N locations for each time step. Each model k makes its prediction $g_n^t(k)$ for location n at time t. This forms a matrix of predictions $\mathbf{G}_k \in \mathbb{R}^{N \times T}$. The model's predictions are put together in a vector $\mathbf{G} = [\mathbf{G}_1, \mathbf{G}_2, \ldots, \mathbf{G}_K]$. The ensemble's collective prediction h_n^t for location n at time t is a weighted average of the member's predictions:

$$h_n^t = \sum_{k=1}^{K} w_k g_n^t(k) \tag{3.1}$$

where w_k is the weight corresponding to model k. The ensemble's predictions for all N locations for all T time steps forms a matrix $\mathbf{H}$, and is given by $\mathbf{H} = \mathbf{GW}$, where $\mathbf{W} = [w_1, w_2, \ldots, w_K]$. Different $\mathbf{W}$ will result in different predictions, and in turn different prediction accuracies. How do we find the optimal set of weights $\hat{\mathbf{W}}$?

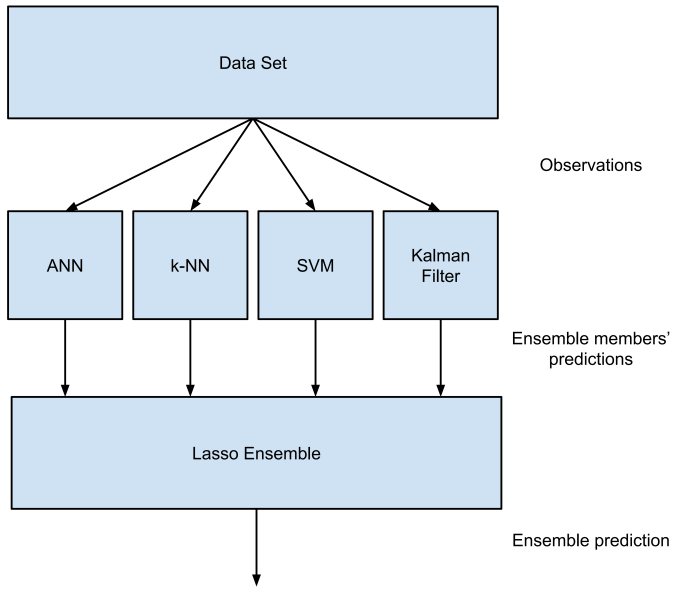

Figure 3.2: Lasso ensemble process

One of the proposed solutions to this question in Li et al. [2014] is lasso ensemble [Tibshirani, 1996; Efron et al., 2004; Hastie et al., 2009]. In this approach the optimal set of weights is given by solving the following equation:

$$\hat{\mathbf{W}} = \arg\min_{\mathbf{W}} \|\mathbf{F} - \mathbf{G}\mathbf{W}\|_2^2 + \lambda\|\mathbf{W}\|_1 \tag{3.2}$$

where $\|\mathbf{F} - \mathbf{G}\mathbf{W}\|_2^2$ is the l_2 norm of the least squares solution, $\|\mathbf{W}\|_1$ is the l_1 norm of the weights, and λ is a regularization term controlling the importance of the l_1 norm of the weights. Li et al. [2014] explains that having this l_1 regularization of the weights leads to a sparse solution, corresponding to many weights being zero. This means that only some of the models are included when making the final prediction. This is advantageous as the lasso ensemble technique selects the best performing models in the ensemble, and accumulates their prediction into the ensemble's final prediction. Li et al. [2014] explains that having a broad range of models in combination with the lasso ensemble approach produces good results as large parts of the parameter space is thus covered, and the best performing of them is included in the final prediction.

The baseline models used in this work's lasso ensemble are ANN, k-NN, SVM and Kalman filter. These four methods are chosen as baselines as they were

IF predicted flow of model **A** is LOW ***AND*** predicted flow of model **B** is LOW, ***THEN*** flow at time $t+1$ is LOW
IF predicted flow of model **A** is LOW ***AND*** predicted flow of model **B** is ***NOT*** LOW, ***THEN*** flow at time $t+1$ is LOW
IF predicted flow of model **A** is MEDIUM ***AND*** predicted flow of model **B** is MEDIUM, ***THEN*** flow at time $t+1$ is MEDIUM
IF predicted flow of model **A** is MEDIUM ***AND*** predicted flow of model **B** is ***NOT*** MEDIUM, ***THEN*** flow at time $t+1$ is MEDIUM
IF predicted flow of model **A** is HIGH ***AND*** predicted flow of model **B** is HIGH, ***THEN*** flow at time $t+1$ is HIGH
IF predicted flow of model **A** is HIGH ***AND*** predicted flow of model **B** is ***NOT*** HIGH, ***THEN*** flow at time $t+1$ is HIGH

Table 3.1: Rule base used in Stathopoulos et al. [2008].

reported in Sjåfjell et al. [2013] as commonly used methods in the traffic literature. Figure 3.2 demonstrates the architecture of the lasso ensemble.

3.1.4 Fuzzy Rule Based System

A FRBS is a machine learning approach where a set of rules define its behaviour. These rules can either be manually created, thus enabling the ability to incorporate expert knowledge, or automatically created based on a data set. FRBSs consist of a fuzzyfication stage, an inference system, and a defuzzyfication stage.

The fuzzyfication stage assigns input values to fuzzy sets. Fuzzy sets are sets whose elements have degrees of membership. What degree of membership an input is given to a fuzzy set is defined by a membership function. Basically what a membership function does is that it returns a probability that some value is a member of a fuzzy set. Imagine a system predicting people's weight based on their height. Two possible fuzzy sets describing a persons height could be *short* and *tall*. The height 185 cm can for example be assigned the degrees 0.2 and 0.8 to *short* and *tall*, respectively.

The inference system is used to generate outputs based on the fuzzyfied input. It consists of a number of if-then rules, referred to as the rule base, that define what to do when some condition is satisfied. The rule base basically constitutes a complete mapping from all possible inputs to outputs. In the example FRBS above, a rule could be "If a person is *tall* then that person is *heavy*". In contrast to typical black-box approaches such as artificial neural networks, the behaviour of FRBSs can be easily interpreted because their behaviour is explicitly described in the rule base.

The responsibility of the defuzzification stage is to convert the values rep-

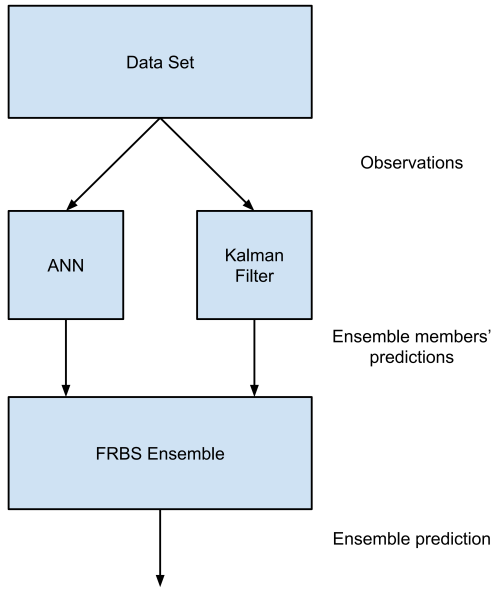

Figure 3.3: Fuzzy Rule Based System Ensemble Process

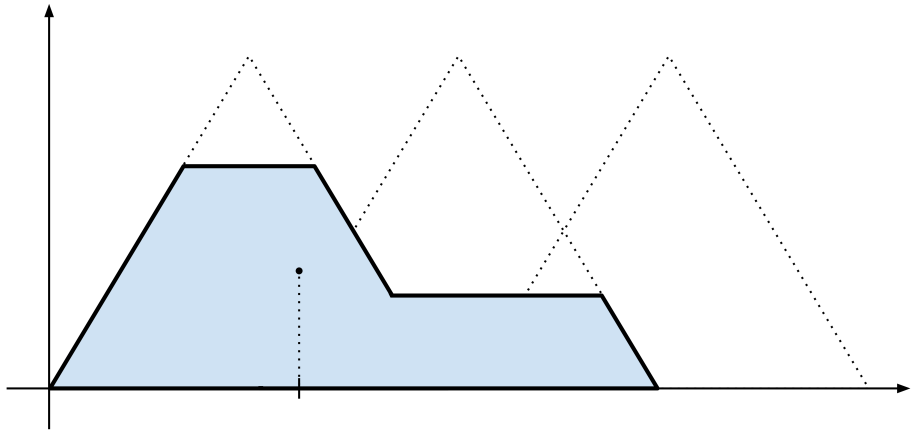

Figure 3.4: Center of gravity defuzzification rule

resenting membership degrees returned by the inference system into one single output. For example converting membership degrees of *heavy* and *light* into kilograms.

Stathopoulos et al. [2008] use a FRBS to combine forecasts from two different models predicting traffic flow: ANN and Kalman filter. The same approach is followed in this work, and the structure of the FRBS ensemble is illustrated in Figure 3.3. The traffic flows are mapped to the fuzzy sets *low*, *medium* and *high*. The FRBS consists of two different rule bases, each preferring one of the models over the other as seen in Table 3.1. When predicting traffic flow at time $t + 1$, the rule base giving priority to the best performing model at time t is selected. The output from the rule base is then converted back to a traffic flow value using the center of gravity algorithm [Takagi and Sugeno, 1985]. The center of gravity defuzzification rule works by first cutting the triangular membership functions at the computed degree of membership. This forms a trapezoid for each function. Next, these trapezoids are combined into one polygon. Next, the center of gravity of this polygon is computed, and the first component of the center of gravity point is used as the defuzzified value. An illustration of the center of gravity defuzzification rule is presented in 3.4.

3.2 Online Learning

This section describes the two online learning approaches used in this study, namely online EKF and LOKRR, which is described in Section 3.2.1 and Section 3.2.2, respectively.

3.2.1 Online Extended Kalman Filter

To explain the online EKF, this section starts by briefly explaining the Kalman filter and its extension, EKF. Next, this section explains how the EKF can be used for training the weights of an ANN, and how the online-delayed EKF and censored EKF conduct this training in an incremental fashion.

Kalman Filter

The Kalman filter is an approach for making estimations about some state vector $\mathbf{x_t}$ at time t from noisy observations $[\mathbf{z_0}, \mathbf{z_1}, \ldots, \mathbf{z_{t-1}}, \mathbf{z_t}]$ from time $t = 0$ to time t. It is assumed that there is a linear relationship between the state vector $\mathbf{x_t}$ at time t and the state vector $\mathbf{x_{t+1}}$ at time $t+1$. Additionally, it is assumed that the process evolves with some uncertainty and that this uncertainty can be modelled as a term following a zero mean Gaussian distribution. These assumptions yield

the basis for the transition model, describing the relationship between successive states in the system, which can be described by the following equation:

$$\mathbf{x_{t+1}} = \mathbf{F}\mathbf{x_t} + \mathbf{B}\mathbf{u_t} + \epsilon_{process} \tag{3.3}$$

where, $\mathbf{x_{t+1}}$ is the state vector at time $t+1$, $\mathbf{x_t}$ is the state vector at time t, $\mathbf{F}$ is a matrix that represents the relationship between the variables in the state vector at time t and $t+1$, $\mathbf{B}$ is a matrix that represents the effect that the input $\mathbf{u_t}$, given to the system at time t, has on the state vector. $\epsilon_{process}$ is the error term following a Gaussian distribution that represents the uncertainty with which the system evolves. $\epsilon_{process}$ has covariance matrix $\mathbf{Q}$.

It is further assumed in the Kalman filter that the state vector $\mathbf{x_t}$ can not be directly observed. One can only gain insight to the state vector through observations of related variables in observation vector $\mathbf{z_t}$. It is assumed that there is a linear relationship between the unobservable state vector $\mathbf{x_t}$ and the observation vector $\mathbf{z_t}$. It is also assumed that the observations are noisy, which means that there is some uncertainty related to the observations. This uncertainty is assumed to follow a zero mean Gaussian distribution. Putting this together, the observation model, describing the relationship between observations and the state vector, can be expressed by the following equation:

$$\mathbf{z_t} = \mathbf{H}\mathbf{x_t} + \epsilon_{observation} \tag{3.4}$$

where $\mathbf{z_t}$ is the observation at time t, $\mathbf{x_t}$ is the state vector at time t, $\mathbf{H}$ is a matrix that represents the relationship between the observations and the state vector. $\epsilon_{observation}$ represents the uncertainty related to the observation, following a zero mean Gaussian distribution with covariance matrix $\mathbf{R}$.

The Kalman filter approach provides the following equations for making predictions about the system in question:

$$\hat{\mathbf{x}}_{t|t-1} = \mathbf{F}\hat{\mathbf{x}}_{t-1|t-1} + \mathbf{B}\mathbf{u_t} \tag{3.5}$$

$$\mathbf{P_{t|t-1}} = \mathbf{F}\mathbf{P_{t-1|t-1}}\mathbf{F}^T + \mathbf{Q} \tag{3.6}$$

where $\hat{\mathbf{x}}_{t|t-1}$ is the prediction of the state vector at time t given observations up to time $t = t-1$, $\hat{\mathbf{x}}_{t-1|t-1}$ is the estimate of the state vector at time $t-1$ given observations up to time $t = t-1$. $\mathbf{P_{t|t-1}}$ is the covariance matrix of $\hat{\mathbf{x}}_{t|t-1}$, representing the uncertainty related to the prediction made, $\mathbf{P_{t-1|t-1}}$ is the covariance matrix representing the uncertainty from the previous iteration.

After making an observation $\mathbf{z_t}$ at time t, the Kalman filter approach provides the following equations for updating the estimation of the state vector and its uncertainty

$$\hat{\mathbf{x}}_{t|t} = \hat{\mathbf{x}}_{t|t-1} + \mathbf{K}(\mathbf{z_t} - \mathbf{H}\hat{\mathbf{x}}_{t|t-1}) \tag{3.7}$$

$$\mathbf{P_{t|t}} = \mathbf{P_{t|t-1}} - \mathbf{KHP_{t|t-1}} \tag{3.8}$$

where $\hat{\mathbf{x}}_{t|t}$ is the updated estimate for the state vector, $\hat{\mathbf{x}}_{t|t-1}$ is the predicted state vector, $\mathbf{K}$ is the Kalman gain matrix which is given by the following expression:

$$\mathbf{K} = \mathbf{P_{t|t-1}}\mathbf{H}^T(\mathbf{HP_{t|t-1}}\mathbf{H}^T + \mathbf{R})^{-1} \tag{3.9}$$

The Kalman filter works in an iterative fashion, starting with an initial state vector $\mathbf{x_{t=0}}$ and a corresponding covariance matrix $\mathbf{P}_{t=0}$ at time $t = 0$. Then it makes a prediction for the state vector $\hat{\mathbf{x}}_{t=1|t=0}$ and its uncertainty $\mathbf{P_{t=1|t=0}}$ for time $t = 1$. Next, it makes an observation $\mathbf{z}_{t=1}$ at time $t = 1$ providing the basis for updating the state vector $\hat{\mathbf{x}}_{t=1|t=1}$ and its uncertainty $\mathbf{P_{t=1|t=1}}$ for time $t = 1$. Now the process starts over again, making a prediction for the next time step, doing an observation, updating the state vector etc.

Extended Kalman Filter

The Kalman filter approach described above assumes that the relationship between successive states, and the relationship between observations and state vector, is linear. However, this may not be sufficient to model all systems and processes. The EKF generalizes the ideas from the Kalman filter and does not assume that these relationships are linear. Allowing non-linear relationships in the model yields the following transition and observation model equations:

$$\mathbf{x_{t+1}} = f(\mathbf{x_t}, \mathbf{u_t}) + \epsilon_{process} \tag{3.10}$$

$$\mathbf{z_t} = g(\mathbf{x_t}) + \epsilon_{observation} \tag{3.11}$$

where f and g are arbitrary continuous, differentiable functions.

Since the transition and observation models are changed, the prediction and update equations have to be updated to reflect this non-linear relationship. The prediction equations for the EKF can be expressed as:

$$\mathbf{x_{t|t-1}} = f(\mathbf{x_{t-1|t-1}}, \mathbf{u_t}) \tag{3.12}$$

$$\mathbf{P_{t|t-1}} = \mathbf{J_f P_{t-1|t-1} J_f}^T + \mathbf{Q} \tag{3.13}$$

where $\mathbf{J_f}$ is the Jacobian of function f.
The update equations for the EKF can be expressed as:

$$\hat{\mathbf{x}}_{t|t} = \hat{\mathbf{x}}_{t|t-1} + \mathbf{K}(\mathbf{z_t} - g(\hat{\mathbf{x}}_{t|t-1})) \tag{3.14}$$

$$\mathbf{P_{t|t}} = \mathbf{P_{t|t-1}} - \mathbf{K J_g P_{t|t-1}} \tag{3.15}$$

where the Kalman gain matrix is expressed as:

$$\mathbf{K} = \mathbf{P J_g}^T (\mathbf{J_g P J_g}^T + \mathbf{R})^{-1} \tag{3.16}$$

and $\mathbf{J_g}$ is the Jacobian of function g.

Artificial Neural Network Weight Training

The above explanations of the Kalman filter and the EKF are general and not written to be specific to any problem, process or system. This section explains how the EKF can be applied for training the weights in an ANN.

An ANN consists of several layers of neurons. Each neuron outputs a signal which is constructed by applying a function, often a non-linear sigmoid shaped function, to a weighted sum of its input signals. A common structure of an ANN is having an input layer, one or more hidden layers, and an output layer. The neurons are connected to each other with weighted edges, propagating signals from inputs to outputs. During training, the network is exposed to a set of input-output pairs. The network propagates the input signals through the network, and the output signal from the output neuron is compared to the correct output in the input-output pair. Then the weights are adjusted to make the difference between the network's output and the correct output as small as possible across all training examples.

The outputs of the network can be seen as a non-linear function of the input signals to the network and the weights in the network. The idea presented in Van Lint [2008] is that one can consider the weights in an artificial neural network during training as a state vector, where the artificial neural network does a non-linear observation of the weights through the network's outputs. Given this state-space definition, one can apply the EKF equations explained in Section 3.2.1 to update the weights in the artificial neural network in an incremental fashion.

$$\begin{cases} \theta_t = \theta_{t-1} + r_t, r_t \sim \mathcal{N}(0, R_t) \\ y_t = G(\mathbf{x_t}, \theta_t) \end{cases} \tag{3.17}$$

Online-Delayed Extended Kalman Filter

A common setting for prediction tasks is that presented with an input $\mathbf{x}_t$ at time t, make a prediction $\hat{\mathbf{y}}_t$, observe the correct output $\mathbf{y}_t$ and update the model given the prediction error $\epsilon = \hat{\mathbf{y}}_t - \mathbf{y}_t$. This approach is applicable for many problems. However, it assumes that at the time that the prediction $\hat{\mathbf{y}}_t$ is made, the correct output $\mathbf{y}_t$ is available. For many problems this is true, however, this is not the case for travel time prediction. Imagine predicting travel times for a road section stretching from point A to point B. When a vehicle arrives at point A at time t_A, one wants to make a prediction $\hat{y}_{AB}$ for how long it will take for that vehicle to arrive at point B, given the current traffic flow, traffic density and vehicle speeds $\mathbf{x_{t_A}}$. Making the prediction is straight forward; the input values $\mathbf{x_{t_A}}$ are fed into the model, and a prediction $\hat{y}_{AB}$ is given as output. However, the model cannot correct for the prediction error $\epsilon = \hat{y}_{AB} - y_{AB}$ until the actual travel time y_{AB} spent driving from point A to point B is available. This travel time is available at time $t = t_A + y_{AB}$. This is the approach taken in online-delayed EKF. A prediction is made when a vehicle enters the road section, and the model is updated when the actual travel time for that vehicle is available. As the name indicates, this approach yields a model that is delayed. The weights of the artificial neural network is not updated until the realized travel time is available, and this makes the model lagging behind the traffic situation.

Censored Extended Kalman Filter

To reduce the delay of when the model is updated, Van Lint [2008] proposes an approach that makes use of a *censored* observation of the realized travel time. Imagine a vehicle when it has just past point B, lets call this point in time p. This vehicle entered the road section at some time m and spent d_m seconds driving from point A to point B, where $m = p - d_m$. Consider all vehicles starting at some time k where $m < k < p$ for which no realized travel time is available yet. Van Lint [2008] suggests that a *censored* observation of the travel time d_k

is $d_k^* = p - k$. This censored observation of the actual travel time can be used as a lower bound estimate for the actual travel time during training, and the model can be updated at an earlier time p instead of at time $k + d_k$ which is at a later point in time. This may lead to a model that reflects the traffic situation better than online-delayed EKF. This effect is best seen during the time when congestion is building up. In online-delayed EKF, as the travel times increases, it takes longer and longer time until the model is update. However, in censored EKF, the censored observations for these travel times provides a better and better estimate of the actual travel time and the model is updated incrementally as more and more censored observations becomes available. This leads to a model that reflects the traffic situation better, and may lead to higher prediction accuracy.

3.2.2 Local Online Kernel Ridge Regression

In Haworth et al. [2014] a novel approach for travel time prediction is introduced, namely LOKRR. The approach is based on using kernel ridge regression to generate a prediction for a data point as follows:

$$g(\mathbf{x}) = \mathbf{y}'(\mathbf{K} + \lambda \mathbf{I_n})^{-1}\mathbf{k} \tag{3.18}$$

In Equation 3.18, $\mathbf{x}$ is a new observation, $\mathbf{K}$ is a kernelized version of matrix $\mathbf{X}$ (a $n \times p$ matrix with n observations of p variables), $\mathbf{y}$ is a vector of size n where element i corresponds to the true value for observation i in $\mathbf{X}$, λ is the regularization parameter, and $\mathbf{k}$ is a vector of size n where element i is the value of a kernel function between the new observation $\mathbf{x}$ and row i in $\mathbf{X}$.

The kernel function used in Haworth et al. [2014] is the Gaussian radial basis function kernel:

$$\mathbf{G}(\mathbf{x}, \mathbf{z}) = \exp\left(-\frac{\|\mathbf{x} - \mathbf{z}\|^2}{2\sigma^2}\right) \tag{3.19}$$

$\mathbf{K}$ is generated by calculating the value of Equation 3.19 between every pair of observations in $\mathbf{X}$. $\|\mathbf{x} - \mathbf{z}\|^2$ is the squared Euclidean distance between vectors $\mathbf{x}$ and $\mathbf{z}$, whilst σ is the kernel bandwidth parameter controlling the smoothness of the function.

The idea in LOKRR is to divide the day into 5-minute intervals. For each interval a data set is created which contains observations of travel times during that specific interval for the last N days. This data set is what is referred to as the $\mathbf{X}$ matrix in Equation 3.18. For each interval a kernel is created by calculating the $\mathbf{K}$ matrix based on $\mathbf{X}$. By creating a kernel for each interval, LOKRR is better able to capture subtle differences in traffic patterns that are local to specific times of the day. If one single big kernel was used for the entire data set, these local differences would be a lot more difficult to detect.

Haworth et al. argue that recurring events such as rush hour not necessarily happen at exactly the same time every single day, which is why each kernel also contains data from its neighboring intervals. For a kernel responsible for interval t, its $\mathbf{X}$ matrix contains data from interval $t-w$ to interval $t+w$, where w is the window size. This is done to create a buffer for when events can happen, whilst still being detectable by a single kernel.

To generate a travel time prediction for an observation $\mathbf{x}$ at time t, LOKRR uses the kernel responsible for interval i containing time t to calculate Equation 3.18. As soon as the true travel time is observed, all the kernels responsible for an interval in the range $[i-w, i+w]$ update their $\mathbf{X}$ matrix. This means that $\mathbf{K}$ and $\mathbf{K}^{-1}$ also need to be updated. Doing this every time a new travel time is realized is computationally expensive. Therefore, a method that significantly reduces the computation time by computing the inverse of $\mathbf{K}$ at time $t+1$ based on the inverse at time t is used. The method is described in detail in [Haworth et al., 2014, p. 156].

Chapter 4

Experiments and Results

This chapter presents the experiments performed in this study along with their results. Section 4.1 describes the intent of each experiment and how they are conducted. Section 4.2 explains the experimental setup, describing the data set, and model parameters. Section 4.3 describes the environment which the experiments are run in. Section 4.4 presents the results of the experiments.

4.1 Experimental Plan

Section 4.1.1 introcudes the different measures that are used to evaluate the performance of the approaches investigated in the experiments in this study. Next, Section 4.1.2 and Section 4.1.3 describe the two experiments conducted in this study, and puts them in context of the research questions they attempt to answer. For convenience, the relevant research question is repeated for each experiment.

4.1.1 Performance metrics

The performance metrics presented below are chosen because they are extensively used when comparing predicted values to observed values. They are measures representing the performance of a model using a single number, making it easy to compare the models to each other.

RMSE

The Root Mean Squared Error (RMSE) is expressed as:

$$RMSE = \sqrt{\frac{\sum_{t=1}^{t=N} (\hat{y}_t - y_t)^2}{N}}, \tag{4.1}$$

where $\hat{y}_t$ is the predicted value, y_t is the observed value and N is the number of observations.

Although RMSE is a commonly used error metric, it should be noted that one of the most prominent drawbacks with the RMSE comes from the fact that it squares the difference between predicted values and observed values. Large errors are therefore weighted exponentially more than small errors, making the RMSE sensitive to extreme values.

MAE

The Mean Absolute Error (MAE) is expressed as:

$$MAE = \frac{1}{N} \sum_{i=1}^{n} |\hat{y}_t - y_t|, \tag{4.2}$$

where $\hat{y}_t$ is the predicted value, y_t is the observed value and N is the number of observations.

4.1.2 Experiment 1 - Ensemble Learning

Experiment 1 is conducted in an attempt to answer Research Question 1, which relates to the prediction accuracy of each ensemble learning method described in Section 3.1.

Research Question 1 Given a set of baseline methods, which ensemble learning technique yields the best prediction accuracy?

Outline Use predictions from a set of baselines to construct the ensemble learning methods bagging, boosting, lasso, and FRBS. Compare the ensembles with each other with respect to the defined performance metrics.

Baseline Models

The first step in this experiment is the tuning of parameters for the baseline methods. This parameter tuning is performed on a designated part of the data set. A grid of possible parameters for each baseline is constructed, and a separate

model for each combination of parameters is trained on the designated part of the data set, using k-fold cross-validation. The parameters of the best performing model, in terms of RMSE, during the grid search is chosen as the parameters for the model to use in Experiment 1.

Given the parameters of the best performing models from the parameter tuning step, the four baseline models are trained on a separate part of the data set. The trained models are used to make predictions for the rest of the data set.

Ensemble models

The next part of the experiment consists of training the ensemble learning models. The predictions from the baselines are divided into a training and a testing set. FRBS and lasso use the training set to tune their parameters and then make predictions on the testing set. Bagging and boosting differ from FRBS and lasso in terms of how they are constructed. Instead of building a model based on predictions from a set of different baselines, they are concerned with building multiple versions of one single baseline model. The baselines in bagging and boosting are built on the same data set as the baselines used in FRBS and lasso. In addition, a naive ensemble learning approach taking the average of the baselines' predictions is constructed. The simple average of the baselines' predictions is included to give a basis for comparison to the two other ensemble learning approaches.

In order to evaluate the performance of the ensemble methods in terms of prediction accuracy, the error metrics and the plots of the error distributions are inspected. Additionally, hypothesis testing is conducted to see if the potential differences between the methods are significant.

4.1.3 Experiment 2 - Online Learning

Experiment 2 sets out to give an answer to Research Question 2, which is concerned with finding the best performing online learning technique among the ones described in Chapter 3.

Research Question 2 Which online learning technique yields the best prediction accuracy?

Outline Train LOKRR and online-delayed EKF on a common data set and compare the two approaches based on a performance measure.

Section 3.2.1 explains two ways of using the EKF to train an ANN in an online fashion, namely online-delayed EKF and censored EKF. Implementing the censored EKF approach is more time consuming than implementing the online-delayed EKF. Van Lint [2008] reports that using the censored EKF in favor of the

online-delayed EKF only results in a slight improvement of prediction accuracy. The combination of an increase in required implementation time, and only a slight improvement in terms of prediction accuracy leads to the conclusion of only investigating the online-delayed EKF approach in this study.

Parameters for the two online learning approaches LOKRR and online-delayed EKF are tuned using a designated part of the data set. The approach taken to tune the parameters of LOKRR and online-delayed EKF is similar to the one taken to tune the parameters of the baselines in Experiment 1. A grid of possible parameters is constructed, and a model for each combination of parameters is trained. The performance of each model is assessed using RMSE, and the model with best performance is chosen to be used in the next step of the experiment.

Next, the models make predictions, and learn when observations become available, for the rest of the data set. Based on their predictions, the performance metrics for each model are computed and used for comparing LOKRR and online-delayed EKF.

4.2 Experimental Setup

This section contains information about how the experiments were set up. Section 4.2.1 presents the data set used, whilst Section 4.2.2 and Section 4.2.3 describe the setup of parameters and training data for the baseline and ensemble methods, respectively.

4.2.1 Data description

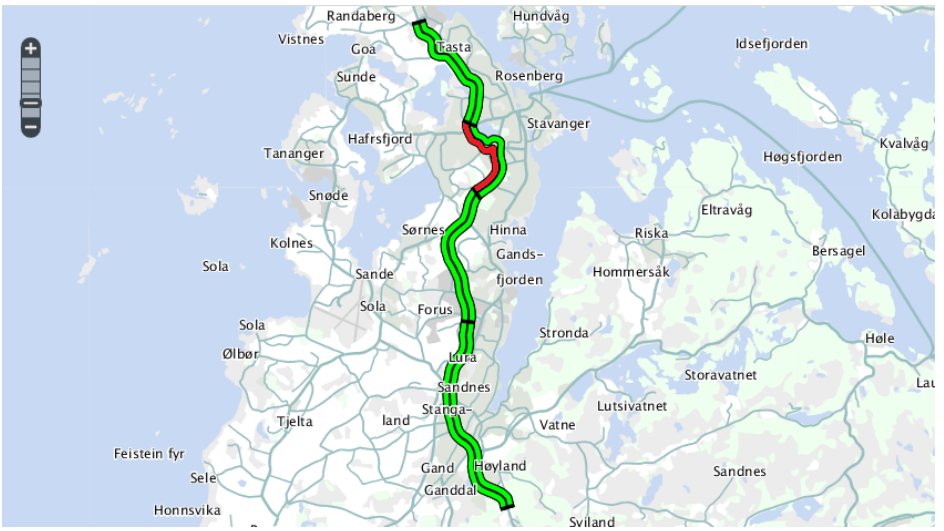

Figure 4.1: Map showing the route where the data is collected from.[1]

The data used in the experiments in this study is collected by the NPRA from highway E39 between Dusavik and Bogafjell in Rogaland, Norway. The route is illustrated in green in Figure 4.1. There are in total five measurement points along this route, illustrated as black lines across the road in Figure 4.1. Data from two consecutive points, Tjensvoll and Auglendshøyden, along this route is chosen to be used in the experiments in this study. The stretch is 4.6 km long, and the data is collected in the southbound direction between January 29, 2015 and March 31, 2015. The road section is highlighted with red in Figure 4.1.

The data set consists of measurements of three variables at the time a vehicle enters the road section: mean travel time the last five minutes, traffic flow at the entry point the last five minutes, and the realized travel time for the vehicle. The data is collected by registering IDs from AutoPASS tags in the vehicles. Travel times are derived by computing the time difference between the registration of the same AutoPASS tag identification number at the two points. The mean travel times are trivial to compute once the travel times are registered. Traffic flow is derived from counting the number of vehicles passing the first registration point. The data set consists of 247074 observations. The first ten rows of the data set

[1]Taken from `http://www.reisetider.no/reisetid/omrade.html?omrade=4` on April 19, 2015

are shown in Table 4.1.

Two data sets are constructed, one containing the original data as described in the previous paragraph, and another one where outliers are removed. The data set without outliers is created to increase the baselines' prediction accuracy for common traffic situations. The outliers may represent commuters stopping along the road section to fill gas or to run other errands. All rows in the data set having a travel time outside the range $[\mu - 3\sigma^2, \mu + 3\sigma^2]$ are removed, where μ is the mean of the travel times in the data set, and σ^2 is the standard deviation of the travel times in the data set.

A plot of individual and five minute mean travel times for January 29, 2015 can be seen in Figure 4.2 and Figure 4.3, respectively. A plot of the traffic flow for the same day is displayed in Figure 4.4. Note that these plots are generated using the original data set, where outliers have not been removed.

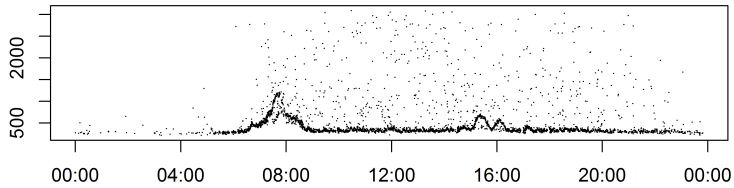

Figure 4.2: Plot of travel times from January 29, 2015

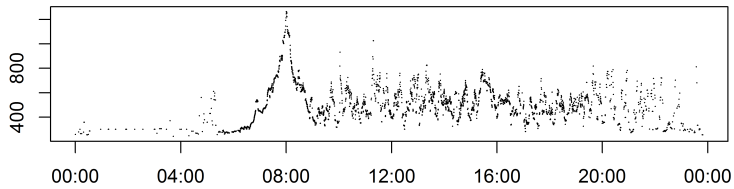

Figure 4.3: Plot of mean travel times from January 29, 2015

Date and time	Mean travel time	Traffic Flow	Travel Time
2015-01-29 00:01:25	298.67	4	242
2015-01-29 00:05:14	690	6	1111
2015-01-29 00:07:05	386.67	6	243
2015-01-29 00:11:55	243	4	241
2015-01-29 00:27:43	1111	2	639
2015-01-29 00:28:25	1111	3	223
2015-01-29 00:35:34	223	3	292
2015-01-29 00:37:27	300	3	249
2015-01-29 00:39:47	639	6	335
2015-01-29 00:40:11	639	8	302

Table 4.1: Example rows from the data set

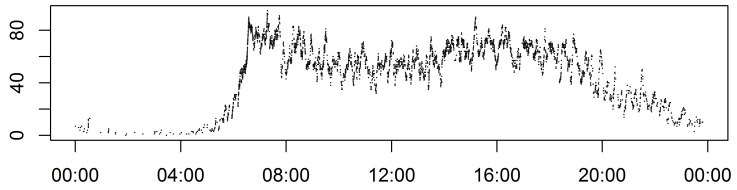

Figure 4.4: Plot of traffic flow from January 29, 2015

4.2.2 Baselines

This section contains descriptions of each baseline method used in Experiment 1. Information about the process applied to every baseline is given in Overview, whilst specific details about each baseline is given in their respective sections.

Overview

The implementation of the baseline methods used in Experiment 1 is done in R [R Core Team, 2014]. Parameter tuning and training for SVM, k-NN and ANN is done through the library `caret` [Kuhn et al., 2015], which is a library meant to make the process of constructing predictive models easier in R. In this

work, `caret` is used as a layer between the authors code in R and the respective implementations of the baseline methods. `caret` provides structures for setting the grid for which parameters are to be searched among, and returns the model with parameters resulting the best performance in terms of RMSE. Tuning and training of the Kalman filter is done with the R library `dlm` [Petris, 2010].

The baselines use the data set were outliers have been removed. In order to find the best set of parameters for each baseline, 10-fold cross-validation on data from January 29, 2015 to February 1, 2015 is used. Each baseline is then trained with the best set of parameters on data from February 5, 2015 to February 25, 2015. The resulting models are then used to generate predictions from Febrary 26, 2015 to March 31, 2015.

Support Vector Machine

The SVM implementation used in this work is from the R library `kernlab` [Karatzoglou et al., 2004]. When using SVMs there is a choice among several kernel functions to use. To determine which kernel to use in the SVM of Experiment 1, a preliminary experiment comparing the RMSE of SVM models using a linear, polynomial and radial basis function kernel functions is conducted. The training data for these models are as described in Overview, and the testing data, for which the RMSE is computed, is from February 2, 2015 to February 4, 2015. For each kernel, a grid of parameters is searched in order to find the best parameters. The SVM with a radial basis function kernel function resulted in the lowest RMSE, and was therefore selected to be used in Experiment 1. Among the parameter values tested in the grid search for the radial basis function SVM models, the parameters $\sigma = 4.1451371$ and $C = 2^{-1}$ produced the best performing model, and is the one used in Experiment 1. For a more detailed description of the parameter search for SVM, please see Appendix B.1.

k-Nearest Neighbors

The k-NN implementation used in this work is from the R library `kknn` [Schliep and Hechenbichler, 2014]. The parameters that are possible to tune in `kknn` are k, distance measure and kernel. The k parameter controls how many neighbors to extract from the instances present in the data set when making predictions. The distance measure parameter controls how the distance between two points in the data set is computed. The kernel parameter controls how to weight the values of the k neighbors based on their distance. In order to find a good set of parameters for the weighted k-NN algorithm a grid search was performed. Please see Appendix B.1 for a more comprehensive description. Based on the grid search, the following parameters lead to the lowest RMSE: $k = 50$, distance measure = 1 (Euclidean) and kernel = Rank.

Artificial Neural Network

The ANN implementation used in Experiment 1 is from the R library `nnet` [Venables and Ripley, 2002]. The `nnet` library provides functions for creating and training feed forward ANNs with a single hidden layer. The parameters being tuned for the ANN is the number of hidden nodes in the network and the weight decay parameter. The parameters producing the lowest RMSE is: number of hidden nodes = 16 and decay= 1×10^{-4}. Please see Appendix B.1 for more a elaborate description of the parameters, and a table presenting the results for each combination of parameters.

Kalman Filter

`dlm` is a R library providing functions for defining dynamic linear models of various types, which is used to perform the Kalman filter predictions in Experiment 1. The actual travel times of the data set forms a time series, and it is assumed that this time series of travel times can be modelled using a first order linear model with the following state space formulation:

$$\begin{cases} y_t = \theta_t + v_t, v_t \sim \mathcal{N}(0, V_t) \\ \theta_t = \theta_{t-1} + w_t, w_t \sim \mathcal{N}(0, W_t) \end{cases} \tag{4.3}$$

where y_t is the observed travel time at time t, θ_t is the *actual* travel time at time t, which is assumed to be unobservable, v_t is the observation noise and w_t is the process noise. Both v_t and w_t are assumed to follow a zero-mean Gaussian distribution with covariance matrices V_t and W_t, respectively.

The parameters that are tuned for this model are the covariance matrices V_t and W_t. A detailed description of the parameter tuning process can be found in Appendix B.1. Since y_t and θ_t are univariate, matrices V_t and W_t are 1×1 matrices, and only one value per covariance matrix is tuned. V_t and W_t are set to 47939 and 122, respectively. In order to give a complete definition of the dynamic linear model, an initial estimate of $y_{t=0}$ and its variance $\sigma^2_{t=0}$ has to be provided. This is set to be the mean and standard deviation of the training observations, which are 242 and 255, respectively.

4.2.3 Ensemble Learning Methods

The setup of each ensemble method used in Experiment 1 is presented in this section. The general procedure is described in Overview, whilst specific details concerning the setup of each ensemble method can be found in their respective sections.

Overview

Bagging and boosting train multiple SVMs on the same data set as the baselines described in Section 4.2.2. Data from February 5, 2015 to Febrary 25, 2015 and generate predictions on data from February 26, 2015 to March 31, 2015. Due to the computational complexity of training SVMs, no parameter tuning is conducted for bagging and boosting.

Lasso and FRBS use the predictions of the baselines described in Section 4.2.2 as input. Lasso is tuned and trained on data from February 26, 2015 to March 18, 2015 and uses data from March 19, 2015 to March 31, 2015 to generate predictions. Due to the computational complexity of FRBS, only one week of data from February 26, 2015 to March 4, 2015 is used to search for the best set of parameters. As the rules in the FRBS are manually created, no training phase is necessary. Predictions are generated from March 5, 2015 to March 31, 2015.

Bagging

The bagging method is implemented in R. There are two parameters that must be specified when doing bagging of a baseline learner: the number of learners K to use in the ensemble, and the number of samples N to do for each learner. In Experiment 1, K is set to 25, and N is set to the number of training examples in the training set. A SVM model with the parameters of the best performing model from the parameter tuning step for SVM reported in Section 4.2.2 is used, i.e. a SVM model with a Gaussian radial basis function as kernel, $\sigma = 4.1451371$ and $C = 2^{-1}$.

Boosting

In order to test the effects of boosting, the AdaBoost algorithm is used. An implementation of the AdaBoost.r2 algorithm [Drucker, 1997], which is a version of the AdaBoost algorithm [Freund and Schapire, 1997] for regression problems, is found in the Python library `scikit-learn` [Pedregosa et al., 2011]. Of the baseline methods used in this work, the only one that is supported in `scikit-learn` is SVM, and is therefore the baseline used for the boosting ensemble in Experiment 1. A SVM model with the same parameters as the one reported in Section 4.2.3 is used for the boosting approach. The only parameter provided to the AdaBoost algorithm is the number of learners K to use in the ensemble, which is set to 25 in Experiment 1.

Lasso Ensemble

The implementation of the lasso method is taken from the R library `elasticnet` Zou and Hastie [2012]. The `elasticnet` library provides functions for for defining

IF ANN is *low* and KF is *low* **THEN** output is *low*
IF ANN is *low* and KF is *not low* **THEN** output is *low*
IF ANN is *medium* and KF is *medium* **THEN** output is *medium*
IF ANN is *medium* and KF is *not medium* **THEN** output is *medium*
IF ANN is *high* and KF is *high* **THEN** output is *high*
IF ANN is *high* and KF is *not high* **THEN** output is *high*

Table 4.2: Rule base preferring ANN over Kalman filter (KF)

elastic nets, of which the lasso method is a special case. Recall from Equation 3.2 in Section 3.1.3 that the optimal weights in lasso is given by solving the following equation:

$$\hat{\mathbf{W}} = \arg\min_{\mathbf{W}} \|\mathbf{F} - \mathbf{G}\mathbf{W}\|_2^2 + \lambda\|\mathbf{W}\|_1$$

where λ is a parameter of the Lasso method. `caret` provides functionality for optimizing this λ parameter through a grid search, where RMSE is used to assess the performance of a given λ value. A λ value of 1 resulted in the lowest RMSE, and is the one used in Experiment 1. More details concerning the parameter tuning of lasso can be found in Appendix B.2.

Fuzzy Rule Based System

The R library `frbs` [Riza et al., 2015] is used to generate a FRBS based on Stathopoulos et al. [2008]. The input to the FRBS is a data set containing predictions from the ANN and the Kalman filter described above. As in Stathopoulos et al. [2008], two different rule bases, each preferring one of the two baselines, are used. The rule base preferring ANN is displayed in Table 4.2. The rule base preferring Kalman filter is similar, except that it gives priority to the Kalman filter's predictions. The travel time predictions from the baselines are mapped with triangular membership functions to three fuzzy sets; low, medium, and high. Data from February 26, 2015 to March 4, 2015 is used to optimize the membership function parameters. The final parameters can be found in Table 4.3 and Table 4.4. Please refer to Appendix B.2 for more details with regards to the FRBS setup.

4.2.4 Online Learning Methods

This section presents the setup of the online methods used in Experiment 2. The general approach is described in Overview, whilst specific details for the individual methods are presented in their respective sections.

	low	medium	high
a	204.5799	292.2150	298.0683
b	287.4319	313.9751	1111.8725
c	302.5835	404.7539	1448.2825

Table 4.3: Membership function parameters for FRBS preferring ANN

	low	medium	high
a	164.3251	240.4815	272.8217
b	328.2455	344.3591	968.0878
c	332.7156	1039.4697	1700.2693

Table 4.4: Membership function parameters for FRBS preferring Kalman filter

Overview

The online learning methods are tuned and trained on two weeks of data from January 29, 2015 to February 11, 2015. The first week is used to build the models with a certain set of parameters, whilst the second week is used to calculate the RMSE of the predictions done during that week. Since the methods are online, the data in the second week is also used to update the models whilst making predictions. The model built on the set of parameters that lead to the lowest RMSE is used to generate predictions from February 12, 2015 to March 31, 2015. The data set used is not preprocessed, i.e. no outliers are removed.

Online-Delayed Extended Kalman Filter

An implementation of the EKF [Cao, a] and using this EKF to train a feed-forward ANN [Cao, b] is found in Matlab [MATLAB, 2014] on the Matlab Central File Exchange[2]. Recall from Equation 3.17 in Section 3.2.1 that the state space definition that is assumed for the EKF can be expressed as follows:

$$\begin{cases} y_t = G(\mathbf{x_t}, \theta_\mathbf{t}) \\ \theta_t = \theta_{t-1} + r_t, r_t \sim \mathcal{N}(0, R_t) \end{cases}$$

where θ_t is the weights in the feed-forward ANN at time t, r_t is the noise with which the weights are assumed to evolve, and G represents the mapping performed by the ANN from inputs x_t to output y_t given weights θ_t.

Recall from Section 3.2.1 that an initial estimate of the state vector $\mathbf{x_{t=0}}$, its covariance matrix $\mathbf{P_{t=0}}$, process noise covariance matrix $\mathbf{Q}$, and observation

[2] `http://www.mathworks.com/matlabcentral/fileexchange/`

noise covariance matrix $\mathbf{R}$ has to be defined in order to run the EKF. For the way EKF is used in the experiments in this study, this means that an initial estimate of the weights θ_{initial} has to be provided. The approach described in Van Lint [2008] is that the weights θ_{initial} are initialized using the Nguyen-Widrow method [Nguyen and Widrow, 1990]. In Van Lint [2008], the weights' covariance matrix $\mathbf{P}_{\text{initial}}$ is initialized to a diagonal matrix with large values, reflecting that it is assumed that the weights are independent, and that there is a large uncertainty connected to the initial guess of the weights. These are also the approaches taken in this work, and the value along the diagonal of θ_{initial} is set to 10000.

As Van Lint [2008] does not describe how the covariance matrices $\mathbf{Q}$ and R is set, a parameter tuning step is performed to find reasonable values for $\mathbf{Q}$ and $\mathbf{R}$, in addition to the number of nodes in the hidden layer in the feed-forward ANN. The covariance matrix $\mathbf{Q}$ is assumed to be a diagonal matrix, once more reflecting that the weights are independent, such that the only value being search for regarding $\mathbf{Q}$ in the parameter tuning step, is the value q along the diagonal of $\mathbf{Q}$: $\mathbf{Q} = q \times \mathbf{I}$. As the output y_t consists of a single value, namely the travel time, the covariance matrix $\mathbf{R}$ of the observation noise is a single element r reflecting the noise related to the observations that the ANN does of the weights.

The parameters that produced the lowest RMSE during the tuning process are: $q = 0.1$, $r = 750$, number of hidden nodes $= 1$. Please see Appendix B.3 for more details on the parameter tuning process.

Local Online Kernel Ridge Regression

No open source implementation of LOKRR was found, so an implementation of LOKRR was developed in Python [Rossum, 1995] from scratch. LOKRR is tuned on the same data set as online-delayed EKF. For each pair of parameter values, the kernels are trained (i.e. the inverse of the regularized kernel matrix was calculated) on the first week of data and predictions are made on the second week of data. The observations in the test set are also used to update the kernels to simulate how LOKRR would normally work. The $\mathbf{X}$ matrix is reset to contain only the training data before every run with new parameters. This way the best pair of parameters can be found for each kernel.

The recommendations for finding possible parameter values for kernel bandwidth σ and regularization constant λ presented in the article are followed. The approach the authors recommend for finding possible λ values is based on Exterkate [2013]. First, the R^2 of an ordinary least squares fit of $\mathbf{y}$ on $\mathbf{X}$ is found. Then λ_0 is determined as $\lambda_0 = 1/\phi_0$, where $\phi_0 = R^2/(1 - R^2)$. The recommended values for λ is $\{1/8\lambda_0, 1/4\lambda_0, 1/2\lambda_0, \lambda_0, 2\lambda_0\}$. The recommended approach for finding possible σ parameters is based on that optimal values of σ lie in the range between the 0.1 and 0.9 quantiles of the pairwise euclidean distance between the points in the kernel [Caputo et al., 2002]. The 0.25, 0.5 and 0.75

quantiles are used as possible values for σ. Due to the amount of data being smaller during the night, the data set used for LOKRR only contains observations from 06:00 to 21:00. Additionally, the window size is set to 1. This is done to reduce the computational complexity. A more detailed description of the LOKRR implementation can be found in Appendix B.3.

4.3 Environment

The experiments described in Section 4.1 are run in two environments, an Ubuntu 14.04 server at Amazon Elastic Computing Cloud (EC2) [3] and a laptop computer. The Ubuntu server is running on a EC2 instance of type m3.xlarge [4], which has a Intel Xeon E5-2670 v2 (Ivy Bridge) processor [5] @ 2.5 GHz and 15 GiB of RAM. The laptop computer is a Samsung ATIV Book 9 Plus NP940X3G running Windows 8.1 with an Intel Core i7-4500U Processor @ 1.8 GHz[6] and 8 GiB of RAM. The Ubuntu server is used to run all experiments, with the exception of the online-delayed EKF which is run in Matlab on the laptop computer.

4.4 Experimental Results

This section presents the results of Experiment 1 and Experiment 2. Section 4.4.1 presents the results from the first experiment, which is concerned with the ensemble learning methods. Section 4.4.2 presents the results from the second experiment, which is concerned with the online learning methods.

4.4.1 Ensemble Learning

In Experiment 1, several models are used to make prediction of travel times for vehicles entering the road section described in Section 4.2.1. Four baselines, SVM, k-NN, ANN and Kalman Filter are used to make predictions for the dates from February 26, 2015 to March 31, 2015. A lasso ensemble approach is used to combine predictions from these four baselines for the dates March 19, 2015 to March 31, 2015. A FRBS Ensemble approach is used to combine predictions from the aforementioned baselines for the dates March 5, 2015 to March 31, 2015. A bagging version of a SVM is used to make predictions for the dates February 26, 2015 to March 31, 2015. A boosted SVM is used to make predictions for the

[3] http://aws.amazon.com/ec2/

[4] http://aws.amazon.com/ec2/instance-types/

[5] http://ark.intel.com/products/75275/Intel-Xeon-Processor-E5-2670-v2-25M-Cache-2_50-GHz

[6] http://ark.intel.com/products/75460/Intel-Core-i7-4500U-Processor-4M-Cache-up-to-3_00-GHz

Model	RMSE	MAE
SVM	233.4601	108.524
k-NN	218.3885	110.4693
ANN	216.5004	107.201
Kalman filter	220.5144	112.6546

Table 4.5: Performance metrics for baselines

Model	RMSE	MAE
Bagging	231.8706	93.88298
Boosting	233.3547	165.567
Lasso	167.0931	95.92011
FRBS	167.705	94.00293
Average	215.6909	98.34549

Table 4.6: Performance metrics for ensemble approaches

dates February 26, 2015 to March 31, 2015. This section presents these results in the form of tables showing RMSE and MAE across all examples in the testing set.

Table 4.5 shows the performance metrics of the four baselines. Table 4.6 displays the performance metrics for bagging, boosting, lasso ensemble, FRBS ensemble and a simple average of the baseline's predictions. The performance metrics in Table 4.5 and Tables 4.6 are computed from predictions and actual travel times from March 19, 2015 to March 31, 2015 as this is the largest span of dates that all models have predictions for. This is done in order to give all methods an equal basis for comparison.

4.4.2 Online Learning

In Experiment 2, two online learning approaches, namely the online-delayed EKF and LOKRR, are used to make predictions for vehicles entering the road section described in Section 4.2.1 for the dates from February 12, 2015 to March 31, 2015. Table 4.7 shows performance metrics values computed from predictions and actual travel times.

Model	RMSE	MAE
Online-delayed EKF	427.1863	230.551
LOKRR	476.1826	233.5318

Table 4.7: Performance metrics for Online-Delayed EKF and LOKRR

Chapter 5

Evaluation and Conclusion

Section 5.1 presents an objective evaluation of the results presented in Section 4.4. Section 5.2 offers a more detailed discussion, presenting possible explanations for the results. Section 5.3 draws the final conclusions based on the evaluation and discussion. Section 5.4 summarizes the contributions from this work, and Section 5.5 suggest possible directions for future work.

5.1 Evaluation

This section evaluates the experimental results presented in Section 4.4. Section 5.1.1 forms the basis for the statistical inference employed in this evaluation, whilst Section 5.1.2 and Section 5.1.3 are concerned with the results from Experiment 1 and Experiment 2, respectively.

5.1.1 Overview

Figures 5.1 to 5.11 display the plots of the error distributions of the predictions from the different models. The error is computed by subtracting the actual travel time from the predicted travel time. Before doing any inference regarding these distributions, it is important to know whether or not they can be assumed to be normally distributed. The distribution of the error affect which hypothesis tests can be employed, and which properties of the distributions that should be compared. Some of the characteristics of the normal distribution is that it is symmetrical about its mean μ; the first derivative is positive for all $x < \mu$ and negative for all $x > \mu$; the mean, median and mode are the same value. By considering the density plots, it can be seen that none of these properties are not

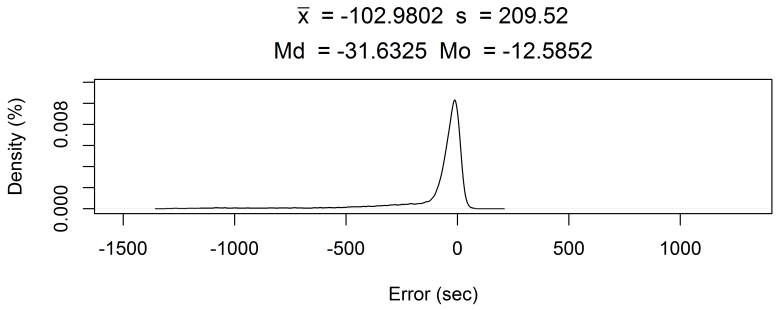

Figure 5.1: Density of errors from SVM Radial

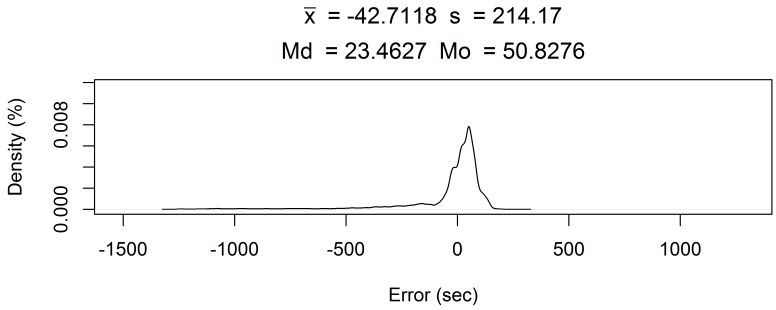

Figure 5.2: Density of errors from k-NN

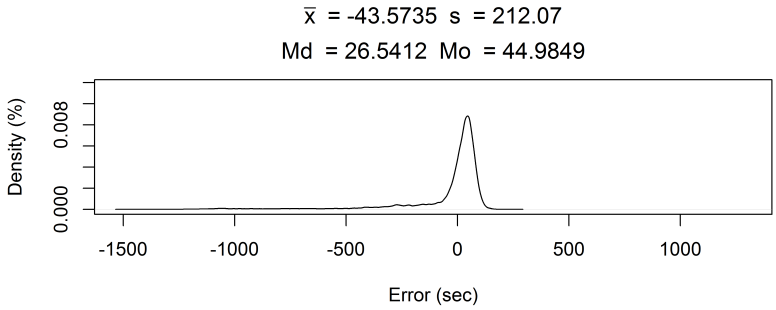

Figure 5.3: Density of errors from ANN

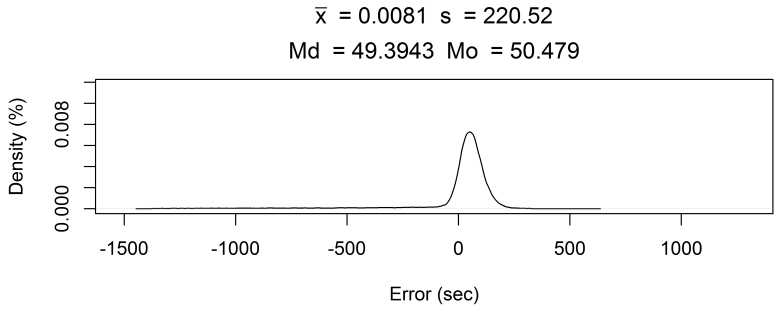

Figure 5.4: Density of errors from Kalman Filter

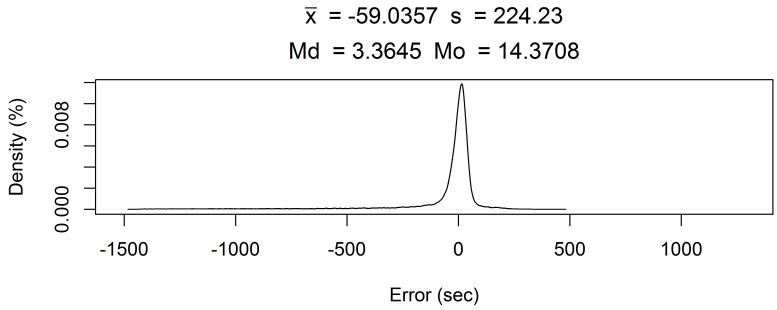

Figure 5.5: Density of errors from Bagging

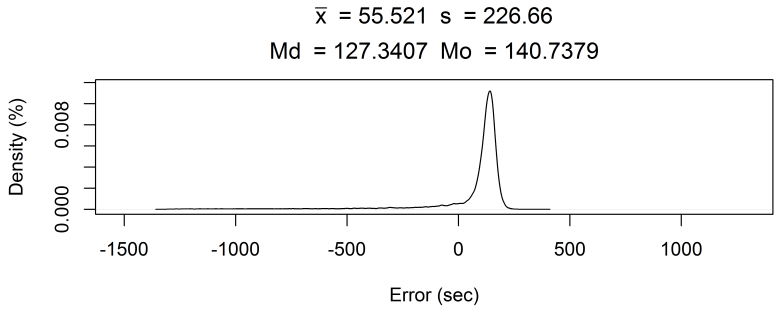

Figure 5.6: Density of errors from Boosting

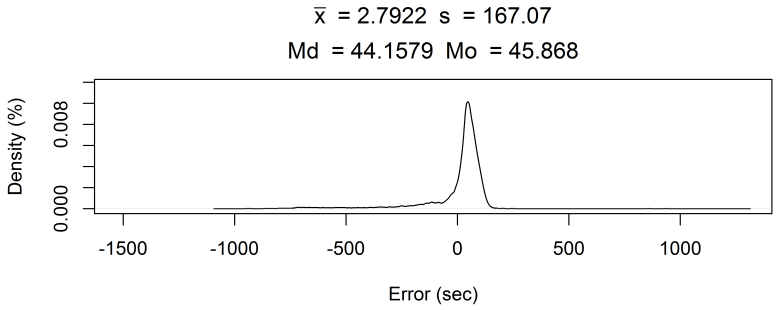

Figure 5.7: Density of errors from Lasso

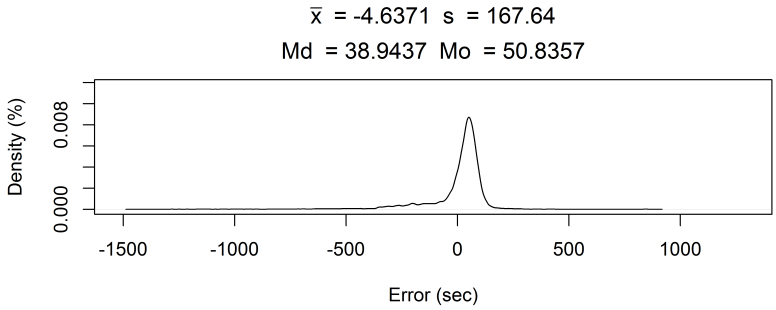

Figure 5.8: Density of errors from FRBS

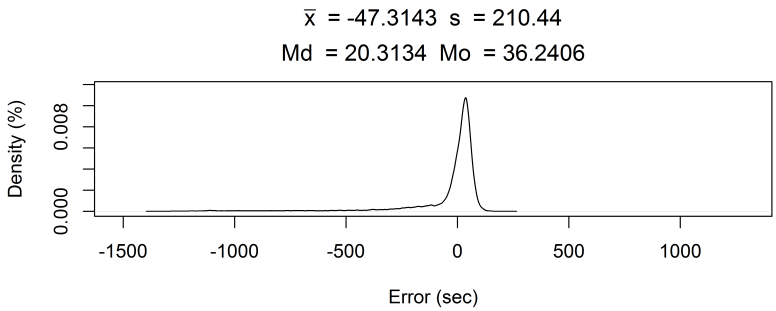

Figure 5.9: Density of errors from average

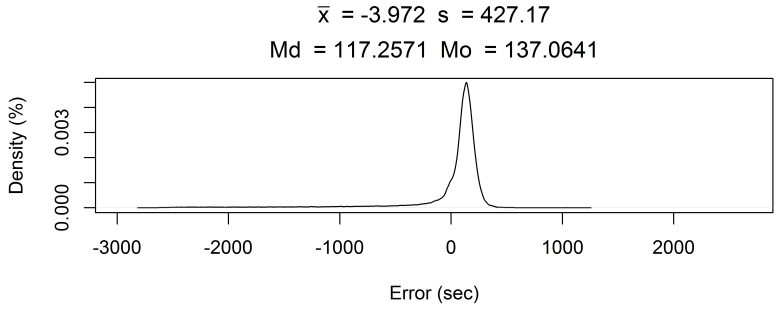

Figure 5.10: Density of errors from online-delayed EKF

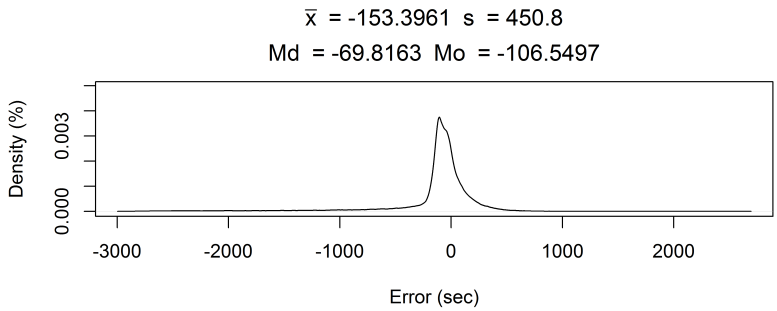

Figure 5.11: Density of errors from LOKRR

Model	A	p
SVM	7817.497	3.7×10^{-24}
ANN	7245.759	3.7×10^{-24}
k-NN	6550.465	3.7×10^{-24}
Kalman filter	8078.293	3.7×10^{-24}
Bagging	8949.996	3.7×10^{-24}
Boosting	8875.103	3.7×10^{-24}
Lasso	6413.819	3.7×10^{-24}
FRBS	5400.46	3.7×10^{-24}
Average ensemble	8124.137	3.7×10^{-24}
Online-delayed EKF	29299.61	3.7×10^{-24}
LOKRR	21770.85	3.7×10^{-24}

Table 5.1: Result of Anderson-Darling normality test

satisfied by the error distributions. This indicates that the errors of the different methods might not follow a normal distribution.

To further investigate this assumption, the Anderson-Darling test [Anderson and Darling, 1952] is employed to check for normality. The Anderson-Darling investigates whether or not a given distribution follow a normal distribution. Its null hypothesis is that the distribution follows a normal distribution, and the alternative hypothesis is that the given distribution is not normally distributed. The Anderson-Darling test statistic and the corresponding p-value for each methods' error distribution is displayed in Table 5.1. For the baselines and ensemble learning approaches, the Anderson-Darling test is based on 50 381 samples, and for the online learning approaches the test is based on 176 953 samples. The results of the Anderson-Darling test indicate, on a 0.05 significance level, that none of the error distributions follow a normal distribution. Based on the considerations of the plots of the different approaches' errors, and the results of the Anderson-Darling, it is assumed that the errors are not normally distributed.

5.1.2 Experiment 1 - Ensemble Learning

In order to investigate Research Question 1, which is repeated below for convenience, a comparison between all the different ensemble methods is conducted. First, this section compares the performance metrics of the different ensemble learning approaches. Second, the plots of the error distributions are evaluated. Finally, the results from the non-parametric significance tests are reported.

Research Question 1 Given a set of baseline methods, which ensemble learning technique yields the best prediction accuracy?

Performance metrics

Table 4.5 displays the performance metrics for the different baselines used in Experiment 1. The best performing baseline in terms of both RMSE and MAE, is ANN with RMSE and MAE of 216.5004 and 107.201, respectively. SVM has a RMSE of 233.4601 and MAE of 108.524. K-NN has RMSE of 218.3885 and MAE of 110.4693, whilst Kalman filter has RMSE and MAE values of 220.5144 and 112.6546, respectively.

The results from Experiment 1 indicate that lasso's RMSE of 167.0931 is the lowest RMSE of all the ensemble learning approaches. However, it is only slightly lower than FRBS's RMSE of 167.705. Taking the simple average of the baseline predictions results in a RMSE of 215.6909. The RMSE values of bagging and boosting are 231.8706 and 233.3547, respectively.

Comparing the MAE for the different models forms a slightly different picture. Bagging has a MAE of 93.88298, and lasso and FBRS have MAE values of 95.92011 and 94.00293, respectively. The average ensemble has a MAE of 98.34549, and boosting has a MAE of 165.567. The most noticeable difference comparing the MAE values, is that bagging has the lowest value among the ensemble learning approaches. Lasso, FRBS and the average have MAE values close to that of bagging, whilst boosting's MAE is considerably higher.

Since the results presented above does not agree upon a winner in both RMSE and MAE, the performance metrics alone may not be sufficient to support any conclusions as to which ensemble learning approach provides the best prediction accuracy. The density plots presented in figures 5.5 to 5.9 contain a lot of information which a single performance metric is not able to represent. Therefore, a detailed examination of the error distributions follow.

Error distributions

Due to the non-normality assumption of the error distributions, the sample mean of an error distribution is not a representative value for where the majority of the mass of the distribution is located. The sample means of the distributions are affected by the long tails present in the error distributions, and are offset in the direction of the most prominent tail. Two measures that are more capable of representing the location of the majority of the error distribution's mass are the sample median and sample mode, as these are not as affected by the tails present in the error distributions as the sample mean. The sample median is chosen as basis for comparison here because it is convenient to use in non-parametric significance tests.

From the error distributions, presented in figures 5.5 to 5.9, it can be seen that bagging has the sample median closest to zero, which is 3.3645. This might indicate that bagging is close to unbiased when predicting travel times. The sam-

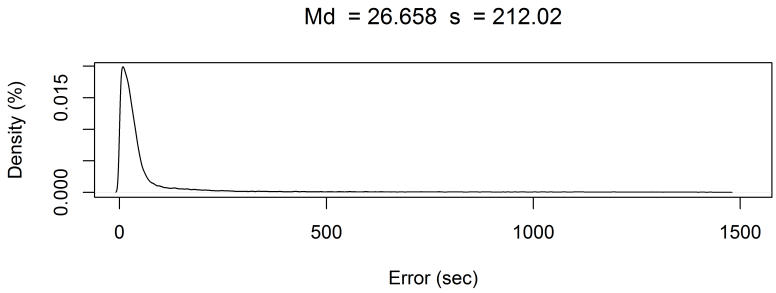

Figure 5.12: Density of absolute errors from Bagging

ple medians of average, FRBS, lasso and boosting are 20.3134, 38.9437, 44.1579 and 127.3407, respectively. Based on the sample median of the error distributions it appears as though these approaches are biased towards overestimating travel times. Furthermore, there are noticeably differences between the distributions' sample deviation. Lasso and FRBS have sample deviations of 167.07 and 167.64, respectively, which are considerably lower than the sample deviations for average, bagging and boosting which are 210.44, 224.23 and 226.66, respectively. Bagging, boosting and the average have long tails towards the left. This indicates that in rare cases the models heavily underestimate the travel time. Both lasso and FRBS have long tails on both sides, indicating that they occasionally overestimate or underestimate the travel time by far.

In order to investigate whether the observed differences between the sample medians of the different models are significant, statistical significance testing should be employed. Usually, the alternative hypotheses in significance tests are expressed as $\text{mean}_A < \text{mean}_B$. This poses a problem in this case, as the distributions contain negative errors, which leads to incorrect orderings. E.g. comparing an error of -100 to an error of $+1$ would prefer the error of -100, which is undesirable because it is a larger error, only in the other direction. For this reason, plots of the distribution of absolute errors are used as basis for making the alternative hypotheses.

Figures 5.12 to 5.16 illustrate the distribution of the absolute errors for the ensemble learning approaches. These plots indicate that the sample median of the absolute error for bagging is lower than boosting, lasso, FRBS and average; the sample median of the average is lower than boosting, lasso and FRBS; the sample median of FRBS is lower than boosting and lasso; the sample median of lasso is lower than boosting. To further investigate these hypotheses, statistical significance testing is employed.

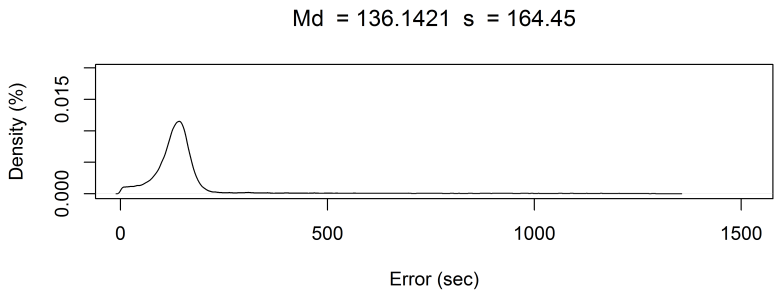

Figure 5.13: Density of absolute errors from Boosting

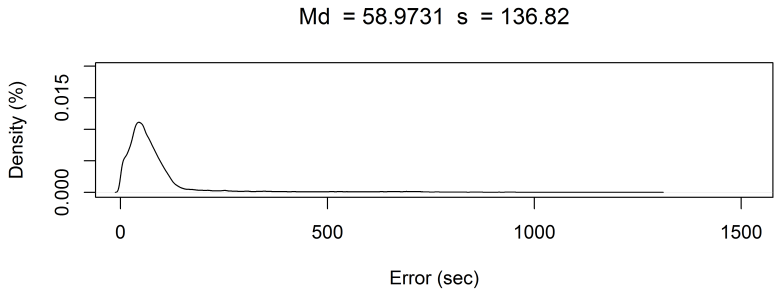

Figure 5.14: Density of absolute errors from Lasso

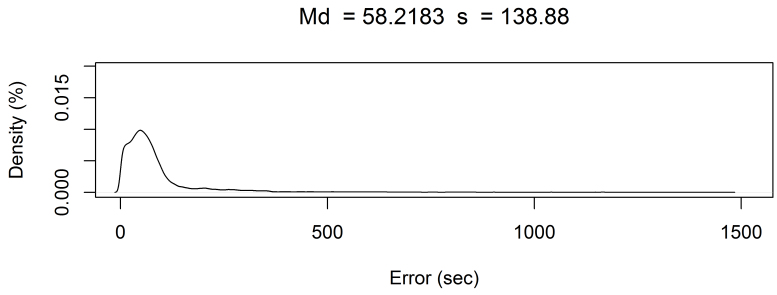

Figure 5.15: Density of absolute errors from FRBS

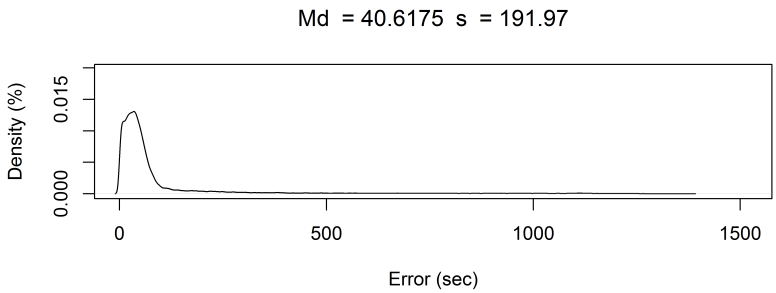

Figure 5.16: Density of absolute errors from average

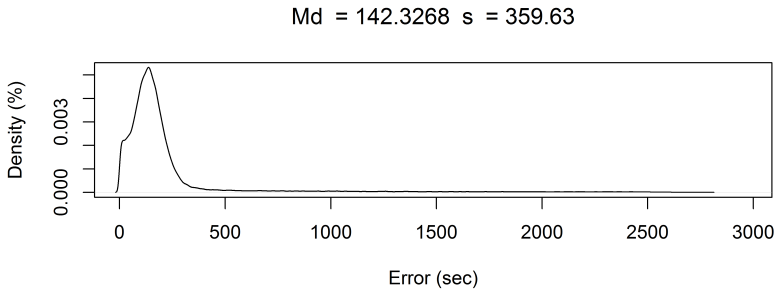

Figure 5.17: Density of absolute errors from online-delayed EKF

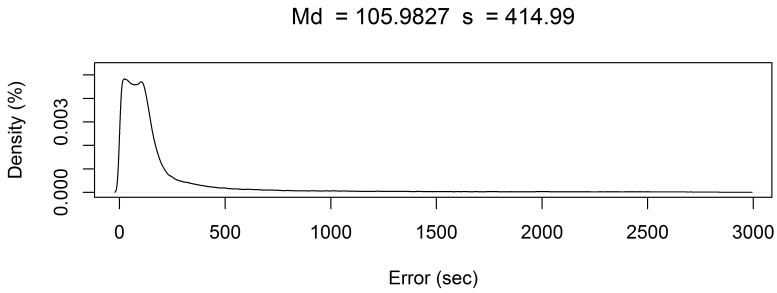

Figure 5.18: Density of absolute errors from LOKRR

Hypothesis Testing

Based on the assumption that the errors do not follow a normal distribution, non-parametric significance testing is used to test the hypotheses described above. In non-parametric significance testing, the test variable used is the sample median in contrast to parametric significance testing, where the sample mean is used.

When doing significance testing on a group of related hypotheses the probability of making type 1 errors increases as the number of hypotheses increases. This probability is referred to as the familywise error rate, and the concept of increasing probability of making type 1 errors when the number of hypotheses increases is also known as the multiplicity effect [Salzberg, 1997]. Trawinski et al. [2012] suggests using the approach described in García and Herrera [2008] to control the familywise error rate, when comparing more than two machine learning regression models. The approach uses the Friedman test [Friedman, 1937], which works by assigning a rank to each model. For every prediction made during testing, the model with the lowest error is assigned the lowest rank and the model with the highest error is assigned the highest rank. The average rank is computed for each model across all predictions. The Friedman test indicates whether any of the rankings are significantly different from each other. However, it does not identify the specific pairs of models that differ. Consequently, one or more post-hoc procedures are needed to determine which pairs of models are significantly different. Note that these procedures do not test whether or not the differences in sample medians are significant. Rather, their test statistic is based on the differences in Friedman rank.

The post-hoc procedures used in García and Herrera [2008] are Nemenyi [Nemenyi, 1962], Holm [Holm, 1979], Shaffer [Shaffer, 1986], and Bergmann-Hommel [Bergmann and Hommel, 1988]. These procedures perform two-sided hypothesis tests, and can therefore only establish whether or not there is a significant difference in Friedman rank between a pair of models. In the cases where a significant difference can be established, the rank from the Friedman test indicates which of the two models is considered best, where lower rank is better.

This study follows the approach recommended in Trawinski et al. [2012]. The Friedman test and post-hoc procedures are conducted using an open source program available on the web page[1] for the research group Soft Computing and Intelligent Information Systems at the University of Granada, Spain. The software is written in Java [Gosling et al., 2013] and follows the procedure reported in García and Herrera [2008]. It is important to note that the program assumes the input to be prediction accuracies. Since the performance of the methods investigated in this study are assessed with error metrics, the inverse of the absolute error is given as input to the program.

[1] `http://sci2s.ugr.es/sicidm/#ten`

The Friedman test is run using 50 381 observations of inverse absolute errors from the five ensemble learning approaches. The result of running the Friedman test indicates, on a 0.05 significance level, that there is a significant difference in performance between the ensemble learning approaches. The rankings resulting from the Friedman test is displayed in Table 5.2.

As the Friedman test indicates a significant difference between the ensemble learning approaches, post-hoc procedures is employed to determine for which pairs of algorithms the difference is significant. The adjusted p-values resulting from running the different post-hoc procedures is illustrated in Table 5.3, where the approach highlighted with bold font is the one with lowest rank in the Friedman test. The post-hoc procedures are unanimous in their decisions to reject all the null hypotheses with a significance level of 0.05. The results indicate that bagging has a significantly lower Friedman rank than all the other ensemble approaches in this study. Furthermore, the Friedman rank of average is significantly lower than that of boosting, lasso and FRBS. The test also reveals that the Friedman rank of FRBS is significantly lower than the Friedman rank of lasso, whilst both lasso and FRBS have a significantly lower Friedman rank than boosting.

Algorithm	Friedman rank	Rank position
Bagging	2.1883845100330297	1st
Average	2.40672078759757	2nd
Lasso	2.980071852483732	3rd
FRBS	3.02907842242099	4th
Boosting	4.395744427460865	5th

Table 5.2: Friedman ranking

Algorithms	Nemenyi	Holm	Shaffer	Bergmann-Hommel
Bagging vs. boosting	0.0	0.0	0.0	4.9×10^{-324}
Boosting vs. **average**	0.0	0.0	0.0	4.9×10^{-324}
Boosting vs. **lasso**	0.0	0.0	0.0	4.9×10^{-324}
Boosting vs. **FRBS**	0.0	0.0	0.0	4.9×10^{-324}
Bagging vs. FRBS	0.0	0.0	0.0	4.9×10^{-324}
Bagging vs. lasso	0.0	0.0	0.0	4.9×10^{-324}
FRBS vs. **average**	0.0	0.0	0.0	4.9×10^{-324}
Lasso vs. **average**	0.0	0.0	0.0	4.9×10^{-324}
Bagging vs. average	1.8022×10^{-105}	3.6044×10^{-106}	3.6044×10^{-106}	3.6044×10^{-106}
Lasso vs. FRBS	8.6857×10^{-6}	8.6857×10^{-7}	8.6857×10^{-7}	8.6857×10^{-7}

Table 5.3: Post-hoc procedure results

5.1.3 Experiment 2 - Online Learning

In order to investigate Research Question 2, which is repeated below for convenience, a comparison of the two online learning approaches is conducted. First,

the performance metrics of online-delayed EKF and LOKRR are compared. Second, the error distributions of the two online methods are examined. Finally, statistical significance testing is employed.

Research Question 2 Which online learning technique yields the best prediction accuracy?

The results from Experiment 2, seen in Table 4.7, indicate that the online-delayed EKF has a lower RMSE than LOKRR. The RMSE of the online-delayed EKF is 427.1863, compared to LOKRR's 476.1826. Online-delayed EKF also has the lowest MAE of 230.551, whilst LOKRR has a MAE of 233.5318. In contrast to the RMSE values, the MAE values do not differ considerably from each other.

Figure 5.10 and Figure 5.11 illustrate the error distributions of online-delayed EKF and LOKRR, respectively. Online-delayed EKF has a sample median of $+117.2571$, whilst LOKRR has a sample median of -69.8163. This might indicate that the two methods have different bias towards predicting travel time, where online-delayed EKF tend to overestimate the travel time and LOKRR tend to underestimate the travel time. Looking at the tails of error distributions for the two methods, it can be seen that LOKRR's right tail is noticeably longer than that of online-delayed EKF. However, both approaches have similarly long tails towards the left. This indicates that both methods greatly underestimates the travel time in some cases, whilst LOKRR also occasionally heavily overestimates the travel time. Online-delayed EKF has a sample deviation of 427.17, and LOKRR has a sample deviation of 450.8. The sample deviations do not differ to the same extent as the sample medians of the two methods.

Following the same approach as with the ensemble learning techniques, the sample medians of the absolute errors of online-delayed EKF and LOKRR are compared. The distributions of the absolute errors for the two online learning approaches are illustrated in Figure 5.17 and Figure 5.18. These two plots suggests that the sample median of LOKRR is lower than the sample median of online-delayed EKF. In order to investigate whether the differences between the two sample medians are significantly different, a statistical significance test is employed.

Section 5.1.2 introduces the multiplicity effect that arises when performing hypothesis testing including many hypotheses. In Experiment 2, only two models are tested for significant difference, and the multiplicity effect is therefore not prominent. A paired non-parametric significance test is thus sufficient to evaluate the results from Experiment 2. More specifically, the Wilcoxon signed rank test [Wilcoxon, 1945] is employed. The null hypothesis is that the two distributions are equal, meaning that the medians of two distributions are equal. The alternative hypothesis is that the sample median of LOKRR is lower than the sample median of online-delayed EKF. The Wilcoxon signed rank test statistic

Alternative hypothesis	V	p
LOKRR < Online-delayed EKF	632913330	2.2e-16

Table 5.4: Result of Wilcoxon sign-rank test

and corresponding p-value for online-delayed EKF and LOKRR can be seen in Table 5.4. The results are based on 176 953 samples, and indicate, on a 0.05 significance level, that LOKRR's sample median is significantly lower than that of online-delayed EKF.

5.2 Discussion

This section offers a discussion on the results from the experiments. Section 5.2.1 discusses the results from Experiment 1, whilst Section 5.2.2 discusses the results from Experiment 2.

5.2.1 Ensemble Learning

One thing that is noticeable from the results in Experiment 1, is that there does not seem to be a strong positive correlation between which method is considered best in terms of sample median, RMSE and MAE. Bagging is the method with the lowest MAE and has the absolute error distribution with lowest sample median, whilst at the same time having the second highest RMSE. Lasso is the method with the lowest RMSE, whilst at the same time having a MAE value close to that of bagging. FRBS has a RMSE value close to that of lasso. The fact that bagging has the lowest MAE and lowest sample median while also having the second highest RMSE might indicate that bagging makes predictions being closer to the actual travel time more often than lasso. This is also indicated by the result of the Friedman test where bagging has the best Friedman rank. Even though bagging has lower MAE and sample median than lasso, lasso has considerably lower RMSE than bagging. Since RMSE is sensitive to large errors, this might indicate that in those rare cases where their errors are very large, bagging's errors are larger than those of lasso.

Both lasso and FRBS are optimized in terms of RMSE, i.e. they combine the predictions of their baselines in order to reduce the RMSE as much as possible. This might explain why lasso and FRBS achieve lower RMSE than the other approaches. When they optimize their performance in terms of RMSE during training, they will most likely be the ones performing best in terms of RMSE during testing as well, as long as they do not overfit the training data.

Bagging's approach, on the other hand, is entirely different. It trains multiple

baselines on a random subset of the training data and then combines them by taking the average over all predictions. It makes no attempt to decrease the RMSE, but instead achieves the lowest median of the ensemble approaches. In theory, the strength of bagging is that it can even out the differences in bias between its underlying baselines by averaging their predictions. However, in Experiment 1, this does not seem to be the reason why bagging achieves a low sample median. Table 5.5 shows the sample medians of the baselines used in bagging. By inspecting these sample medians, it can be seen that all of them are centered around three. Consequently, when bagging averages the predictions from its baselines, bagging's sample median is also approximately three.

Recall from Section 4.2.3 that the implementation of boosting for regression models provided in `scikit-learn` is used in Experiment 1. The only method of the four baselines used in this study, SVM, k-NN, ANN and Kalman filter, that both is implemented in `scikit-learn` and supports weighting of training examples is SVM. That puts a restriction on the baseline being used in boosting, as SVM is the only baseline available. In order to compare bagging and boosting on a fair basis, SVM is also chosen as the baseline in bagging. The use of SVM as baseline in bagging may have lead to suboptimal performance for said method. Section 3.1.1 explains that bagging may work well with unstable learners. However, there is little to gain from bagging when using stable learners. As mentioned in Section 5.2.1, SVMs are stable learners and are therefore unlikely to get completely different bias when trained on slightly different data. This might explain why all the underlying baselines in bagging are very similar, even though they have been trained on different subsets of the training set. Consequently, when using SVMs as baselines for bagging, this study does not realize bagging's full potential as en ensemble learner. This makes it difficult to conclude whether any of the other ensemble learners are better than bagging.

Boosting performs considerably worse than the other methods in terms of sample median and MAE. Its sample median of 127.43 illustrates a considerable bias towards overestimating the travel time. Additionally, boosting comes out as the worst performing approach in terms of RMSE and Friedman ranking. During training, boosting was set to use a maximum of 25 baselines in order to be similar to bagging. In `scikit-learn` the boosting algorithm stops if it achieves perfect fit on the training data. In Experiment 1, boosting ended up with 5 baselines. This indicates that it achieved perfect fit on the training data, and its poor performance might be due to overfitting the training set. Although the results for boosting in this study are not promising, it does not mean it is not a viable ensemble learning approach for travel time prediction.

The fact that the sample median of the average ensemble is farther away from zero than bagging suggests that the average ensemble has a stronger bias towards overestimating travel times than that of bagging. However, the average

Baseline	Sample median
1	3.3158
2	3.3944
3	3.2888
4	3.5681
5	3.409
6	3.3392
7	3.2783
8	3.2737
9	3.3757
10	3.3157
11	3.3435
12	3.3958
13	3.0521
14	3.0536
15	3.2123
16	3.0686
17	3.2234
18	3.2264
19	3.6155
20	3.2237
21	3.1358
22	3.3673
23	3.2151
24	2.998
25	3.5768

Table 5.5: Sample medians of bagging baselines

ensemble has a sample median closer to zero than lasso, FRBS and boosting, which indicates that the average ensemble has a weaker bias than the mentioned approaches. The average ensemble's MAE value is considerably lower than that of boosting, and close to those of bagging, lasso and FRBS, albeit higher than the three latter methods. The average ensemble's RMSE and sample deviation illustrates that its predictions might have larger deviations than those of lasso and FRBS. These results illustrates that even a simple approach like taking the average of the baselines' predictions may be a viable approach to constructing an ensemble.

As the different performance metrics and other criteria for comparison do not agree on which method is best, it is interesting to explore their interpretation and importance. The RMSE is tightly coupled with the sample deviation, as their computations are carried out in a similar fashion. Therefore, both RMSE and sample deviation can be seen as measures of variation in the predictions of a method. The median, on the other hand, is more representative for the model's bias and is a good indicator for what an expected error is. Similarly, the MAE is a measure indicating what a typical absolute error is. Therefore it is not surprising that the approach with the lowest sample median also has the lowest MAE. Additionally, it is also expected that the method with the lowest RMSE also has the lowest sample deviation.

Since these measures represent different properties, what measure to consider when deciding what model is best depends on what properties are most important. Presented with two methods, method A having low bias, but large deviation; and method B having larger bias, but smaller deviation, which method is most desirable? Method B will in most cases either overestimate or underestimate travel times, depending on the direction of the bias. This is not the case for method A since it is unbiased, but its errors will fluctuate more than method B because of its higher deviation. It is simpler to correct for method B's bias than it is to correct for method A's varying errors. One can, in theory, correct for method B's bias by adding or subtracting a constant to its prediction, depending on the direction of its bias. In terms of predicting travel time, having consistent predictions is important. One can imagine commuters having more confidence in a travel time prediction system if its predictions are consistent. However, it is also important to have accurate predictions, meaning that the travel time predictions are as close to the actual travel time as possible. If a bias is present in a prediction model, it may be better having a bias towards overestimating travel time than underestimating it. Commuters might be more pleased using less time than predicted in contrast to end up using longer time than predicted. In total, both accurate predictions and consistent predictions are desirable from a travel time prediction system.

To summarize the results, there is no ensemble approach among the ones

investigated in this study that clearly outperforms the other methods on all measures. This makes it difficult to conclude which ensemble learning technique has the best prediction accuracy.

Recall from Section 2.1.2 that ensemble learners work best when its baselines have different bias. By inspecting figures 5.1 to 5.4, it can be seen that k-NN, ANN and Kalman filter have sample medians of 23.46, 26.54 and 49.39, respectively, whilst SVM has a sample median of -31.63. This might indicate that k-NN and ANN have similar bias towards slightly overestimating the travel time, whilst Kalman filter overestimates the travel time even more. In contrast, SVM has a bias towards underestimating the travel time. Including even more baselines, or baselines with greater diversity in terms of bias could potentially lead to even better performance for the ensemble learning approaches.

Figures 5.1 to 5.11 in Section 5.1 illustrates that the error distributions of the baselines, ensemble learning approaches and the online learning approaches have long tails. These tails may be explained by the presence of outliers in the data set. The traffic flow and mean travel time for those vehicles having an abnormally large travel time might be the same as vehicles having a normal travel time. When the methods make predictions for the vehicles having an extreme travel time, they will have seen the normal case more times, and therefore predict that the current vehicle will have a normal travel time. However, due to this vehicle stopping along the road stretch, the resulting travel time is much larger than normal, and an extreme error occurs. This will both affect the tail of the error distribution, in addition to affecting the performance metrics, especially the RMSE.

5.2.2 Online Learning

Based on the performance metrics alone, online-delayed EKF appear to be better than LOKRR. However, based on the error distribution plots and the significance tests regarding the sample medians, LOKRR's sample median seem to be lower than the sample median of online-delayed EKF. The fact that online-delayed EKF has lower RMSE and MAE than LOKRR, might be explained by that online-delayed EKF does not have as extreme errors as LOKRR, and in this way does not get punished as much through the performance metrics. At the same time, LOKRR has an absolute error distribution with lower sample median than online-delayed EKF. A possible explanation for this is that in majority, LOKRR makes predictions being closer to the actual travel time than online-delayed EKF, and thus makes less biased predictions than online-delayed EKF.

Note that the performance metrics and sample deviations of the two online learning approaches are considerably higher than those of the ensemble approaches. This relates to the fact that the online learning approaches have learned from and made predictions for a data set where no attempt to remove outliers is

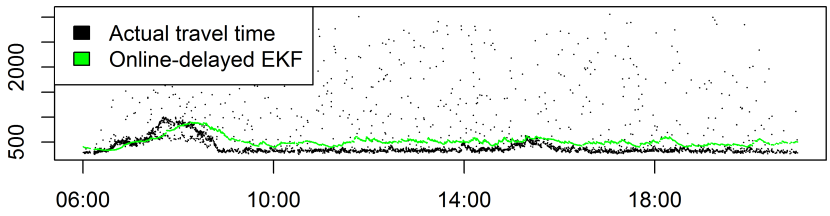

Figure 5.19: Predictions from online-delayed EKF and actual travel time March 4, 2015

done. In contrast, the ensemble learning approaches have used a data set where outliers have been removed to some degree. Additionally, the offline learners can look at data more than once and therefore have a better chance of discovering patterns in the data. However, the focus of this study is not to compare online learners with offline learners. What is important is to assess the performance of them as online learners and to compare them to each other.

In Figure 5.19 and Figure 5.20 the predictions from online-delayed EKF and LOKRR March 4, 2015 are plotted, respectively. Online-delayed EKF tends to overestimate the travel times. This can be seen in Figure 5.19, where the online-delayed EKF predictions tend to lie above the actual travel time during normal traffic conditions. LOKRR, on the other hand, tends to underestimate the travel times as its predictions tend to lie below the actual travel times. It can be seen in Figure 5.20 that the predictions from LOKRR vary greatly from one prediction to the next. In contrast to LOKRR, online-delayed EKF's predictions do not fluctuate a lot. Online-delayed EKF seems to be able to follow the curve of the true travel time, except with a delay.

In theory, the strength of online learning approaches are their ability to adapt to abnormal traffic scenarios, i.e. scenarios that are not present in the training set. As seen in Figure 5.21 and Figure 5.22, the afternoon peak on March 13, 2015 is considerably higher than usual. In order to investigate how well the two online learning approaches foresee this peak, their predictions are inspected. Interestingly, LOKRR is unable to detect the increase in travel time at all. One possible explanation for LOKRR's inability to detect this peak is that it is an instance based approach. Each kernel contains data from the previous week of

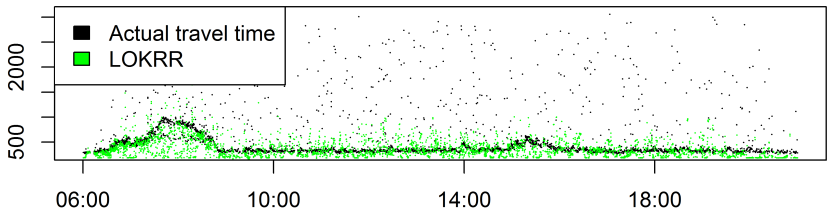

Figure 5.20: Predictions from LOKRR and actual travel time March 4, 2015

data, approximately. If this data does not contain any abnormally high travel times, the kernel is unable to predict that such travel times can happen. An inspection of the travel times during the week leading up to March 13 reveals that no similar peaks in travel times occurred. It is also interesting to note that the abnormally high travel times observed during the afternoon on March 13 do not seem to affect the predicted travel times the following days. The online-delayed EKF is better able to detect that there is a peak in travel times. However, there is a considerable latency between when the congestion builds up and the online-delayed EKF is able to detect the increase. Similarly, the online-delayed EKF is slow to detect the decrease in travel time. Consequently, the online-delayed EKF predicts a peak in travel time when the traffic is almost back to normal.

In the original paper [Haworth et al., 2014], LOKRR uses five minute aggregated travel time data. However, in this study the kernels contain individual observations. This leads to several challenges.

First of all, using individual travel times heavily limits how far back in time one can keep data. In Haworth et al. [2014], the kernels consisted of observations from the 80 previous days. Including a window size of 3, this leads to each kernel containing 560 observations. In this study, the biggest kernel contained 698 observations. However, this only comprised a week of data with a window size of 1. One week of data might not be enough to detect cyclic patterns. Additionally, the limited window size further decreases its ability to detect cyclic patterns, since it is unlikely that e.g. rush hour occurs at the same exact time every day.

Secondly, the kernels end up with different amounts of data. The number of travels conducted during the morning and afternoon rush hour far exceed the amount of travels at 8 PM. Kernels responsible for periods of the day with less

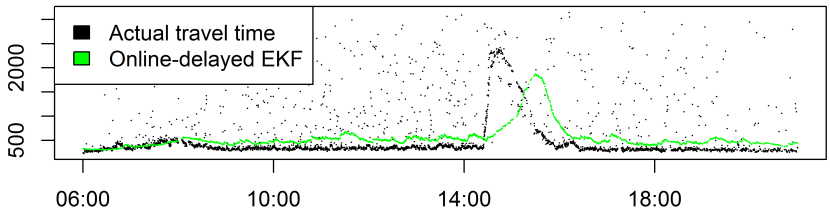

Figure 5.21: Predictions from online-delayed EKF and actual travel time March 13, 2015

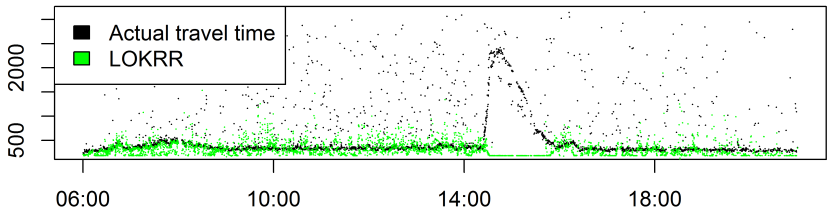

Figure 5.22: Predictions from LOKRR and actual travel time March 13, 2015

traffic have less data to base their predictions on, which is expected to lead to lower prediction accuracy. This could be solved by setting a limit to how much data each kernel could contain. However, that would mean that the kernels responsible for intervals with less traffic would contain data farther back in time than kernels responsible for periods of the day with heavy traffic. Whether or not that is a desirable property should be considered.

It may be unfair to employ LOKRR on a type of data set that it is not designed for and the implications using individual travel times have for LOKRR makes it difficult to compare it to online-delyaed EKF or any other online learner.

5.2.3 Limitations

The results from Experiment 1 and Experiment 2 are products of several factors. This section presents possible factors that may have contributed to the results.

The methods investigated in this study are data driven approaches, and is inherently affected by the data it operates on. Consequently, the data preprocessing step will affect the final results both in terms of the trained models and the performance metrics. Figure 4.2 shows the travel times registered for January 29, 2015. The figure clearly illustrates the presence of vehicles with abnormally high travel times, which is seen as dots high above the line illustrating the normal travel time. An attempt to remove such outliers is performed. However, more sophisticated schemes may be employed. When the methods are evaluated based on RMSE, the approaches that best adapt to the outliers may appear better than those who are unable to adapt to outliers. This may suggest the use of other performance metrics than RMSE, when outliers are present in the data set. However, it may also indicate that having a greater focus on removing outliers from the data set is important in order to assess the performance more based on normal cases and less based on outliers.

Another aspect affecting the method's performance is the amount of training data that they are provided with. Having more training data increases the probability that the methods generalize better on the test set. In Experiment 1, the baselines are trained on three weeks of data. In order for the baselines to perform well on the test set, these three weeks have to contain enough data to include all the different scenarios in the test set. Additionally, three weeks of data is used to train the weights in the lasso ensemble.

The methods are also affected by their parameters. In this study relatively small amounts of data is used for parameter tuning. Additionally, the parameter search is fairly simple. Larger grids could have been employed to begin with, and several steps of using finer grids could have been used to optimize the parameters even further.

5.3 Conclusion

- What conclusions can be drawn from this?

5.4 Contributions

- State contributions

5.5 Future Work

- Present suggestions for future work
- Things we did not have time to do, but think are important or interesting to investigate further

Bibliography

Anderson, T. W. and Darling, D. A. (1952). Asymptotic theory of certain "goodness of fit" criteria based on stochastic processes". *Ann. Math. Statist.*, 23(2):193–212.

Bajwa, S. U. I., Chung, E., and Kuwahara, M. (2003). A travel time prediction method based on pattern matching technique. volume 21, pages 997 – 1010, Cairns, QLD, Australia.

Bergmann, B. and Hommel, G. (1988). Improvements of general multiple test procedures for redundant systems of hypotheses. In Bauer, P., Hommel, G., and Sonnemann, E., editors, *Multiple Hypothesenprüfung / Multiple Hypotheses Testing*, volume 70 of *Medizinische Informatik und Statistik*, pages 100–115. Springer Berlin Heidelberg.

Bouillet, E., Chen, B., Cooper, C., Dahlem, D., and Verscheure, O. (2013). Fusing traffic sensor data for real-time road conditions. Rome, Italy.

Breiman, L. (1996). Bagging predictors. *Machine Learning*, 24(2):123–140.

Cao, Y. Learning the extended kalman filter. `http://www.mathworks.com/matlabcentral/fileexchange/18189-learning-the-extended-kalman-filter`. From MATLAB Central File Exchange, accessed: 2015-02-12.

Cao, Y. Neural network training using the extended kalman filter. `http://www.mathworks.com/matlabcentral/fileexchange/18289-neural-network-training-using-the-extended-kalman-filter`. From MATLAB Central File Exchange, accessed: 2015-02-12.

Caputo, B., Sim, K., Furesjo, F., and Smola, A. (2002). Appearance-based object recognition using svms: which kernel should i use?

Chen, D.-W. and Zhang, J.-P. (2005). Time series prediction based on ensemble anfis. In *Machine Learning and Cybernetics, 2005. Proceedings of 2005 International Conference on*, volume 6, pages 3552–3556 Vol. 6.

Chen, L. and Chen, C. (2007). Ensemble learning approach for freeway short-term traffic flow prediction. In *System of Systems Engineering, 2007. SoSE '07. IEEE International Conference on*, pages 1–6.

Chung, E., Warita, H., ul Islam Bajwa, S., and Kuwahara, M. (2004). Travel time prediction: issues and benefits. `http://its.iis.u-tokyo.ac.jp/pdf/2004-021.pdf`, accessed November 21, 2014.

Cohen, P. R. and Howe, A. E. (1988). How evaluation guides AI research: The message still counts more than the medium. *AI Magazine*, 9(4):35–43.

Commision of the European Communities (2001). European transport policy for 2010: time to decide. Technical report. `http://ec.europa.eu/transport/themes/strategies/doc/2001_white_paper/lb_com_2001_0370_en.pdf`, accessed November 21, 2014.

Deb Nath, R. P., Lee, H.-J., Chowdhury, N. K., and Chang, J.-W. (2010). Modified k-means clustering for travel time prediction based on historical traffic data. volume 6276 LNAI, pages 511 – 521, Cardiff, United kingdom.

Dembczynski, K., Kotlowski, W., Gawel, P., Szarecki, A., and Jaszkiewicz, A. (2013). Matrix factorization for travel time estimation in large traffic networks. volume pt.II, pages 500 – 10, Berlin, Germany.

Dharia, A. and Adeli, H. (2003). Neural network model for rapid forecasting of freeway link travel time. *Engineering Applications of Artificial Intelligence*, 16(7-8):607 – 613.

Drucker, H. (1997). Improving regressors using boosting techniques. In *Proceedings of the Fourteenth International Conference on Machine Learning*, ICML '97, pages 107–115, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

Efron, B., Hastie, T., Johnstone, I., and Tibshirani, R. (2004). Least angle regression. *Ann. Statist.*, 32(2):407–499.

Exterkate, P. (2013). Model selection in kernel ridge regression. *Computational Statistics and Data Analysis*, 68(0):1 – 16.

Freund, Y. and Schapire, R. E. (1997). A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences*, 55(1):119 – 139.

Friedman, M. (1937). The use of ranks to avoid the assumption of normality implicit in the analysis of variance. *Journal of the American Statistical Association*, 32(200):pp. 675–701.

García, S. and Herrera, F. (2008). An extension on "statistical comparisons of classifiers over multiple data sets" for all pairwise comparisons. *Journal of Machine Learning Research*, 9:2677–2694.

Gosling, J., Joy, B., Steele, Jr., G. L., Bracha, G., and Buckley, A. (2013). *The Java Language Specification, Java SE 7 Edition*. Addison-Wesley Professional, 1st edition.

Hao, L., Xiao-liang, Z., and Ke, Z. (2008). Travel time prediction for urban arterials based on rough set. *Journal of Highway and Transportation Research and Development*, 25(10):117 – 22.

Hastie, T., Tibshirani, R., and Friedman, J. (2009). *The elements of statistical learning: data mining, inference and prediction*. Springer, 2 edition.

Haugen, T. and Aakre, A. (2014). BA6058 Trafikkteknikk og trafikksikkerhet, Kompendium del Trafikkteknikk. Norwegian University of Science and Technology.

Haworth, J., Shawe-Taylor, J., Cheng, T., and Wang, J. (2014). Local online kernel ridge regression for forecasting of urban travel times. *Transportation Research Part C: Emerging Technologies*, 46:151 – 178.

Hofleitner, A., Herring, R., and Bayen, A. (2012). Arterial travel time forecast with streaming data: a hybrid approach of flow modeling and machine learning. *Transportation Research, Part B (Methodological)*, 46(9):1097 – 122.

Holm, S. (1979). A simple sequentially rejective multiple test procedure. *Scandinavian Journal of Statistics*, 6(2):pp. 65–70.

Karatzoglou, A., Smola, A., Hornik, K., and Zeileis, A. (2004). kernlab – an S4 package for kernel methods in R. *Journal of Statistical Software*, 11(9):1–20.

Kofod-Petersen, A. (2014). How to do a structured literature review in computer science, version 0.2. `https://research.idi.ntnu.no/aimasters/files/SLR_HowTo.pdf`, accessed September 1, 2014.

Kuhn, M., Wing, J., Weston, S., Williams, A., Keefer, C., Engelhardt, A., Cooper, T., Mayer, Z., Kenkel, B., the R Core Team, Benesty, M., Lescarbeau, R., Ziem, A., and Scrucca., L. (2015). *caret: Classification and Regression Training*. R package version 6.0-41.

Li, L., Chen, X., and Zhang, L. (2014). Multimodel ensemble for freeway traffic state estimations. *Intelligent Transportation Systems, IEEE Transactions on*, 15(3):1323–1336.

Li, Y., Fujimoto, R., and Hunter, M. (2009). Online travel time prediction based on boosting. In *Intelligent Transportation Systems, 2009. ITSC '09. 12th International IEEE Conference on*, pages 1–6.

Liu, H., van Lint, H., van Zuylen, H., and Zhang, K. (2006a). Two distinct ways of using kalman filters to predict urban arterial travel time. In *Intelligent Transportation Systems Conference, 2006. ITSC '06. IEEE*, pages 845–850.

Liu, H., Van Zuylen, H., Van Lint, H., and Salomons, M. (2006b). Predicting urban arterial travel time with state-space neural networks and kalman filters. Number 1968, pages 99 – 108.

Lu, C.-C. (2012). An adaptive system for predicting freeway travel times. *International Journal of Information Technology; Decision Making*, 11(4):727 – 47.

Ma, Y., Chowdhury, M., Sadek, A., and Jeihani, M. (2012). Integrated traffic and communication performance evaluation of an intelligent vehicle infrastructure integration (vii) system for online travel-time prediction. *Intelligent Transportation Systems, IEEE Transactions on*, 13(3):1369–1382.

Mak, W., Viti, F., Hoogendoorn, S., and Hegyi, A. (2009). Online travel time estimation in urban areas using the occupancy of long loop detectors. pages 383 – 390, Redondo Beach, CA, United states.

MATLAB (2014). *8.4.0.150421 (R2014b)*. The MathWorks Inc., Natick, Massachusetts.

Mitchell, T. M. (1997). *Machine Learning*. McGraw-Hill, international edition.

Mu, T., Jiang, J., and Wang, Y. (2013). Heterogeneous delay embedding for travel time and energy cost prediction via regression analysis. *Intelligent Transportation Systems, IEEE Transactions on*, 14(1):214–224.

Nelder, J. A. and Mead, R. (1965). A simplex method for function minimization. *The Computer Journal*, 7(4):308–313.

Nemenyi, P. (1962). Distribution-free multiple comparisons. In *Biometrics*, volume 18, page 263. International Biometric SOC 1441 I ST, NW, Suite 700, Washington, DC 20005-2210.

Nguyen, D. and Widrow, B. (1990). Improving the learning speed of 2-layer neural networks by choosing initial values of the adaptive weights. In *Neural Networks, 1990., 1990 IJCNN International Joint Conference on*, pages 21–26 vol.3.

Nikovski, D., Nishiuma, N., Goto, Y., and Kumazawa, H. (2005). Univariate short-term prediction of road travel times. In *Intelligent Transportation Systems, 2005. Proceedings. 2005 IEEE*, pages 1074–1079.

Norwegian Public Roads Administration (2007). ITS-Strategi for Statens vegvesen. Technical Report 7. `http://www.vegvesen.no/_attachment/60628`, accessed November 20, 2014.

Ohra, Y., Koyama, T., and Shimada, S. (1997). Online-learning type of traveling time prediction model in expressway. In *Intelligent Transportation System, 1997. ITSC '97., IEEE Conference on*, pages 350–355.

Park, J., Murphey, Y., McGee, R., Kristinsson, J., Kuang, M., and Phillips, A. (2014). Intelligent trip modeling for the prediction of an origin-destination traveling speed profile. *IEEE Transactions on Intelligent Transportation Systems*, 15(3).

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

Petris, G. (2010). An r package for dynamic linear models. *Journal of Statistical Software*, 36(12):1–16.

R Core Team (2014). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria.

R Core Team (2015). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria.

Riza, L. S., Bergmeir, C., Herrera, F., and Benitez, J. M. (2015). *frbs: Fuzzy Rule-based Systems for Classification and Regression Tasks*. R package version 3.0-0.

Rossi, A., Carvalho, A., and Soares, C. (2012//). Meta-learning for periodic algorithm selection in time-changing data. pages 7 – 12, Piscataway, NJ, USA.

Rossum, G. (1995). Python reference manual. Technical report, Amsterdam, The Netherlands, The Netherlands.

Russell, S. and Norvig, P. (2010). *Arificial Intelligence, A Modern Approach*. Pearson, third edition.

Salzberg, S. (1997). On comparing classifiers: Pitfalls to avoid and a recommended approach. *Data Mining and Knowledge Discovery*, 1(3):317–328.

Schliep, K. and Hechenbichler, K. (2014). *kknn: Weighted k-Nearest Neighbors*. R package version 1.2-5.

Shaffer, J. P. (1986). Modified sequentially rejective multiple test procedures. *Journal of the American Statistical Association*, 81(395):826–831.

Sjåfjell, A., Dahl, E., and Skogen, S. (2013). Intelligent transportation systems and artificial intelligence - a state of the art review.

Stathopoulos, A., Dimitriou, L., and Tsekeris, T. (2008). Fuzzy modeling approach for combined forecasting of urban traffic flow. *Computer-Aided Civil and Infrastructure Engineering*, 23(7):521 – 535.

Sun, S. (2009). Traffic flow forecasting based on multitask ensemble learning. pages 961 – 964, Shanghai, China.

Takagi, T. and Sugeno, M. (1985). Fuzzy identification of systems and its applications to modeling and control. *Systems, Man and Cybernetics, IEEE Transactions on*, SMC-15(1):116–132.

Tam, M. L. and Lam, W. H. K. (2007). Real-time travel time estimation using automatic vehicle identification data in hong kong. volume 4413 LNAI, pages 352 – 361, Jeju Island, Korea, Republic of.

Tibshirani, R. (1996). Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society (Series B)*, 58:267–288.

Trawinski, B., Smetek, M., Telec, Z., and Lasota, T. (2012). Nonparametric statistical analysis for multiple comparison of machine learning regression algorithms.

van Hinsbergen, C., van Lint, J., and van Zuylen, H. (2009). Bayesian committee of neural networks to predict travel times with confidence intervals. *Transportation Research Part C: Emerging Technologies*, 17(5):498 – 509.

Van Lint, J. (2008). Online learning solutions for freeway travel time prediction. volume 9, pages 38 – 47.

Vanajakshi, L. and Rilett, L. (2007). Support vector machine technique for the short term prediction of travel time. In *Intelligent Vehicles Symposium, 2007 IEEE*, pages 600–605.

Venables, W. N. and Ripley, B. D. (2002). *Modern Applied Statistics with S*. Springer, New York, fourth edition. ISBN 0-387-95457-0.

Wahl, R. and Haugen, T. (2005). DynamIT - Dynamiske Informasjonstjenester for Transportsektoren, Systembeskrivelse og evaluering. Technical report, SINTEF Teknologi og samfunn, Transportsikkerhet og -informatikk.

Wang, Z. and Poslad, S. (2013). A personalised online travel time prediction model. In *Systems, Man, and Cybernetics (SMC), 2013 IEEE International Conference on*, pages 3327–3332.

Wilcoxon, F. (1945). Individual Comparisons by Ranking Methods. *Biometrics Bulletin*, 1(6):80–83.

Wu, T., Xie, K., Xinpin, D., and Song, G. (2012). A online boosting approach for traffic flow forecasting under abnormal conditions. In *Fuzzy Systems and Knowledge Discovery (FSKD), 2012 9th International Conference on*, pages 2555–2559.

Yang, Z.-L., Li, Y., Song, Y.-W., Chen, X., and Zhang, J. (2012). Empirical study of robust combination of forecasts for short-term highway traffic flow forecast. In *Machine Learning and Cybernetics (ICMLC), 2012 International Conference on*, volume 4, pages 1372–1375.

Yao, X., Fischer, M., and Brown, G. (2001//). Neural network ensembles and their application to traffic flow prediction in telecommunications networks. volume vol.1, pages 693 – 8, Piscataway, NJ, USA.

Zheng, F. and van Zuylen, H. (2013). Trip travel time distribution prediction for urban signalized arterials. In *Intelligent Transportation Systems - (ITSC), 2013 16th International IEEE Conference on*, pages 1829–1834.

Zhu, J. (2011). An ensemble learning short-term traffic flow forecasting with transient traffic regimes. *Applied Mechanics and Materials*, 97-98:849 – 53.

Zhu, J. and Shen, L. (2012). A real-time layered neural network ensemble learning system for short-term traffic prediction. *International Journal of Advancements in Computing Technology*, 4(10):18 – 26.

Zhu, T., Kong, X., and Lv, W. (2009). Large-scale travel time prediction for urban arterial roads based on kalman filter. In *Computational Intelligence and Software Engineering, 2009. CiSE 2009. International Conference on*, pages 1–5.

Zou, H. and Hastie, T. (2012). *elasticnet: Elastic-Net for Sparse Estimation and Sparse PCA*. R package version 1.1.

Øien Lunde, E. and Wolff, T. (2014). Travel time prediction, a state of the art review.

Appendices

A Structured Literature Review Protocol

This section describes in detail how the structured literature review was conducted. Two search engines were used: IEEEXplore[2] and Engineering Village[3].

A.1 Specifying the Search Term

When doing a structured literature review, it is necessary to have a search string that represents the subjects of interest. Having this search string serves two purposes: it makes it possible to reproduce the search results given the archive(s) used. In addition it captures the different aspects of the research, and therefore yields a reduced number of results, hopefully with the relevant literature.

The search string is built up using three groups of terms. The different terms contained in one group are synonyms, have the same semantics or cover similar concepts. The terms in one group are joined using the logical OR operator, and each group is joined using the logical AND operator. In this way the search string is meant to yield the research that is the intersection of the different groups. The search string used is:

$$(\text{“prediction”}\ OR\ \text{“forecasting”}\ OR\ \text{“estimation”})$$
$$AND$$
$$(\text{“travel time”}\ OR\ \text{“transit time”}\ OR\ \text{“driving time”}\ OR\ \text{“traffic flow”}\ OR\ \text{“congestion”})$$
$$AND$$
$$(\text{“ensemble learning”}\ OR\ \text{“machine learning”}\ OR\ \text{“artificial intelligence”})$$

By examining this search string one can identify three aspects that this research focuses on: prediction or estimation of traffic variables using ensemble learning or machine learning in general.

A.2 Search Results

The search was conducted on the 24th of September, 2014. Engineering Village returned 670 results and IEEE returned 477 results. Because many of the articles are unlikely to be suitable for this research, a filtering process is employed to narrow down the number of articles being read in its entirety.

The filtering process consists of three stages: filtering by title, filtering by abstract and finally filtering by full text. In this way irrelevant literature is filtered out as early in the process as possible. The next sections explains in more detail how the filtering stages are executed.

[2] `http://ieeexplore.ieee.org/search/advsearch.jsp?expression-builder`
[3] `http://www.engineeringvillage.com/search/expert.url?CID=expertsearch`

A.3 Filtering by Title

This stage filters the articles by looking at their title. In this way articles that obviously do not concern the current research are excluded, and do not go further in the filtering process. An article is included if its title indicate that it:

Inclusion Criterion 1 Is about Intelligent Transportation Systems

or

Inclusion Criterion 2 Is about ensemble learning

or

Inclusion Criterion 3 Predicts, estimates or models road traffic variables, e.g. traffic flow, speed, congestion or travel time.

Inclusion criterion 1 ensures that articles in the ITS domain are included. These articles are valuable to this research, as they can give insight to the domain in general and contributes to the pool of knowledge in the area. By using inclusion criterion 2 articles about ensemble learning are included. The reason that research done on ensemble learning in the ITS domain is interesting, is that it that an ad hoc search prior to this literature review showed that ensemble learning is a possible direction to go in to achieve more accurate predictions [Russell and Norvig, 2010]. Inclusion criterion 3 ensures that research done on predicting or estimating traffic variables no matter what method that is employed is included. After filtering the articles by these three criteria, 401 articles remained in total.

A.4 Filtering by Abstract

After filtering the articles by title, a more thorough approach has to be employed to identify the valuable articles. For an article to be included in the last filtering stage, its abstract has to indicate one or more of the following:

Inclusion Criterion 1 The main focus of the article is using AI method(s) to predict travel time, traffic flow, speed or congestion.

Inclusion Criterion 2 That the article gives insight to which data sources that are relevant in the ITS domain.

Inclusion Criterion 3 The article gives insight to how data aggregation or preprocessing impacts prediction or estimation of travel time, traffic flow, speed of congestion.

Inclusion Criterion 4 That the article solves a problem using ensemble learning.

Inclusion Criterion 5 The article describe a solution that can easily be extended or adapted to fit our research.

Since investigating what machine learning techniques that have been used to predict traffic variables is of interest, inclusion criterion 1 is employed to include articles that would be relevant in that regard.

Preliminary discussions with the NPRA revealed that it would be interesting to use several data sources in the prototype system to increase prediction accuracy. Inclusion criterion 2 ensures that articles that can give insight to what data sources that are relevant to use is included.

There is an ongoing discussion at the NPRA about how to aggregate the data, and how the aggregation method affects the estimation of traffic variables. Currently, the most common technique used for travel times at the NPRA is to use an average of the travel times the last five minutes. Inclusion criterion 3 ensures that any literature that touches upon this issue is included.

As stated in the previous section, ensemble learning is one of the main focuses in this research and inclusion criterion 4 ensures that research that have employed ensemble learning to solve a problem is included.

If there is work that is done in the ITS domain whose solution can be extended or adapted to predicting travel time, we can draw knowledge from their findings. Inclusion criterion 5 ensures that such research is included.

Employing these filtering criteria yielded 294 articles. Given that this work is conducted by two students for a specialization project, it is infeasible due to time limitations, to read 294 articles in the last stage of the process. In order to further reduce the number of articles, a ranking system is developed. The ranking system consists of ten criteria by which the articles are rated. Each criterion is given an integer weight, and the total score of each article is the sum of all the weights that article has received for each criterion. The criteria, with the corresponding weights shown in parentheses, are as follows:

Ranking Criterion 1 The article is poorly written (-5)

Ranking Criterion 2 The article has empirical results ($+3$)

Ranking Criterion 3 The work concerns predicting travel time ($+2$)

Ranking Criterion 4 The models used are based on travel times for road sections, not GPS data etc. ($+2$)

Ranking Criterion 5 The work concerns short term prediction ($+1$)

Ranking Criterion 6 The work concerns urban/signalized roads ($+1$)

Ranking Criterion 7 The work employs online learning ($+1$)

Ranking Criterion 8 The work predicts traffic variables for public transport (−1)

Ranking Criterion 9 The work employs ensemble learning (+4)

Ranking Criterion 10 The work uses multiple data sources (+1)

Because of the previous stages of the filtering process, all the articles given as input to this stage have a certain degree of relevance, and this stage is meant to find the most relevant amongst them. As indicated by the weight of ranking criterion 2, the fact that the articles base their findings on empirical results are important, as this increases the credibility and reproducibility of their work. Ranking criterion 9 shows that articles concerned with ensemble learning are also favorable because they give insight to the one of the main topics in our study. Articles that are poorly written are given a big penalty, as seen by ranking criterion 1, as this makes them hard to read and time is wasted. The rest of the criteria correspond to features that would be beneficial to this research, but not essential.

After ranking all 294 articles using the aforementioned ten criteria, the total scores were in the range [-5, 10]. A threshold of 6 was set, where articles with a score of 6 or higher are included. This yielded 54 articles, a reasonable number of articles to read for two people. The number of articles was further reduced due to duplicates being removed and that some full texts were not available. The final number of articles entering the full text filtering was 36, and can be found in Table A.2 and Table A.3. It can be seen in from the tables that all articles with a total score of 6 or better have empirical results, and most of them predict travel time in addition to having other interesting features like using online learning or employing short term prediction. From Table A.2 and Table A.3 we can also see that articles that do not predict travel time can be included, as long as they have enough other interesting features.

A.5 Filtering by Full Text

In this final filtering stage the goal is to assess the quality of the work done in the remaining articles. In order to do this, another ranking system is developed where each article is evaluated using the following seven quality criteria [Kofod-Petersen, 2014]:

Quality Criterion 1 Are system or algorithmic design decisions justified?

Quality Criterion 2 Is the method/algorithm thoroughly explained?

Quality Criterion 3 Is the experimental procedure thoroughly explained and reproducible?

Quality Criterion 4 Is it clearly stated in the study which other algorithms the study's algorithm(s) have been compared with?

Quality Criterion 5 Are the performance metrics used in the study explained and justified?

Quality Criterion 6 Are the test results thoroughly analysed?

Quality Criterion 7 Does the test evidence support the findings presented?

Six of the above quality criteria are taken directly from [Kofod-Petersen, 2014]. Question 2 is added because it is important that the methods described are reproducible. Each question is given a weighted answer: yes (1), in some degree (0.5) and no (0).

To ensure that the different questions were interpreted in the same way by both students reading the articles, a calibration round was done on 8 of the articles where both students read and evaluated all 8 of them. After agreeing on how the questions should be interpreted, the remaining 30 articles were divided equally and read separately.

After assessing all articles, their score were in the range [2.5, 7]. As this range shows, not all articles had the same level of quality. To ensure a certain level of quality for the articles that are going to be included as our final references, a threshold of 5 is set, where articles with score higher than or equal to this threshold is included as references in our state of the art review. The final articles are shown in Table A.1.

Final Articles	
Ensemble	[Sun, 2009] [Li et al., 2014] [Zhu and Shen, 2012] [Stathopoulos et al., 2008] [van Hinsbergen et al., 2009]
Online	[Liu et al., 2006a] [Liu et al., 2006b] [Van Lint, 2008] [Haworth et al., 2014] [Lu, 2012] [Wu et al., 2012]
Hybrid Approach	[Hofleitner et al., 2012]
Vehicle Infrastructure Integration	[Ma et al., 2012]
Other	[Park et al., 2014] [Bouillet et al., 2013] [Dharia and Adeli, 2003] [Nikovski et al., 2005] [Tam and Lam, 2007] [Vanajakshi and Rilett, 2007] [Mu et al., 2013]

Table A.1: Articles resulting from the structured literature review

Title	RC1	RC2	RC3	RC4	RC5	RC6	RC7	RC8	RC9	RC10	Total Score
Li et al. [2009]	0	3	2	0	0	0	1	0	4	0	10
Ma et al. [2012]	0	3	2	2	0	0	1	0	0	1	9
Rossi et al. [2012]	0	3	2	0	0	0	0	0	4	0	9
van Hinsbergen et al. [2009]	0	3	2	0	0	0	0	0	4	0	9
Zhu [2011]	0	3	0	0	1	0	0	0	4	0	8
Yang et al. [2012]	0	3	0	0	1	0	0	0	4	0	8
Zhu and Shen [2012]	0	3	0	0	1	0	0	0	4	0	8
Chen and Chen [2007]	0	3	0	0	1	0	0	0	4	0	8
Wu et al. [2012]	0	3	0	0	0	0	1	0	4	0	8
Tam and Lam [2007]	0	3	2	2	1	0	0	0	0	0	8
Sun [2009]	0	3	0	0	0	0	0	0	4	0	7
Mu et al. [2013]	0	3	2	0	1	0	0	0	0	1	7
Mak et al. [2009]	0	3	2	0	0	1	0	0	0	1	7
Li et al. [2014]	0	3	0	0	0	0	0	0	4	0	7
Liu et al. [2006a]	0	3	2	0	1	1	0	0	0	0	7
Zheng and van Zuylen [2013]	0	3	2	0	0	1	0	0	0	1	7
Chen and Zhang [2005]	0	3	0	0	0	0	0	0	4	0	7
Deb Nath et al. [2010]	0	3	2	2	0	0	0	0	0	0	7
Dembczynski et al. [2013]	0	3	2	2	0	0	0	0	0	0	7
Yao et al. [2001]	0	3	0	0	0	0	0	0	4	0	7

Table A.2: Articles included in the quality screening process, part I

Title	RC1	RC2	RC3	RC4	RC5	RC6	RC7	RC8	RC9	RC10	Total Score
Bouillet et al. [2013]	0	3	2	0	0	0	1	0	0	1	7
Park et al. [2014]	0	3	0	2	1	0	0	0	0	1	7
Hofleitner et al. [2012]	0	3	2	0	0	1	0	0	0	0	6
Bajwa et al. [2003]	0	3	2	0	1	0	0	0	0	0	6
Dharia and Adeli [2003]	0	3	2	0	0	0	0	0	0	1	6
Zhu et al. [2009]	0	3	2	0	0	1	0	0	0	0	6
Wang and Poslad [2013]	0	3	2	1	0	0	0	0	0	0	6
Vanajakshi and Rilett [2007]	0	3	2	0	1	0	0	0	0	0	6
Ohra et al. [1997]	0	3	2	0	0	0	1	0	0	0	6
Van Lint [2008]	0	3	2	0	0	0	1	0	0	0	6
Hao et al. [2008]	0	3	2	0	0	1	0	0	0	0	6
Haworth et al. [2014]	0	3	2	0	0	1	0	0	0	0	6
Stathopoulos et al. [2008]	0	3	0	0	1	1	1	0	0	0	6
Lu [2012]	0	3	2	0	0	0	1	0	0	0	6
Liu et al. [2006b]	0	3	2	0	0	1	0	0	0	0	6
Nikovski et al. [2005]	0	3	2	0	1	0	0	0	0	0	6

Table A.3: Articles included in the quality screening process, part II

B Experimental Setup Protocol

B.1 Baselines

This section contains descriptions of each baseline method used in Experiment 1. Information about the process applied to every baseline is given in Overview, whilst specific details about each baseline is given in their respective sections.

Overview

The implementation of the baseline methods used in Experiment 1 is done in R [R Core Team, 2014]. Parameter tuning and training for support vector machine, k-nearest neighbors and artificial neural network is done through the library `caret` [Kuhn et al., 2015], which is a library meant to make the process of constructing predictive models easier in R. In this work, `caret` is used as a layer between the authors code in R and the respective implementations of the baseline methods. `caret` provides structures for setting the grid for which parameters are to be searched among, and returns the model with parameters resulting the best performance in terms of RMSE. Tuning and training of the Kalman filter is done with the R library `dlm` [Petris, 2010].

he baselines use the data set were outliers have been removed. In order to find the best set of parameters for each baseline, 10-fold cross-validation on data from January 29, 2015 to February 1, 2015 is used. Each baseline is then trained with the best set of parameters on data from February 5, 2015 to Febrary 25, 2015. The resulting models are then used to generate predictions from Febrary 26, 2015 to March 31, 2015.

Support Vector Machine

The SVM implementation used in this work is from the R library `kernlab` [Karatzoglou et al., 2004]. When using SVMs there is a choice among several kernel functions to use. To determine which kernel to use in the SVM of Experiment 1, a preliminary experiment comparing the RMSE of SVM models using a linear, polynomial and radial basis function kernel functions was conducted. In addition to experimenting with different kernel functions, a grid of parameters is searched for each kernel function in order to find the parameters to be used for the best performing kernel.

For the linear kernel the only parameter to optimize is the cost parameter C. For the polynomial kernel there are three parameters to optimize: the degree of the polynomial *degree*, the cost parameter C and the scale parameter *scale*. For the radial basis function kernel there are two parameters to optimize: σ and the cost parameter C. Tables B.4, B.5 and B.7 presents the range of parameters

C	RMSE
2^{-5}	0.1594510
2^{-1}	0.1594400
2	0.1594396
2^{5}	**0.1594387**
2^{10}	0.1604016
2^{15}	0.1843528

Table B.4: Parameter tuning for Linear SVM

used during the parameter tuning of the SVM models using a linear, polynomial and radial basis function kernel functions, respectively. Table B.6 summarizes the results from the different degrees used in polynomial kernel function.

Table B.8 summarizes the results of running the best performing models for each kernel type on the test set. The SVM with a radial basis function kernel function yielded the lowest RMSE of 243.3726 and was therefore selected to be used in Experiment 1. Among the parameter values tested in the grid search for the radial basis function SVM models, the parameters $\sigma = 4.1451371$ and $C = 2^{-1}$ yielded the best performing model, and is the one used in Experiment 1.

k-Nearest Neighbors

The kNN implementation used in this work is from the R library `kknn` [Schliep and Hechenbichler, 2014]. The parameters that are possible to tune in `kknn` are k, distance measure and kernel. The k parameter controls how many neighbors to extract from the instances present in the data set when making predictions. The distance measure parameter controls how the distance between two points in the data set is computed. The kernel parameter controls how to weight the values of the k neighbors based on their distance. In order to find a good set of parameters for the weighted kNN algorithm a grid search was performed.

The distance parameter is more specifically the parameter used to calculate the Minkowski distance between two points. In the grid search the values 1 and 2 were selected to use the Manhattan and Euclidean distance, respectively. The kernel is a function representing the distribution of the underlying data set. Since k-NN does not require any training, testing for a large number of combinations is possible. Therefore all the available kernel functions in the kknn library were tested. Small k values makes k-NN sensitive to noise in the data, but makes it easier to distinguish different traffic scenarios from each other. Choosing large k values reduce the algorithms sensitivity to noise, but reduces its ability to detect special cases. Tables B.9 and B.10 shows the parameters tested for each kernel

C	Scale	RMSE
2^{-5}	0.001	0.1678236
2^{-1}	0.001	0.1603247
2	0.001	0.1596821
2^{5}	0.001	0.1594829
2^{10}	0.001	0.1594655
2^{15}	0.001	0.1602348
2^{-5}	0.010	0.1607862
2^{-1}	0.010	0.1595467
2	0.010	0.1594870
2^{5}	0.010	0.1594715
2^{10}	0.010	0.1594667
2^{15}	**0.010**	**0.1590753**
2^{-5}	0.100	0.1596038
2^{-1}	0.100	0.1594786
2	0.100	0.1594710
2^{5}	0.100	0.1594700
2^{10}	0.100	0.1594310
2^{15}	0.100	0.2060724

First degree polynomial

C	Scale	RMSE
2^{-5}	0.001	0.1648976
2^{-1}	0.001	0.1597790
2	0.001	0.1591884
2^{5}	0.001	0.1577153
2^{10}	0.001	0.1572685
2^{15}	0.001	0.1576086
2^{-5}	0.010	0.1595339
2^{-1}	0.010	0.1575453
2	0.010	0.1573096
2^{5}	0.010	0.1572620
2^{10}	0.010	0.1572505
2^{15}	**0.010**	**0.1570599**
2^{-5}	0.100	0.1573271
2^{-1}	0.100	0.1572610
2	0.100	0.1572575
2^{5}	0.100	0.1572554
2^{10}	0.100	0.1573637
2^{15}	0.100	0.1835173

Second degree polynomial

C	Scale	RMSE
2^{-5}	0.001	0.1632949
2^{-1}	0.001	0.1594510
2	0.001	0.1586526
2^{5}	0.001	0.1573733
2^{10}	0.001	0.1569685
2^{15}	**0.001**	**0.1564797**
2^{-5}	0.010	0.1586273
2^{-1}	0.010	0.1571663
2	0.010	0.1568624
2^{5}	0.010	0.1565607
2^{10}	0.010	0.1565122
2^{15}	0.010	0.1628453
2^{-5}	0.100	0.1566293
2^{-1}	0.100	0.1565251
2	0.100	0.1565129
2^{5}	0.100	0.1565076
2^{10}	0.100	0.1576626
2^{15}	0.100	0.1660079

Third degree polynomial

Table B.5: Parameter tuning for Polynomial SVM

Degree	C	Scale	RMSE
1	2^{15}	0.010	0.1590753
2	2^{15}	0.010	0.1570599
3	$\mathbf{2^{15}}$	**0.001**	**0.1564797**

Table B.6: Summary of best performing parameters per degree for Polynomial SVM

σ	C	RMSE
0.1005521	2^{-5}	0.1560983
0.1005521	2^{-1}	0.1558789
0.1005521	2	0.1558185
0.1005521	2^{5}	0.1556929
0.1005521	2^{10}	0.1557148
0.1005521	2^{15}	0.1565749
0.5023972	2^{-5}	0.1558294
0.5023972	2^{-1}	0.1556032
0.5023972	2	0.1555518
0.5023972	2^{5}	0.1554235
0.5023972	2^{10}	0.1554382
0.5023972	2^{15}	0.1575169
4.1451371	2^{-5}	0.1560392
4.1451371	$\mathbf{2^{-1}}$	**0.1549082**
4.1451371	2	0.1549102
4.1451371	2^{5}	0.1552377
4.1451371	2^{10}	0.1564157
4.1451371	2^{15}	0.1647936

Table B.7: Parameter tuning for Radial SVM

Kernel	Parameters	RMSE
Linear	C = 2^5	248.3869
Polynomial	Degree = 3, C = 2^{15}, Scale = 0.001	243.3781
Radial Basis Function	**Sigma = 4.1451371, C = $\mathbf{2^{-1}}$**	**243.3726**

Table B.8: Summary of the best performing SVM models per kernel type

type. Notice that the best performing set of parameters per kernel is highlighted with bold font. Table B.11 lists the best performing set of parameters for each kernel, and enhances the best performing set of parameters across all parameters with bold font. The following parameters yielded the lowest RMSE of 0.1501901: $k = 50$, distance measure = 1 (Euclidean) and kernel = Rank.

Artificial Neural Network

The ANN implementation used in Experiment 1 is from the R library `nnet` [Venables and Ripley, 2002]. The `nnet` library provides functions for creating and training feed forward ANNs with a single hidden layer. The parameters being tunes for the ANN is the number of hidden nodes in the network and the weight decay parameter. Table B.12 presents the different values tested during the parameter tuning for ANN, and the resulting RMSE for each combination of parameters. The parameters yielding lowest RMSE of 0.1497285 was: number of hidden nodes = 16 and decay= 1×10^{-4}, highlighted with bold font in Table B.12.

Kalman Filter

`dlm` is a R library providing functions for defining dynamic linear models of various types, which is used to perform the Kalman Filter predictions in Experiment 1. The actual travel times of the data set forms a time series, and it is assumed that this time series of travel times can be modelled using a first order linear model with the following state space formulation:

$$\begin{cases} y_t = \theta_t + v_t, v_t \sim \mathcal{N}(0, V_t) \\ \theta_t = \theta_{t-1} + w_t, w_t \sim \mathcal{N}(0, W_t) \end{cases} \tag{1}$$

where y_t is the observed travel time at time t, θ_t is the *actual* travel time at time t, which is assumed to be unobservable, v_t is the observation noise and w_t is the process noise. Both v_t and w_t are assumed to follow a zero-mean Gaussian distribution with covariance matrices V_t and W_t, respectively.

The parameters that are tuned for this model are the covariance matrices V_t and W_t. This is done through a grid search using maximum likelihood estimation. The `dlm` library provides a function for finding the set of parameters α having the highest probability given a set of observations x. The optimal set of parameters $\hat{\alpha}$ is found through searching through a grid of possible α values, and choosing the one maximizing the log-likelihood of α given the observations x. The observations that are given to find these parameters is the same set of training data that the other baseline models are given.

Kmax	Distance	RMSE
3	1	0.1703484
3	2	0.1701785
5	1	0.1610623
5	2	0.1613434
7	1	0.1575735
7	2	0.1577541
10	1	0.1546021
10	2	0.1547042
20	1	0.1517638
20	2	0.1517947
50	**1**	**0.1502186**
50	2	0.1502300

Rectangular

Kmax	Distance	RMSE
3	1	0.1781971
3	2	0.1786599
5	1	0.1669801
5	2	0.1680861
7	1	0.1614506
7	2	0.1624606
10	1	0.1574673
10	2	0.1581775
20	1	0.1531711
20	2	0.1536536
50	**1**	**0.1503567**
50	2	0.1506450

Triangular

Kmax	Distance	RMSE
3	1	0.1837144
3	2	0.1842058
5	1	0.1728479
5	2	0.1739973
7	1	0.1660075
7	2	0.1675166
10	1	0.1608040
10	2	0.1620943
20	1	0.1549380
20	2	0.1558122
50	**1**	**0.1512792**
50	2	0.1519595

Triweight

Kmax	Distance	RMSE
3	1	0.1761805
3	2	0.1761574
5	1	0.1653235
5	2	0.1654804
7	1	0.1603163
7	2	0.1603996
10	1	0.1566807
10	2	0.1567659
20	1	0.1525314
20	2	0.1525958
50	**1**	**0.1502085**
50	2	0.1502746

Optimal

Kmax	Distance	RMSE
3	1	0.1764503
3	2	0.1769599
5	1	0.1651597
5	2	0.1663247
7	1	0.1600756
7	2	0.1610494
10	1	0.1564606
10	2	0.1571335
20	1	0.1527153
20	2	0.1531223
50	**1**	**0.1502012**
50	2	0.1504361

Epanechnikov

Kmax	Distance	RMSE
3	1	0.1770035
3	2	0.1775128
5	1	0.1657162
5	2	0.1668814
7	1	0.1604792
7	2	0.1614875
10	1	0.1567521
10	2	0.1574551
20	1	0.1528490
20	2	0.1532873
50	**1**	**0.1502529**
50	2	0.1505107

Cos

Table B.9: kNN Parameter Tuning Results pt. 1

Kmax	Distance	RMSE
3	1	0.1837840
3	2	0.1836059
5	1	0.1781395
5	2	0.1781810
7	1	0.1755044
7	2	0.1754793
10	1	0.1733839
10	2	0.1732052
20	1	0.1703310
20	2	0.1700676
50	1	0.1675241
50	**2**	**0.1671295**

Inv

Kmax	Distance	RMSE
3	1	0.1734786
3	2	0.1733456
5	1	0.1640946
5	2	0.1642367
7	1	0.1596231
7	2	0.1597906
10	1	0.1562839
10	2	0.1563961
20	1	0.1524152
20	2	0.1524823
50	**1**	**0.1501901**
50	2	0.1502636

Rank

Kmax	Distance	RMSE
3	1	0.1709502
3	2	0.1708552
5	1	0.1620715
5	2	0.1624392
7	1	0.1583586
7	2	0.1587008
10	1	0.1555217
10	2	0.1557561
20	1	0.1524583
20	2	0.1526986
50	**1**	**0.1504290**
50	2	0.1506819

Gaussian

Table B.10: kNN Parameter Tuning Results pt. 2

Kmax	Distance	Kernel	RMSE
50	1	Rectangular	0.1502186
50	1	Optimal	0.1502085
50	1	Triangular	0.1503567
50	1	Epanechnikov	0.1502012
50	1	Triweight	0.1512792
50	1	Cos	0.1502529
50	2	Inv	0.1671295
50	1	Gaussian	0.1504290
50	**1**	**Rank**	**0.1501901**

Table B.11: Summary of best performing parameters per kernel for kNN

Number of hidden nodes	Weight Decay	RMSE
1	0	0.2270246
1	1×10^{-4}	0.1504067
1	1×10^{-1}	0.1513350
2	0	0.2270246
2	1×10^{-4}	0.1580870
2	1×10^{-1}	0.1509677
4	0	0.2192657
4	1×10^{-4}	0.1498524
4	1×10^{-1}	0.1509192
8	0	0.2111320
8	1×10^{-4}	0.1499537
8	1×10^{-1}	0.1510321
16	0	0.2195452
16	$\mathbf{1 \times 10^{-4}}$	**0.1497285**
16	1×10^{-1}	0.1510171

Table B.12: Parameter tuning for ANN

$$\hat{\alpha} = \max_{\alpha} \mathcal{L}(\alpha|x)$$

Since y_t and θ_t are univariate, matrices V_t and W_t are 1×1 matrices, and only one value per covariance matrix is tuned. V_t and W_t are set to 47939 and 122, respectively. In order to give a complete definition of the dynamic linear model, an initial estimate of $y_{t=0}$ and its variance $\sigma^2_{t=0}$ has to be provided. This is set to be the mean and standard deviation of the training observations, which are 242 and 255, respectively.

B.2 Ensemble Learning Methods

The setup of each ensemble method used in Experiment 1 is presented in this section. The general procedure is described in Overview, whilst specific details concerning the setup of each ensemble method can be found in their respective sections.

Overview

Bagging and Boosting train multiple SVMs on the same data set as the baselines described in Section 4.2.2. Data from February 5, 2015 to Febrary 25, 2015 and generate predictions on data from February 26, 2015 to March 31, 2015. Due to the computational complexity of training SVMs, no parameter tuning is conducted for Bagging and Boosting.

Lasso and FRBS use the predictions of the baselines described in Section 4.2.2 as input. Lasso is tuned and trained on data from February 26, 2015 to March 18, 2015 and uses data from March 19, 2015 to March 31, 2015 to generate predictions. Due to the computational complexity of FRBS, only one week of data from February 26, 2015 to March 4, 2015 is used to search for the best set of parameters. As the rules in the FRBS are manually created, no training phase is necessary. Predictions are generated from March 5, 2015 to March 31, 2015.

Bagging

The Bagging method is implemented in R. There are two parameters that must be specified when doing Bagging of a baseline learner: the number of learners K to use in the ensemble, and the number of samples N to do for each learner. In Experiment 1, K is set to 25, and N is set to the number of training examples in the training set. A SVM model with the parameters of the best performing model from the parameter tuning step for SVM reported in Section B.1 is used, i.e. a SVM model with a Gaussian radial basis function as kernel, $\sigma = 4.1451371$ and $C = 2^{-1}$.

Boosting

In order to test the effects of boosting, the AdaBoost algorithm is used. An implementation of the AdaBoost.r2 algorithm [Drucker, 1997], which is a version of the AdaBoost algorithm [Freund and Schapire, 1997] for regression problems, is found in the Python library `scikit-learn` [Pedregosa et al., 2011]. Of the baseline methods used in this work, the only one that is supported in `scikit-learn` is SVM, and is therefore the baseline used for the boosting ensemble in Experiment 1. A SVM model with the same parameters as the one reported in Section B.2 is used for the Boosting approach. The only parameter provided to the AdaBoost algorithm is the number of learners K to use in the ensemble, which is set to 25 in Experiment 1.

Lasso Ensemble

The implementation of the Lasso method is taken from the R library `elasticnet` Zou and Hastie [2012]. The `elasticnet` library provides functions for for defining elastic nets, of which the Lasso method is a special case. Recall from Section 3.1.3 that the optimal weights in Lasso is given by solving the following equation:

$$\hat{\mathbf{W}} = \arg\min_{\mathbf{W}} \|\mathbf{F} - \mathbf{G}\mathbf{W}\|_2^2 + \lambda\|\mathbf{W}\|_1$$

where λ is a parameter of the Lasso method. `caret` provides functionality for optimizing this λ parameter through a grid search, where RMSE is used to assess the performance of a given λ value. `elasticnet`, which `caret` uses to perform the Lasso approach, requires that this λ value is controlled by specifying a fraction f which represents how much the weight regularization term contributes to the sum relative to the norm of the full least squares solution. A search grid of values from 0.05 up to 1 in steps of 0.05 is used in the parameter tuning step for Lasso. Table B.13 summarizes the performance for the different fraction values tested during the parameter tuning step. Having the fraction parameter set to 1 resulted in the lowest RMSE of 169.4987, and is the parameter used in Experiment 1.

Fraction	RMSE
0.05	228.9219
0.10	220.7864
0.15	213.0905
0.20	205.8835
0.25	199.2188
0.30	193.1524
0.35	187.7423
0.40	183.0462
0.45	179.1200
0.50	176.0145
0.55	173.7729
0.60	172.4239
0.65	171.6533
0.70	170.9984
0.75	170.4612
0.80	170.0428
0.85	169.7440
0.90	169.5771
0.95	169.5259
1.00	169.4987

Table B.13: Parameter tuning for Lasso Ensemble

Fuzzy Rule Based System

The R library `frbs` [Riza et al., 2015] is used to generate a Fuzzy Rule Based System (FRBS) based on Stathopoulos et al. [2008]. The input to the FRBS is a data set containing predictions from the artificial neural network and the Kalman filter described above. As in Stathopoulos et al. [2008], two different rule bases, each preferring one of the two baselines, are used. The rule base preferring ANN is displayed in Table B.14. The rule base preferring Kalman filter is similar, except that it gives priority to the Kalman filter's predictions. The travel time predictions from the baselines are mapped with triangular membership functions to three fuzzy sets; low, medium, and high. The values of the membership functions are optimized with the `constrOptim` function in the `stats` library [R Core Team, 2015], in which the Nelder-Mead [Nelder and Mead, 1965] algorithm is used to search for an optimal set of parameters bounded by the constraints shown in Table B.15. Data from February 26, 2015 to March 4, 2015 is used to tune the FRBS. The final set of membership function parameters for each rule base can be found in Table B.16 and Table B.17, and illustrations of the

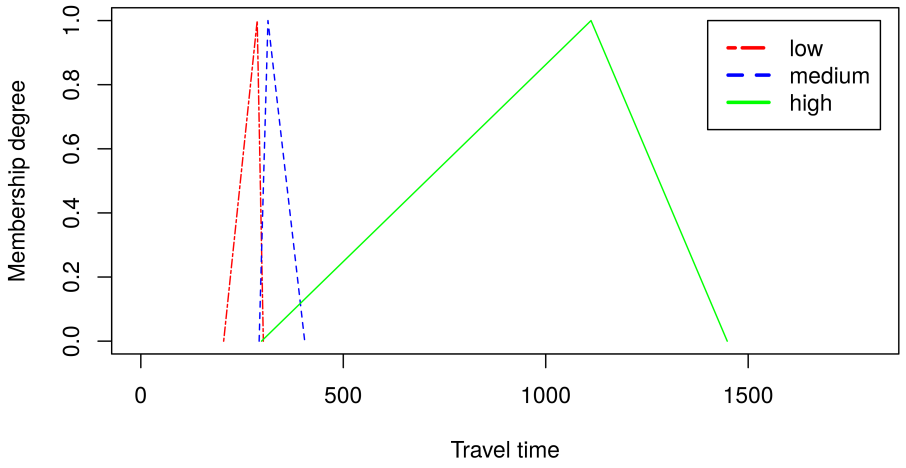

Figure B.1: Illustration of membership functions for FRBS preferring ANN

membership functions are given in Figure B.1 and Figure B.2.

B.3 Online Learning Methods

This section presents the setup of the online methods used in Experiment 2. The general approach is described in Overview, whilst specific details for the individual methods are presented in their respective sections.

IF ANN is *low* and KF is *low* **THEN** output is *low*
IF ANN is *low* and KF is *not low* **THEN** output is *low*
IF ANN is *medium* and KF is *medium* **THEN** output is *medium*
IF ANN is *medium* and KF is *not medium* **THEN** output is *medium*
IF ANN is *high* and KF is *high* **THEN** output is *high*
IF ANN is *high* and KF is *not high* **THEN** output is *high*

Table B.14: Rule base preferring ANN over Kalman filter (KF)

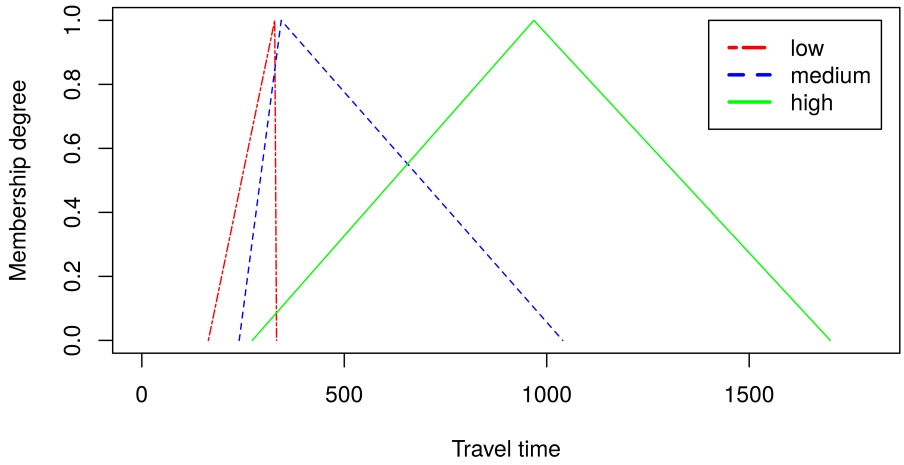

Figure B.2: Illustration of membership functions for FRBS preferring Kalman filter

low_a	$\geq$	min
low_b	$\geq$	low_a
low_c	$\geq$	low_b
low_c	$\geq$	$medium_a$
$medium_a$	$\geq$	low_a
$medium_b$	$\geq$	$medium_a$
$medium_c$	$\geq$	$medium_b$
$medium_c$	$\geq$	$high_a$
$high_a$	$\geq$	$medium_a$
$high_b$	$\geq$	$high_a$
$high_c$	$\geq$	$high_b$
$high_c$	$\leq$	max

Table B.15: Constraints for the parameters of the membership functions

	low	medium	high
a	204.5799	292.2150	298.0683
b	287.4319	313.9751	1111.8725
c	302.5835	404.7539	1448.2825

Table B.16: Membership function parameters for FRBS preferring ANN

	low	medium	high
a	164.3251	240.4815	272.8217
b	328.2455	344.3591	968.0878
c	332.7156	1039.4697	1700.2693

Table B.17: Membership function parameters for FRBS preferring Kalman Filter

Overview

The online learning methods are tuned and trained on two weeks of data from January 29, 2015 to February 11, 2015. The first week is used to build the models with a certain set of parameters, whilst the second week is used to calculate the RMSE of the predictions done during that week. Since the methods are online, the data in the second week is also used to update the models whilst making predictions. The model built on the set of parameters that lead to the lowest RMSE is used to generate predictions from February 12, 2015 to March 13, 2015. The data set used is not preprocessed, i.e. no outliers are removed.

Online-Delayed Extended Kalman Filter

An implementation of the EKF [Cao, a] and using this EKF to train a feed-forward ANN [Cao, b] is found in Matlab [MATLAB, 2014] on the Matlab Central File Exchange[4]. The state space definition that is assumed can be expressed as follows:

$$\begin{cases} y_t = G(\mathbf{x_t}, \theta_\mathbf{t}) \\ \theta_t = \theta_{t-1} + r_t, r_t \sim \mathcal{N}(0, R_t) \end{cases} \tag{2}$$

where θ_t is the weights in the feed-forward ANN at time t, r_t is the noise with which the weights are assumed to evolve, and G represents the mapping performed by the ANN from inputs x_t to output y_t given weights θ_t. That is, the weights are assumed to follow a random walk, and the ANN does a noisy observation of the weights through its outputs. In the experiments performed in

[4] `http://www.mathworks.com/matlabcentral/fileexchange/`

this study, the input vectors input_t consists of the mean travel time the previous five minutes and traffic volume the previous five minutes. The target values target_t for the output of the ANN is the actual travel time for each vehicle.

$$\text{input}_t = [\text{mean travel time}, \text{traffic volume}]$$

$$\text{target}_t = \text{actual travel time}$$

Recall from Section 3.2.1 that an initial estimate of the state vector $\mathbf{x_{t=0}}$, its covariance matrix $\mathbf{P_{t=0}}$, process noise covariance matrix $\mathbf{Q}$, and observation noise covariance matrix $\mathbf{R}$ has to be defined in order to run the Extended Kalman Filter. For the way EKF is used in the experiments in this study, this means that an initial estimate of the weights θ_{initial} has to be provided. The approach described in Van Lint [2008] is that the weights θ_{initial} are initialized using the Nguyen-Widrow method [Nguyen and Widrow, 1990]. In Van Lint [2008], the weights' covariance matrix $\mathbf{P}_{\text{initial}}$ is initialized to a diagonal matrix with large values, reflecting that it is assumed that the weights are independent, and that there is a large uncertainty connected to the initial guess of the weights. These are also the approaches taken in this work, and the value along the diagonal of θ_{initial} is set to 10000.

As Van Lint [2008] does not describe how the covariance matrices $\mathbf{Q}$ and R is set, a parameter tuning step is performed to find reasonable values for $\mathbf{Q}$ and $\mathbf{R}$, in addition to the number of nodes in the hidden layer in the feed-forward ANN. The covariance matrix $\mathbf{Q}$ is assumed to be a diagonal matrix, once more reflecting that the weights are independent, such that the only value being search for regarding $\mathbf{Q}$ in the parameter tuning step, is the value q along the diagonal of $\mathbf{Q}$.

$$\mathbf{Q} = q \times \mathbf{I}$$

As the output y_t consists of a single value, namely the travel time, the covariance matrix $\mathbf{R}$ of the observation noise is a single element r reflecting the noise related to the observations that the ANN does of the weights.

The parameter tuning for Online-Delayed EKF is run on two weeks of data from January 19, 2015 to February 11, 2015. Given a set of parameters the Online-Delayed EKF is trained on the first week of data from January 19, 2015 to February 4, 2015. Next, the model is used to train on the second week of data from Februar 5, 2015 to February 11, 2015, and the predictions made during this final training phase is used as a basis for computing the RMSE for that combination of parameters. The parameters yielding the lowest RMSE on the test set is the ones used for the Online-Delayed EKF model in Experiment 2.

Table B.18 illustrates the values for each parameter that is evaluated during the parameter tuning. All combinations of the parameters is evaluated. As

q	r	Number of hidden nodes
0.1	10	1
0.01	25	2
0.001	50	4
0.0001	100	8
	250	16
	500	32
	750	
	1000	

Table B.18: Values tested during parameter tuning for Online-Delayed Extended Kalman Filter

this leads to 192 combinations, only the combination yielding lowest RMSE is repeated here: $q = 0.1$, $r = 750$, number of hidden nodes = 1. This set of parameters yielded a RMSE = 406.42. The RMSEs during the parameter tuning was in the range $[406.42, 735.13]$. Notice that the RMSE of this parameter tuning step lies within another range than the ones resulting from the parameter tuning for the baselines in Experiment 1. This is due to the fact that the RMSE values reported from the parameter tuning for the baselines in Experiment 1 are computed using normalized travel times. This is not the case for Online-Delayed EKF, where the RMSE is computed on non-normalized values.

LOKRR

To find optimal parameters for each kernel two weeks of data from January 29, 2015 to February 11, 2015 was used. The first week was used to train the kernels on different parameter values and the last week was used to make predictions. For each pair of parameter values, the kernel was trained (i.e. the inverse of the regularized kernel matrix mas calculated) on the first week of data and predictions were made on the second week of data. The observations in the test set were also used to update the kernel to simulate how LOKRR would normally work. The $\mathbf{X}$ matrix was reset to contain only the training data before every run with new parameters. This way the best pair of parameters could be found for each kernel. The recommendations for finding possible parameter values for σ and λ presented in the article were followed. The approach the authors recommend for finding possible λ values is based on Exterkate [2013]. First, the R^2 of an ordinary least squares fit of $\mathbf{y}$ on $\mathbf{X}$ is found. Then λ_0 is determined as $\lambda_0 = 1/\phi_0$, where $\phi_0 = R^2/(1 - R^2)$. The recommended values for λ is $\{1/8\lambda_0, 1/4\lambda_0, 1/2\lambda_0, \lambda_0, 2\lambda_0\}$. The recommended approach for finding possible σ parameters is based on that optimal values of σ lie in the range between the 0.1 and 0.9 quantiles of the

pairwise euclidean distance between the points in the kernel [Caputo et al., 2002]. The 0.25, 0.5 and 0.75 quantiles were used as possible values for σ.

No open source implementation of LOKRR was found, so a n implementation of LOKRR was developed in Python [Rossum, 1995] from scratch. Several issues were encountered during this process.

Data Set 1 contains individual travel times as opposed to Haworth et al.'s experiments with five minute aggregated data. Since the amount of traffic is a lot bigger during the day than during the night, the kernels in our LOKRR implementation ended up with different sizes. An upper bound on how much data one kernel could contain was not set, but the method was slightly changed for kernels with little data. In the article the kernel sizes are kept static, meaning a data point is removed from the kernel as soon as a new point is added. For kernels with number of observations below some threshold (200) a decision was made not to remove the oldest observation. This way kernels responsible for intervals with low traffic were aloud to grow also during the testing and verification stages. Otherwise the kernels responsible for intervals with low traffic would be limited by the amount of observations in the training set. By adding this functionality kernels with less than 200 observations during the training period could keep data further back in time than the other kernels when new observations were made. This was done in hope of increasing prediction accuracy.

It was also discovered that the regularized kernel matrix sometimes ended up with a determinant of zero, in which case an inverse can not be found. To avoid getting an error during run time when attempting to calculate the inverse of a singular matrix, a check was added to check if adding a new point would cause a determinant of zero in the regularized kernel matrix. In those cases the point was not added to the kernel. This means that some observations may have been skipped by LOKRR during the testing and verification stage.

During parameter tuning two other issues were encountered. In cases where a kernel only has two points in its $\mathbf{X}$ matrix, the R squared of an ordinary least squares fit of $\mathbf{y}$ on $\mathbf{X}$ equals 1. This causes division by zero when calculating ϕ. Kernels with a high percentage of similar observations also caused problems when selecting parameter values for σ. Similar observations means that the euclidean distance between them is zero. This led to one or more of the possible σ values to become zero, which again caused division by zero when calculating the kernel matrix. These two issues seemed to occur during intervals at night with few observations in the kernels. Therefore a decision was made to only consider intervals with a significant amount of traffic such that the probability of these events happening was minimized. The data set used for LOKRR contained observations from 06:00 to 21:00. Additionally, the window size is set to 1. This is done to reduce the computational complexity.