NTNU

Norwegian University of
Science and Technology

Predicting Bike-Sharing Traffic Flow using Machine Learning

Paul Philip Mitchell

Master of Science in Informatics
Submission date: May 2018
Supervisor: Trond Aalberg, IDI
Co-supervisor: Hans Martin Espegren, UIP

Norwegian University of Science and Technology
Department of Computer Science

Paul Philip Mitchell

Predicting Bike-Sharing Traffic Flow using Machine Learning

Data and Artificial Intelligence (DART) Group
Department of Computer Science
Faculty of Information Technology and Electrical Engineering

Abstract

Bike-Sharing Systems (BSSs) have rapidly grown in popularity worldwide in recent years. The driving forces of this explosive growth are attributed to access to modern ubiquitous technology, increased urbanization, a desire to decrease pollution, and the need for flexible and integrated mobility in city environments. However, in order to keep BSSs in a balanced state where bikes and stations are readily available for users, companies are seeking ways to accurately predict future demand.

This project aims to objectively compare and evaluate various machine learning algorithms for the problem of predicting cluster-level BSS traffic flow. All models are evaluated using two different station clustering techniques, zone-based and grid-based, representing ways to group stations both with and without expert knowledge of the system.

Furthermore, this thesis presents a rigorous state-of-the-art review of current research on demand prediction in BSSs and other on-demand transport services. Random Forest (RF), Feed-Forward Neural Network (FFNN) and Deep Residual Network (ResNet) emerged as the most promising models from this review, and were therefore implemented. Additionally, this thesis presents the first evaluation of using Recurrent Neural Networks within the context of BSSs demand prediction. FFNN proved to be the most successful model, reaching 21.1% and 36.65% improvements over the best baseline using grid-based- and zone-based clustering respectively. The developed system is designed to run in a cloud production environment, and elaborations are made as to how it can be extended to handle real-time streaming data from Oslo City Bike users.

Sammendrag

Bysykkelsystemer har opplevd en eksplosiv vekst verden rundt de siste årene. Drivkreftene bak denne enorme utviklingen har vært tilgang til moderne teknologi, økt urbanisering, et ønske om å redusere forurensning og behov for fleksibel og integrert mobilitet i bymiljøer.

Dette prosjektet tar sikte på å objektivt sammenligne og evaluere en rekke maskinlæringsalgoritmer for å predikere bysykkelflyt på klyngenivå. Alle modeller er evaluert ved å bruke to forskjellige teknikker for å gruppere sykkelstasjoner i klynger, sone-basert og rutenett-basert, som representerer måter å gruppere stasjoner både med og uten ekspertisekunnskap om systemet.

Denne masteroppgaven presenterer en grundig gjennomgang av dagens forskning på etterspørselsprediksjon i bysykkelsystemer og andre transporttjenester. Random Forests (RF), Feed-Forward Neural Networks (FFNNs) og Deep Residual Networks (ResNets) viste seg å være de mest lovende modellene fra denne gjennomgangen, og ble derfor implementert. I tillegg fremlegges den første evalueringen av å bruke Recurrent Neural Networks (RNNs) i kontekst av etterspørselsprediksjon i bysykkelsystemer i denne oppgaven. FFNN var den mest vellykkede modellen, og nådde 21.1% og 36.65% forbedring over den beste baselinemodellen for henholdsvis rutenett-basert og sone-basert gruppering. Det utviklede systemet er designet for å kjøre i et skybasert produksjonsmiljø, og utdypninger foreligger for hvordan systemet kan utvides til å håndtere strømmende sanntidsdata fra Oslo Bysykkel sine brukere.

Preface

This thesis was written for the Department of Computer Science (IDI) at the Norwegian University of Science and Technology (NTNU). The project aims to objectively compare and evaluate machine learning algorithms for the problem of predicting flow in Bike-Sharing Systems (BSSs). The data used was provided by Oslo City Bike, and the results of this thesis are likely to be included in their production systems to provide accurate predictions of flow. Trond Aalberg (IDI) was the supervisor for this thesis, and Hans Martin Espegren (UIP) served as a co-supervisor. I would like to thank them both for being invaluable resources throughout this project. I would also like to thank family and friends for endless support.

Paul Philip Mitchell
Trondheim, May 26, 2018

Contents

List of Figures

List of Tables

List of Abbreviations

AdaBoost Adaptive Boosting. 32, 35

Adagrad Adaptive Gradient algorithm. 14, 74

Adam Adaptive Moment Estimation. 14

AI Artificial Intelligence. 9

ANN Artificial Neural Network. 11, 34, 35, 38, 41

ARIMA Autoregressive Integrated Moving Average. 33, 36, 39, 41, 79

ARMA Autoregressive Moving Average. 33, 34, 40

BN Bayesian Network. 33, 79

BO Bayesian Optimization. 25, 53, 54, 74, 75

BPTT Back-Propagation Through Time. 18

BSS Bike-Sharing System. 1, 2, 4, 5, 7, 8, 12, 29–32, 35–39, 41, 43, 49, 50, 52, 54, 59, 73–76, 78, 79, 81, 83, 86

CART Classification and Regression Tree. 23, 35

CNN Convolutional Neural Network. 8, 20–22, 40, 41, 48, 49, 57

DNN Deep Neural Network. 52, 57, 58, 63, 66, 74–78, 80–83

DTW Dynamic Time Warping. 36

EI Expected Improvement. 25

FFNN Feed-Forward Neural Network. 14, 17, 20, 32, 48, 50, 52, 54, 79

GAM Generalized Additive Model. 33, 34, 37

GBDT Gradient Boosted Decision Tree. 32, 39, 40

GRNN Generalized Regression Neural Network. 35, 79

GRU Gated Recurrent Unit. 18–20, 50, 54, 75

HA Historical Average. 34, 39, 40, 51, 76

ILSVRC ImageNet Large Scale Visual Recognition Challenge. 21, 74

KNN K-Nearest Neighbors. 37

LASSO Least Absolute Shrinkage and Selection Operator. 39

LSBoost Least Squares Boosting. 34, 38

LSTM Long Short-Term Memory. 18, 20, 41, 50, 54, 75

LV Last Value. 33, 34, 52, 76, 77, 80, 81

MAE Mean Absolute Error. 34, 36, 39, 59

ML Machine Learning. 2, 9, 50, 55, 73, 75–77, 79, 80, 82

MLP Multi-Layer Perceptron. 13

MLR Multiple Linear Regression. 38

MNIST Mixed National Institute of Standards and Technology (database of handwritten digits). 10

MSE Mean Squared Error. 14

NAB Normalized Available Bikes. 34

NAS Normalized Activity Score. 34

PLSR Partial Least Squares Regression. 36

POI Point-Of-Interest. 31

R2 Coefficient of Determination. 35

RA Random Algorithm. 50, 51, 76

RBF Radial Basis Function. 35

ReLU Rectified Linear Unit. 18

ResNet Deep Residual Network. 22, 23, 39, 41, 47–50, 54–58, 61, 63, 68, 74, 77–82

RF Random Forest. 7, 23, 32, 34, 37–39, 48, 50, 52, 54, 55, 57, 63, 68, 74, 76–82

RMSE Root Mean Squared Error. 32, 34, 39, 59, 66, 68, 75–78, 80

RMSLE Root Mean Squared Logarithmic Error. 36

RMSProp Root Mean Square Propagation. 14, 24, 74

RNN Recurrent Neural Network. 14, 17, 18, 20, 41, 48, 50, 54, 56–58, 61, 63, 66, 68, 74–82

RQ Research Question. 4

SARIMA Seasonal Autoregressive Integrated Moving Average. 31, 41

SGD Stochastic Gradient Descent. 14, 24, 35

SLR Structured Literature Review. 5, 27, 54, 79

SQL Structured Query Language. 44

SVM Support Vector Machine. 35, 37

SVR Support Vector Regression. 32, 35

Tanh Hyperbolic Tangent. 18

XGBoost Extreme Gradient Boosting. 36, 39

Chapter 1

Introduction

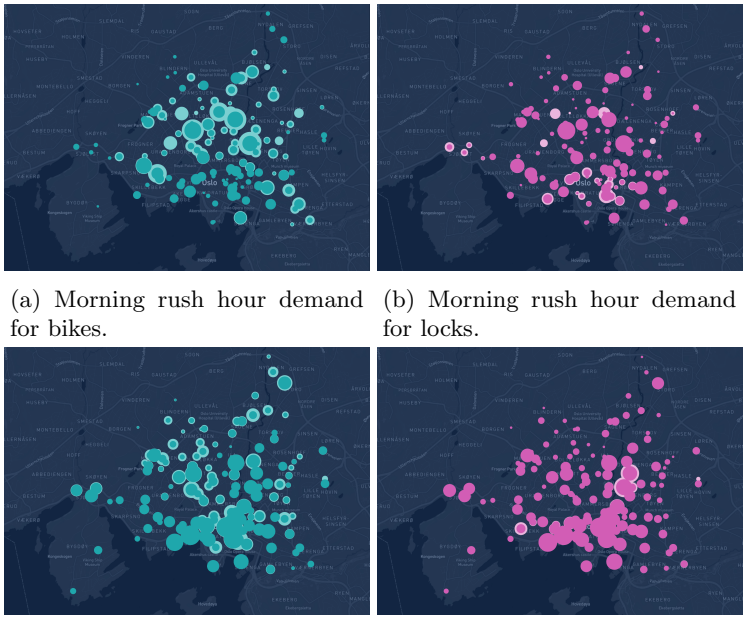
In this thesis, the problem of predicting the flow[1] of bikes in a bike-sharing system is explored through the use of machine learning algorithms. Several machine learning algorithms are evaluated and compared, and are implemented based on state-of-the-art techniques from the literature. This chapter introduces the background and motivation for carrying out this research, followed by a description of the goals and research questions defined, and the research method applied. A short summary of the contributions of this thesis is also presented, before finally the outlining of the thesis structure is described.

1.1 Background and Motivation

As major cities are increasingly facing difficulties regarding energy dependence, air pollution and climate change issues, governments are seeking alternative environmentally friendly means of transportation. Bike-Sharing Systems (BSSs) have grown exponentially in popularity in recent years, spreading from 17 BSSs worldwide in 2005 to more than 2000 currently operating or under planning or construction in 2018 (Metrobike [2012], Meddin and DeMaio [2018]). BSSs offer users a flexible, efficient and affordable means of transportation, while simultaneously reducing CO_2-emission and providing health benefits. According to Shaheen et al. [2010], for every kilometer someone rides a bike instead of driving a car, about 0.72 kilograms of carbon dioxide is kept out of the atmosphere. Oslo City Bike users rode an estimated total of 3.9 million kilometers spread across 2.65 million trips in 2017, which would equal a reduction of 2808 metric tons of carbon dioxide emission, assuming every bike trip replaced a car ride.

[1] see Chapter 2.1 for a definition of flow.

However, the provided flexibility of BSSs comes at the price of uncertainty; bike stations have limited capacity, so users are not guaranteed that bikes or locks are available when needed. Figure 1.1 shows heatmaps of the Oslo BSS demand during morning and evening rush hours. It is clear from the figures that the Oslo BSS follows a typical bike-sharing system usage pattern, where demand for bikes tends to increase in residential areas during morning rush hours and in downtown areas during evening rush hours. Likewise, demand for locks increases in downtown areas during morning rush hours and in residential areas during evening rush hours. Every time the system is unable to meet the demand, i.e. when there are no available bikes for a user to lend or there are no available locks for a user to return a bike, a *violation* occurs, represented as lighter shaded circles in the figures.



(a) Morning rush hour demand for bikes.

(b) Morning rush hour demand for locks.

(c) Evening rush hour demand for bikes.

(d) Evening rush hour demand for locks.

Figure 1.1: Oslo City Bike demand during morning and evening rush hour Tuesday, 27th of June 2017, from 08:00 - 09:00 (a, b) and 16:00 - 17:00 (c, d). Lighter shaded circles indicate violations.

In order to keep the system in a balanced state where as few violations as possible occur, most BSSs use service vehicles (shown in figure 1.2) to redistribute bikes within the system from nearly full stations to nearly empty stations. The redistribution process at Oslo City Bike today is done on an ad-hoc basis or by using naive demand estimators. By using more sophisticated Machine Learning (ML) approaches to accurately predict demand, the goal is that service vehicles can more precisely redistribute bikes in order to reduce the number of violations. The machine learning regressors presented in this thesis are able to accurately predict the flow of bikes within every clustered area of a BSS, and can be used either as direct way of dispatching service vehicles to areas where the predicted flow of bikes results in suboptimal states, or as input features to more fine-grained demand estimators which predict station-level demand.

Figure 1.2: A Oslo City Bike service vehicle. Credits: OsloCityBike [2016].

1.2 Goals and Research Questions

The research conducted during the course of this thesis aims to establish a thorough understanding of current state-of-the-art demand prediction solutions within BSSs and to apply ideas and algorithms from these studies in the context of flow prediction using machine learning methods. The overarching goal that the result of this thesis aims to accomplish is defined below.

Goal To enable Bike-Sharing System decision makers to proactively rebalance service stations based on accurate predictions of future bicycle flow.

The goal will specifically be achieved by performing experiments on a multitude of machine learning algorithms in order to objectively determine which

type of algorithm is best suited for the problem of bicycle flow prediction in BSSs. The current state-of-the-art literature will be rigorously examined to both identify unexplored areas of machine learning which may be suitable for such a problem, and to build upon existing solutions. Four Research Questions (RQs) have been defined to approach this goal:

Research question 1 What is state-of-the-art within on-demand transport services demand prediction?

Research question 2 Which neural network based techniques have been used within the domain of on-demand transport services demand prediction?

These two RQs relate to the structured literature review part of this thesis. By choosing to focus on the broader area of on-demand transport services, rather than restricting the area of research to bike-sharing systems, the idea is that state-of-the-art solutions within other domains (such as car-sharing services like Uber) will be retrieved in the literature search, which will likely be transferrable to the domain of BSSs.

Research question 3 How can machine learning be used to predict the flow of bikes in bike-sharing systems?

Research question 4 Which features have the highest impact on the learning algorithm's ability to accurately perform predictions?

RQ3 and RQ4 concerns the experimental part of this thesis, and are defined to ensure that the experiments performed are strictly objective.

1.3 Research Method

In order to address these research questions, a Structured Literature Review (SLR) (presented in Chapter 3) will be conducted to systematically explore relevant research to this thesis. The review will be used as the foundation to design experiments for machine learning algorithms that address RQ3 and RQ4. Conclusions will then be drawn based on the results of the experiments.

1.4 Contributions

The contributions of this thesis are primarily:

1. A state-of-the-art review of on-demand transport services demand prediction using machine learning models, with a primary focus on the domain of bike-sharing systems.

2. A comparison of the performance of multiple machine learning algorithms evaluated on real BSS data provided by Oslo City Bike.

3. A proposed machine learning model that is designed to run in a cloud environment and that is potentially capable of serving real-time predictions on streaming flow data provided by Oslo City Bike.

These contributions are described in greater detail in Chapter 6.3.

1.5 Thesis Structure

The rest of this thesis is structured as follows. Chapter 2 introduces the relevant background theory needed to gain sufficient understanding of the contributions of this thesis. Chapter 3 presents a thorough synthetization and discussion of state-of-the-art research within on-demand transport service demand prediction through the results of a rigorous SLR process. Chapter 4 presents the different models that were used for the experiments and how they were implemented. The experiment details and results are described in Chapter 5. Finally, Chapter 6 presents an evaluation and discussion of the experiment results, a detailed description of the contributions of this thesis, and some proposed extensions.

Chapter 2

Background Theory

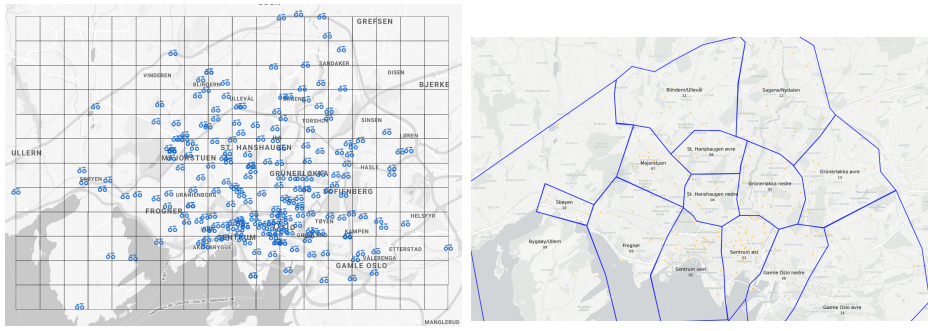
This chapter introduces background theory and ideas that are relevant to this thesis in order for the reader to acquire adequate knowledge to understand the contributions. Section 2.1 describes the definitions of clustering and flow in the context of Bike-Sharing Systems. Section 2.2 introduces the general concepts of machine learning, and Section 2.3 presents an introduction in each of the neural network models implemented in this thesis, as well as some more general concepts related to neural networks. Random Forests (RF), a different type of machine learning algorithm which has been implemented in this thesis, is explained in Section 2.4. Finally, Hyperparameter Optimization is described in section 2.5.

2.1 Definition of clustering and flow

In the BSSs literature on demand prediction, there are predominantly four levels of abstraction by which a BSS is viewed. The highest level of abstraction is system-level where the entire BSS, with all of its stations, is regarded as one entity and predictions are performed on the system as a whole, e.g. predicting the total number of bike trips that will occur in a city during a specific hour. The next level is cluster-level, where groups of stations are clustered together based on some criteria such as usage pattern similarity or spatial proximity, and predictions are performed on a per-cluster basis. Station-level is the third level of abstraction, handling prediction-related tasks on individual stations. The lowest level of abstraction, bike-level, is less prevalent in the literature. Predicting the end station of a recently picked up bike is a prediction task that is handled on bike-level.

Cluster-level, the second level of abstraction, is the primary topic of this the-

sis. Two different types of clustering methods are explored and evaluated against each other; grid-based clustering (shown in figure 2.1a) and zone-based clustering (shown in figure 2.1b). Grid-based was chosen as a naive clustering method because it does not require any prior domain knowledge of a BSS, and can easily be implemented in any BSS. For this thesis, 216 cells were defined in a 12x18 grid, where each cell is approximately 500×500 meters in size. As explained in Chapter 4.1.1, grid-based clustering also happens to be well suited for Convolutional Neural Networks (CNNs) due to the fact that the BSS at a specific point in time can easily be represented as an image with three dimensions. The zone-based clustering is how stations are clustered today by Oslo City Bike in order to more effectively visualize demand and to convey inventory-related information to rebalancing employees. The stations are clustered in 14 zones of approximately equal size based on multiple criteria, including similar demand patterns, spatial proximity, input from rebalancing truck drivers, Oslo's geography and topography, and several other soft criteria.



(a) Grid-based clustering (216 cells, of which 81 contain stations.)

(b) Zone-based clustering (14 zones)

Figure 2.1: Grid- vs. zone-based clustering

Furthermore, the focus of this thesis is to predict the flow of bikes within each clustering area (grid cell or zone) for some time interval. Three different types of flow is covered; outflow, inflow and internal flow. Informally, for an area a_i, **outflow** is defined as the number of trips that originated in area a_i and ended in an area a_j where $i \neq j$, for some time interval t. Similarly, **inflow** is defined as the number of trips that originated in an area a_j and ended in area a_i where $j \neq i$, for some time interval t. Finally, **internal flow** is defined as the number of trips that originated in area a_i and ended in area a_i for some time interval t. A formal definition of outflow, inflow and internal flow is given in Definition 1.

The three types of flow are conceptualized in figure 2.2, where a single grid cell is shown.

Definition 1. Let C be a collection of trips at the t^{th} time step. For a clustered area (zone or grid cell) a, the outflow, inflow and internal flow at time step t is defined respectively as

$$x_t^{out,a} = \sum_{T \in C} |\{(g_o \in g_a) \wedge (g_d \notin g_a)\}|$$
$$x_t^{in,a} = \sum_{T \in C} |\{(g_o \notin g_a) \wedge (g_d \in g_a)\}|$$
$$x_t^{internal,a} = \sum_{T \in C} |\{(g_o \in g_a) \wedge (g_d \in g_a)\}|$$

where $T : g_o \rightarrow g_d$ is a trip in C, g_o is the geographical coordinates for the origin station and g_d is the geographical coordinates for the destination station; $g_{o|d} \in g_a$ means the station $g_{o|d}$ lies within the geographic boundaries of g_a (definition adapted from Zhang et al. [2016]).

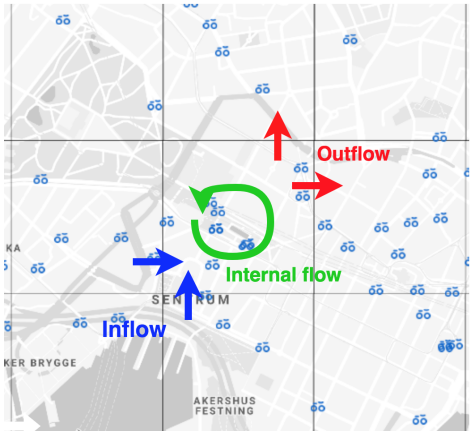


Figure 2.2: This figure shows the three different types of flow which are handled in this thesis: outflow is the number of trips that started in this cell and ended in another cell, inflow is the number of trips that started in another cell and ended in this cell, and finally internal flow is the number of trips that started and ended in this cell.

2.2 Machine Learning

Machine Learning (ML) is a subset of Artificial Intelligence (AI) focused on the science of using statistical methods to make a computer system autonomously learn some function by feeding it data of real-world events, without explicitly being programmed. The most commonly quoted definition of machine learning was given by Michalski, Carbonell and Mitchell in *"Machine Learning: An Artificial Intelligence Approach"* (Michalski et al. [2013]):

> *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.*

When defined within the bounds of this thesis, experience E relates to the historical data of bike trips transformed to flow as the number of incoming, outgoing and internal trips within each grid cell or zone per hour, tasks T is the task of predicting some future flow for a given grid cell or zone, and performance P is how well the network approximates the non-linear function of flow.

Machine learning methods are typically categorized as either supervised or unsupervised.

Supervised machine learning methods works by presenting the program with numerous example inputs and their accompanying labels. The learning algorithm produces a function that maps inputs to outputs, and iteratively modifies its own weights based on the error between its produced output and the correct provided output. After a sufficient amount of training iterations, the model is able to apply the learned function to make predictions about previously unseen inputs. The classic example of supervised learning is the MNIST classification task (LeCun [2007]), where the program is presented multiple images of handwritten digits and their associated labels.

When it comes to **unsupervised machine learning methods**, the program is not presented with any labels for its examples. The goal is for the learning algorithm to construct some function that describes the hidden structure in its data. A typical example of unsupervised learning is anomaly detection to automatically identify characteristics of fraud, where the learning algorithm is given large amounts of unlabeled credit card transactions, and the goal is to identify outliers which can be regarded as fraudulent.

There are multiple applications of machine learning, and thus it can also be categorized based on what the desired output should be.

Classification is the problem of assigning one or more labels to some unseen input. The classic example of classification problems in supervised learning is

spam filtering, where the learner is fed multiple examples of email and their associated label of *spam* or *not spam*, and the goal is to predict the correct label for unseen examples.

For **regression** problems, the goal is to predict some continuous output. For example, a common regression problem in supervised machine learning is to predict the price of a household based on its location, size, number of rooms etc.

Clustering is another common problem in machine learning, where the goal is to assign some example input to a group based on similarity to other examples.

This thesis focuses mainly on supervised machine learning for a regression problem. It is supervised because we present the learning algorithm with historical data of flow per grid cell or zone per hour, and it is regression because the goal is to predict the future continuous values for inflow, outflow and internal flow. There are many different approaches within machine learning to model this problem. The main branch of models used in this thesis, Artificial Neural Networks (ANNs), is explained in section 2.3.

2.3 Artificial Neural Networks

Artificial neural networks are computational models mimicking the way biological neural networks process information in the human brain. At the most basic level, artificial neural networks consist of multiple *neurons*. A neuron i receives a number of input signals $x_1, ..., x_n$ from other neurons, each of which is multiplied by a weight w_i which is computed based on the input's relative importance to other inputs, and finally summed. A bias weight w_0 is also added to provide an extra trainable parameter which leads to a better fit. The output z (called the pre-activation value) of a single neuron is defined by equation 2.1.

$$z_i = w_0 + \sum_{j=1}^{N} x_j w_{ij} \tag{2.1}$$

The final output signal of neuron i is computed by feeding the pre-activation value z_i through an activation function $g(z)$. The simplest type of neuron, called a *perceptron*, uses a simple binary activation function called the Heaviside step function (Bracewell and Bracewell [1986]), which outputs 1 if its input is positive and 0 if it is negative (see equation 2.2).

$$step(z) = \begin{cases} 1 & \text{if } z > 0 \\ 0 & \text{otherwise} \end{cases} \tag{2.2}$$

The perceptron is able to approximate linear functions, and is therefore only useful in cases where data is linearly separable. However, most real-world data (including the BSS data used in this thesis) is non-linear. See figure 2.3 for the difference between linearly vs. not linearly separable data.

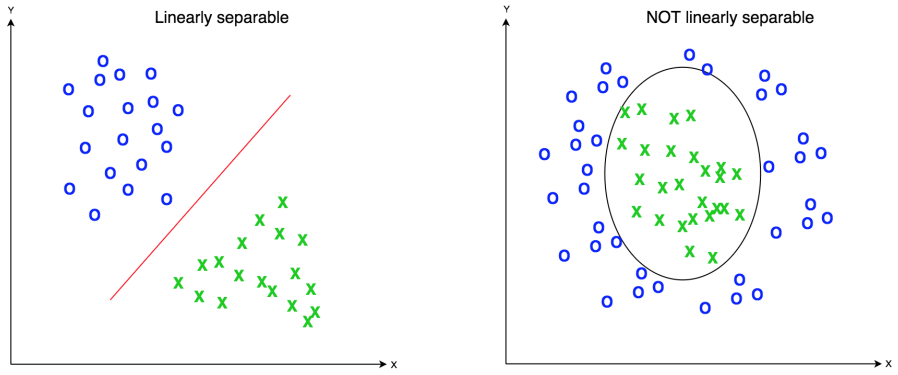


Figure 2.3: Linearly vs. not linearly separable data

There exist other activation functions which enable neural networks to approximate non-linear functions. The most commonly used are:

- *sigmoid*, which takes a real-valued input $x \in \mathbb{R}$ and squashes it to a range between 0 and 1.

$$sigmoid(z) = \frac{1}{1 + e^{-z}} \tag{2.3}$$

- *hyperbolic tangent*, which takes a real-valued input $x \in \mathbb{R}$ and squashes it to a range between -1 and 1.

$$tanh(z) = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}} \tag{2.4}$$

- *rectified linear unit*, which simply replaces negative values with 0.

$$ReLU(z) = max(0, z) \tag{2.5}$$

See figure 2.4 for a graphical representation of the previously mentioned activation functions and figure 2.5 for a diagram of how a neuron computes its output.

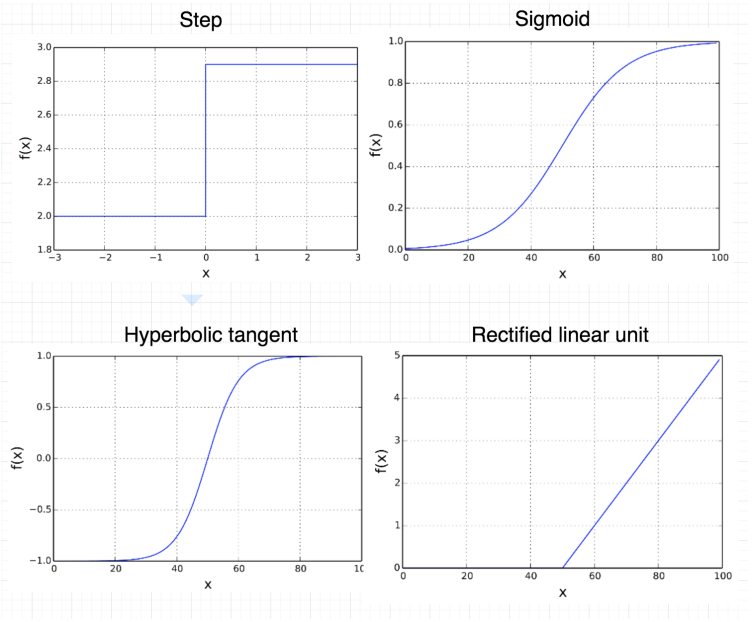


Figure 2.4: Comparison of activation functions

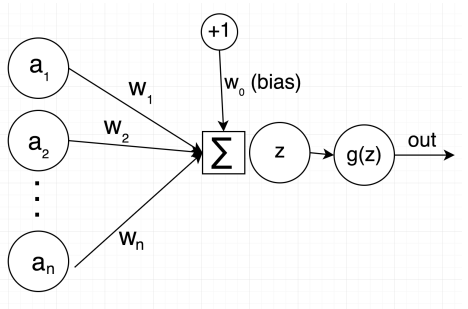


Figure 2.5: How a neuron computes its output

2.3.1 Multi-Layer Perceptron

A single perceptron is able to approximate linear functions. However, by constructing a network of perceptrons arranged in one or more hidden layers us-

ing a non-linear activation function, the network is able to compute non-trivial problems. In fact, the universal approximation theorem states that Multi-Layer Perceptrons (MLPs) can approximate any continuous function from one finite dimensional space to another, provided enough hidden units (Hornik et al. [1989]).

The output of every neuron in layer i is strictly dependant on the output of all neurons in layers $i-1$, and every neuron in layer i is fully connected to all neurons in layer $i-1$ through edges with an associated weight. See figure 2.6 for an example of a simple multi-layer perceptron with one hidden layer. A multi-layer perceptron is also called a Feed-Forward Neural Network (FFNN), because information only flows in one direction; from input, to hidden layers, to output. The edges between each neuron do not form cycles, as opposed to Recurrent Neural Networks (RNNs).

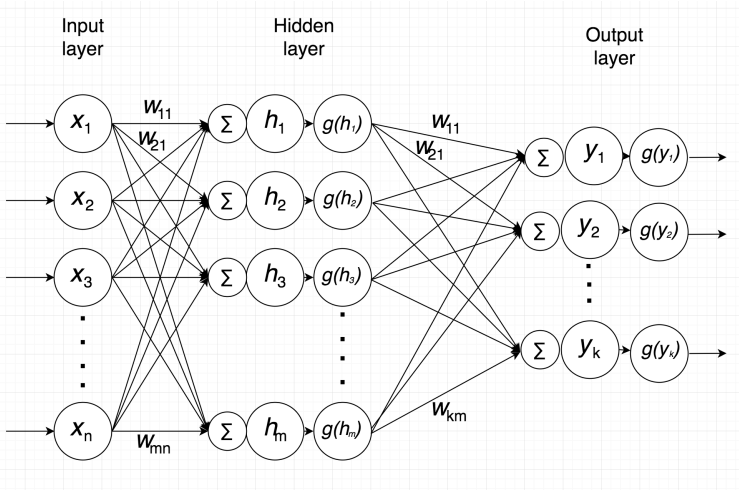


Figure 2.6: Diagram of a multi-layer perceptron with n input nodes, m hidden nodes in a single hidden layer and k output nodes. For regression tasks, the output layer activation function is usually linear, while for classification tasks softmax is used. Bias nodes are not shown in this diagram for simplicity.

2.3.2 Training artificial neural networks

The main goal when training an artificial neural network is to optimize the network's weights in order to minimize the error produced, such that the constructed non-linear function best fits the data. This is usually done in a two-step process.

In supervised learning, the network first processes a single vector of input features, and the attached known target for that vector of input features. Initially, the network weights are arbitrarily initialized. Given this example of data, and these weights, the network produces some output. The error between the produced output and the actual target value is then calculated by computing the Mean Squared Error (MSE), as shown in equation 2.6. One such calculation is called *forward-propagation*.

$$MSE = \frac{1}{2}(output - target)^2 \tag{2.6}$$

Next, the summed errors in the output layer is propagated back through the network in order to calculate the *gradients* by using the *back-propagation algorithm*, popularized by Rumelhart et al. [1986]. These gradients are in turn used by some optimization technique, such as *Stochastic Gradient Descent*, *Adam*, *Adagrad* or *RMSProp*, in order to minimize the loss function by adjusting the network weights with the goal of reducing the error in the output layer. To find the function minimum, most optimization techniques use some form of *gradient descent*, which works by taking steps proportional to the negative direction of the gradient at the current point. Specifically, given an error function $F(x)$ at point a, the fastest way to decrease $F(x)$ is given by equation 2.7

$$a_{n+1} = a_n - \gamma \nabla F(a_n) \tag{2.7}$$

where γ is the *learning rate*, which represents the speed in which we move in a given direction.

When all rows of the provided dataset have been processed in this manner, the network has completed one *epoch*. This procedure can be repeated for multiple epochs in order to fine-tune the weights.

2.3.3 Bias-variance trade-off

In supervised machine learning, the objective is to construct a model of the underlying process that produces the given training data in order to capture the patterns present in that data, while simultaneously managing to generalize to previously unseen data (the test data). The bias-variance trade-off is the problem of minimizing two metrics of error:

- **error due to bias**, which happens when the model is not complex enough to capture core patterns that are present in the training data.
- **error due to variance**, which is the result of an overly complex model that is able to represent the training data well, but also captures the noise present in the training data, leading to lower accuracy on unseen examples.

In common machine learning terms, models with high bias and low variance are *underfitting* their training data, while models with high variance and low bias are *overfitting* their training data. Moore, McCabe and Craig represents the bias-variance trade-off problem visually through a bulls-eye diagram, as seen in figure 2.7. The center of the bulls-eye diagram represents a model that perfectly captures the underlying process that produces some training data. A model with low bias and low variance will produce outputs that are consistently within the center of the bulls-eye. However, if the training data contains a lot of noise, the predicted outputs produced by the model will likely be poorer, leading to a model that either underfits or overfits. Figure 2.8 further illustrates the cases of underfitting and overfitting when modelling the function $f(x) = cos(\frac{3}{2}\pi x)$ with some added noise. The optimal model complexity to capture some underlying process results in the lowest prediction error, which occurs at the intersection of bias and variance, as illustrated in figure 2.9.

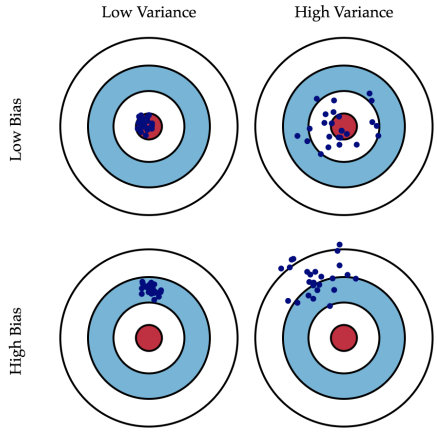


Figure 2.7: Visual representation of bias vs. variance (adapted from Moore et al. [2009])

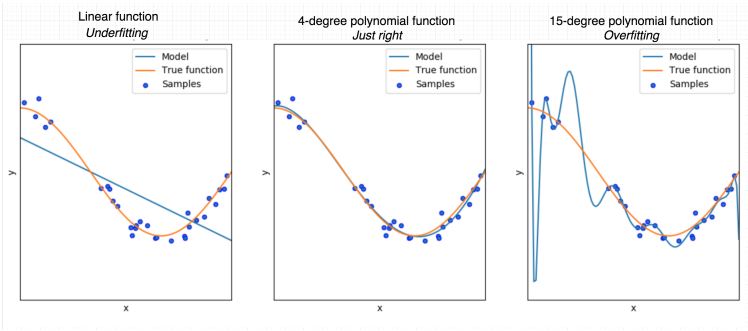


Figure 2.8: Illustration of underfitting vs. overfitting when modelling the function $f(x) = cos(\frac{3}{2}\pi x)$ with some added noise. A linear function underfits the training data, and a 15-degree polynomial function overfits the training data. A 4-degree polynomial function seems to fit the true function just right (adapted from the online *scikit-learn* documentation (Scikit-Learn [2011]))

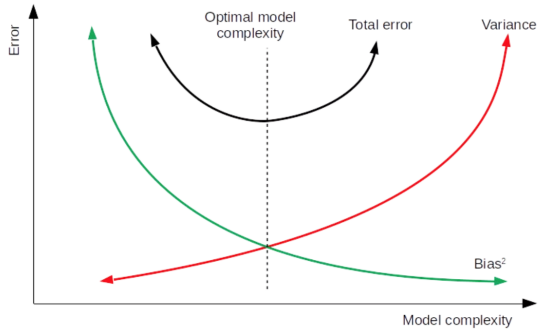


Figure 2.9: Optimal model complexity occurs at the intersection of variance and bias (adapted from the *machinelearning-wiki* website (MachineLearning-Wiki [2011]))

In practice, overfitting is a more prevalent issue than underfitting. There are many ways to avoid overfitting, including regularization, batch normalization and early stopping. Early stopping was primarily used in this thesis.

Early stopping The standard way to assess the performance of a machine learning model is to divide the training data into a training set, a validation set and a test set. The machine learning model is presented with a num-

ber of samples from the training set in order to reduce the loss function with respect to this dataset. After training on this dataset for some number of steps, the model is evaluated on the validation set, which contains previously unseen examples. The loss on the validation set is used as an indication of generalization capability. When employing *validation-based early stopping* this process is repeated until the error on the validation set ceases to decrease. This tends to happen when the model starts to overfit to the training set. Ultimately, the weights of the model at the step with best validation loss is used on the test set to assess the final generalization ability of the network.

2.3.4 Recurrent Neural Networks

A Recurrent Neural Network differs from a traditional Feed-Forward Neural Network by forming cycles between the edges of each neuron. By allowing the network to propagate information back through itself, the network obtains a form of memory which captures important information about what has been previously computed. This type of neural network architecture is designed to learn from sequential data. Intuitively, a recurrent neural network should perform well on the task of predicting bicycle traffic flow of a given zone or grid cell, as this data is inherently sequential, where the flow at each time step is at least partially dependent on the flow of the preceding N time steps.

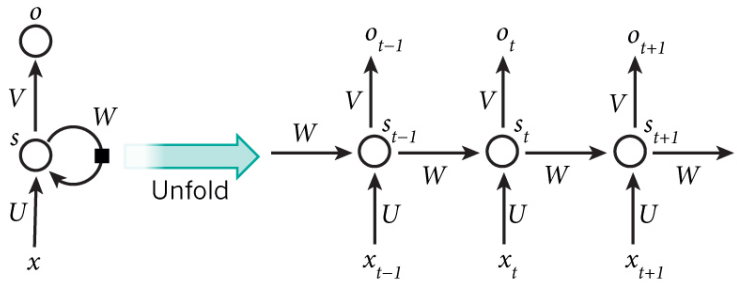


Figure 2.10: An unfolded representation of a recurrent neural network (original illustration by Hochreiter and Schmidhuber [1997])

Figure 2.10 illustrates how a typical RNN unfolds into a network for a sequence of length 3. x_t is the input data passed to the network at time step t, e.g. the inflow, outflow and internal flow of a zone at a specific hour. s_t is the hidden state at time step t. Each hidden state s_t is computed based on based on the

previous hidden state, where each computation follows equation 2.8, where Θ is an activation function such as Rectified Linear Unit (ReLU) or Hyperbolic Tangent (Tanh). s_0 is usually a zero-vector. o_t is the output at step t, e.q. the predicted inflow, outflow and internal flow for the next hour.

$$s_t = \Theta(Ux_t + Ws_{t-1}) \tag{2.8}$$

Training a Recurrent Neural Network requires using a modified version of the back-propagation algorithm called Back-Propagation Through Time (BPTT), because the network weights are shared by all time steps in the current sequence. This involves calculating and summing up all gradients for the previous time steps in a sequence in order to compute the gradient for time step $t + 1$. In practice, Recurrent Neural Networks suffer from loss of long term context due to the *vanishing/exploding gradient problem*. However, several modified versions of RNNs, such as Long Short-Term Memory (LSTM) and Gated Recurrent Unit (GRU), have been engineered to address this issue.

LSTMs and GRUs are simply a different way of computing the hidden state given the previous hidden state and the current input. An illustration of a single LSTM cell is shown in figure 2.11, and all of its computing steps are shown in equation 2.9.

$$\begin{aligned} i_t &= \sigma(x_t U_i + h_{t-1} W_i) \\ f_t &= \sigma(x_t U_f + h_{t-1} W_f) \\ o_t &= \sigma(x_t U_o + h_{t-1} W_o) \\ \tilde{C} &= tanh(x_t U_C + h_{t-1} W_C) \\ C_t &= C_{t-1} \odot f_t + \tilde{C}_t \odot i_t \\ h_t &= tanh(C_t) \odot o_t \end{aligned} \tag{2.9}$$

U and W are the input weights and recurrent weights, respectively. The input gate i_t controls how to squash the computed state for the current input. The forget gate f_t controls how to squash the previous state. The output gate o_t controls how much of the internal state that is let through to the rest of the network. The candidate hidden state $\tilde{C}_t$ defines what can be added to the next state. The cell state C_t is the internal memory, and is a combination of what should be forgotten from the previous cell state C_{t-1} and the candidate hidden state $\tilde{C}_t$. The final hidden state h_t is the Hadamard product ($\odot$ element-wise multiplication) of the memory C_t and the squashed output gate. Note that σ is the sigmoidal activation function. The GRU cell computes its hidden state in a similar fashion, except it has two gates instead of three. The reader is referred

to the original paper on LSTMs (Hochreiter and Schmidhuber [1997]) and GRUs (Cho et al. [2014]) for further details on these RNN architectures.

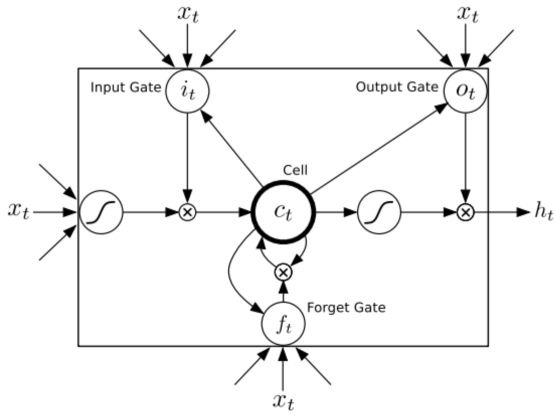


Figure 2.11: Diagram of a single Long Short-Term Memory cell (original illustration by Graves et al. [2013])

2.3.5 Convolutional Neural Networks

In contrast to traditional feed-forward neural networks, Convolutional Neural Networks (CNNs) are not fully connected; every neuron in layer i is only partially connected to the neurons in layer $i + 1$, except for the last layer. CNNs are usually applied to problems where the network input can be represented as an image with a width dimension, a height dimension, and a depth dimension (corresponding to for example the pixel color intensities). Feed-Forward Neural Networks can also be applied to such problems, but their performance suffer greatly due to the fact that network complexity (e.g. number of parameters) grows exponentially with larger image sizes (due to its fully-connected nature). CNNs aim to solve this problem by reducing the network parameters through a sequence of computational layers; convolution, activation, pooling (subsampling), and fully-connected.

In a **convolution** layer, filters are applied to the input matrix iteratively with a pre-defined stride length, which over a number of training steps learn to recognize a specific feature, such as the edges in an image. Multiple filters can be applied in a convolutional layer to create feature maps. In figure 2.12, an image of size 32x32x3 is passed through a single convolutional filter of size 5x5x3 and stride length 1. The Hadamard product is taken of the pixel intensities in all channels

for that filter, and results in a scalar, which is then passed through an **activation** function, and corresponds to one "pixel" in the output of the convolutional layer. When the filter is slid across the entire input image, the output feature map is of size 28x28x1. In this example, N filters of size 5x5x3 could be applied to the input image, which would result in an output of dimension 28x28xN.

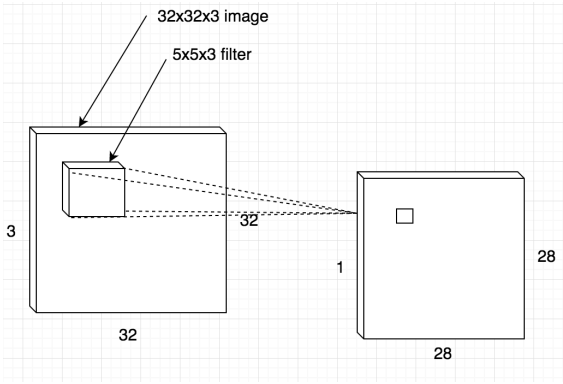


Figure 2.12: Diagram of how a convolutional layer works in a Convolutional Neural Network.

The output of the convolutional layer is then passed through a **pooling** layer to reduce the dimensionality of the image, allowing for faster training. In a pooling layer, a number of input nodes are pooled together, and some singular value is passed on. A popular choice is max pooling, which simply picks the highest value in the current window. A simple illustration of max pooling is shown in figure 2.13. If the output of the convolution layer has size 28x28x10 and is passed through a max pooling layer with window size 2x2 and stride 2, the result is of size 14x14x10, effectively reducing the number of parameters to $\frac{1}{4}$ of the input.

Finally, the output of the max pooling layer is flattened to a 1-dimensional vector and passed through to a **fully-connected** layer, which is the same type of layer as those used in traditional feed-forward neural networks. For more details on how Convolutional Neural Networks work, the reader is referred to the original paper by LeCun et al. [1998].

2.3.6 Deep Residual Networks

Recent advances in the annual ImageNet Large Scale Visual Recognition Challenge (ILSVRC) show that deeper convolutional networks tend to perform better

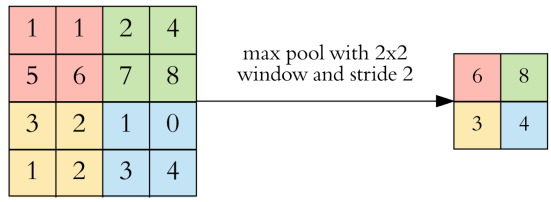


Figure 2.13: Diagram of how a max pooling layer works in a Convolutional Neural Network.

than shallow networks. This is also supported by the research performed by Eldan and Shamir [2016], Szegedy et al. [2015] and Simonyan and Zisserman [2014]. However, simply adding ten-folds of layers to a CNN tends to lead to vanishing/exploding gradients, and saturated degrading accuracy as the network depth increases. It is also very expensive to train such very deep CNNs. Deep Residual Networks (ResNets), first introduced by He et al. [2016] at the ImageNet competition in 2015 with a 152-layer deep network, allow for extremely deep CNNs by learning the residual functions with reference to the layer inputs, as opposed to learning unreferenced functions. The main idea is that by stacking a few non-linear layers in a network, it can fit an underlying function $H(x)$, and thus it can also fit another function $F(x) = H(x) - x$. This can be written as $H(x) = F(x)+x$, where $F(x)$ is a residual function and x is the layer input (called a skipped connection, or identity mapping). The stacked non-linear layers are then used to fit the residual function $F(x)$, as opposed traditional CNNs which approximate $H(x)$. The authors show that it is easier to optimize the residual function $F(x)$ than the unreferenced function $H(x)$. Figure 2.14 conceptualizes this idea through what the authors call a *residual block*. The reader is referred to the original paper by He et al. [2016] for further reading on Deep Residual Networks. The details of how Deep Residual Convolutional Neural Networks can be applied to the problem of predicting bicycle flow is described in Chapter 4.1.1.

2.4 Random Forest

Random Forest is an ensemble machine learning algorithm first introduced by Leo Breiman in 2001 (Breiman [2001]). It works by constructing multiple Classification and Regression Trees (CARTs) where each individual tree outputs some classification or regression prediction, and the final output of the random forest is either the mode of the classes or the mean prediction of all trees. Random Forests are less prone to overfitting than regular decision trees due to this voting

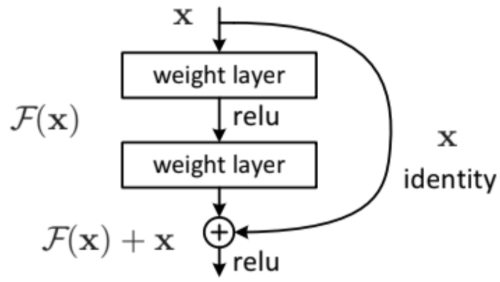


Figure 2.14: Diagram of how residual learning works in a Deep Residual Network with multiple such *Residual blocks* (original diagram by He et al. [2016]).

mechanism. Training a Random Forest algorithm involves a slightly modified version of the bootstrap aggregating (bagging) technique. Given a training set X of size n, bagging produces K new training sets X_i of size n where X_i is a random sample with replacement from X. A CART tree f_i is then trained to fit X_i. Random sampling with replacement ensures that about $\frac{n}{3}$ examples, called the Out-of-Bag samples x' are unseen to the learner, which is used to evaluate the forest by averaging the predictions of all individual trees, as seen in equation 2.10.

$$\hat{f} = \frac{1}{K} \sum_{i=1}^{K} f_i(x') \tag{2.10}$$

In the standard bagging technique, the set of features used at split j are all the features not used at split $j - 1$. Random Forests instead choose a random subset of all features at each candidate split j in a method called *feature bagging*, first introduced by Ho in 1995 (Ho [1995]). For regression problems, it is typical to choose $\frac{p}{3}$ random features at each candidate split j, where p is the total number of features. Random Forests have become very popular lately due to its short training time, no need for data preprocessing in the form of normalization, few hyperparameters to tweak and its ability to measure feature importance during training.

2.5 Hyperparameter Optimization

When working with neural networks, two types of parameters become apparent; network parameters (i.e. weights) and hyperparameters (e.g. learning rate or

number of layers). The whole point of training neural networks is to optimize the network weights through the use of some optimizer such as Stochastic Gradient Descent or RMSProp. However, determining which hyperparameters to use is a non-trivial task, usually requiring many years of experience in implementing neural networks and expertise knowledge in the domain of application. However, this process of determining optimal hyperparameters for some network for some problem can be automated by utilizing a form of meta-optimization called hyperparameter optimization. A hyperparameter optimization algorithm searches the hyperparameter space for a combination of hyperparameters that optimizes some objective function, typically the mean squared error on a validation set for regression problems. For neural networks, the model is usually trained on a training set with some combination of hyperparameters for a number of epochs, before being evaluated on a validation set, and then repeats the process with a different combination of hyperparameters. Finally, the combination that yielded the best performance on the validation set is returned. There are mainly three types of hyperparameter optimization:

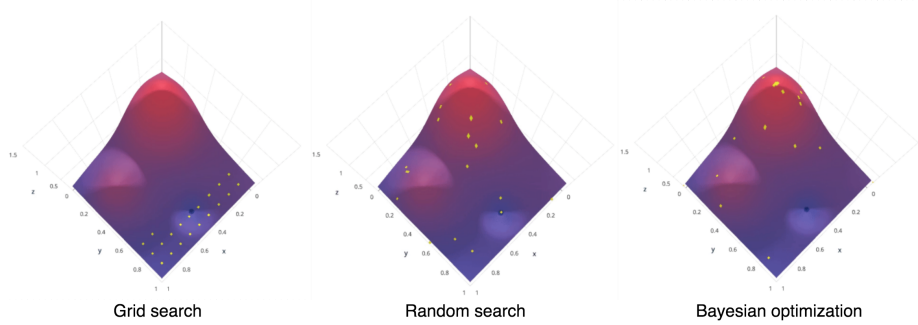


Figure 2.15: Illustrations of how different hyperparameter optimization algorithms work. In the diagrams, the x and y dimensions represent two different hyperparameters (e.g. learning rate and number of layers), and the z dimension represents performance on a validation set. The hill surface is the true performance for all combinations of hyperparameters. Each yellow dot is the result on the validation set for one combination of hyperparameters. Illustration adapted from a SIGOPT blog post by Alexandra Johnson [2017].

- **Grid search**, shown on the left in figure 2.15, is an exhaustive search through a manually selected subset of the hyperparameter space. Given a neural network model with two hyperparameters to optimize, *learning rate* and *number of layers*, the practitioner selects a finite set of values for each

parameters, e.g. learning rate $\eta \in \{0.1, 0.001, 0.0001\}$ and number of layers $L \in \{2, 4, 6\}$. Consequently, grid search trains the model with each pair (η, L) of the Cartesian product of these sets and returns the combination that yielded the best performance on the validation set.

- **Random search**, shown in the middle in figure 2.15, instead searches on some randomly selected values from the range $\{\eta \in \mathbb{R} : 0.1 \leq \eta \leq 0.0001\}$ and $\{L \in \mathbb{Z} : 2 \leq L \leq 6\}$. Random search is usually a better choice than Grid search when the hyperparameter search space is of high dimensionality.

- In contrast to Grid Search and Random Search, **Bayesian Optimization (BO)** searches the hyperparameter space intelligently by modelling distributions over objective functions as Gaussian Processes. Given the model performance on a validation set as a function of hyperparameters, $f(x)$, the task is to optimize $f(x)$ by picking the best hyperparameters, as shown in equation 2.11.

$$x^* = \overset{\text{argmax}}{x \in \chi} f(x) \tag{2.11}$$

  In figure 2.16, the dotted line represents an unknown objective function which BO tries to find the max of. BO selects some point on the graph where the mean is high (exploitation) and the variance is high (exploration), which creates a exploitation-exploration trade-off which is encoded in an acquisition function, shown as the green surface in the figure. Expected Improvement (EI) is a popular choice of acquisition function, which measures the expected increase in objective function given the next point picked. EI is defined as $EI(x) = \mathbb{E}[max\{0, f(x) - f(\hat{x})\}]$, where $\hat{x}$ is the current best combination of hyperparameters. In summary, the following steps are run for N iterations when using Bayesian Optimization:

  1. Update posterior expectation of f given observed values $f(x)$ using GP.
  2. Find some x_j which optimizes EI: $x_j = argmaxEI(x)$
  3. Compute value of f for x_j

  This technique has been used extensively throughout this thesis through Google CloudML's HyperparameterTuning module, as explained in Chapter 5.3.

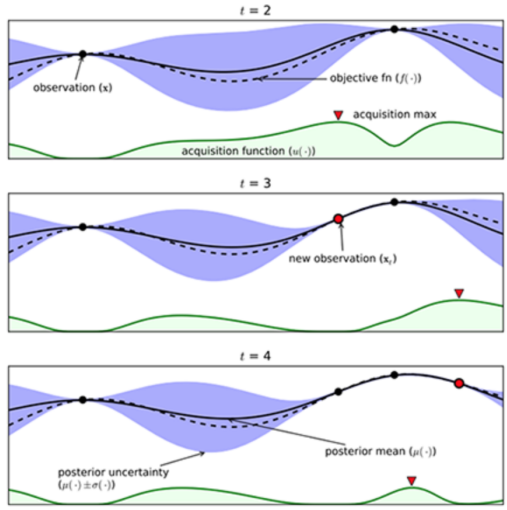


Figure 2.16: Gaussian process approximation of objective function (original figure by Brochu et al. [2010]).

Chapter 3

Structured Literature Review

This section discusses the results of the Structured Literature Review (SLR). A structured literature review is a systematic approach to collect and critically examine research studies related to a set of research questions. The goals of an SLR are primarily to map out existing solutions to some problem, identify gaps of knowledge, highlight areas where further research is necessary, and synthesize available information in order to build upon existing research (Kofod-Petersen [2012]).

3.1 Structured Literature Review Protocol

In order to ensure objectiveness of the review, a protocol was developed defining how the review was to be conducted. The primary goals of the SLR protocol were 1) to thoroughly cover as much of the relevant literature as possible given the time constraints on this thesis, and 2) to ensure the results of the literature search were reproducible. The developed SLR protocol was heavily inspired by the guidelines presented in Keele et al. [2007] and can be reviewed in its entirety in Appendix A. Table 3.1 presents the final set of papers after a rigorous SLR process.

3.2 State of the Art Review

This section discusses the synthesized information from the final set of papers presented in table 3.1. The discussions presented here are to be considered as

Authors	**Title**
Ashqar et al. [2017]	Modeling bike availability in a bike-sharing system using machine learning
Bacciu et al. [2017]	An experience in using machine learning for short-term predictions in smart transportation systems
Bei et al. [2013]	Uncertainty in urban mobility: Predicting waiting times for shared bicycles and parking lots
Caggiani et al. [2017]	Spatio-temporal clustering and forecasting method for free-floating bike sharing systems
Dali and Mladenic [2012]	BICIKELJ: Environmental data mining on the bicycle
Fei et al. [2017]	Predicting public bicycle rental number using multi-source data
Froehlich et al. [2009]	Sensing and predicting the pulse of the city through shared bicycling
Gallop et al. [2011]	A seasonal autoregressive model of Vancouver bicycle traffic using weather variables
Ghanem et al. [2017]	Bike share travel time modeling: San Francisco bay area case study
Giot and Cherrier [2014]	Predicting bikeshare system usage up to one day ahead
Han et al. [2014]	Towards bicycle demand prediction of large-scale bicycle sharing system
Kaltenbrunner et al. [2010]	Urban cycles and mobility patterns: Exploring and predicting trends in a bicycle-based public transport system
Li et al. [2015]	Traffic prediction in a bike-sharing system
Liu et al. [2015]	Station Site Optimization in Bike Sharing Systems
Malani et al. [2013]	Forecasting Bike Sharing Demand
Milenković et al. [2014]	Comparison of SARIMA-ANN and SARIMA-Kalman methods for railway passenger flow forecasting
Moreira-Matias et al. [2013]	Predicting taxi-passenger demand using streaming data
Rudloff and Lackner [2014]	Modeling demand for bikesharing systems: neighboring stations as source for demand and reason for structural breaks
Salaken et al. [2015]	Forecasting Bike Sharing Demand Using Fuzzy Inference Mechanism
Wang et al. [2017]	DeepSD: Supply-demand prediction for online car-hailing services using deep neural networks
Yang et al. [2015]	Public bicycle prediction based on generalized regression neural network
Yang and Zhang [2016]	A Novel Travel Adviser Based on Improved Back-Propagation Neural Network
Yoon et al. [2012]	Cityride: a predictive bike sharing journey advisor
Zhang et al. [2016]	Deep Spatio-Temporal Residual Networks for Citywide Crowd Flows Prediction

Table 3.1: The final set of primary studies after a rigorous SLR process.

answers to RQ1 and RQ2, and will be further summarized in Chapter 6.

Introduction

Transport demand forecasting is a vital core function to all transportation companies as it provides the basic input for planning and controlling all functional areas including transport operations planning, marketing and finance. Temporal demand patterns and their intensities significantly influences capacity levels and financial needs of any transportation business (Milenković et al. [2014]).

The fundamental issue within the domain of BSS research is to understand mobility patterns (Han et al. [2014]), and thus, the bike redistribution strategy is an essential part of the operational level. Because the distribution of rides is not uniform, certain stations may fill up or become empty, leading to a demand for bikes that cannot be fulfilled. Two approaches to redistribution are suggested by Rudloff and Lackner [2014]; incentive-based user redistribution and truck-based redistribution. The authors claim that although incentive-based user redistribution is cheap and sustainable, two major problems exist; stations in close proximity to transportation hubs tend to be at max capacity, and monetary incentives may be ineffective due to the already low cost of BSSs.

Truck-based redistribution can either be solved as a static optimization problem during the night, or as a more complex dynamic optimization problem during the day where user traffic must be taken into consideration. Although the dynamic rebalancing problem is well studied in the literature (Pillac et al. [2013]), most dynamic rebalancing algorithms use past demand as a proxy for future demand. The underlying current demand is not modeled to improve distribution. Obtaining an accurate approximation of future demand may be a crucial component to both dynamic rebalancing schemes and to increase user satisfaction.

The problem of forecasting BSS traffic flow has been very moderately studied. However, because the underlying goal of such a forecast is to predict demand, other types of problems that have been well researched within BSSs forecasting are very likely to yield insightful knowledge. Therefore, this literature review will cover a range of issues within BSSs, such as analysis of mobility patterns, system- and station-level demand forecasting, how to cluster stations to increase prediction accuracy, and other prediction tasks. In addition, because demand forecasting has been extensively studied in other related domains, some state-of-the-art solutions within these related domains are also presented.

Spatio-Temporal Usage Patterns

The majority of BSSs exhibit identical spatio-temporal usage patterns; during morning rush hours, the demand for bike rentals tends to surge in residential areas of the system, while the demand for bike rentals in city centres and business areas declines. Conversely, demand for bike rentals tends to surge in city centres and business areas during evening rush hours, while the demand for locks tends to increase in residential areas (Bei et al. [2013], Dali and Mladenic [2012], Froehlich et al. [2009], Han et al. [2014], Kaltenbrunner et al. [2010], Li et al. [2015], Rudloff and Lackner [2014]). Froehlich et al. [2009] argue that understanding and modelling these mobility patterns is essential in order to successfully predict future demand within the BSS

The Bicing system has been thoroughly studied in the literature (Froehlich et al. [2009], Kaltenbrunner et al. [2010], Yang and Zhang [2016], Yang et al. [2015]) and closely resembles the BSS in Oslo in terms of geography and topography, where residential areas are located on an outer incline edge of the city and business areas are located in the city centre close to sea-level. Froehlich et al. [2009] developed two separate clustering methods for this system focusing on two different metrics;

- activity clustering measuring how active a station is. This clustering algorithm resulted in five activity clusters, where the number of stations for each cluster increased as one moves from the outer edges of the city and inwards, confirming the intuition that stations on the outer edges of Barcelona's incline topography are less active than the inner stations.

- bicycle clustering measuring the average number of available bikes at each station. This clustering algorithm resulted in six clusters with three classes of behavior: outgoing (C1, C2), incoming (C3, C4), and flat (C5, C6). The outgoing clusters show a sharp drop in number of available bikes during morning rush hour, and an increase during evening rush hour, and are located around residential areas. The incoming clusters show a sharp increase in number of available bikes during morning rush hour and a decrease during evening rush hours, and are located around business areas. Cluster C6 tends to have a relatively flat usage pattern with an average of 66% available bikes, and is located in high altitudes. Cluster C5 has an average of only 15% available bikes, and are located at sea-level.

Dali and Mladenic [2012] simulated the most common bike paths by using a stochastic cumulative Gaussian distribution method and found similar usage patterns. However, since each record of the BICIKELJ dataset only describes the

number of available bikes at each station, not information per individual bike, their results could not be verified.

Critically Influential Variables

Several factors have been shown to impact bike-sharing demand significantly. The most obvious and intuitive exogenous factor is weather, which has been extensively studied in the literature (Ashqar et al. [2017], Caggiani et al. [2017], Dali and Mladenic [2012], Fei et al. [2017], Gallop et al. [2011], Ghanem et al. [2017], Giot and Cherrier [2014], Li et al. [2015], Malani et al. [2013], Rudloff and Lackner [2014], Salaken et al. [2015]). However, other variables such as traffic, spatio-temporal correlations between stations, and a station's proximity to transportation hubs or points of interest have also been researched to some extent.

Gallop et al. [2011] principally studied impact of weather on bicycle traffic by analyzing the findings of a Seasonal Autoregressive Integrated Moving Average (SARIMA) model of the bicycle traffic in Vancouver. The authors conclude that temperature and precipitation are particularly significant factors, where bike traffic counts increase by 1.65% per degree Celsius increase from the mean, whereas rain in the previous three hours decreases traffic counts by 23.54%, which is more impactful than it being a holiday or Saturday (-13.8%) or a day during the Olympics (+15.7%). Interestingly, the authors found that rain during the previous hour was more impactful than rain during the current hour, decreasing traffic counts by 8.86% and 3.68% respectively. Ashqar et al. [2017] and Malani et al. [2013] agrees that temperature considerably influences bike-sharing demand, and adds that humidity is negatively correlated with demand.

Variables related to geography and the internal structure of a BSS have also been shown to be highly correlated with station-level demand. Results presented by Ashqar et al. [2017] and Liu et al. [2015] indicate that the demand of nearby stations in terms of geographical proximity is positively correlated with the demand of an individual station. Malani et al. [2013] further studied station-level demand in Washington D.C. with a socio-economic perspective and found that population size within the census tract associated with a particular station is highly correlated with the amount of rentals.

Liu et al. [2015] were the first researchers to demonstrate a clear correlation between otherwise unexplored factors and station-level demand:

- Transportation related variables: the number of taxi pick-ups in a stations associated Voronoi region, and walking distance to nearby parking lots, subway stations or bus hubs

- Point-Of-Interest (POI) related one-hot encoded variables: four POI categories (entertainment, restaurant, shopping, education) where, if the POI is within some threshold distance, its value is 1.

Other factors related to the actual experimental details of a machine learning demand prediction approach have also been shown to influence prediction accuracy. Results presented by Ashqar et al. [2017], Froehlich et al. [2009], Yang and Zhang [2016], and Bei et al. [2013] indicate that accuracy decreases as the prediction window increases, with best performances observed at very short prediction windows (less than 10 minutes). However, Giot and Cherrier [2014] found that prediction accuracy is roughly the same for prediction windows of size 1 hour and 25 hours, due to the regularity of temporal patterns observed at a single station throughout weekdays.

Furthermore, a machine learning algorithm's prediction performance may suffer if the utilized dataset spans a relatively short period of time. Bacciu et al. [2017] emphasizes that BSS user behaviors may exhibit regularity in the initial months which is later lost due to seasonality effects in the process being modeled.

System-level Demand Prediction

System-level demand can be defined as either the total number of rentals during some time interval, or the total number of available bicycles or locks at some point in time, for the entire BSS as a whole. Having an estimate of future system-level demand may be beneficial on an operational level in order to approximate the number of redistribution trucks that must be in service during a particular time interval. Giot and Cherrier [2014] claim that future system-level demand cannot be predicted based solely on the historical mean. System-level demand is influenced by a range of external factors, making it highly non-linear, as confirmed in the findings presented by Froehlich et al. [2009].

In research performed by Giot and Cherrier [2014] to predict system-level rental amount per hour, the authors assume that the system follows a time series pattern where each record is at least partially temporally dependent on the previous. Thus, the authors augment each record in the original dataset by adding 24 autoregressive features; the number of available bikes from one hour to twenty-four hours ago. Results show that Ridge Regression achieved a lower Root Mean Squared Error (RMSE) than all other regressors (Adaptive Boosting (AdaBoost), Support Vector Regression (SVR), RF, Gradient Boosted Decision Tree (GBDT)) for all 24 hour lags.

Interestingly, because Ridge Regression is a linear regressor, their model assigned a low Pearson coefficient to the time-of-day feature. However, in line with what Ghanem et al. [2017] found about breaking assumptions of homoscedasticity

(see *Similar Problems in Bike-Sharing Systems*), Ridge Regression fails to grasp the impact of time-of-day because it is highly non-linear.

Salaken et al. [2015] built a fuzzy inference model using the Wang-Mendel rule generation method on data from the Capital Bikeshare program in Washington D.C., which contained both date- and weather-related information. The authors suggest that fuzzy inference predictors can potentially outperform FFNNs in terms of prediction accuracy when it comes to predicting the total number of available bicycles at a future time. However, the authors emphasize the fact that work was not carried out to optimize the baseline single-layered 10-neuron neural network, hence it can be argued that the obtained results may not be comparable.

Station-level Demand Prediction

Station-level demand prediction has been widely researched in the literature and far outweighs research on system-level demand prediction. This may be due to the fact that the obtained predictive demand granularity for station-level models would be more advantageous on an operational level because it allows for more accurate bike redistribution and provides rebalancing trucks information about expected city activity (Froehlich et al. [2009]).

Autoregressive Moving Average (ARMA) is a statistical forecasting model for time series data that is inadequate for station-level demand prediction because it fails to capture the non-stationary nature of station availability caused by the fact that the mean and variance of availability changes over time (Yoon et al. [2012]). However, Yoon et al. [2012] dealt with this problem by using Autoregressive Integrated Moving Average (ARIMA), which captures the spatio-temporal correlations between stations, i.e. a reduction of bikes in an origin station is correlated with the increase of bikes in a destination station. ARIMA has also been successfully used to predict demand for other transportation modes (Milenković et al. [2014], Moreira-Matias et al. [2013]). Kaltenbrunner et al. [2010] used an ARIMA approach to predict station-level availability for a 30-minute horizon by using the autoregressive features of past availability data for both the station in question, and the 15 nearest neighbors in terms of geographical proximity. The authors highlight that modelling neighboring stations decreases prediction error considerably. However, a substantial flaw in this study is that meteorological factors, which have been shown to significantly influence demand, are not integrated in the prediction model.

Froehlich et al. [2009] used a Bayesian Network (BN) with only three features; time of day, last known number of bicycles and a prediction window of 10 to 120 minutes. However, the model was only trained on weekdays, which follow a pe-

riodic temporal pattern, and thus demand prediction for weekends and holidays would most likely perform worse due to the irregular nature of the usage patterns. Additionally, like Kaltenbrunner et al. [2010], the BN model fails to incorporate exogenous contextual features such as weather and seasonality. Yoon et al. [2012] further argues that Froehlich and Neumann's results are only marginally better than the baseline Last Value (LV) prediction because too few features are used, and that adding more features to the BN would lead to an exponential increase of the probability table, leading to infeasible training times.

Bei et al. [2013] proposes a statistical approach to model spatio-temporal features of bike availability at station-level using a class of statistical algorithms called Generalized Additive Models (GAMs). The models are trained on three prediction horizons:

- Short-term (5 minutes): uses current weather
- Medium-term (1 hr): uses current weather and autoregressive features, namely y_{t-1} and y_{t-2} as available bikes 30 min and 60 min ago.
- Long-term (24 hr): no weather or autoregressive features; rather uses Historical Average (HA) at the given time of day as exogenous variable.

Results show that the proposed GAM method outperforms LV, HA and ARMA baselines for both short-, medium-, and long-term predictions, obtaining up to 50% lower RMSE and WRMSE (mean average error of all stations as a function of station size) than ARMA. If the algorithm outputs an availability of zero, the waiting time before a bike arrives is calculated (see *Similar Problems in Bike-Sharing Systems*).

Ashqar et al. [2017] uses ensemble and bagging methods, specifically RF and Least Squares Boosting (LSBoost), as univariate regression algorithms to predict station-level availability. RF achieved the best score with a Mean Absolute Error (MAE) of 0.37 bikes per station, outperforming LSBoost with an MAE of 0.58 bikes per station for a time horizon of 15 minutes.

Many scholars use normalized values to measure station-level availability; Yang and Zhang [2016] and Yang et al. [2015] calculates the Normalized Available Bikes (NAB) for each station at each timestep to reflect percentage of available bikes by dividing the number of bikes by the sum of the number of bikes and locks. In addition, Normalized Activity Score (NAS) is calculated to indicate the activeness of a station. The authors show that the temporal patterns for NAB and NAS differ greatly from weekdays to weekends, and therefore concentrate on only one of those situations.

$$NAB_t = \frac{\alpha_t}{\alpha_t + \beta_t}$$

$$NAS_t = \frac{|\alpha_{t-1} - \alpha_t|}{\alpha_t + \beta_t}$$

where α is the number of available bikes and β is the number of available locks.

ANNs have been widely used in the literature either as baseline or primary methods to predict future station-level demand. Yang and Zhang [2016] employs a ANN with a single hidden layer of 25 neurons, where the initial weights and biases are configured using a genetic algorithm. The authors show that for short time horizons (30 minutes), the model achieves a MAE of 1.15 when including the NAB scores of the 10 nearest neighbors in terms of temporal patterns, as opposed to 1.86 when including the NAB score of the 10 geographically nearest neighbors. A major drawback of their model is that it fails to incorporate meteorological features. In another study, Yang et al. [2015] used a Generalized Regression Neural Network (GRNN) on the exact same problem and dataset, but found that a classic ANN performed better.

Fei et al. [2017] also predicts station availability by using a neural network, where the network's hyperparameters are computationally tweaked by employing a particle swarm optimization approach, bearing resemblance to Yang and Zhang's genetic algorithm approach to initializing hyperparameters. Liu et al. [2015] observes a faster convergence rate when training their neural network using the Levenberg-Marquardt algorithm as opposed to SGD, but the authors fail to describe the network architecture. Nonetheless, their solution performs well with an Coefficient of Determination (R2) score of 0.88, while baselines such as AdaBoost, SVR and CART achieve below 0.75.

A problem uncovered by Rudloff and Lackner [2014] is the complex issue of estimating demand when there are no available bikes or locks, named censored demand by the authors. A simple solution offered is to add a dummy variable of 1 if there are no available bikes or locks. Similarly, an equivalent dummy variable is added to the three geographically closest stations to see if demand shifts to these neighboring stations in the case of censored demand.

Dali and Mladenic [2012] states that since the problem of availability prediction exhibits properties such as few features, no sparsity, and non-linearity, a Support Vector Machine (SVM) with Radial Basis Function (RBF)-kernel may be an appropriate fit for such a regression problem.

Clustering

Clustering techniques have been explored extensively in the literature as a method to gain insight to activity patterns within BSSs. The overall aim of clustering is to partition a group of dissimilar patterns into multiple smaller homogeneous groups, where there is low correlation between different clusters, and a high correlation between the elements within each cluster (Caggiani et al. [2017]).

Li et al. [2015] discovered that by grouping stations into clusters based on geographical and temporal patterns, the periodicity and regularity of each cluster becomes much easier to predict than that of a single station. In addition, both check-out proportions and inter-cluster transitions is more robust for clusters than for individual stations. This is confirmed by graphing the standard deviation of check-out proportion and inter-cluster transition for clusters and stations for each hour of the day, where it is observed that the standard deviation is lower for clusters than for individual stations for all hours.

Furthermore, Li et al. [2015] argue that the number of clusters must be carefully chosen based on expertise. Having as many clusters as there are stations means that the clustering algorithm fails to capture the underlying patterns in the system, while having just one large cluster offers no applicable information to redistribution trucks.

Cluster Analysis in Bike-Sharing Systems

Froehlich et al. [2009] investigates how the spatio-temporal usage patterns are affected by geographical layout and topography within Barcelona by using hierarchical agglomerative clustering over the average station activity for each station (activity clusters) and the average percentage of available bikes for each station (bicycle clusters). At each iteration of the clustering algorithm, the two most similar clusters were grouped together by calculating the cluster-to-cluster similarity matrix using a Dynamic Time Warping (DTW) distance metric (which, in contrast to Euclidean distance, captures the temporal patterns), until a cluster-to-cluster distance threshold was reached. The authors emphasize that the clustering algorithm had no prior knowledge of the station's three-dimensional geocoordinates.

Similarly, Dali and Mladenic [2012] observed that different groups of stations showed similar behavioral patterns, and therefore clustered the stations based on hourly activity using hierarchical agglomerative clustering.

Fei et al. [2017] used a classic centroid-based clustering method called K-means to partition the 3000-station Hangzhou dataset into 4 temporally distinct classes.

Malani et al. [2013] use a simpler approach and cluster each station to its

associated census tract, which is a geographical region defined by socio-economic studies, in Washington. The total number of rentals for each census tract in the next hour is then predicted using Extreme Gradient Boosting (XGBoost), which outperformed all other regressors with a Root Mean Squared Logarithmic Error (RMSLE) of 0.405.

Ashqar et al. [2017] introduced Partial Least Squares Regression (PLSR) as a dimensionality reduction technique in order to reduce the number of required models from 70 (corresponding to each station) to 5. The authors examined an adjacency matrix of bike trips by finding the highest 10 in-degree stations for station i originating from stations $j \neq i$, where they found that each of the 5 resulting regions corresponded to 5 different zip codes, indicating that the majority of trips occurred within the same spatial region. Without using PLSR, their ensemble prediction algorithm had an MAE of 0.37 bikes per station. Using PLSR, the authors achieved a MAE of 0.6 bikes per station, which the authors argue is still within acceptable limits for a large-scale BSS prediction system, and is more feasible in a production setting due to the considerable decrease in amount of models that must be maintained.

Yoon et al. [2012] explored three different clustering strategies to integrate in their ARIMA model;

- Voronoi regions, generated via geographical proximity.
- K-Nearest Neighbors (KNN) by comparing the temporal patterns of the number of available bikes for all stations, with $K = 3$.
- Since a user can pick up a bike at station i and return it at station j, there exists both positive (similar patterns) and negative (opposite pattern) correlations. The authors cluster stations into significantly positive, significantly negative and non-significant using a Generalized Linear Regression model to calculate the correlation coefficients for the two aforementioned correlations.

However, it is shown that all three clustering strategies yield almost identical results in terms of prediction accuracy.

Rudloff and Lackner [2014] argues that surrounding stations significantly influences an individual station's demand. The authors tested their Poisson count model for demand forecasting on both individual stations and on spatially clustered stations, and found that their model significantly improved on individual stations. Rudloff and Lackner claims that this finding is due to the fact that spatial clustering groups stations with highly dissimilar temporal patterns, thus

leading to poor forecasting capabilities. Li et al. [2015] also observed poor results when only considering spatial similarities when clustering.

Similar Problems in Bike-Sharing Systems

Bacciu et al. [2017] aims to solve a classification task rather than a regression task; namely to predict the destination station of circulating bikes within the Pisa BSS. The task at hand is divided into three levels of granularity: city-level, station-level and user-level. SVM with Gaussian kernel and RF are the two proposed machine learning methods for each level. City-level uses a single unified model to predict destination station given a new rental, and achieves an F1 score of 0.2. Station-level uses a separate model tailored to each specific departure station, and achieves an F1 score of 0.28. User-level partitions the rental data by each of the 2999 user's rental history and trains a separate model tailored to each user, and achieves an F1 score of 0.64. The authors note that the user-level model tends to perform significantly worse for holidays and Sundays, as these days break the seasonality of the data. Another interesting point is that if users with less than 100 total rentals are excluded from the data set, the user-level prediction model performs extremely well with an F1-score of over 0.95.

The second stage of the two-stage GAM in Bei et al. [2013] computes the distribution of waiting time until a bike becomes available if the first stage predicts an availability of zero bikes for a given station. The fundamental assumption is that bike arrival times follow an exponential distribution with time-varying intensity, explained by the fact that demand rises during rush hours. As such, the model significantly underperforms at computing waiting time during periods where the amount of arrivals is low.

Ghanem et al. [2017] found that using Multiple Linear Regression (MLR) to predict travel time on the Bay Area dataset containing 33 features, including weather, time-of-day, distance, and subscription type, was infeasible due to the fact that the studentized residuals violates the underlying assumption of homoscedasticity within the data. Because RF does not assume normality of the data, the authors instead used this to predict travel time and it outperformed other methods such as LSBoost and ANN.

Similar Problems in Related Domains

Other transport industries such as bus, taxi or car-sharing services (Uber, Lyft) also benefit greatly by having an estimate of future demand. The demand patterns for both car-related services and bike sharing systems share similar characteristics;

- It is periodical, meaning that demand on a Monday afternoon for a taxi stand, car-sharing region or bike station does not deviate by much from its mean.

- It is seasonal, meaning that demand for a taxi stand, car-sharing region or bike station may surge during specific seasons, such as stations near parks during summer.

- Both follow the same spatio-temporal patterns; outwards demand tends to surge in residential areas during morning rush hours and inwards demand surges in residential areas during evening rush hours. Conversely, outwards demand surges in business areas during evening rush hours, and inwards demand surges during morning rush hours.

However, the influence of meteorological factors exhibits inverse effects on demand for car-related services compared to bike sharing systems; demand tends to increase during periods of precipitation, as opposed to bike sharing systems where demand decreases during precipitation.

Thus, the problem of predicting the number of orders that will emerge at a given taxi stand or in a car-sharing region at a future time point can be seen as analogous to station-level demand prediction for BSSs, as they both share many of the same characteristics.

Moreira-Matias et al. [2013] presented a model to predict the spatio-temporal distribution of taxi-passenger demand on a 30-minute horizon using an ensemble learning method, modeling the number of orders that will emerge at a given taxi stand at a given future time duration. The authors ensemble method achieved better results than Poisson, Weighted Poisson and ARIMA.

Wang et al. [2017] investigates how a ResNet performs compared to classical ensemble methods such as RF and XGBoost on the problem of predicting demand for the car sharing service Didi in Hangzhou, China. The problem is analogous to demand prediction in BSSs because of the aforementioned similar characteristics, demand is predicted for a cell in a city-wide N-sized grid which can be compared to predicting demand for a bike station, and the dataset used for order data is exactly the same as trip data in BSSs.

Compared to other machine learning methods, the ResNet proposed by Wang et al. [2017] requires little to no feature engineering, as the network is able to learn patterns across several spatio-temporal attributes (e.g. geolocation, time intervals, day of week) on its own using embedding layers. Exogenous factors such as meteorology and traffic are easily incorporated into the model, and the

model is easily extendable by simply fine-tuning an already trained network if new data attributes are available, making it suitable for online learning.

Instead of one-hot encoding categorical values, the authors introduce an embedding layer which maps each categorical value to a low-dimensional space (relative to its vocabulary size), where the parameters of the embedding matrix are trained alongside the entire network, not separately. This leads to two benefits; categorical values with a large vocabulary is significantly cheaper to represent using embedding compared to one-hot encoding, and spatio-temporal attributes (such as location or timeslot) are not treated independently (as with one-hot encoding), but are clustered together based on similarity using embedding. Embedding categorical features results in a lower MAE and RMSE, and a lower training time, compared to one-hot encoding categorical values. The ResNet achieved an MAE of 3.56 and RMSE of 15.57, outperforming RF, GBDT, Least Absolute Shrinkage and Selection Operator (LASSO) and HA.

Flow Prediction

Li et al. [2015] used a novel approach to predict the traffic flow between clusters of bike stations in both New York and Washington. The authors describe a five-step process.

1. In order to mitigate to issue of fluctuating traffic at individual stations, the authors use a bipartite clustering algorithm based on the stations geographical locations and historical transition patterns.

2. The total number of rentals within the bike sharing system within a period is predicted by feeding temporal and meteorological features into a GBDT.

3. The traffic is allocated to each cluster by predicting each cluster's check-out proportions by calculating temporal, weather and temperature similarities using a multi-similarity inference model.

4. Each cluster's check-in is predicted based on the check-outs calculated in the previous step. The authors use the same multi-similarity-based inference model to predict an inter-cluster transition matrix, e.g. a matrix $T_{t,m*m}$ ($_m$ = # clusters) corresponding to period t, where each entry T_{t,C_i,C_j} is a transition probability from cluster C_i to C_j in time t.

5. Finally, trip duration between each pair of clusters is obtained by using maximum likelihood estimation on a lognormal distribution.

The proposed method achieved better results than HA and ARMA for both common and anomalous periods.

Zhang et al. [2016] aims to solve a different, but similar problem, namely crowd flows prediction. The authors propose a deep spatio-temporal residual network (ST-ResNet) to simultaneously predict inflow and outflow of crowds in every region of a city. The flow within a region is affected by three factors:

- spatial dependencies: the inflow of some region is affected by the outflow of other regions. Likewise, the outflow of some region, affects the inflow of other regions.
- temporal dependencies: flow is largely affected by periodicity and seasonality.
- exogenous factors: weather and events may drastically affect the flow of crowds.

The authors emphasize the fact that the flow of a given region is dependant on the flow of other regions in spatial proximity, and formulates the prediction task using CNNs which have been shown to hierarchically capture spatial structural information. Every region in the city is represented as a cell in a grid with two channels; inflow and outflow. Hence, the historical flow data for the city can be encoded as multiple images with two channels, where every region at a specific time step corresponds to one pixel in an image, and CNNs can be used to capture spatial dependencies within each "image". By employing a ResNet-type of CNN, the authors are able to train a very deep network in a feasible amount of time.

The authors show that their ST-ResNet model outperforms a relatively simple ANN, as well as ARIMA and SARIMA, on the Citibike data set.

Gaps in the Literature

Demand forecasting in BSSs has been widely researched in the last decade, following the rapid development of such systems in urban environments around the world. However, several gaps within the literature have been uncovered in this literature review, and must be further studied to gain a more complete understanding of the dynamics within BSSs.

Many demand prediction algorithms in the literature are based on the fact that the underlying data follows a time series pattern. As such, many scholars have attempted to use typical time series analysis tools such as ARIMA to forecast demand with varying levels of success. Recurrent Neural Networks (RNNs) however, have not been studied at all in the literature, despite the promising results that have been shown for time series prediction with RNNs and LSTMs.

ResNet is a relatively new neural network model, and is therefore sparingly studied within the literature. Nonetheless, it has been shown to be very effective at both predicting demand for car-sharing services (Wang et al. [2017]) and predicting crowd flows between regions of cities (Zhang et al. [2016]). This calls for further research on ResNets within the context of BSSs.

The impact of events on a BSSs demand patterns has not been studied, possibly due to the fact that obtaining accurate event information has been difficult.

In addition, it was observed during the course of this literature review that there is no unified frame for testing models within the research community. This is problematic because results cannot be properly validated.

Another problem is that some researchers opt to include weather as exogenous factors, while others do not. This also causes different results to be incomparable.

Chapter 4

Architecture and Models

This chapter describes the overall system architecture and the individual models implemented in this thesis.

4.1 Architecture

An overview of the system architecture is presented in Figure 4.1. The system is divided into several modules, each responsible for handling a specific part of the system. This modular architecture allows for easy modification in each module without having to be worried about breaking functionality in the other modules. In addition, new custom modules developed in future work can easily be integrated. The stapled lines and rectangles in the figure represent parts of the system that are not within the scope of this thesis, but are proposed extensions that are suitable in a production setting, described in Chapter 6.4.

Everything related to data handling is managed by the Data Module, including querying the Google BigQuery[1] databases for station locations and historical trips, transforming this into a flow format, extracting features, and building ML-ready datasets.

The Machine Learning Module consumes datasets produced by the Data Module. All baseline- and ML-models are implemented in the Machine Learning Module, which uses Google CloudML[2] capabilities to optimize hyperparameters in some of the models, and consequently trains and evaluates each model.

The visualization module was used during implementation to better understand the dynamics of the BSS, and visualizes the historical flow through the use

[1] https://cloud.google.com/bigquery/
[2] https://cloud.google.com/ml-engine/

of a Google Maps[3] application. In the future, this module will also be responsible for presenting predicted flow.

Sections 4.1.1, 4.1.2 and 4.1.3 describe each module in greater detail.

4.1.1 Data Module

The Data Module is responsible for handling all data-related tasks. All bike trips from April 4th 2016 to November 22nd 2017 were registered and stored in a BigQuery database, and was used as the base dataset in this thesis. In addition, a database containing all stations was used to aggregate bike trips into zones or grid cells, and a database containing hourly meteorological data was used to supply the final datasets with important weather information. See Table B1, B2 and B3 in Appendix B for an overview of what types of relevant data each of these databases contain. This section describes how these three datasets were combined, how features were extracted, and finally how this data was processed in order to produce the final datasets used in the Machine Learning Module.

Data Aggregation

Recall from Chapter 2.1 that the primary focus of this thesis is cluster-level flow prediction. Two different types of clustering methods were applied; grid-based (figure 2.1a) and zone-based (figure 2.1b). Two additional tables are presented in Table B4 and B5 in Appendix B describing the format of the JSON[4] files defining which stations belong to which clustering area. In order to assign every station to a grid cell or zone, the procedure described in Algorithm 1 was used.

Algorithm 1: Assigning stations to grid cells or zones

inputs: A list of grid cells or zones $C = c_1 \dots c_n$; a list of stations $S = s_1 \dots s_n$

```
foreach c_i ∈ C do
    foreach s_i ∈ S do
        if s_i.coordinates is within c_i.bounds then
            c_i.stations ← s_i.id;
```

By having an overview of which stations belong to which grid cells or zones, the Oslo City Bike Trip dataset can be grouped by clustered areas. The procedure to do this for both grid-based clustering and zone-based clustering is equivalent, and therefore only the procedure for zone-based clustering will be explained here. A Structured Query Language (SQL) query, shown in Listing 4.1, was ran in a for-loop iterating over every zone in the *clustering-by-zone.json* dataset, where the

[3] https://cloud.google.com/maps-platform/

[4] JavaScript Object Notation: an alternative to CSV.

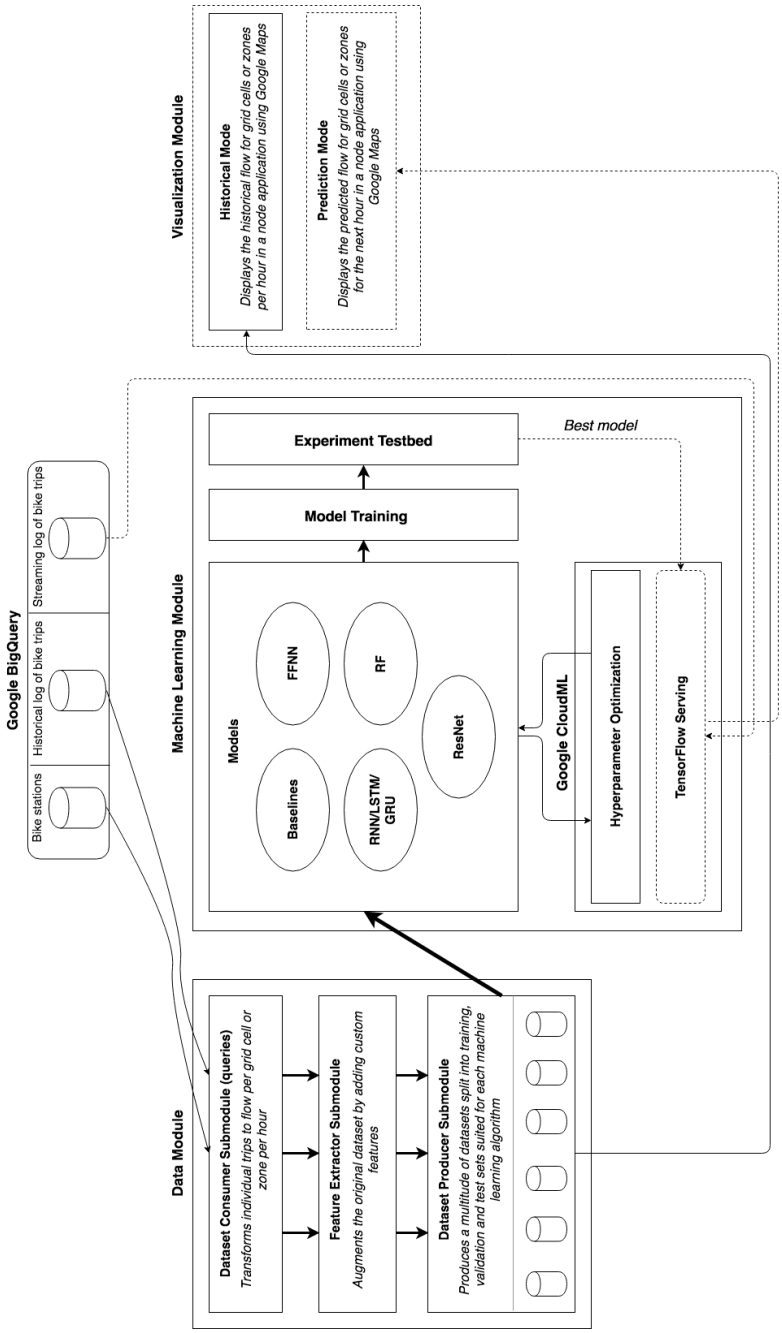


Figure 4.1: Overview of the overall system architecture.

result was appended to a CSV-file after every iteration. Note that the timestamp is truncated hourly and grouped, so that the resulting CSV-file has an hourly time interval. *index* is the current iteration and *stations* is the list of stations for the current zone. If the *stations* list was empty, the iteration was skipped. All zones had a non-empty stations lists, but only 81 of 216 cells had non-empty stations lists. The resulting CSV-file contained 125 356 rows for zone-based clustering, and 725 274 rows for grid-based clustering. Finally, the grid-flow and zone-flow CSV-files were *left joined* with the weather CSV (Table B3) on the *timestamp* field using *pandas*[5].

Listing 4.1: SQL query to extract the hourly outflow, inflow and internal flow for a zone.

```
SELECT
{index} as zone,

TIMESTAMP_TRUNC(
    TIMESTAMP(DATETIME(started_at, 'Europe/Oslo')),
    HOUR) AS timestamp,

COUNTIF(trips.start_station_id IN ({stations}) AND
        trips.end_station_id NOT IN ({stations}))
        AS outflow,

COUNTIF(trips.start_station_id NOT IN ({stations}) AND
        trips.end_station_id IN ({stations}))
        AS inflow,

COUNTIF(trips.start_station_id IN ({stations}) AND
        trips.end_station_id IN ({stations}))
        AS internal_flow

FROM `oslo_bysykkel_trips` as trips
GROUP BY timestamp
ORDER BY timestamp
```

Feature Engineering

Feature engineering is a fundamental part of machine learning research to make a learning algorithm work efficiently. It requires domain knowledge of the data

[5] pandas is a data structuring library for Python

used, and is often used to generalize certain features of a dataset to a format that is understood by the learning algorithm. This section will describe some of the feature engineering techniques that were applied in this thesis. Feature engineering was performed on the two CSV-files described in the previous section, hereby denoted as *flow-by-grid* and *flow-by-zone*.

First, a number of datetime-related features were extracted from the *timestamp* field. This includes the year, month, week number, day of the week, and hour. In addition, whether or not that date was a Norwegian holiday was extracted, and the season in which the date belongs to (winter, spring, summer or autumn).

In Vancouver, Canada, Gallop et al. [2011] found that bike traffic counts for the current hour decreases by 23.54% if there was rain in the previous three hours. This was found to be one of the most significantly influential variables for the Vancouver dataset, and hence it was also chosen to include it in this dataset. A boolean feature *rain_last_3* was added to encode this information. It is set to true if there was measured more than 0.8mm rain in the previous three hours combined. Additionally, other meteorological features used include current rain in millimeters, temperature in Celsius, wind speed in km/h and whether it was sunny or not.

Next, a number of features capturing the temporal dependencies of past flow were engineered, inspired by research conducted by Zhang et al. [2017] where the authors used Convolutional ResNets to predict city-wide crowd flows. Specifically, *closeness*, *period* and *trend* were calculated as moving averages for the outflow, inflow and internal flow of a grid cell or zone for a given hour, resulting in nine added features (see Table B6 in Appendix B).

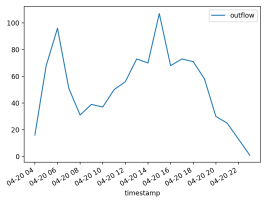
Closeness is defined as the average outflow, inflow or internal flow of the previous *three hours* of a given cell or zone at a given hour, and hence captures very recent temporal dependencies.

Period is defined as the average outflow, inflow or internal flow of the previous *four days at the same hour* of a given cell or zone at a given hour, and hence captures temporal dependencies related to daily periodicity, i.e. that flow tends to be relatively equal for 24 hour intervals.

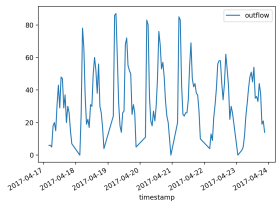
Trend is defined as the average outflow, inflow or internal flow of the previous *four weeks at the same day at the same hour* of a given cell or zone at a given hour, and hence captures the trend in the data, i.e. that flow steadily increases as peak season approaches.

Figure 4.2 shows the outflow of zone "St. Hanshaugen Nedre" for three different time intervals. Figure 4.2a indicates that outflow of recent time intervals are more relevant than outflow of distant time intervals, implying temporal closeness.

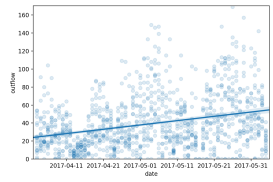
Figure 4.2b shows the clear pattern of daily periodicity (period) in this area. Figure 4.2c shows how the flow increases as the year approaches peak season (trend).



(a) Closeness of St. Hanshaugen Nedre (20.04.2017 05:00 - 23:00)



(b) Period of St. Hanshaugen Nedre (17.04.2017 - 23.04.2017)



(c) Trend of St. Hanshaugen Nedre (03.04.2017 - 04.06.2017)

Figure 4.2: Closeness, period and trend temporal patterns of outflow for zone St. Hanshaugen Nedre

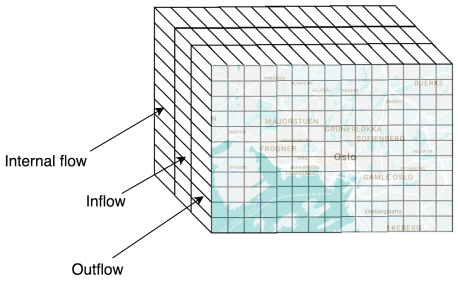
Producing the Final Datasets

The final feature engineered datasets *flow-by-grid* and *flow-by-zone* were then split into a training set, a validation set and a test set. The validation set is used for early stopping. 75% of the dataset is used as a training set, 10% as validation set and 15% as test set. The split was performed a total of four times; before the first two splits the datasets were ordered first by cell or zone, then by timestamp. This is a suitable format for RNNs which expect sequential data, as well as many of the baselines. Before the last two splits, the datasets were shuffled randomly. This format is suitable for FFNNs. Ordering does not affect RF performance, so either can be used for this algorithm. See Table B6 in Appendix B for a description of every feature in the final datasets. The ResNet model however expects the data to be structured in an image-like manner. This is explained in the following section.

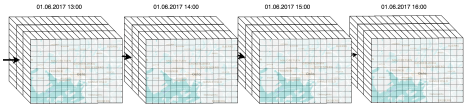
Transforming Flow Datasets to Images for ResNet

The ResNet used in this thesis is actually a CNN with a slightly different architecture. Since it is a CNN, it expects input in the form of images with a height dimension, width dimension and depth dimension. Every image represents one timestep (one hour in the case of this thesis). Every pixel in an image represents the flow of a specific cell or zone. In addition, every pixel has three channels; outflow, inflow and internal flow. All temporal features described in Table B6 are

discarded, and the images are rather sequenced by each timestamp. In addition, all meteorological features are also discarded due to time constrains in this thesis, but Chapter 6.4 describes how this can be added in the future. All moving average features are calculated in the actual ResNet model itself, rather than being pre-calculated in this module. Figure 4.3 shows a conceptual diagram of how the data is structured for the ResNet-model. The ResNet-model that was used in this thesis expects data for every "pixel" and every hour; therefore, all 216 cells were used for grid clustering, and all 14 zones were used for zone clustering. The original data only included data for hours where the BSS actually was open, but this had to be upsampled to include all 24 hours. The upsampled rows were given flow values of 0. Furthermore, to limit the sparsity of the dataset such that most images actually contain flow values, the time span of the dataset was limited to April 16th 2017 - October 15th 2017.



(a) A single input image for the ResNet representing the flow of all grid cells for a specific hour



(b) Multiple input images arranged in sequence.

Figure 4.3: Conceptual diagram showing the expected input for the ResNet-model. These figures show how the input is structured when using grid clustering.

Grid clustering was rather trivial to represent as an image, as the BSS is already grouped in an "image-like" manner when using grid clustering. However, zone clustering uses polygons defining the bounds of each area, and was therefore not trivial to represent as images. Ultimately, zone-based clustering was chosen to be represented as a 2x7 image, which means that the CNN is not able to accurately capture spatial dependencies between each zone. This is explained in further detail in Chapter 5. The ResNet-compatible datasets are 4D tensors of shape (T, d, h, w) where $T = timesteps = 4392$ and $d = dimensions = 3$ for both grid and zone, $h = height = 12$ and $w = width = 18$ for grid, and $h = 2$ and $w = 7$ for zone.

4.1.2 Machine Learning Module

The Machine Learning Module is, as the name implies, responsible for handling everything related to ML. All baselines, RF, FFNN, RNN/LSTM/GRU and ResNet models are implemented, trained and evaluated in this module using TensorFlow, Keras or SKLearn. All models consume datasets produced in the Data Module. The FFNN and RNN-models are factored out to their own separate packages within this module in order to have a code structure that complies with the requirements of Google CloudML. Each model is described in greater detail in Section 4.2.

4.1.3 Visualization Module

The Visualization Module was developed to get a greater understanding of the flow dynamics of the Oslo BSS. Albeit not within the scope of this thesis, this module is proposed as a visual front-end for the predictions made by the ML Module. A Node.js[6] application using the Google Maps Platform API was initially developed to visualize historical flow. As this module is not a central part of this thesis, further elaborations will not be made in this section. Chapter 6.4 describes how this module is intended to be used in the future. Figure 4.4 shows a screenshot of the application.

4.2 Models

This section describes the models implemented in this thesis.

4.2.1 Baselines

It is crucial to establish baselines in machine learning projects from which models can be compared with. Baselines serve as a simple reference point of the complexity of the problem modelled. The more sophisticated machine learning models implemented in this thesis were iteratively improved and compared with the baselines in order to determine whether the predictive power of the ML-models was stronger than naive prediction methods. The three baselines implemented are described below.

Random Algorithm

Random Algorithm (RA) is frequently used in the literature as a very naive baseline. It is the baseline with the least predictive power in this thesis. RA

[6]Node.js is a JavaScript run-time environment for building web applications

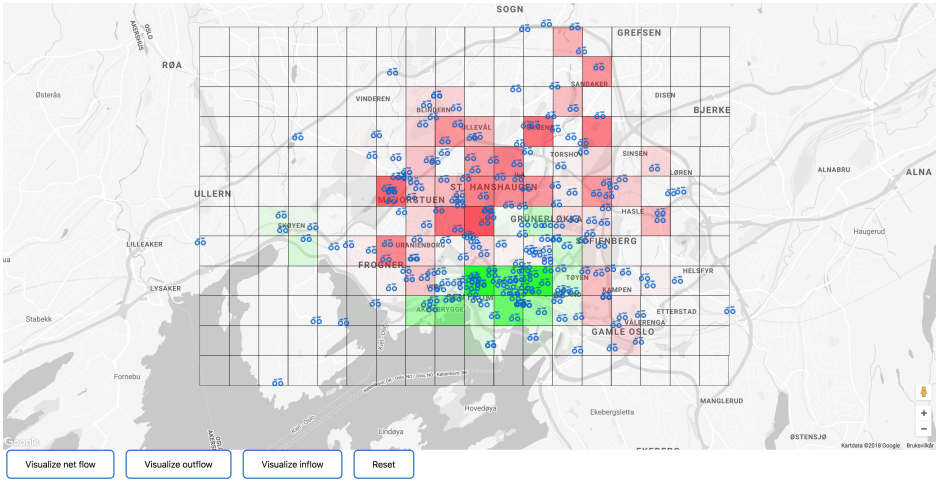


Figure 4.4: Screenshot from the Visualization Module using Grid mode. The overall flow in Oslo is here shown for a morning rush hour. Green cells indicate positive net flow (higher amount of inflow than outflow). Red cells indicate negative net flow. Darker colors represent a larger difference. Internal flow is shown by clicking on a cell.

simply samples a random target value from the distribution of target values within the training set. Since there are three target values in this thesis, RA picks an inflow value between in_{min} and in_{max}, an outflow value between out_{min} and out_{max} and internal flow value between $internal_{min}$ and $internal_{max}$.

Historical Average

Historical Average (HA) is one of the most popular choices of baseline in the literature. HA calculates the average outflow, inflow and internal flow for all clustered areas for all time steps in the training set. Given the historical collection of flow $F = Y_1, Y_2, \ldots, Y_t$, the predicted value $\hat{Y}_{t+h}$ is the mean of the values in F.

$$\hat{Y}_{t+h} = \frac{Y_1 + Y_2 + \cdots + Y_t}{t} \tag{4.1}$$

Last Value

In the BSS industry, Last Value (LV) is commonly used as a naive prediction method because it tends to perform well in this domain, where traffic counts usually do not fluctuate significantly every hour. LV works by setting the last observed value Y_t as the estimate of $\hat{Y}_{t+h}$, where h is the prediction horizon (one hour in this thesis).

$$\hat{Y}_{t+h} = Y_t \tag{4.2}$$

In machine learning, a problem that may occur is that models predict the identity function for the last observed value, effectively rendering the model equal in performance to a LV algorithm. This baseline is used to identify if such an issue arises. In addition, the machine learning models implemented must perform better than this baseline in order to be of any business value.

4.2.2 Random Forest

RF differs from the other machine learning algorithms implemented in that it is not a neural network. RF has been used with great success when it comes to prediction tasks within the context of BSS (Ashqar et al. [2017], Bacciu et al. [2017], Ghanem et al. [2017], Wang et al. [2017]). It is a machine learning model that requires little feature engineering, data preprocessing and hyperparameter optimization, while still being able to achieve promising results. In addition, RF enables the practicioner to get an overview of which features contributed the most by performing feature importance calculation during training. In this thesis, the SciKit-Learn implementation of RF was used. The only hyperparameter that was optimized using Grid Search (see Chapter 2.5) was the number of trees in the forest. The reader is referred to Chapter 2.4 for more details on Random Forests.

4.2.3 Feed-Forward Neural Network

FFNNs have been used extensively in the literature, either as a main contribution, or to serve as baselines for other models (Fei et al. [2017], Milenković et al. [2014], Yang et al. [2015], Yang and Zhang [2016], Zhang et al. [2016]). The FFNN models implemented in this thesis are commonly referred to as Deep Neural Networks (DNNs) because they consist of more than one hidden layer. Numerous configurations were evaluated through the use of Bayesian Optimization on Google CloudML's Hyperparameter Tuning[7] service. Common for all configurations was that the *ReLU* activation function (see Chapter 2.3) was used.

[7] https://cloud.google.com/ml-engine/docs/tensorflow/hyperparameter-tuning-overview

In addition, the features were processed in the same manner for all configurations.

Categorical features with few classes were encoded as one-hot vectors. This is important because without one-hot encoding, high categorical values would effectively be perceived as more important by the neural network. For example, a value of 21 for the *hour* feature would be more important than a value of 8, which in reality may not be the case. One-hot encoding performs binarization on a categorical feature, such that a feature with N categorical values translates to a binary vector of length N where only one value is 1, the rest are 0. Only binary categorical features (F10, F11, F14 and F16 in Table B6) were one-hot encoded in this thesis.

Categorical features with many classes were mapped to dense real-valued vectors using *embeddings*. Neural networks do not train particularly well when using sparse high-dimensional one-hot encoded vectors for features with many categories. Therefore, features with a large vocabulary size (i.e. many categories) were transformed into dense low-dimensional continuous vectors. A general rule of thumb for the dimensions of the embedding vector is to use the square root of the number of categories in the original feature vector, e.g. a feature vector with 64 categories would be transformed to an 8-dimensional embedding vector. Embedding mapping was performed on features F1, F6, F7, F8, F9 and F12 in Table B6.

Finally, continuous features (F13, F15, F16, F18-F26 in Table B6) were normalized using standardization. Standardization works by first determining the distribution mean and standard deviation of each feature, then subtracting the mean for each feature before finally dividing the values of each feature by its standard deviation. This is shown in Equation 4.3.

$$x' = \frac{x - \bar{x}}{\sigma} \tag{4.3}$$

where x is the original feature vector, $\bar{x}$ is the mean of that vector and σ is the standard deviation.

Every model had a different configuration of the following hyperparameters as they were optimized by BO: batch size, size of first hidden layer, number of hidden layers, layer size decay rate, optimizer, and learning rate. The layers in the network were constructed with exponential decay, meaning that with a decay rate of 0.5 and a first layer size of 512, the second layer would be of size $512 \times 0.5 = 256$, the third of size $256 \times 0.5 = 128$ and so on.

4.2.4 Recurrent Neural Network

As explained in Chapter 3.2, one of the major gaps in the literature that was identified during the SLR process was the lack of RNNs. RNNs have been shown to excel when applied to sequential data. Since the flow data used in this thesis is inherently sequential, and due to the fact that RNNs have not been used in the BSS literature, it was decided to implement two RNN variations, LSTM and GRU, and evaluate their performance against both each other and other models. Several model hyperparameters were optimized using BO; the type of RNN cell (LSTM or GRU), the number of units for the RNN cell, learning rate, sequence length, and batch size. Common for all model configurations was the use of RMSprop as optimizer and *tanh* as the activation function. Initially, *ReLU* was used as the activation function, but this caused the model to diverge quickly, resulting in exploding gradients. The *tanh* function fixes this problem by squashing the output to a value in the range $(-1, 1)$. All the same feature processing steps as FFNNs were used for this model.

A considerable limitation of using RNN models for this prediction problem was uncovered in this thesis. Due to the fact that each clustered area (grid cell or zone) is represented as a sequence in and of itself, and that the dataset used contains the flow for all areas for all timesteps, RNNs can not be directly used on this dataset. This is because the RNNs may use sequences that span multiple areas, which intuitively is not desirable. Instead, a separate RNN model has to be trained and optimized for each clustering area since the sequence patterns are unique for each area. Naturally, this is far more computationally expensive than other models, requiring 14 or 216 models to be optimized, trained and maintained for zone-based clustering and grid-based clustering respectively, as opposed to a single unified model for each of the clustering methods when using RF, FFNN or ResNet. Nevertheless, RNNs modelling individual clustered areas did perform well on the validation set, as described in Chapter 5. Due to time constraints in this thesis, only three zones and three cells were modelled using RNN variations.

4.2.5 Deep Residual Network

In Zhang et al. [2016] the authors show that a Deep Residual Network is able to effectively model crowd flows using data from both taxi trips and bike trips in New York City. Additionally, Wang et al. [2017] demonstrated that ResNets outperformed all other models when used to predict demand in a car-sharing service company in China. Based on this, it was decided to use ResNet in this thesis, and evaluate its performance when applied to the Oslo City Bike flow dataset. The author of this thesis strongly emphasizes that the ResNet model used in this thesis is a clone of a publicly available implementation by Zhang et al.

[2016], and the only modifications made is the dataset used and simple manual tuning of a few select hyperparameters, specifically the optimizer and number of residual units. The ResNet implementation by Zhang et al. is available at `https://github.com/lucktroy/DeepST`. It is included as a candidate model in this thesis to establish a solid foundation on which objective conclusions can be made as to which machine learning algorithm is best suited for the problem of predicting cluster-level flow. Because this model essentially is a copy of Zhang et al's implementation, the reader is referred to the original study by Zhang et al. [2016] and their GitHub repository for further implementation details.

4.3 Frameworks, libraries and programming languages

All development concerning the data- and machine learning modules is done using Python 3.5. Python has become the standard language to use when working with machine learning, because most ML libraries are developed primarily for this language. Furthermore, it is an easily readable programming language that has powerful interoperability with low-level C code, making it efficient for complex mathematical operations. The visualization module was built using JavaScript, and is built as a Node.js application.

For data structuring and analysis, the Pandas[8] library was used. Pandas has exceptionally high performance when handling large datasets, and includes a wide range of operations to transform data.

The data module utilizes *pandas-gbq*[9], which is a Pandas-supported data querying library for Google BigQuery. All data provided by Oslo City Bike in this thesis is stored in Google BigQuery[10]. *pandas-gbq* queries the databases using SQL syntax and returns the data in Pandas data structures, which can then be transformed and easily exported to different file formats.

The RF model is implemented using *Scikit-Learn*[11] which provides high-level interfaces for various regression, classification and clustering algorithms.

The other ML models are implemented with *TensorFlow*[12], which has become the most popular framework for machine learning. TensorFlow is developed by Google and features multiple levels of abstractions for implementing a wide range of machine learning models. Developers can make use of prebuilt Estimators, which are fully-equipped ML models following best practices that require very little actual coding, or choose to use one of the low-level libraries which provides

[8] https://pandas.pydata.org/
[9] https://github.com/pydata/pandas-gbq
[10] https://cloud.google.com/bigquery/
[11] http://scikit-learn.org/
[12] https://www.tensorflow.org/

endless options for customization. TensorFlow works by building computational graphs where every node is a mathematical operation and every edge represents flow of data between nodes. By using a predefined computational graph to represent computational steps, TensorFlow is able to organize operations in the most efficient manner, allowing high performance computation. TensorFlow models can also be trained in the cloud, where large clusters of GPU-equipped computers can train and optimize models in a fraction of the time it would have taken on a regular desktop.

The RNN and ResNet models are also essentially TensorFlow models, but are implemented using *Keras*[13], which is a high-level neural networks API that runs on top of TensorFlow. Keras provides very high level abstractions of popular neural network implementations, and was designed to enable developers to quickly test various models. Since Keras uses TensorFlow as the backend, the developed models are also capable of being trained and optimized on Google CloudML.

[13] https://keras.io/

Chapter 5

Experiments and Results

This chapter describes the experiments performed in this thesis. First, Section 5.1 introduces the experimental plan. Next, the experimental setup is described in Section 5.2. Finally, the experiment results are presented in Section 5.3.

5.1 Experimental Plan

The experiments performed in this thesis are designed to answer RQ3 and RQ4 through rigorous testing of multiple models and configurations.

Research question 3 How can machine learning be used to predict the flow of bikes in bike-sharing systems?

In order to answer RQ3, several machine learning algorithms originally intended for different applications were implemented: RNNs are intended for data of sequential nature and can model the process responsible for every consequence. ResNets, which are variants of CNNs, are intended for data structured in an image-like manner and can identify spatial and temporal relationships between data. DNNs are the classical type of neural networks and can be used in various scenarios to model complex non-linear relationships by mapping a set of input features to some output. RF, unlike the other models implemented, are not neural network-based models, but can nevertheless be used for many of the same applications as DNNs. Evaluating a wide range of different machine learning algorithms ensures that RQ3 can be answered objectively.

Experiment 1: Hyperparameter Optimization involves finding the optimal hyperparameters for each machine learning algorithm, and was specifically designed to answer RQ3. RF uses Grid Search, described in Chapter 2.5, to

find the optimal number of trees in the forest, which is the only hyperparameter for this model. The DNN and RNN models are implemented in TensorFlow and structured in compliance with Google CloudML's packaging requirements in order to use CloudML's Hyperparameter Tuning service. Cloud ML Hyperparameter Tuning uses Bayesian Optimization as the optimization technique and allows for parallel optimization of multiple models which saves a lot of time. The hyperparameters tuned for DNN and RNN can be found in Table 5.1 in Section 5.2. Since the ResNet model used in this thesis is the same as in Zhang et al. [2016], where the authors optimized the model on a very similar dataset, not much work was put into optimizing this model. Manual hyperparameter search was performed by evaluating a few different combinations of hyperparameters, described in Table 5.1 in Section 5.2.

Since this thesis focuses on two different types of clustering methods, hyperparameter optimization was performed for both zone-based clustering and grid-based clustering for each model. Additionally, because the number of features used in each of the sub-experiments in Experiment 2 varies from 8 features to 22 features, separate models were optimized for experiments with few features (*NONE, M*) and many features (*C, CP, CPT, CPTM*). Furthermore, as explained in Section 5.2.2, separate RNN models had to be optimized for low-, mid- and high-traffic volume areas. This resulted in a total of 24 optimized models ((3 RNN models + 3 other ML models) $\times$ 2 clustering methods $\times$ 2 levels of feature amount).

Research question 4 Which features have the highest impact on the learning algorithm's ability to accurately perform predictions?

In order to determine which features are most influential both for each individual model and across all models, two approaches are taken. The first approach is to evaluate every model with different feature configurations. In **Experiment 2: Determining Influential Features**, all models are evaluated in sub-experiments with 6 different sets of features for both zone-based clustering and grid-based clustering:

- *NONE* uses only temporal features F6-F12 and the zone/cell feature F1 described in Table B6 in Appendix B.
- *M* uses F1, temporal features F6-12 and meteorological features F13-F17.
- *C* uses F1, temporal features F6-12 and moving average closeness features F18, F21, and F24.
- *CP* uses F1, temporal features F6-12, moving average closeness features F18, F21 and F24, and moving average period features F19, F22 and F25.

- *CPT* uses F1, temporal features F6-12, moving average closeness features F18, F21 and F24, moving average period features F19, F22 and F25, and moving average trend features F20, F23 and F26.
- *CPTM* uses all features.

These sub-experiments will also supply answers to RQ3 by determining which configuration of features are optimal for each model.

The second approach to answering RQ4 is to make use of the built-in feature importance computation in SciKit-Learn's Random Forest implementation. This is done for both zone-based and grid-based clustering using the *NONE*, *M* and *CPTM* configurations of features. However, some drawbacks of this approach are discussed in Chapter 6.

Evaluation Metrics

The standard method for measuring performance of machine learning models in the BSS literature is to use Root Mean Squared Error (RMSE), shown in Equation 5.1.

$$RMSE = \sqrt{\frac{\sum_{i=1}^{n}(\hat{y}_i - y_i)^2}{n}} \tag{5.1}$$

RMSE, in contrast to MAE, punishes large errors heavily. If the predicted flow deviates by much from the true flow, it could ultimately lead to a completely false sense of the BSS demand, so punishing large errors is desired. Furthermore, RMSE outputs values in the same units as the dependant variables being modelled, e.g. RMSE of 10 means that the predicted outflow, inflow and internal flow was off by 10 trips on average across all flows, which is an easy measure to understand.

5.2 Experimental Setup

5.2.1 Hyperparameters

As mentioned earlier, several hyperparameters were optimized through different optimization techniques for each model. The model configurations and hyperparameters are summarized in Table 5.1. The final selected hyperparameters for each model after optimization is presented in Table 5.2 and 5.3.

Parameter	**RF**	**RNN**	**DNN**	**ResNet**
num_trees	[50, . . . , 200]	-	-	-
rnn_cell[1]	-	gru $\vee$ lstm	-	-
rnn_units	-	[16, . . . , 256]	-	-
learning_rate	-	[5e-4, . . . , 8e-3]	[1e-4, . . . , 5e-2]	2e-4
seq_length	-	[18, 54, 126, 252]	-	-
activation_fn	-	tanh	ReLU	ReLU
optimizer[2]	-	Θ_R	$\Theta_R \vee \Theta_{adam} \vee \Theta_{ag}$	$\Theta_{adam} \vee \Theta_R$
first_layer_size	-	-	[64, . . . , 700]	-
num_layers	-	1	[1, . . . , 6]	-
decay_rate	-	-	[0.1, . . . , 0.8]	-
batch_size	-	[128, . . . , 1024]	[128, . . . , 1024]	32
residual_units	-	-	-	[4, . . . , 12]

[1] C_G = GRU, C_L = LSTM
[2] Θ_R = RMSprop, Θ_{adam} = Adam, Θ_{ag} = AdaGrad

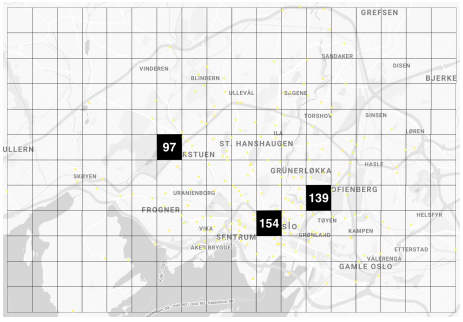
Table 5.1: Parameters for every model. Parameters with continuous parameter ranges that were optimized are denoted with brackets []. Parameters with categorical parameters that were optimized are denoted with logical ORs $\vee$ in between. RF used Grid Search, RNN and DNN used Bayesian Optimization, ResNet used manual search.

5.2.2 Dataset alterations

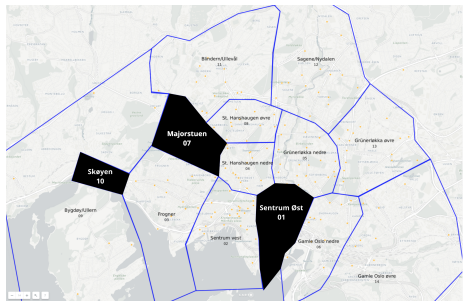
The datasets used in the experiments and how they were produced are described in greater detail in Chapter 4.1.1. For each dataset *flow-by-grid* and *flow-by-zone*, multiple different combinations of features were used as described in Section 5.1. However, some alterations had to be made to the datasets in order for them to be used by RNN and ResNet.

Recall from Chapter 4.2.4 that a limitation of the RNN model is that a separate model must be optimized and trained for every clustered area (cell or zone).

Due to time constraints it was not feasible to train and optimize separate RNN models for all 216 grid cells and 14 zones. Therefore, it was decided to choose three different grid cells and three different zones with varying levels of traffic volume and optimize models for these areas, which is used as a proxy for the prediction performance of all areas. Low-, medium- and high-traffic volume areas were chosen for each of the clustering methods. Figure 5.1 shows the geospatial locations for each of the areas highlighted in black. The traffic volumes for each of the selected areas is presented in pie charts in Figure 5.2, which shows that the three different areas chosen for each clustering method are of different levels of traffic volume.

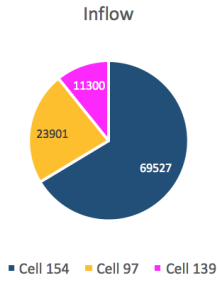


(a) Cell 154 (high-volume, near Youngstorget), cell 97 (mid-volume, near Majorstuen) and cell 139 (low-volume, Herslebs gate)

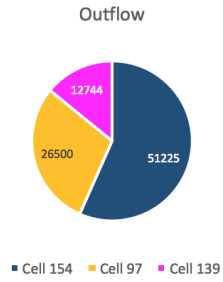


(b) Zone 1 (high-volume, Sentrum Øst), zone 7 (mid-volume, Majorstuen) and zone 10 (low-volume, Skøyen)

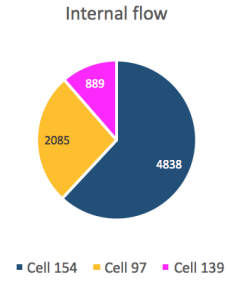
Figure 5.1: Low-, medium- and high-traffic volume grid cells and zones used for the RNN model.



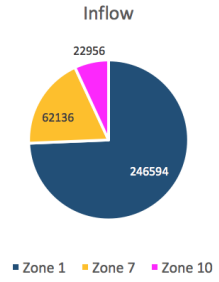
(a) Grid cell inflow



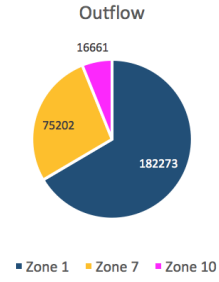
(b) Grid cell outflow



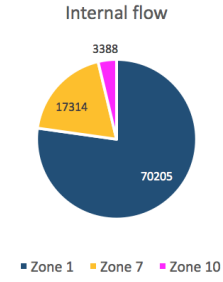
(c) Grid cell internal flow



(d) Zone inflow



(e) Zone outflow



(f) Zone internal flow

Figure 5.2: Pie charts describing the traffic volumes for each of the zones and grid cells chosen for the RNN model. The data is sampled from the period April 4th 2017 - August 4th 2017. The charts clearly show that low-, medium- and high-traffic volume areas were chosen.

Additionally, as explained in the section titled *Transforming Flow Datasets to Images for ResNet* in Chapter 4.1.1, the ResNet datasets were limited in time span from April 16th 2017 - October 15th 2017 in order to reduce sparsity.

5.2.3 Environment

Development and prototyping of all models was done on a personal MacBook Pro laptop. Optimization, training and experiments for the RF and ResNet models was also done on this laptop. Optimization, training and experiments for the DNN and RNN models was done on a Google CloudML virtual machine. In total, 435 hours was spent on optimization and training of the RNN and DNN models on Google CloudML.

MacBook Pro running macOS High Sierra version 10.13.3. 2.7 GHz Intel Core i5 processor, 16 GB 1867 MHz DDR3 memory, Intel Iris Graphics 6100 1536 MB GPU.

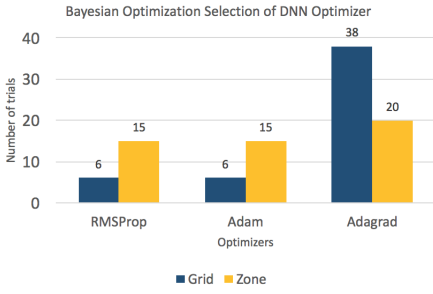
Google CloudML Virtual Machine running a single BASIC_GPU worker instance on a standard_gpu Compute Engine machine. It has one NVIDIA Tesla K80 GPU.

5.3 Experimental Results

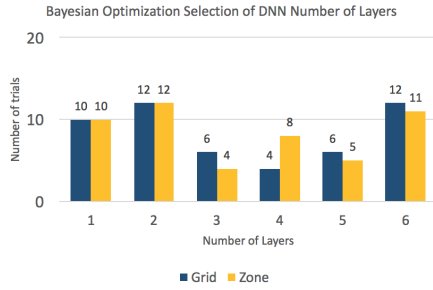
5.3.1 Experiment 1: Hyperparameter Optimization

Hyperparameter Selection Charts

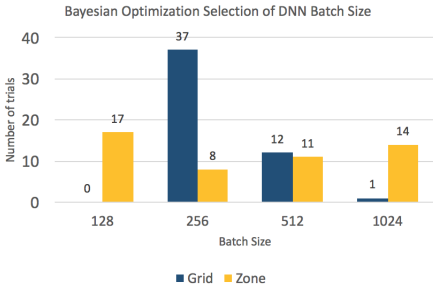
This section presents multiple charts showing the selections of hyperparameters that were chosen by Bayesian Optimization for the DNN (Figure 5.3) and RNN (Figure 5.4) high-feature models for both grid- and zone-based clustering. Low-feature models were optimized in the same manner, but charts for these are omitted due to similarity. Charts are not presented for the RF and ResNet hyperparameter selection process due to the fact that grid search and manual search was performed on these models. Bayesian Optimization was run for 50 trials for DNN models and 30 trials for RNN models. Please note that the charts of RNN hyperparameter selections (Figure 5.4) are for the medium-traffic models (cell 97 / zone 7).



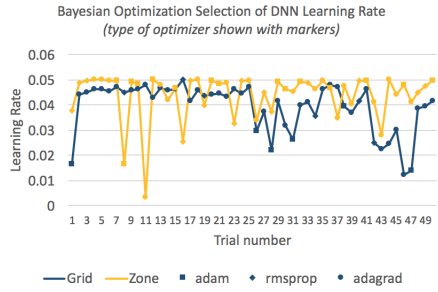
(a) Optimizer selection distribution



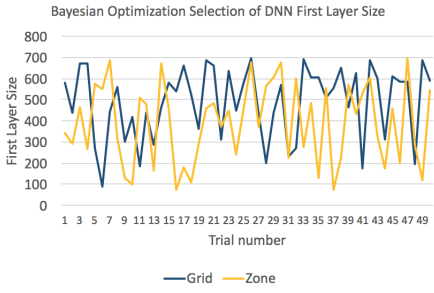
(b) Number of layers selection distribution



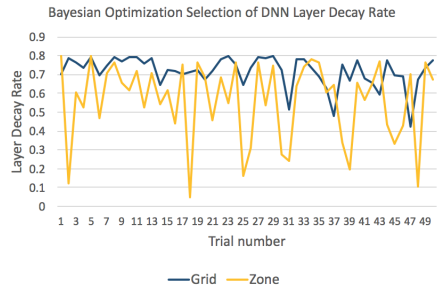
(c) Batch size selection distribution



(d) Learning rate selection per trial

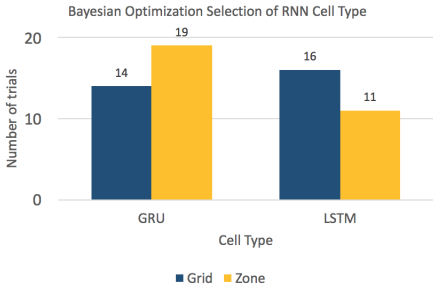


(e) First layer size selection per trial

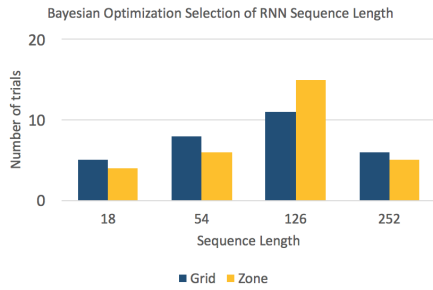


(f) Layer decay rate selection per trial

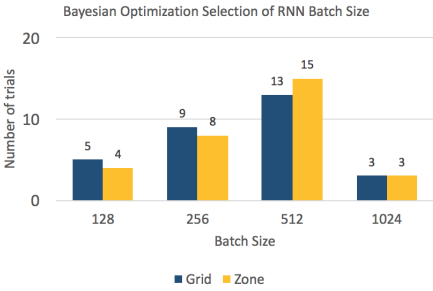
Figure 5.3: Hyperparameter selections using Bayesian Optimization for DNN. Grid-based models in dark blue, zone-based models in yellow. Figure a, b, and c shows the number of times the respective categorical parameters were selected over 50 trials. Figure d, e and f shows the respective parameter values per trial run for continuous parameters.



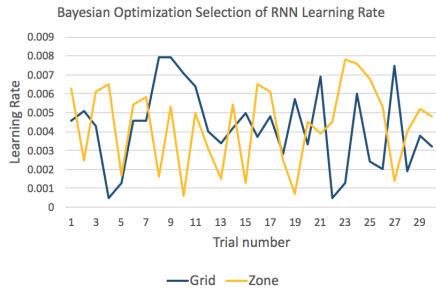
(a) Cell type selection distribution



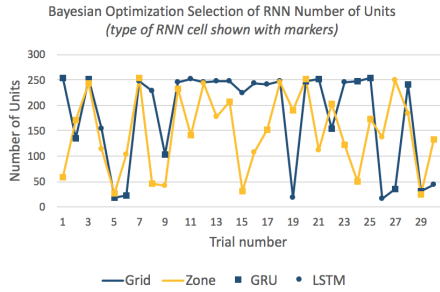
(b) Sequence length selection distribution



(c) Batch size selection distribution



(d) Learning rate selection per trial

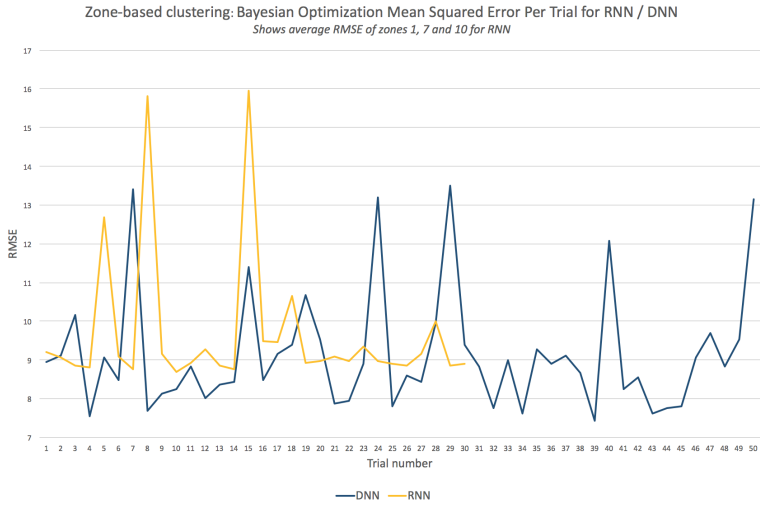


(e) Number of units selection per trial

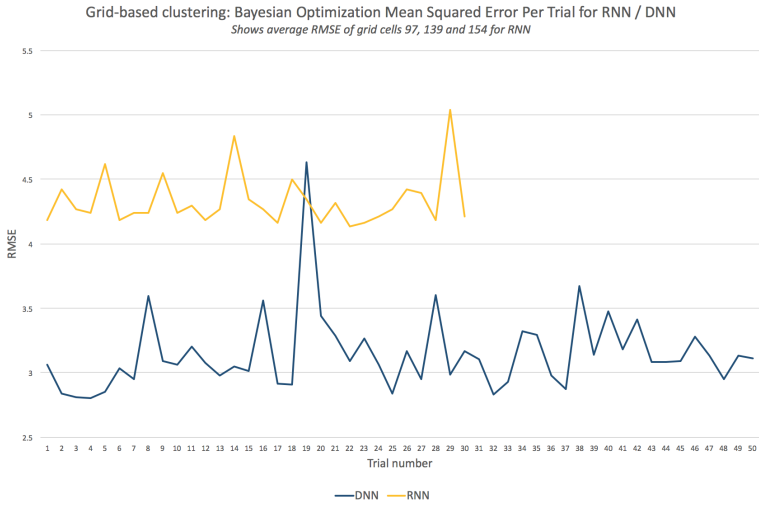
Figure 5.4: Hyperparameter selections using Bayesian Optimization for RNN (medium-traffic model). Grid-based models in dark blue, zone-based models in yellow. Figures a, b, and c shows the number of times the respective categorical parameters were selected over 30 trials. Figures d and e shows the respective parameter values per trial run for continuous parameters.

Optimal Hyperparameters

After running Bayesian Optimization for RNN and DNN, the hyperparameters in the trial that yielded the lowest RMSE were picked. Figure 5.5a shows the RMSE per trial of Bayesian Optimization for zone-based clustering, while Figure 5.5b shows the same for grid-based clustering. Note that the RNN graphs show the average RMSE of low-, medium- and high-traffic volume models. For reproducibility, the final optimal hyperparameters for most models are presented in Table 5.2 for zone-based clustering and Table 5.3 for grid-based clustering, except the hyperparameters for low- and high-traffic RNN models, which can be found in Appendix C due to space limitations.



(a) RMSE for every trial of hyperparameter optimization for zone-based DNN and RNN.



(b) RMSE for every trial of hyperparameter optimization for grid-based DNN and RNN.

Figure 5.5: RMSE for every trial of hyperparameter optimization for both zone-, and grid-based clustering for DNN and RNN. Note that the RNN graphs are the average of low-, mid- and high-traffic volume models.

Optimal hyperparameters for zone-based models
Format: *low-feature models (NONE, M) / high-feature models (C, CP, CPT, CPTM)*

Parameter	RF	$\text{RNN}_{\text{medium-traffic}}$	DNN	ResNet
num_trees	100 / 200	-	-	-
rnn_cell	-	gru / gru	-	-
rnn_units	-	232 / 249	-	-
learning_rate	-	0.005 / 0.0058	0.0499 / 0.0403	2e-4
seq_length	-	126 / 126	-	-
activation_fn	-	tanh	ReLU	ReLU
optimizer[1]	-	Θ_R	Θ_{adam} / Θ_{ag}	Θ_{adam}
first_layer_size	-	-	572 / 589	-
num_layers	-	1	3 / 2	-
decay_rate	-	-	0.2215 / 0.1955	-
batch_size	-	512 / 512	512 / 256	32
residual_units	-	-	-	4

[1] Θ_R = RMSprop, Θ_{adam} = Adam, Θ_{ag} = AdaGrad

Table 5.2: Optimal hyperparameters for every model when using zone-based clustering. Low-feature models (NONE, M) on the left side of the slash (/), high-feature models (C, CP, CPT, CPTM) on the right. Note that only the hyperparameters for medium-traffic RNN-models is presented; hyperparameters for low-traffic and high-traffic RNN-models can be found in Appendix C.

Optimal hyperparameters for grid-based models
Format: *low-feature models (NONE, M) / high-feature models (C, CP, CPT, CPTM)*

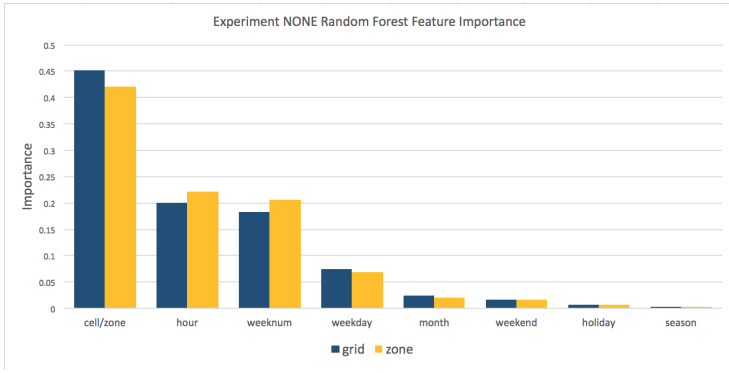
Parameter	RF	$\text{RNN}_{\text{medium-traffic}}$	DNN	ResNet
num_trees	100 / 200	-	-	-
rnn_cell	-	lstm / gru	-	-
rnn_units	-	154 / 251	-	-
learning_rate	-	0.00046 / 0.00054	0.0417 / 0.0462	2e-4
seq_length	-	126 / 126	-	-
activation_fn	-	tanh	ReLU	ReLU
optimizer[1]	-	Θ_R	Θ_{ag} / Θ_{ag}	Θ_{adam}
first_layer_size	-	-	655 / 671	-
num_layers	-	1	5 / 2	-
decay_rate	-	-	0.792 / 0.7381	-
batch_size	-	256 / 512	128 / 256	32
residual_units	-	-	-	8

[1] Θ_R = RMSprop, Θ_{adam} = Adam, Θ_{ag} = AdaGrad

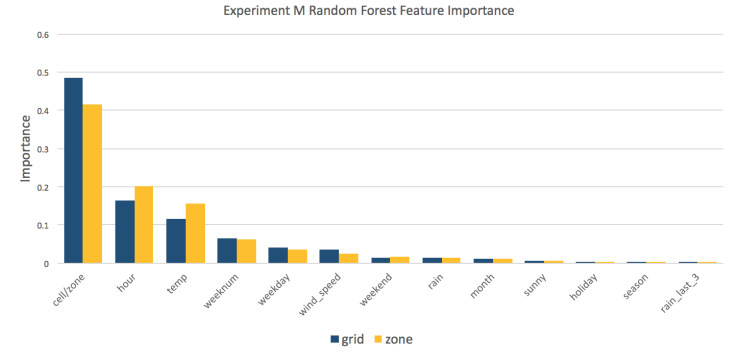
Table 5.3: Optimal hyperparameters for every model when using grid-based clustering. Low-feature models (NONE, M) on the left side of the slash (/), high-feature models (C, CP, CPT, CPTM) on the right. Note that only the hyperparameters for medium-traffic RNN-models is presented; hyperparameters for low-traffic and high-traffic RNN-models can be found in Appendix C.

5.3.2 Experiment 2: Determining Influential Features

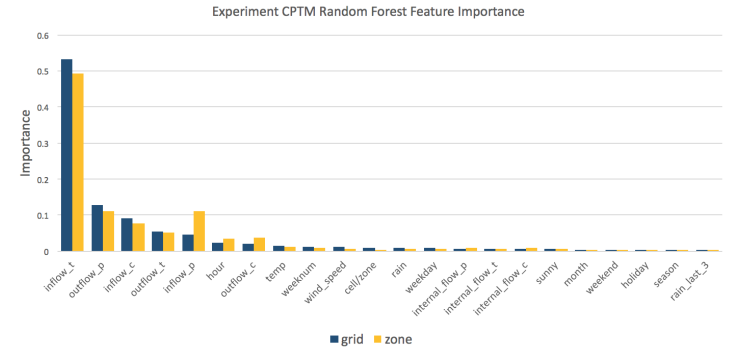
Two approaches were taken in order to determine which features were most influential. The first approach was to use the built-in feature importance measurement that comes with Scikit-Learns RF implementation. Figure 5.6a, 5.6b and 5.6c shows the measured feature importances when running RF using the NONE, M and CPTM feature configurations respectively. The second approach was to test all models using all six configurations of features. RMSE was used as a performance metric for all tests. Results for all models using grid-based clustering is presented in Table 5.4, and using zone-based clustering is presented in Table 5.5. Note that the values presented for RNN are the averages of low-, medium- and high-traffic RNN models. This experiment also provides answers to RQ3 by determining which model and feature configuration combination yields the lowest RMSE value for both zone- and grid-based clustering. Experiments using meteorological information was not performed for the ResNet model due to time constraints, and is proposed as future work in Chapter 6.4.



(a) RF feature importance using NONE feature configuration



(b) RF feature importance using M feature configuration



(c) RF feature importance using CPTM feature configuration

Figure 5.6: Random Forest feature importance measurements for grid- and zone-based clustering. Figure a shows feature importance using the NONE configuration of features, Figure b using the M configuration and Figure C using the CPTM configuration.

RMSE Evaluation metrics for grid-based clustering

Features	RA	HA	LV	RF	RNN[1]	ResNet	DNN
NONE	100.01	8.58	3.60	3.46	4.65	4.60	4.33
M	-	-	-	3.18	3.85	-	3.16
C	-	-	-	3.27	3.94	4.58	3.26
CP	-	-	-	3.28	3.91	4.57	3.00
CPT	-	-	-	3.28	3.87	4.58	2.95
CPTM	-	-	-	3.18	4.13	-	**2.84**

[1] Shows the average RMSE of low-, medium- and high-traffic RNN models

Table 5.4: RMSE of all models for all configurations of features when using grid-based clustering. DNN performed best with the CPTM configuration, reaching an RMSE of 2.84.

RMSE Evaluation metrics for zone-based clustering

Features	RA	HA	LV	RF	RNN[1]	ResNet	DNN
NONE	221.02	30.34	11.76	10.6	7.62	12.49	14.62
M	-	-	-	9.22	7.74	-	9.56
C	-	-	-	9.11	7.5	10.65	9.00
CP	-	-	-	9.40	8.22	10.23	8.26
CPT	-	-	-	9.41	9.11	9.85	7.96
CPTM	-	-	-	9.06	8.54	-	**7.45**

[1] Shows the average RMSE of low-, medium- and high-traffic RNN models

Table 5.5: RMSE of all models for all configurations of features when using zone-based clustering. DNN performed best with the CPTM configuration, reaching an RMSE of 7.45.

Chapter 6

Evaluation and Conclusion

Recall from Chapter 1 the overarching goal of this thesis:

Goal To enable Bike-Sharing System decision makers to proactively rebalance service stations based on accurate predictions of future bicycle flow.

In order to accomplish this goal, it was decided to focus on cluster-level flow prediction using machine learning techniques. Both grid-based clustering and zone-based clustering was explored, where grid-based clustering can be regarded as a naive clustering method which requires no prior knowledge of the Bike-Sharing System (BSS), while zone-based clustering can more accurately define regions based on spatial and temporal patterns, but requires expert knowledge of the system. Several machine learning algorithms principally intended for different use cases were implemented and rigorously optimized, trained and tested in order to determine which ML algorithm is most suitable for the problem of predicting cluster-level flow in BSSs. The entire system architecture, as well as the implementation details of each ML model and how these can potentially be used in a production setting, was described in detail in Chapter 4.

In this chapter, an evaluation of the experiment results from Chapter 5 is presented in Section 6.1. The results of the experiments, as well as the overall merits and limitations of this thesis, is discussed in Section 6.2. Section 6.3 describes the contributions of this thesis, before finally Section 6.4 describes several proposed extensions and solutions to limitations of this work.

6.1 Evaluation

This section presents an evaluation of the experiment results presented in Chapter 5. The first experiment was designed to empirically find the best hyperparameters

for every model such that optimal performance was ensured for each one, so that the models could be objectively compared against each other. The results for this experiment are evaluated in Section 6.1.1. The second experiment relates to the feature processing part of this thesis, and was designed to thoroughly test which variables were most important in order to obtain satisfactory prediction performance. The results of this experiment are evaluated in Section 6.1.2.

6.1.1 Experiment 1: Hyperparameter Optimization

This experiment was mainly designed to provide answers to RQ3.

Research question 3 How can machine learning be used to predict the flow of bikes in bike-sharing systems?

It was decided to implement four machine learning algorithms originally intended for different use cases. Random Forest and Deep Neural Network had been used in the literature with success for a range of different problems related to BSS prediction. Additionally, it was decided to adapt the Deep Residual Network (ResNet) model by Zhang et al. [2016] for this thesis for two reasons; first, because of the authors compelling results when applying the model to a BSS use case, and secondly due to the fact that ResNet is currently one of the best performing models in the ILSVRC image recognition competition. Finally, because the problem of predicting flow in BSSs is naturally a time series prediction problem, and due to the fact that RNNs had not been used in the literature, two unique RNN variants were also tested.

In order to ensure that every model would perform well, a lot of time was spent optimizing each model. From the experiment details provided in Chapter 5, it is clear that the primary focus was on Bayesian Optimization (BO) for the RNN and DNN models. The RF model only had a single parameter to optimize, *num_trees*, so a simple grid search sufficed for this model. Furthermore, it was assumed that the ResNet model adapted from Zhang et al. [2016] already was optimized to some extent - however, some manual optimization was performed on this model by trying out different combinations of a few hyperparameters.

From the hyperparameter selection charts for DNN and RNN in Figure 5.3 and Figure 5.4 respectively, a few insights from the BO process becomes apparent.

- During initial prototyping of the DNN model, RMSProp was mainly used. However, it is clear that BO favored Adagrad for both grid- and zone-based clustering.
- There seems to be no apparent pattern for the number of layers used in the DNN models, but two and six layers were chosen marginally more often than the others.

- For grid-based clustering using the DNN model, 256 as batch size was by far the most popular choice, while the extremes at 128 and 1024 were almost not chosen at all. Interestingly, there does not seem to be a clear favor of batch size when using zone-based clustering.
- For the RNN model, GRU was a more frequent choice for zone-based clustering, while LSTM was more frequent for grid-based clustering.
- A sequence length of 126 was most frequently picked for both clustering methods for RNN. This may be due to the fact that a sequence length of 126 equals approximately seven days of recurrent behavior. It is intuitive to think that a sequence length of seven days would be desirable due to the periodicity of BSS traffic flow (ref Figure 4.2).

However, perhaps the most interesting take-away from this experiment can be found by looking at the graph of RMSE per trial of BO, presented in Figure 5.5. As explained in Chapter 2.5, BO is supposed to search the hyperparameter space intelligently, as opposed to Grid Search and Random Search. Therefore, it should be expected that the graph of RMSE per trial of BO should gradually converge towards a lower RMSE as the search continues. Some fluctuations should appear due to the exploitation/exploration trade-off that BO makes use of. Nevertheless, it is clear from the graphs in Figure 5.5 that BO searches the hyperparameter space rather randomly for both RNN and DNN. The RMSE per trial run fluctuates wildly, and does not steadily decrease. This suggests that BO did not work as intended, but rather performed similarly to how Random Search would have performed. It is unclear why this is the case, but the author of this thesis speculates that it may be because too many hyperparameters were optimized simultaneously. If the number of hyperparameters to optimize was reduced, or the optimization process was allowed to continue for more trials, the RMSE may have steadily decreased as expected. Nevertheless, Random Search is widely used with great success in the ML community, so even if BO behaved like Random Search, it can be expected that near-optimal hyperparameters were found.

6.1.2 Experiment 2: Determining Influential Features

This experiment simultaneously aimed to provide answers to RQ3 and RQ4. This is because during this experiment, all models were tested with different feature configurations in order to determine which model and feature combination yielded the lowest RMSE on the test set.

Research question 4 Which features have the highest impact on the learning algorithm's ability to accurately perform predictions?

Two approaches were taken in order to answer this question. First, feature importance measurement was performed using the built-in functionality provided by Random Forest. RF was run with three different configurations of features: NONE, M and CPTM. The results of these three runs are presented in Figure 5.6 in Chapter 5.3.2. However, a major limitation of using RF to measure feature importance was identified in this experiment, which in essence rendered all measured importances invalid. A general pattern in the RF feature importance sub-experiments is that features with a high number of categorical values or with a wide range of continuous values are consistently weighted heavier than other features. For example, when introducing moving average features in the CPTM experiment, these CPT features generally dwarf the importance of most other features. This has to do with how importance is measured in Scikit-Learns RF implementation, where the Gini impurity measure artificially prefers potentially suboptimal predictors solely because of the scale of measurement or number of categories (Strobl et al. [2007]). Unfortunately, a consequence of this is that the feature importances measured by RF can **not** be deemed as accurate, because this thesis uses a combination of both categorical features with as few as two possible values, and continuous features with a much wider range of possible values, which skews the results in favor of the latter.

Nevertheless, the second approach to answer RQ4 does provide some answers as to which features were most influential on the prediction performance, while simultaneously determining how machine learning best can be used to predict flow in BSSs. In this approach, all ML models were evaluated using six different configurations of features; NONE, M, C, CP, CPT, and CPTM. In order for a model to be regarded as successful, it had to perform better than all baselines.

It is clear from the final RMSE tables presented in Table 5.4 and Table 5.5 that Last Value is decidedly the best baseline for both grid- and zone-based clustering, outperforming Historical Average and Random Algorithm. Note that all baselines simply perform predictions based on past values of flow, and therefore do not make use of any extra features. In order for an ML model to be regarded as successful, it had to perform better than all baselines.

Grid-based clustering experiments

For grid-based clustering, only RF and DNN actually performed better than the best baseline LV. RNN did not perform better than LV with any of the feature configurations; in fact, the best configuration for RNN was M, which yielded a RMSE of 3.85. By adding moving average features CPT to the RNN model, we are essentially adding features that the network is supposed to learn through recurrent unfolding. Hence, adding these features manually may just add noise to the network, which in turn reduces the prediction accuracy. For the DNN model

the opposite is observed; prediction performance increases as more features are added.

Surprisingly, ResNet does not perform well when using grid-based clustering, although the findings presented by Zhang et al. [2016] suggests otherwise. This may be due to the fact that the ResNet model does not make use of meteorological information in any of the experiments. However, in the C, CP, and CPT experiments, it is observed that ResNet performs worse than all other ML models.

RF performs best with the M and CPTM configurations, with a RMSE value of 3.18. This suggests that RF is not able to make full use of the moving average features CPT, and that meteorological features are more important to this particular model.

The best performance of all models when using grid-based clustering is found using DNN with the CPTM configuration, which yielded an RMSE value of **2.84**, outperforming LV. It is observed that when integrating meteorological features in the M experiment, the performance drastically increases. When moving average features CPT are included as well in the CPTM experiment, the performance increases further. This suggests that for the DNN model, the flow of bikes is strongly dependant on both the current weather and temporal patterns of closeness, period and trend.

Zone-based clustering experiments

In the zone-based clustering experiments, all ML models with some feature configuration are able to perform better than the best baseline LV. ResNet is the worst ML model, but still manages to perform quite well with an RMSE value of 9.85 using the CPT configuration, outperforming LV with an RMSE of 11.76. Prior to the experiments, it was assumed that ResNet would perform much worse using zone-based clustering than grid-based clustering, due to the fact that the "image" supplied as the dataset has a much lower resolution using zone-based clustering (2×7) than grid-based clustering (12×18). Although ResNet did perform worse than all other ML models in this experiment, it still managed to beat the best baseline.

RNN showed some of the same symptoms as in the grid-based experiments, namely that adding moving average features CPT resulted in added noise. However, it is observed that using the C configuration actually yielded better results than M or NONE. Additionally, the NONE experiment had better results than M, which indicates that the network was unable to make full use of meteorological information. It is unclear why this is the case, and further testing using all zones is required to gain knowledge on this subject.

When using the DNN model, the same pattern as in the grid-based experiments is observed; the model performance increases as more features are added.

There is a significant jump in performance when adding meteorological information, but the decidedly best performance presents itself when using all features in the CPTM experiment, which resulted in an RMSE of **7.45**, outperforming all other models.

From the grid- and zone-based experiments, it becomes apparent that bike flow is generally dependant on both meteorological factors and temporal patterns of closeness, period and trend. This finding presents itself most clearly when testing the DNN model using different configurations of features. Meteorological features seem to be most important for RF. A shortcoming of this thesis is that ResNet was not tested with meteorological features, so it could not be decided whether these features are important to this model. However, from the zone-based experiments, it is clear that ResNet benefits greatly from introducing closeness, period and trend features. RNN performs well when introducing meteorological features, but performance declines when using moving average features, indicating that this is just seen as noise for this model.

In conclusion, the machine learning model that is best suited for the task of predicting both grid- and zone-level flow is a standard DNN model using meteorological features and information on temporal closeness, period and trend.

6.2 Discussion

One of the most crucial aspects of operating any transportation company is the ability to predict future demand, as it provides essential input to a wide range of functional areas within the business. Bike-Sharing Systems are a relatively new mode of transport in the modern urban world, and benefits from accurate demand prediction just as much as any other transportation mode. The research conducted in this thesis aims to predict future cluster-level flow in BSSs by leveraging machine learning techniques. In order to get an overview of the current research literature on demand prediction, two research questions were defined. Because BSSs are in their industrial infancy compared to other transportation industries, it was decided to broaden the literature search by including the more general topic of on-demand transport services demand prediction, but focusing on mainly BSSs. The idea was that demand prediction research in other transportation industries is likely to use techniques that are applicable to the domain of BSSs.

Research question 1 What is state-of-the-art within on-demand transport services demand prediction?

Research question 2 Which neural network based techniques have been used within the domain of on-demand transport services demand prediction?

These research questions were explored by conducting a thorough Structured Literature Review (SLR), presented in Chapter 3. A wide range of aspects related to demand prediction were considered, including analysis of spatio-temporal usage patterns, examination of critically influential variables, different levels of granularity on which prediction is performed, how entities can be clustered to increase accuracy, and other similar problems related to demand prediction. Several prediction techniques have been used in the literature, ranging from classical statistical forecasting tools such as Autoregressive Integrated Moving Average (ARIMA), to very recent advances in neural network research using ResNets.

In the BSS domain, the research community agrees that bike traffic is strongly affected by primarily two factors; weather and spatio-temporal patterns. Bad weather intuitively reduces system usage drastically, so integrating meteorological information in prediction systems is crucial. Additionally, traffic in one area is spatially affected by traffic in other areas, and temporally affected by past patterns of traffic in the system. These factors were tightly integrated in the models implemented in this thesis through the M and CPT features. However, the SLR uncovered that no researchers had studied the impact of events, which is likely a strong influencer on bike traffic. Unfortunately, due to both technical details and time constraints, this was not studied in this thesis either, but is left as an important area of research for the community.

A fundamental issue with the current state-of-the-art on BSS demand prediction is that there is no universal frame for testing models within the research community. Models are evaluated on different datasets, causing an inability to properly compare results between researchers.

Several neural network based techniques has been used in the literature, including classic FFNNs using various methods of initializing the weights, GRNN, BN and more recently ResNets. Of the neural network based models reviewed, ResNet emerged as the model with most potential due to the overwhelming success when applied to car-sharing services demand prediction by Wang et al. [2017] and crowd flow prediction by Zhang et al. [2016]. Additionally, variations of FFNN has been widely applied with promising results. RF, although strictly not neural network based, has also been widely used, and generally delivers good performance without the need for tedious feature preprocessing or hyperparameter optimization. Surprisingly, usage of RNN was not found during the SLR, despite the fact that demand in all transportation industries follows a time series pattern. It was decided to implement these four ML models in order to answer RQ3 and RQ4 by performing the experiments described in Section 6.1.

Despite the promising results achieved by Wang et al. [2017] and Zhang et al. [2016], ResNet actually turned out to be worst performing model for both grid- and zone-based clustering in this thesis. The best performance for grid-based

clustering was seen using the CPT feature configuration at 4.58 RMSE, whereas LV achieved 3.60. Using zone-based clustering, ResNet with CPT beat LV with an RMSE of 9.85 against 11.76, but still performed worse than all other ML models. There are several reasons as to why this may be the case. The ResNet model does not use meteorological information, which is shown to be important for the other ML models. Furthermore, the model hyperparameters used by Zhang et al. [2016] were assumed to work well for this dataset too, and therefore little time was spent on optimization.

RF is commonly used within the research community with good reason. The algorithm requires little to no feature processing or hyperparameter tuning, and still performs well. In this thesis, RF was found to be better than the best baseline using all feature combinations, and CPTM achieved the best results for both clustering methods. RF was the second best predictor for grid-based clustering and third best for zone-based. However, a major limitation of utilizing feature importance measurements was uncovered. The unfortunate consequence of this was that we could not measure the importance of each feature individually. Furthermore, because this model is implemented using Scikit-Learn, it is not a straight-forward task to serve it in the cloud for online batch predictions using Oslo City Bike's Google Cloud environment.

The RNN model was the second best model in the zone-based experiments, but did not perform better than LV in the grid-based experiments. This may be due to the prominent limitation of this model, namely that a separate RNN model must be optimized and trained for each area. Due to time constraints in this thesis, the RNN model was only tested on three areas in each clustering method, which were picked to serve as representative areas of three different levels of traffic volume. Since there are far more areas using grid-based clustering than zone-based clustering, the three grid cells picked may not have been enough to accurately represent the entire system, which caused the RMSE to be artificially large. However, this can also be said for the opposite case; RNN performed very well in the zone-based experiments, but this can be an artificially low RMSE based on the same reasons as for grid-based. Thus, further testing of this model using all clustering areas is required to conclusively determine the feasibility of using RNNs for this prediction task. The RNN model is implemented using Google Cloud-compliant TensorFlow code, and can therefore be extended to be used in a production setting managing online batch predictions. However, because a separate model is required for each area, this model is far more complicated and expensive to maintain than DNN in such a production environment.

DNN was found to be the best performing machine learning model for predicting both grid- and zone-level flow. In the grid-based clustering experiments, DNN achieved a RMSE of 2.84 using the CPTM configuration of features, whereas LV achieved 3.60, resulting in **21.1%** improvement over the best baseline. For zone-

based clustering, DNN with the same feature configuration achieved 7.45 RMSE, while LV achieved 11.76, which is an improvement of **36.65%**. In contrast to RNN, the DNN model is a single unified model capable of modelling flow processes of all areas simultaneously, as opposed each area individually. This makes DNN a suitable candidate as a production-ready predictor. The DNN model is also implemented with Google Cloud-compliant TensorFlow code, and can easily be extended to receive and handle online batch predictions in a production environment.

In conclusion, this thesis found that there are multiple feasible machine learning algorithms to predict cluster-level flow in the Oslo bike-sharing system. DNN proved to be most successful for both grid-based and zone-based clustering, with improvements of 21.1% and 36.65% respectively over the leading baseline. The developed system is designed to run on Google Cloud, and can potentially serve as a handler for online batch prediction jobs where the results can be fed through to a proposed flow visualization module. In a production setting, this enables bike-sharing system decision makers to proactively rebalance service stations more precisely than the current naive baseline predictor LV. The predictions made by this system can also be used as input to other demand prediction systems that work on the more granular station level.

6.3 Contributions

There are primarily three contributions presented in this thesis:

1. An in-depth exploration and discussion of current state-of-the-art solutions within on-demand transport services demand prediction using machine learning models, with a primary focus on the domain of bike-sharing systems. A thorough reproducible structured literature review was conducted in Chapter 3 in order to gain insights to what models could be studied further in this thesis.

2. A comparison of RF, RNN, DNN and ResNet when applied to real BSS data provided by Oslo City Bike. The comparison discusses strengths and weaknesses regarding performance, implementation details and applicability to the problem of predicting cluster-level flow. Based on the literature search, this thesis presents the first evaluation of using RNNs within the domain of BSSs. Furthermore, an examination of important features is provided, concluding that meteorological factors and temporal features of closeness, period and trend have a strong impact on prediction performance for most models.

3. A DNN-based machine learning model that outperforms the best baseline and all other ML models. The DNN model is optimized, trained and tested on Google CloudML using TensorFlow, and can with few modifications be capable of serving real-time predictions on streaming flow data provided by Oslo City Bike, making the model a suitable candidate that fits into Oslo City Bike's existing cloud environment. A description of the model architecture and the proposed production pipeline was described in Chapter 4.

6.4 Future Work

This section presents potential extensions that can be made to the system, improvements that can increase prediction accuracy and possible solutions to limitations that were uncovered in this thesis.

6.4.1 Deep Residual Network

The ResNet model used in this thesis was a fork an open-source repository by Zhang et al. [2016], and it was assumed that the model would work well for this dataset without much optimization. However, ResNet turned out to be inferior to the other models, despite promising results by Zhang et al. [2016] and Wang et al. [2017]. Further exploration on optimal hyperparameters and network topology for this particular dataset, as well as inclusion of meteorological information, is suggested as an open research area for the future.

6.4.2 Modelling spatial dependencies

The ResNet model is the only model that inherently models spatial dependencies between neighboring stations due to the fact that it is a convolutional neural network. The traffic of neighboring stations has been shown to be positively correlated with the traffic of an individual station (Ashqar et al. [2017], Liu et al. [2015], Zhang et al. [2016]). In order for the RF, RNN and DNN models to exploit spatial dependencies, a proposed extension is to include the outflow, inflow and internal flow of the X nearest neighbors as input features to these models. The number of neighbors to include can be found be found by using K-Nearest Neighbors analysis in terms of either spatial proximity or temporal usage pattern similarity.

6.4.3 Events and point-of-interest features

Using events and point-of-interest proximity as features would also be interesting to explore. If there are 200 people attending an event in an area at some time, that is highly likely to influence bike flow in that area. One way to include include events as features would be to exploit Facebook Events, as this is likely to yield the most accurate data on events. However, to the best of this author's knowledge, it is currently not possible to access information on Facebook Events through public APIs. Google measures the activity of certain areas at certain times by determining the number of GPS signals received from mobile phones, but is unclear whether this information is publicly available or not. Liu et al. [2015] used a station's proximity to several points of interest as features to increase prediction accuracy, so this is also proposed as an extension to this thesis.

6.4.4 Production environment pipeline

The recommended model to predict cluster-level flow was found to be DNN. The DNN model is implemented using Google CloudML-compliant code, and is trained and optimized on Google CloudML. In order for the model to be used in a production setting, a few modifications to the code must be implemented. Specifically, the *predict* function must be rewritten to handle a prediction batch which is supplied to the model in a specific format through API calls. The model can then be deployed to TensorFlow Serving, Google's production environment for TensorFlow models. All trips in the Oslo City Bike BSS are recorded and streamed through a data logging service. In a production setting, Google Dataflow[1] could be used to periodically batch process all previous X trips and transform them to a flow format following the description laid out in Chapter 4.1.1. This data, combined with information on the current weather status, could then be used to make predictions of flow in all areas for the next hour. These predictions could be propagated through to the visualization module, which would be hosted as a Node.js app, and could serve as vital information to rebalancing trucks. Additionally, the predictions made by this system could be used as input to an algorithm which predicts station-level demand, which could drastically improve accuracy of such an algorithm.

6.4.5 Predicting flow trajectory

This thesis focuses only on predicting the number of incoming, outgoing and internal trips in each area of the Oslo BSS. An interesting extension of this is to not only predict the amount of trips in each area, but to predict where these

[1] https://cloud.google.com/dataflow/

trips are going or coming from. For example, if it is predicted 300 incoming trips to the Sentrum Vest zone, it would be desirable to know how many of these trips originated in each of the other zones. This would allow BSS management to not only estimate the demand in each zone individually, but provide greater insight to which zones rebalancing trucks should pick up bikes from and deliver bikes to, which would likely result in a greater balance across the entire system.

Bibliography

Alexandra Johnson, S. (2017). Common problems in hyperparameter optimization. [Accessed: 2018-04-01].

Ashqar, H. I., Elhenawy, M., Almannaa, M. H., Ghanem, A., Rakha, H. A., and House, L. (2017). Modeling bike availability in a bike-sharing system using machine learning. In *2017 5th IEEE International Conference on Models and Technologies for Intelligent Transportation Systems (MT-ITS), 26-28 June 2017*, 2017 5th IEEE International Conference on Models and Technologies for Intelligent Transportation Systems (MT-ITS), pages 374–8. IEEE.

Bacciu, D., Carta, A., Gnesi, S., and Semini, L. (2017). An experience in using machine learning for short-term predictions in smart transportation systems. *Journal of Logical and Algebraic Methods in Programming*, 87:52–66.

Bei, C., Pinelli, F., Sinn, M., Botea, A., and Calabrese, F. (2013). Uncertainty in urban mobility: Predicting waiting times for shared bicycles and parking lots. In *16th International IEEE Conference on Intelligent Transportation Systems (ITSC 2013)*, pages 53–58.

Bracewell, R. N. and Bracewell, R. N. (1986). *The Fourier transform and its applications*, volume 31999. McGraw-Hill New York.

Breiman, L. (2001). Random forests. *Machine learning*, 45(1):5–32.

Brochu, E., Cora, V. M., and De Freitas, N. (2010). A tutorial on bayesian optimization of expensive cost functions, with application to active user modeling and hierarchical reinforcement learning. *arXiv preprint arXiv:1012.2599*.

Caggiani, L., Ottomanelli, M., Camporeale, R., and Binetti, M. (2017). Spatio-temporal clustering and forecasting method for free-floating bike sharing systems. In *Advances in Intelligent Systems and Computing*, volume 539, pages 244–254.

Cho, K., Van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., and Bengio, Y. (2014). Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*.

Dali, L. and Mladenic, D. (2012). Bicikelj: Environmental data mining on the bicycle. In *Proceedings of the International Conference on Information Technology Interfaces, ITI*, pages 331–336.

Eldan, R. and Shamir, O. (2016). The power of depth for feedforward neural networks. In *Conference on Learning Theory*, pages 907–940.

Fei, L., Shihua, W., Jian, J., Weidi, F., and Yong, S. (2017). Predicting public bicycle rental number using multi-source data. In *2017 International Joint Conference on Neural Networks (IJCNN), 14-19 May 2017*, 2017 International Joint Conference on Neural Networks (IJCNN), pages 1502–9. IEEE.

Froehlich, J., Neumann, J., and Oliver, N. (2009). Sensing and predicting the pulse of the city through shared bicycling. In *IJCAI*, volume 9, pages 1420–1426.

Gallop, C., Tse, C., and Zhao, J. (2011). A seasonal autoregressive model of vancouver bicycle traffic using weather variables. *i-Manager's Journal on Civil Engineering*, 1(4):9.

Ghanem, A., Elhenawy, M., Almannaa, M., Ashqar, H. I., and Rakha, H. A. (2017). Bike share travel time modeling: San francisco bay area case study. In *2017 5th IEEE International Conference on Models and Technologies for Intelligent Transportation Systems (MT-ITS), 26-28 June 2017*, 2017 5th IEEE International Conference on Models and Technologies for Intelligent Transportation Systems (MT-ITS), pages 586–91. IEEE.

Giot, R. and Cherrier, R. (2014). Predicting bikeshare system usage up to one day ahead. In *Computational intelligence in vehicles and transportation systems (CIVTS), 2014 IEEE symposium on*, pages 22–29. IEEE.

Graves, A., Mohamed, A.-r., and Hinton, G. (2013). Speech recognition with deep recurrent neural networks. In *Acoustics, speech and signal processing (icassp), 2013 ieee international conference on*, pages 6645–6649. IEEE.

Han, Y., Côme, E., and Oukhellou, L. (2014). Towards bicycle demand prediction of large-scale bicycle sharing system. In *Transportation Research Board 93rd Annual Meeting*.

He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778.

Ho, T. K. (1995). Random decision forests. In *Document analysis and recognition, 1995., proceedings of the third international conference on*, volume 1, pages 278–282. IEEE.

Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, 9(8):1735–1780.

Hornik, K., Stinchcombe, M., and White, H. (1989). Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5):359–366.

Kaltenbrunner, A., Meza, R., Grivolla, J., Codina, J., and Banchs, R. (2010). Urban cycles and mobility patterns: Exploring and predicting trends in a bicycle-based public transport system. *Pervasive and Mobile Computing*, 6(4):455–466.

Keele, S. et al. (2007). Guidelines for performing systematic literature reviews in software engineering. In *Technical report, Ver. 2.3 EBSE Technical Report. EBSE*. sn.

Kofod-Petersen, A. (2012). How to do a structured literature review in computer science. *Ver. 0.1. October*, 1.

Labadi, K., Benarbia, T., Barbot, J.-P., Hamaci, S., and Omari, A. (2015). Stochastic petri net modeling, simulation and analysis of public bicycle sharing systems. *IEEE Transactions on Automation Science and Engineering*, 12(4):1380–1395.

LeCun, Y. (2007). Mnist handwritten digit database. [Accessed: 2018-02-14].

LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324.

Li, Y., Zheng, Y., Zhang, H., and Chen, L. (2015). Traffic prediction in a bike-sharing system. In *Proceedings of the 23rd SIGSPATIAL International Conference on Advances in Geographic Information Systems*, page 33. ACM.

Liu, J., Li, Q., Qu, M., Chen, W., Yang, J., Xiong, H., Zhong, H., and Fu, Y. (2015). Station site optimization in bike sharing systems. In *2015 IEEE International Conference on Data Mining*, pages 883–888.

MachineLearning-Wiki (2011). Machine learning wiki: Bias-variance trade-off. [Accessed: 2018-02-15].

Malani, J., Sinha, N., Prasad, N., and Lokesh, V. (2013). Forecasting bike sharing demand.

Meddin, R. and DeMaio, P. (2018). The bike-sharing world map. [Accessed: 2018-05-01].

Metrobike (2012). Bike-sharing systems growth 2002-2012. [Accessed: 2018-05-01].

Michalski, R. S., Carbonell, J. G., and Mitchell, T. M. (2013). *Machine learning: An artificial intelligence approach*. Springer Science & Business Media.

Milenković, M., Bojović, N., GliÅ¡ović, N., and NuhodÅ¾ić, R. (2014). Comparison of sarima-ann and sarima-kalman methods for railway passenger flow forecasting. *Civil-Comp Proceedings*, 104.

Moore, D. S., McCabe, G. P., and Craig, B. A. (2009). *Introduction to the Practice of Statistics*. WH Freeman New York.

Moreira-Matias, L., Gama, J., Ferreira, M., Mendes-Moreira, J., and Damas, L. (2013). Predicting taxiâpassenger demand using streaming data. *IEEE Transactions on Intelligent Transportation Systems*, 14(3):1393–1402.

OsloCityBike (2016). Own work. [Accessed: 2018-05-01].

Pillac, V., Gendreau, M., Guéret, C., and Medaglia, A. L. (2013). A review of dynamic vehicle routing problems. *European Journal of Operational Research*, 225(1):1–11.

Rudloff, C. and Lackner, B. (2014). Modeling demand for bikesharing systems: neighboring stations as source for demand and reason for structural breaks. *Transportation Research Record: Journal of the Transportation Research Board*, (2430):1–11.

Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986). Learning representations by back-propagating errors. *nature*, 323(6088):533.

Salaken, S. M., Hosen, M. A., Khosravi, A., and Nahavandi, S. (2015). Forecasting bike sharing demand using fuzzy inference mechanism. In *Neural Information Processing. 22nd International Conference, ICONIP 2015, 9-12 Nov. 2015*, volume pt.III of *Neural Information Processing. 22nd International Conference, ICONIP 2015. Proceedings: LNCS 9491*, pages 567–74. Springer International Publishing.

Scikit-Learn (2011). Scikit-learn: validation curves. [Accessed: 2018-02-15].

Shaheen, S., Guzman, S., and Zhang, H. (2010). Bikesharing in europe, the americas, and asia: past, present, and future. *Transportation Research Record: Journal of the Transportation Research Board*, (2143):159–167.

Simonyan, K. and Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*.

Strobl, C., Boulesteix, A.-L., Zeileis, A., and Hothorn, T. (2007). Bias in random forest variable importance measures: Illustrations, sources and a solution. *BMC bioinformatics*, 8(1):25.

Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., Rabinovich, A., et al. (2015). Going deeper with convolutions. Cvpr.

Wang, D., Cao, W., Li, J., and Ye, J. (2017). Deepsd: Supply-demand prediction for online car-hailing services using deep neural networks. In *Proceedings - International Conference on Data Engineering*, pages 243–254.

Yang, M., Guang, Y., and Zhang, X. (2015). Public bicycle prediction based on generalized regression neural network. In *2nd International Conference on Internet of Vehicles ndash; Safe and Intelligent Mobility, IOV 2015, December 19, 2015 - December 21, 2015*, volume 9502 of *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, pages 363–373. Springer Verlag.

Yang, M. and Zhang, X. (2016). A novel travel adviser based on improved back-propagation neural network. In *2016 7th International Conference on Intelligent Systems, Modelling and Simulation (ISMS)*, pages 283–288.

Yoon, J. W., Pinelli, F., and Calabrese, F. (2012). Cityride: a predictive bike sharing journey advisor. In *Mobile Data Management (MDM), 2012 IEEE 13th International Conference on*, pages 306–311. IEEE.

Zhang, J., Zheng, Y., and Qi, D. (2016). Deep spatio-temporal residual networks for citywide crowd flows prediction. *CoRR*, abs/1610.00081.

Zhang, J., Zheng, Y., and Qi, D. (2017). Deep spatio-temporal residual networks for citywide crowd flows prediction. In *AAAI*, pages 1655–1661.

Appendices

A Structured Literature Review Protocol

A.1 Framework for search in academic search engines

Multiple academic search engines were selected to ensure that all studies relevant to the domain and problem were retrieved. These search engines are presented in Table A1. To keep the resulting document size manageable, some additional search constrains were introduced. Since the field of machine learning is rapidly evolving, all studies published before the year 2000 were excluded. In addition, where applicable, only studies within the disciplines of either *Computer Science*, *Artificial Intelligence* or *Engineering* were included.

Academic search engine	**URL**
Scopus	https://www.scopus.com/
ACM Digital Library	https://dl.acm.org/
Engineering Village	https://www.engineeringvillage.com/
IEEE Xplore Digital Library	https://ieeexplore.ieee.org/
ProQuest	https://www.proquest.com//
Web of Science	https://www.webofknowledge.com/
ScienceDirect	https://www.sciencedirect.com/

Table A1: Academic search engines used

The searches were performed by grouping key terms in a boolean search string. Each group contains terms that are either synonyms or semantically related within the domain, and each term is chosen based on relatedness to the research questions. Only studies that were at the intersection of the sets (see Figure A1) were retrieved, i.e. studies that contained at least one term from every group. Table A2 presents the groups of terms.

Group 1 focuses on the domain of the problem; the term *transportation* was

included in this group to capture forecasting solutions to other closely related modes of transportation, such as car-sharing services.

Group 2 targets the variable that is predicted; studies that perform prediction on these variables likely apply similar forecasting techniques on similar data sets, and are therefore valuable to this study. Furthermore, it is reasonable to assume that studies that address other issues related to BSSs, such as optimal vehicle rebalancing (Labadi et al. [2015]), include methods or references relevant to this study.

Group 3 focuses on the problem the study aims to solve.

Group 4 addresses the techniques used to solve the problem; the primary subject of this study is machine learning, but because statistical methods are extensively used within the research community for forecasting problems, the terms *statistical analysis* and *big data* are included.

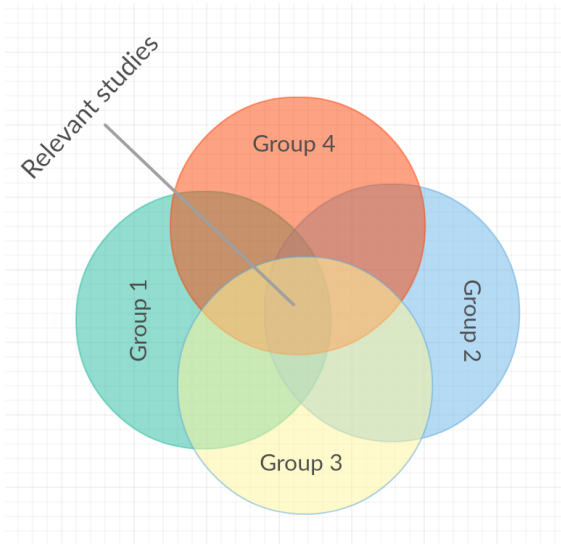


Figure A1: Venn diagram representing relevant studies

The terms were combined using boolean operators, specifically OR-operators within the groups and AND-operators between the groups. Subsequently, the resulting boolean query string is:

$$([G1,T1] \vee [G1,T2] \vee [G1,T3]) \wedge ([G2,T1] \vee [G2,T2] \vee [G2,T3] \vee [G2,T4] \vee [G2,T5] \vee [G2,T6] \vee [G2,T7])([G3,T1] \vee [G3,T2] \vee [G3,T3])([G4,T1] \vee [G4,T2] \vee [G4,T3] \vee [G4,T4] \vee [G1,T5])$$

	Group 1	Group 2	Group 3	Group 4
Term 1	transportation	demand	predict*	machine learning
Term 2	bike	congestion	forecast*	deep learning
Term 3	bicycle	balance	estimat*	neural network
Term 4		flow		statistical analysis
Term 5		rental		big data
Term 6		trip		
Term 7		availab*		

Table A2: Groups of synonymously or semantically related terms used for boolean search string in academic search engines. Terms suffixed with an asterisk (*) utilizes the search engine's capability to perform *stemming*, i.e. *predict** searches for predict*ed*, predict*ing*, predict*ion* etc.

Running this query string on every academic search engine listed in Table A1 resulted in a total of 491 studies. Duplicates were removed either automatically by Microsoft EndNote, or manually by keeping the highest ranking source. In addition, studies of the wrong document type, e.g. entire books, were removed. The number of studies retrieved from each source after duplicate and document type filtration is presented in Table A3.

Academic search engine	Number of studies retrieved
Scopus	112
ACM Digital Library	7
Engineering Village	27
IEEE Xplore Digital Library	23
ProQuest	24
Web of Science	45
ScienceDirect	6
Total	**244**

Table A3: The number of studies retrieved from each academic search engine after duplicate and document type filtration.

A.2 Selection of primary studies

This step was carried out in order to further reduce the set of studies to a manageable size, and ensure that the final set of primary studies were topically relevant to this thesis. This was accomplished by systematically reviewing all remaining

studies by following a three-step filtration strategy:

1. Title inclusion criteria filtration
2. Abstract inclusion criteria filtration
3. Full-text inclusion criteria filtration

The inclusion criteria defined are presented in the list below and are conditions that must be met for each study in order to not be excluded. In the first step of the three-step filtration strategy, title inclusion criteria filtration, only the titles of every study were reviewed and required only *IC1* to be satisfied. In the next step, abstract inclusion criteria filtration, the abstracts were reviewed and needed to satisfy *IC1*, *IC2* and *IC3*. In the final step of the inclusion criteria filtration, the complete full texts were reviewed and had to satisfy all six inclusion criteria. Following this three-step strategy allowed to author of this thesis to spend less time evaluating each study.

IC1 The study is performed with a Computer Science perspective.

IC2 The study's main focus is forecasting or prediction of some future value, preferably related to bike-sharing systems, but transferrable domains are also accepted.

IC3 The study is a primary study presenting empirical results.

IC4 The study clearly describes the models used in a reproducible manner.

IC5 The study uses machine learning methods.

IC6 The study performs prediction on aggregated data sources, such as bike-sharing systems data combined with weather data.

The resulting number of studies from each source after each step is presented in Table A4. Additionally, because some relevant studies may have failed to be retrieved during the initial searches, all references of the remaining *15* studies after full-text filtration were processed in the same three-step manner. Ultimately, *27* studies remained after inclusion criteria filtration.

A.3 Quality assessment

All remaining studies were assessed with respect to their quality. Nine quality criteria, presented below, were defined to evaluate the strength of the evidence presented in each study. Each quality criterion received received a score of either 1 if it was fully satisfied, 0.5 if it was partially satisfied, or 0 if it was not satisfied. Studies that received a total score of 6.5 or lower were regarded as low-quality studies and dismissed.

	Number of studies remaining after IC filtration step			
Source	**Title**	**Abstract**	**Full-text**	**Final**
Scopus	49	15	4	13
ACM Digital Library	2	0	0	0
Engineering Village	12	7	6	8
IEEE Xplore	14	3	3	3
ProQuest	5	1	1	1
Web of Science	19	5	1	1
ScienceDirect	2	1	0	0
Total	**103**	**32**	**15**	**27**

Table A4: The resulting number of primary studies from each academic search engine after each step of the three-stage filtration process. The *Final* column presents the number of studies remaining after all references of all studies in the *Full-text* column went through the same three-step filtration process.

QC1 There is a clear statement of the aim of the research.

QC2 The study is put into context of other studies and research.

QC3 It is clearly stated which other methods/algorithms the study's algorithm(s) has been compared with.

QC4 System- or algorithmic design decisions are justified.

QC5 The test data set is reproducible.

QC6 The experimental procedure is thoroughly explained and reproducible.

QC7 The performance metrics used in the study are explained and justified.

QC8 The test results are thoroughly analyzed.

QC9 The test evidence supports the findings presented.

The first quality criterion, QC1, ensures that there is a clear purpose for the study. This is important, because some initially retrieved papers were summaries of other studies, not independent research. The second and third quality critera, QC2 and QC3, ensures that the author(s) have compared their research to other similar studies, and that it is state-of-the-art. To be able to justify all reasoning and design choices in this thesis, it is essential that research of which this

thesis builds upon also justify their choices. QC4 ensures that design choices are justified. Quality criterion 4 and 5 makes certain whether or not the study is reproducible. Studies that are not reproducible cannot be empirically proven wrong, which limits the quality of the study. The final three quality critera, QC7, QC8 AND QC9, ensures that the results presented, the metrics for measuring success, and the conclusions drawn are thoroughly justified.

Three studies did not pass the quality assessment, and conclusively, 24 studies remained after the complete filtration process. These studies are presented in Table 3.1. The complete matrix of quality assessment scores for each study is shown in figure A2.

Paper	QC1	QC2	QC3	QC4	QC5	QC6	QC7	QC8	QC9	Sum
[1] Ashqar, H. I., et al. (2017)	1	1	1	1	1	1	1	1	1	**9**
[2] Bacciu, D., et al. (2017)	1	1	1	0.5	1	1	1	1	0.5	**8**
[3] Bei, C., et al. (2013)	1	0.5	1	1	1	1	1	1	1	**8.5**
[4] Borgnat, P., et al. (2011)	1	0	0.5	1	0	1	1	0.5	1	**6**
[5] Caggiani, L., et al. (2017)	1	0.5	0	1	1	1	1	1	1	**7.5**
[6] Dali, L. and D. Mladenic (2012)	1	0.5	1	1	1	1	1	0.5	1	**8**
[7] Fei, L., et al. (2017)	1	1	1	1	1	1	1	1	1	**9**
[8] Froehlich, J., et al. (2009)	1	0	1	1	1	1	1	1	1	**8**
[9] Gallop, C., et al. (2011)	1	1	1	1	0	1	0.5	1	1	**7.5**
[10] Ghanem, A., et al. (2017)	1	1	1	1	1	1	0.5	1	1	**8.5**
[11] Giot, R. and R. Cherrier (2014)	1	1	1	0.5	1	1	1	0.5	0.5	**7.5**
[12] Han, Y., et al. (2014)	1	0.5	1	1	1	1	0.5	1	1	**8**
[13] Kaltenbrunner, A., et al. (2010)	1	0.5	0.5	1	1	0.5	1	1	1	**7.5**
[14] Li, Y., et al. (2015)	1	0.5	1	1	0	1	1	1	1	**7.5**
[15] Lin, F., et al. (2016)	0	0.5	0.5	1	0	1	1	0.5	1	**5.5**
[16] Liu, J., et al. (2015)	1	0	1	1	1	1	0.5	0.5	1	**7**
[17] Malani, J., et al. (2015)	1	1	1	1	1	1	0.5	0	1	**7.5**
[18] Milenković, M., et al. (2014)	1	1	0	1	1	1	1	1	1	**8**
[19] Moreira-Matias, L., et al. (2013)	1	1	1	1	0	1	1	1	1	**8**
[20] Rudloff, C. and B. Lackner (2014)	1	1	1	1	1	1	1	0.5	1	**8.5**
[21] Salaken, S. M., et al. (2015)	1	1	0	1	1	1	1	1	1	**8**
[22] Wang, D., et al. (2017)	1	1	1	1	1	1	1	1	1	**9**
[23] Xu, J. X. and J. S. Lim (2007)	1	0	1	1	0	1	0.5	0.5	1	**6**
[24] Yang, M., et al. (2015)	1	1	1	1	1	1	0	1	1	**8**
[25] Yang, M. and X. Zhang (2016)	1	1	1	1	1	1	0.5	1	1	**8.5**
[26] Yoon, J. W., et al. (2012)	1	1	1	1	1	1	1	0.5	1	**8.5**
[27] Zhang et al (2017)	1	1	1	1	1	1	1	1	1	**9**

Figure A2: [4] Borgnat, P., et al. (2011), [15] Lin, F., et al. (2016), and [23] Xu, J. X. and J. S. Lim (2007) received scores lower than 6.5, and were therefore dismissed from the final set of papers. Total number of papers in final set: 24.

This page is intentionally left blank.

B Datasets

Dataset: Oslo Bysykkel Trips **Time span:** 20.04.2016 - 22.11.2017 **Number of records:** 4 993 801	
Column	**Description**
id	A unique identification number for a trip.
start_station_id	The unique identification number for the station a trip originated from.
end_station_id	The unique identification number for the destination station of a trip.
started_at	A ISO-8601 datetime in UTC format specifying the exact start time of a trip.
ended_at	A ISO-8601 datetime in UTC format specifying the exact end time of a trip.
end_method	How a trip ended. Can be one of *null, cancelled, lost_from_station, forced* or *delivered*.

Table B1: Description of the Oslo Bysykkel Trips dataset.

Dataset: Oslo Bysykkel Stations **Number of records:** 528	
Column	**Description**
id	A unique identification number for a station.
title	The name of the geographic location where a station is placed (used in the Visualization Module).
latitude	The latitudinal coordinate of a station with a precision of 6 decimals.
longitude	The longitudinal coordinate of a station with a precision of 6 decimals.
deleted	Boolean specifying whether the station is deleted or not.

Table B2: Description of the Oslo Bysykkel Stations dataset.

Dataset: Oslo Weather **Time span:** 01.01.2016 - 31.12.2017 **Time interval:** 1 hour **Number of records:** 17 544	
Column	**Description**
rain	The amount of rain in millimeters that was measured at an hour.
timestamp	A ISO-8601 datetime in UTC format.
temp	The average temperature in Celsius that was measured at an hour.
wind_speed	The average wind speed in km/h that was measured at an hour.
sunny	Boolean, set to true if the amount of sun minutes at an hour exceeded 30 minutes.

Table B3: Description of the Oslo Weather dataset.

Dataset: Stations by Grid Cells **Type:** JSON **Number of records:** 216	
Column	**Description**
cell	A cell number of the 12x18 grid used to cluster stations.
bounds	The geographical bounds that define a cell expressed as the latitudinal and longitudinal coordinates of the lower left and upper right corner of the cell.
stations	An array of station IDs. A station is added to this list if that station's coordinates lie within the bounds of the respective cell.

Table B4: Description of the Stations by Grid Cell JSON-file.

Dataset: Stations by Zones **Type:** JSON **Number of records:** 14	
Column	**Description**
zone	A unique zone number.
name	The name of the city district a zone encompasses, such as "Oslo City Centre West".
bounds	A list of latitudinal and longitudinal coordinates that defines the geographical polygon shape of a zone .
stations	An array of station IDs. A station is added to this list if that station's coordinates lie within the bounds of the respective zone.

Table B5: Description of the Stations by Zones JSON-file.

Dataset: Feature-engineered flow-by-zone and flow-by-grid **Type:** CSV **Total number of features:** 26 **Total number of records:** 125 356 (zone) / 725 274 (grid) **Training set examples:** 94 017 / 543 955 **Validation set examples:** 12 535 / 72 527 **Test set examples:** 18 804 / 108 792		
ID	**Feature**	**Description**
F1	zone/cell	A unique zone or cell number.
F2	outflow	The total number of outgoing trips during an hour that originated in this zone or cell and ended in another zone or cell.
F3	inflow	The total number of incoming trips during an hour that ended in this zone or cell and originated in another zone or cell.
F4	internal_flow	The total number of trips that originated in this zone or cell and ended in this zone or cell.
Temporal features		
F5	year	The year extracted from the timestamp.
F6	month	The month extracted from the timestamp.
F7	weeknum	The week number extracted from the timestamp.
F8	season	The season (winter, spring, summer or autumn) extracted from the timestamp.
F9	weekday	The day of the week extracted from the timestamp.

F10	weekend	Boolean indicating if this day is a Saturday or Sunday.
F11	holiday	Boolean indicating if this day is a Norwegian holiday.
F12	hour	The hour extracted from the timestamp.
Meteorological features		
F13	rain	The amount of rain in millimeters that was measured at an hour.
F14	rain_last_3	Boolean indicating if there was measured more than 0.8mm rain in the previous three hours combined.
F15	temp	The average temperature in Celsius that was measured at an hour.
F16	wind_speed	The average wind speed in km/h that was measured at an hour.
F17	sunny	Boolean, set to true if the amount of sun minutes at an hour exceeded 30 minutes.
Moving average features		
F18	outflow_c	Outflow closeness; the average outflow of the previous three hours.
F19	outflow_p	Outflow period; the average outflow of the previous four days at the same hour.
F20	outflow_t	Outflow trend; the average outflow of the previous four weeks at the same day at the same hour.
F21	inflow_c	Inflow closeness; the average inflow of the previous three hours.
F22	inflow_p	Inflow period; the average inflow of the previous four days at the same hour.
F23	inflow_t	Inflow trend; the average inflow of the previous four weeks at the same day at the same hour.
F24	internal_flow_c	Internal flow closeness; the average internal flow of the previous three hours.
F25	internal_flow_p	Internal flow period; the average internal flow of the previous four days at the same hour.
F26	internal_flow_t	Internal flow trend; the average internal flow of the previous four weeks at the same day at the same hour.

Table B6: Descriptions of every feature in the final feature engineered datasets.

C Optimal hyperparameters for RNN models

Optimal hyperparameters for zone-based RNN models
Format: *low-feature models (NONE, M) / high-feature models (C, CP, CPT, CPTM)*

Parameter	$\mathbf{RNN_{low\text{-}traffic}}$	$\mathbf{RNN_{medium\text{-}traffic}}$	$\mathbf{RNN_{high\text{-}traffic}}$
rnn_cell	gru / lstm	gru / gru	gru / gru
rnn_units	168 / 133	232 / 249	242 / 245
learning_rate	0.0063 / 0.0061	0.005 / 0.0058	0.007 / 0.0059
seq_length	54 / 126	126 / 126	126 / 126
activation_fn	tanh	tanh	tanh
optimizer[1]	Θ_R	Θ_R	Θ_R
num_layers	1	1	1
batch_size	256 / 256	512 / 512	256 / 512

[1] Θ_R = RMSprop, Θ_{adam} = Adam, Θ_{ag} = AdaGrad

Table C1: Optimal hyperparameters for every RNN model when using zone-based clustering. Low-feature models (NONE, M) on the left side of the slash (/), high-feature models (C, CP, CPT, CPTM) on the right. Note that only the hyperparameters for medium-traffic RNN-models is presented; hyperparameters for low-traffic and high-traffic RNN-models can be found in Appendix C.

Optimal hyperparameters for grid-based RNN models
Format: *low-feature models (NONE, M) / high-feature models (C, CP, CPT, CPTM)*

Parameter	$\mathbf{RNN_{low\text{-}traffic}}$	$\mathbf{RNN_{medium\text{-}traffic}}$	$\mathbf{RNN_{high\text{-}traffic}}$
rnn_cell	lstm / gru	lstm / gru	gru / gru
rnn_units	145 / 187	154 / 251	201 / 243
learning_rate	0.007 / 0.00048	0.00046 / 0.00054	0.00049 / 0.0005
seq_length	54 / 126	126 / 126	126 / 126
activation_fn	tanh	tanh	tanh
optimizer[1]	Θ_R	Θ_R	Θ_R
num_layers	1	1	1
batch_size	512 / 256	256 / 512	256 / 512

[1] Θ_R = RMSprop, Θ_{adam} = Adam, Θ_{ag} = AdaGrad

Table C2: Optimal hyperparameters for every RNN model when using zone-based clustering. Low-feature models (NONE, M) on the left side of the slash (/), high-feature models (C, CP, CPT, CPTM) on the right. Note that only the hyperparameters for medium-traffic RNN-models is presented; hyperparameters for low-traffic and high-traffic RNN-models can be found in Appendix C.