DEGREE PROJECT IN COMPUTER SCIENCE AND ENGINEERING,
SECOND CYCLE, 30 CREDITS
STOCKHOLM, SWEDEN 2020

A time series forecasting approach for queue wait-time prediction

ANTON STAGGE

KTH ROYAL INSTITUTE OF TECHNOLOGY
SCHOOL OF ELECTRICAL ENGINEERING AND COMPUTER SCIENCE

A time series forecasting approach for queue wait-time prediction

ANTON STAGGE

Master in Computer Science
Date: June 10, 2020
Supervisor: Mats Nordahl
Examiner: Pawel Herman
School of Electrical Engineering and Computer Science, KTH
Swedish title: Förutsägelser av väntetid med hjälp av tidsserieprognoser

Abstract

Waiting in queues is an unavoidable part of life, and not knowing how long the wait is going to be can be a big source of anxiety. In an attempt to mitigate this, and to be able to manage their queues, companies often try to estimate wait-times. This is especially important in healthcare, since the patients are most likely already under some distress.

In this thesis the performance of two different machine learning (ML) approaches and a simulation approach were compared on the wait-time prediction problem in a digital healthcare setting. Additionally, a combination approach was implemented, combining the best ML model with the simulation approach.

The ML approaches used historical data of the patient queue in order to produce a model which could predict the wait-time for new patients joining the queue. The simulation algorithm mimics the queue in a virtual environment and simulates time moving forward until the new patient joining the queue is assigned a clinician, thus producing a wait-time estimation. The combination approach used the wait-time estimations produced by the simulation algorithm as an additional input feature for the best ML model.

A Temporal Convolutional Network (TCN) model and a Long Short-Term Memory network (LSTM) model were implemented and represented the *sequence modeling* ML approach. A Random Forest Regressor (RF) model and a Support Vector Regressor (SVR) model were implemented and represented the traditional ML approach. In order to introduce the temporal dimension to the traditional ML approach, the *exponential smoothing* preprocessing technique was applied.

The results indicated that there was a statistically significant difference between all models. The TCN model and the simulation algorithm had the lowest Mean Square Error (MSE) of all individual models. Both sequence modeling models had lower MSE compared to both of the traditional ML models. The combination model had the lowest MSE of all, adopting the best performance traits from both the ML approach and the simulation approach. However, the combination model is the most complex, and thus requires the most maintenance.

Due to the limitations in the study, no single approach can be concluded as optimal. However, the results suggest that the sequence modeling approach is a viable option in wait-time prediction, and is recommended for future research or applications.

Sammanfattning

Att vänta i köer är en oundviklig del av livet. Att inte veta hur lång väntan kommer att bli kan framkalla ångest. I ett försök att lindra denna ångestkälla, samt för att kunna hantera sina köer, försöker företag ofta uppskatta väntetiden. Detta är särskilt viktigt inom hälso- och sjukvården, eftersom patienterna troligtvis redan upplever någon typ av oro.

Syftet med denna uppsats är att jämföra prestandan hos tre olika metoder för att förutspå väntetiden hos en digital vårdstjänst. Två olika maskininlärningsmetoder (ML) samt en simuleringsmetod jämfördes. Utöver detta jämfördes även en kombinationsmetod, som kombinerade den bästa ML-modellen med simuleringsmetoden.

ML-metoderna använde sig av historisk data från patientkön för att skapa en modell som kunde förutsäga väntetiden för nya patienter som ställer sig i kön. Simuleringsalgoritmen imiterar kön i en virtuell miljö och simulerar att tiden går framåt i denna miljö tills den nya patienten som anslöt sig till kön kan tilldelas en ledig kliniker. På detta sätt kan en prediktion av väntetiden ges till patienten. Kombinationsmetoden använde simuleringsprediktionerna som ytterligare indata till den bästa ML-modellen.

En Temporal Convolutional Network (TCN)-modell samt en Long Short-Term Memory (LSTM)-modell implementerades och representerade sekvensmodelleringsmetoden (eng: *sequence modeling*). En Random Forest Regressor (RF)-modell samt en Support Vector Regressor (SVR)-modell implementerades och representerade den traditionella ML-metoden. För att den traditionella ML-metoden skulle få tillgång till tidsdimensionen applicerades förbehandlingstekniken exponentiell utjämning på dess data.

Resultatet visade att det fanns en statistiskt signifikant skillnad i kvadratfelet mellan alla modellerna. TCN-modellen samt simulationsalgoritmen hade lägst medelkvadratfel av de ensamstående modellerna. Båda sekvensmodelleringsmodellerna hade lägre medelkvadratfel än de traditionella ML-modellerna. Kombinationsmodellen hade absolut lägst medelkvadratfel, då modellen behöll fördelarna från både ML- samt simuleringsmetoden. Däremot är kombinationsmetoden den metod som kräver mest underhåll.

På grund av begränsningarna i studien kan ingen enstaka metod hävdas vara optimal. Resultaten tyder emellertid på att sekvensmodelleringsmetoden kan användas för väntetidsprediktion i ett kösystem, och rekommenderas därför för framtida forskning eller applikationer.

Contents

Terminology

The following terms and abbreviations are used in this report:
ML - Machine Learning
TCN - Temporal Convolutional Network
LSTM - Long Short-Term Memory
RF - Random Forest
SVM - Support Vector Machine
SVR - Support Vector Regressor
Feature / predictor / parameter - Input variable
QL - Queue length, i.e. the number of consumers actively waiting in a queue
ED - Emergency Department
DLAMI - Deep Learning Amazon Machine Instance
Consultation / meeting - Used interchangeably to describe a consultation meeting between a patient and a clinician.

Chapter 1

Introduction

A large number of industries that provide products or services require consumers to wait in queues. Several studies have indicated that the waiting process has a strong impact on the perceived quality of the service [1], [2]. In certain queues it is relatively easy for the consumer to estimate how long the wait is going to be, for example cash register queues, due to the fact that the progress of the queue is visible. In other queues, however, the wait-time can not be easily estimated by the consumer, e.g. online or healthcare queues. You might know your position in the queue, but not how long each person in front of you is going to need for their errand. An unknown wait-time is a bigger source of anxiety compared to a known, finite, wait-time [1]. For this reason, and to be able to manage the queues, companies (service/product providers) often try to estimate the wait-time.

The accuracy of wait-time predictions is very important. If the predictions are too short the consumers tend to become frustrated as they were essentially lied to. If the predictions are too long the consumers might consider other alternatives or rival companies instead [3], [4]. Wait-time prediction is especially important in healthcare, since the patients seeking help are most likely already under some distress [4], [5].

Digital healthcare has become increasingly popular during recent years. Digital healthcare providers offer services where patients can have video or text consultations with clinicians online [6]. One of the main advantages of digital healthcare compared to traditional healthcare is time efficiency. In traditional healthcare, one has to physically move to the treatment location and then wait for an unknown amount of time before getting to see a clinician. In digital healthcare, the inconvenience of having to move location does not exist. Furthermore, since faster access to more information is available, predictions

for the wait-time can be made more easily. Patients seeking help consequently do not have to waste their time waiting idly in a queue.

A substantial amount of research has been conducted with focus on wait-time prediction [5], [7]–[13]. Some of this work will be covered in detail in section 2.3. Several approaches to the problem have previously been investigated: Queuing Theory, Simulations and Machine Learning [6], [9], [10], [13]. In the Machine Learning (ML) approach, historical data of the state of the queue is used in order to obtain a model which can make predictions of the future state of the queue. The state-of-the-art ML model used for this purpose is the Random Forest (RF) [14] model but other models have been examined [7], [9], [13].

Another research area that has had some advances more recently is *Sequence Modeling*, which consists of mapping an input sequence to an output. This results in a new dimension added to the input, namely the temporal dimension. Sequence modeling is traditionally applied on problems involving texts, speech, DNA or other naturally occurring sequences [15]. The current state-of-the-art sequence modeling models are the Long Short-Term Memory (LSTM) [16] network and the (more recently presented) Temporal Convolutional Network (TCN) [15]. The historical data of a queue can be considered a sequence, since the current state of the queue is highly dependent on previous states of the queue. While simpler linear techniques have been applied [9], [17], no studies applying the more advanced techniques from the sequence modeling research area on the wait-time prediction problem have been identified.

The aim of this thesis is to compare the performance of three different approaches to the queue wait-time prediction problem in digital healthcare: a sequence modeling approach, the traditional ML approach and a simulation approach implemented by the company KRY [6]. In order to incorporate the temporal dimension to the traditional non-sequence modeling ML approach, the exponential smoothing [18] preprocessing technique is applied.

1.1 Problem Statement

The research question of this thesis is the following:

> *Does a sequence modeling approach, trained on data from the queue's history, perform better at predicting patient wait-times in digital healthcare compared to a non-sequence modeling ML*

approach utilizing exponential smoothing, and a simulation approach?

In order to answer the research question stated above the performance of two sequence modeling models (TCN and LSTM), two non-sequence modeling ML models utilizing exponential smoothing (RF and SVR), and the simulation algorithm implemented by the company KRY [6] will be compared. Additionally, the performance of a combination model, comprising of the best performing ML model and the simulation algorithm, will be examined.

1.2 Purpose

The purpose of this research is to investigate whether the more advanced techniques from sequence modeling research area can be applied to the queue wait-time prediction problem in healthcare, and if so, investigate how it performs compared to the traditional approaches. Recent studies have applied other ML approaches with success, but the ML models used do not take the temporal dimension into account. The TCN model has recently demonstrated its capability on a wide variety of tasks and datasets [15]; thus, there lies an interest in exploring its usability on the wait-time prediction problem. If the sequence modeling approach proves to be viable, it could be applied to other wait-time prediction problems, not just wait-time prediction in a digital healthcare setting, as long as there exists historical data of the state of the queue.

1.3 Scope

This paper compares four different ML models, where two are sequence modeling models and the other two are not. Other models are not considered. The scope does not include the application of queuing theory to the problem. Furthermore, the data used in this thesis comes from a single digital healthcare provider and from Swedish patients only. The consumer behaviour might differ in other markets.

1.4 Outline

This report is divided into six chapters. In Chapter 2 (Background) the research areas are introduced, the theory behind all models and techniques applied in this paper is explained, and finally related work is presented. In Chap-

ter 3 (Methodology) the experiments conducted in this thesis are presented in detail. In the following chapter, Chapter 4 (Results), the results of the experiments are reported. Next, in Chapter 5 (Discussion), the results are interpreted and a discussion is held regarding both the results and the limitations in the methodology used in finding said results. Finally, the report and its findings are concluded in Chapter 6 (Conclusion).

Chapter 2

Background

This chapter starts by introducing the research areas within which this thesis is carried out. Secondly, the theoretical background of all methods used is explained in detail. Lastly, related work and their similarities and differences are presented.

2.1 Research Areas

2.1.1 Sequence Modeling

A sequence is a collection of observations in which order matter. The research area known as *sequence modeling* consists of mapping an input sequence to an output sequence, where every output depends only on input observations before it. More formally, as described in [15], a sequence modeling network is a function that given an input sequence $x_0, ..., x_T$ predicts an output sequence $y_0, ..., y_T$

$$f(x_0, ..., x_T) = y_0, ..., y_T \tag{2.1}$$

under the constraint that every y_t only depends on $x_0, ..., x_t$ and *not* on any of the future input $x_{t+1}, ..., x_T$. This is known as the *causal* constraint. The goal of any sequence modeling network is to minimize the prediction error between the predicted values and the actual observations. This is commonly done with ML and Artificial Neural Networks [15], [19]. The current state-of-the-art network models are LSTMs [16] and TCNs [15]. The LSTM model has been a popular choice for recurrent networks for some time [19]. However, the TCN has proven to be superior to several network models (including the LSTM) in multiple cases more recently [15], [19]–[21].

Time Series Forecasting

A special type of sequence known as a *time series* is a sequence that consists of observations made at subsequent points in time for one or more attributes. A single attribute measured/observed is known as univariate time series, whereas if multiple attributes are considered, it is known as a multivariate time series. The research area of predicting future values in a time series is known as *time series forecasting* [22]. Time series forecasting is a rather hot research topic in which several papers have been released in the most recent years [23]–[26]. Time series forecasting techniques most often predicts the next value in the series, which makes it a special case of sequence modeling explained by the function:

$$f(x_0, ..., x_t) = y_t \tag{2.2}$$

where $x_0, ..., x_t$ are input vectors (in the case of multivariate time series forecasting) and y_t is the output variable. This is known as one-step forecasting.

2.1.2 Queuing Systems

The most classical type of queuing system is a single-queue single-server, which follows the First-in First-out (FIFO) discipline [27]. In this system consumers are waiting for a single limited resource, as shown in Figure 2.1. Since they can not access it all at the same time a queue is required [13]. The next logical extension of this queuing system is the single-queue multi-server system, in which a single line of consumers are waiting for multiple servers that provide the same resource, as shown in Figure 2.2. [27]

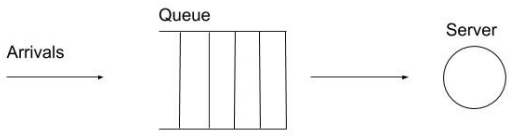


Figure 2.1: A single-queue single-server system.

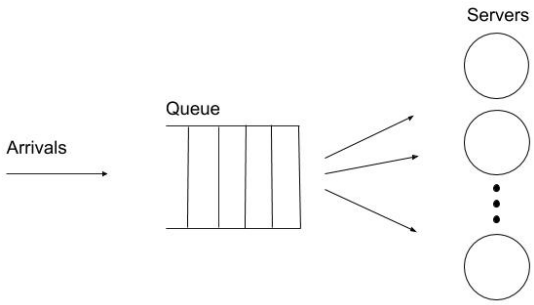


Figure 2.2: A single-queue multi-server system.

Queue Modeling

The research area known as *queuing theory* focuses on using mathematical models to explain a queuing system. This is the traditional way of modeling a queue [13]. The different mathematical formulas provided by queuing theory each have different constrains which have to be met [12], [27]. Alternative less restricted methods of modeling a queue include simulations or the use of ML.

Simulations can be used to model a queue. This is the algorithmic approach. A simulation is designed to mimic a queue as perfectly as possible, and then time is moved forward in a virtual environment. By using assumptions or calculated estimations for the amount of time each consumer is going to spend at the server, time can be moved forward until every consumer has visited a server. This queuing model, therefore, heavily relies on the accuracy of the estimated service time. A detailed description of the simulation algorithm investigated in this thesis is given in subsection 3.2.2.

When modeling a queue with ML, historical data of the state of the queue is used. A set of features is selected to represent the state, and a model is trained on the data of said features. The goal is for the model to portray the system as accurately as possible [5], [10], [13], [28]. This approach is a viable alternative to queuing theory [10], and all it requires is data regarding the history of the queue. When a model of the queue has been acquired it can be used to make predictions about future states of the queue, and certain variables in its state, for instance the wait-time.

2.2 Theory

2.2.1 Exponential Smoothing

The technique and use of exponential smoothing and exponential windows have existed for a long time. In the statistical literature it is often credited to Brown [29], and referred to as Brown's simple exponential smoothing. The method was originally introduced as a forecasting method, but have in the more recent years been proven to be a good preprocessing method [30]. The forecasting method is described as:

$$\hat{y}_{t+1} = \alpha x_t + (1 - \alpha)\hat{y}_t \tag{2.3}$$

where x_t is the raw observation at time t, $\hat{y}_t$ is the smoothed observation at time t, and α is a constant [31]. When used as a preprocessing method, the same formula is applied, but the forecasts for time $t + 1$ are used as inputs for time t.

2.2.2 Artificial Neural Networks

The first model of an artificial neuron was introduced as early as 1943 in [32], a mathematical representation of the neurons in a human brain. An artificial neuron is designed to take multiple inputs and multiply every input by a weight. The result is summed to produce a single value which is fed through an activation function f to produce an output [33]. This is illustrated by Figure 2.3. Ever since the introduction of artificial neurons, more sophisticated models,

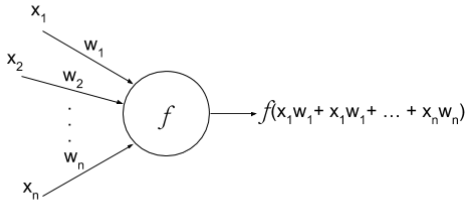


Figure 2.3: A generic artificial neuron.

that connect multiple neurons into a large network, known as Artificial Neural Networks (ANNs), have been invented. Rosenblatt introduced the first *perceptron* model in [34]. An example of a multi-layer perceptron (MLP) network,

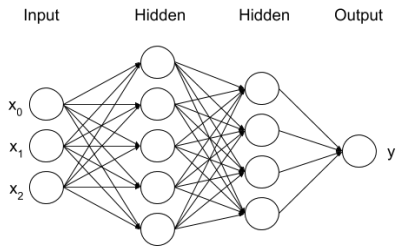


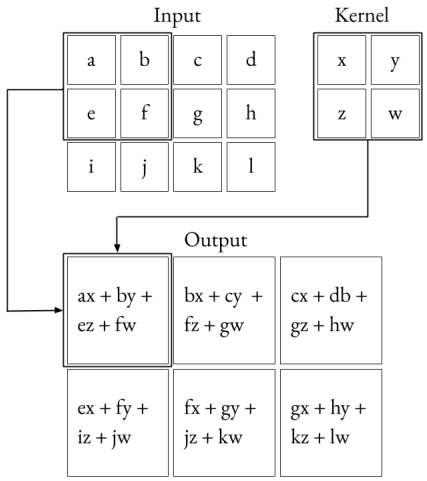
Figure 2.4: An example of a multi-layer perceptron network. The network has 2 hidden layers, with 5 and 4 neurons respectively, and a single output neuron.

connecting multiple artificial neurons together to form a network with 2 layers and a single output neuron is displayed in Figure 2.4.

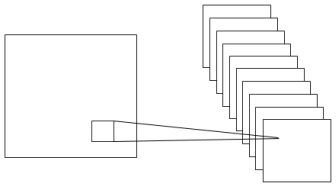
ANNs are supervised universal function approximators. These networks can learn to approximate any given function by being trained on labeled data. During training the weights in the network are slightly updated in order to reduce the error over all data. This is done with an algorithm known as the *backpropagation* [35]–[37] training algorithm. [33]

2.2.3 Convolutional Neural Networks

A Convolutional Neural Network (CNN) [38] is a regular ANN that uses the mathematical operation called convolution in at least one of its layers [39], making it a convolutional layer. Convolutional layers take the spatial relation of grid-like inputs into account. A spatial relation in, for example, an image is that closer pixels are more likely to be correlated. In a convolutional layer, window formations of weights (known as *filters* or *kernels*) stride over the input in order to produce an output known as a *feature map*. An example of a convolution with a single filter over a 2-dimensional input image can be seen in Figure 2.5a [39]. In each step the filter stops at, it produces one new value to the feature map by taking the dot product of its weights and the inputs the filter covers and passing the result through an activation function. The filters are thus made of neurons. [39]



(a) An example of a convolution on a 2-dimensional input image of size 3x4, with a filter of size 2x2, and the resulting feature map of size 2x3.



(b) 10 feature maps stacked together to form a single output with depth 10.

Figure 2.5: A detailed illustration of a convolutional layer.

A single convolutional layer can apply multiple filters to the same input. The feature maps produced by each filter are stacked on top of each other in order to produce a single output, as shown in Figure 2.5b. The number of filters, the size of the filters and the size of the stride are hyperparameters to the layer. [39]

2.2.4 Temporal Convolutional Networks (TCN)

The technique applied by a TCN has existed for a long time with many variations, but the name was established in 2018 by Bai, Kolter, and Koltun. The name was said to be a descriptive term for a family of architectures, and a generic TCN model was provided which is described below [15].

A TCN applies the concepts of a CNN on sequential input data by applying causal 1-dimensional convolutions on multiple subsequent inputs together instead of on a single one [20]. The filter size k determines how many inputs are considered in a single layer. The output is always zero padded to be the size of the input, i.e. zero padding of length $(k-1)$. Figure 2.6 illustrates this further.

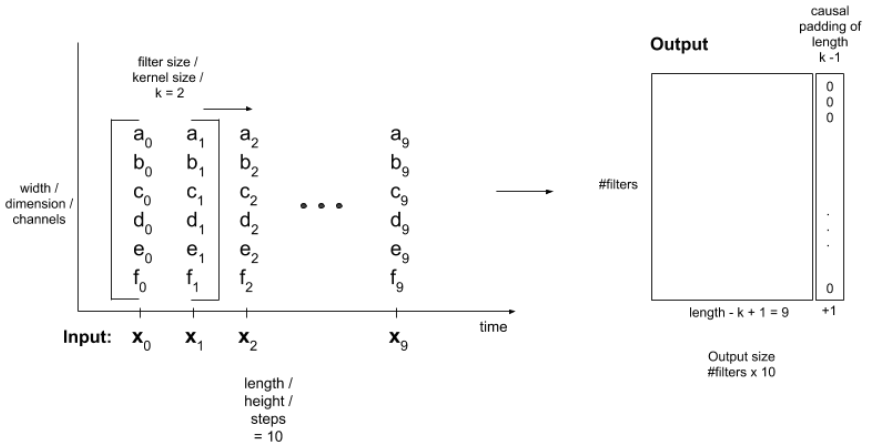


Figure 2.6: An illustration of a 1-dimensional convolution applied to a sequence of length 10 with 6 features and a kernel size $k = 2$.

The convolutional layers are stacked to provide depth to the network as shown in Figure 2.7.

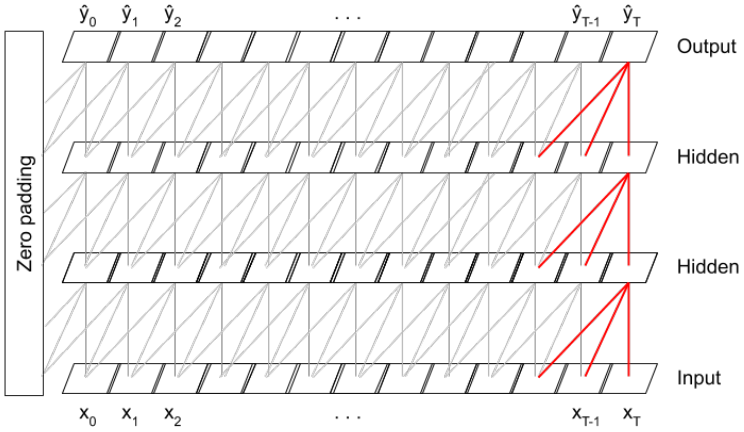


Figure 2.7: An example of a basic TCN architecture with 3 stacked layers each with $k = 3$.

In order to expand the number of inputs used to compute the output (known as the *receptive field*) beyond linear with the filter size and network depth, a technique knows as *dilated* convolutions [40] are made. A dilation is a fixed step d between two filter taps. The filter skips the inputs between the taps as shown in Figure 2.8 [15].

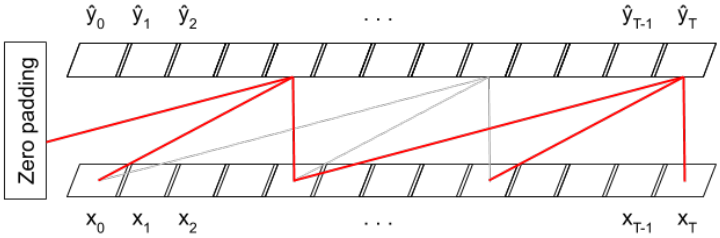


Figure 2.8: An example of a dilation layer with $d = 4$ and $k = 3$. Not all lines have been drawn to simplify the image. In reality, each output $\hat{y}_t$ is connected to k inputs x_t.

Stacking dilated convolutional layers and increasing the dilation d exponentially for each layer makes the receptive field exponentially large while still making sure that there is some filter tapping each input [15]. An example of

such a network is shown in Figure 2.9.

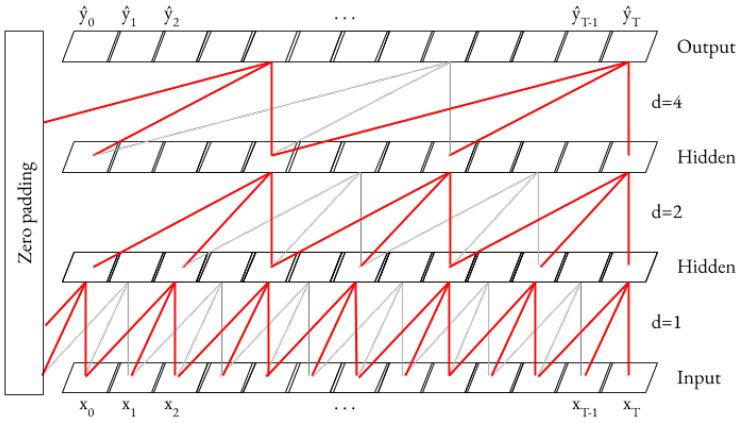


Figure 2.9: An example of 3 dilation layer stacked with $d = 1, 2, 4$, $k = 3$ yielding a receptive field of 29.

A temporal convolutional layer is made of a *residual block*, which additionally connects the input directly to the output in order to allow the layer to learn modifications to the mapping instead of the whole transformation [15]. There exists substantial evidence that the use of residual blocks make optimization easier and improves accuracy for deep architectures [15], [41]. As is shown in Figure 2.10, a single residual block consists of 2 repeated sub-blocks containing a dilated causal convolution followed by a *weight normalization* [42], a *ReLU* activation function [43] and a *Dropout* [44] layer. Moreover, a single 1-dimensional convolutional layer with a kernel size of 1 connects the input to the output with element-wise addition [15].

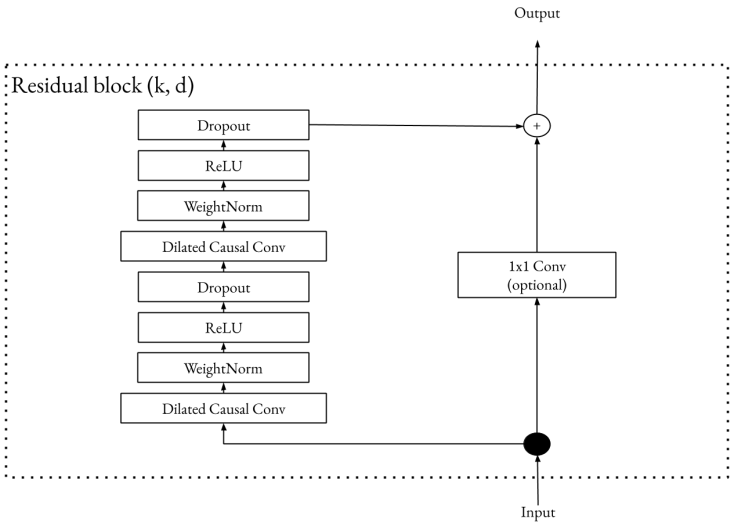


Figure 2.10: The architecture of a single residual block, as described by [15].

Because of the residual blocks, the calculation of the receptive field changes. Without residual blocks, the formula for the receptive field R would be:

$$R(n) = R(n-1) + (k-1) \times d(n) \tag{2.4}$$

where n is the layer, $d(n)$ is the dilation for layer n and $R(0) = 1$. For example, looking back at Figure 2.7 which has $k = 3$ and $d = 1$, the receptive field for the output layer is 15. However, since each residual block contains 2 causal convolutions, the formula becomes:

$$R'(n) = R'(n-1) + 2 \times (k-1) \times d(n) \tag{2.5}$$

If the dilation rate increases exponentially with a base 2, the formula can be simplified to:

$$\begin{aligned} R'(n) &= 1 + 2 \times (k-1) \times (2^0 + 2^1 + 2^2 + ... + 2^{n-1}) \\ &= 1 + 2 \times (k-1) \times (2^n - 1) \end{aligned} \tag{2.6}$$

An illustration of how the residual blocks affect the receptive field is shown in Figure 2.11. As one can see the number of inputs considered for a single output is 13. Using Equation 2.6:

$$R(2) = 1 + 2 \times (3-1) \times (2^2 - 1) = 1 + 2 \times 2 \times 3 = 13 \tag{2.7}$$

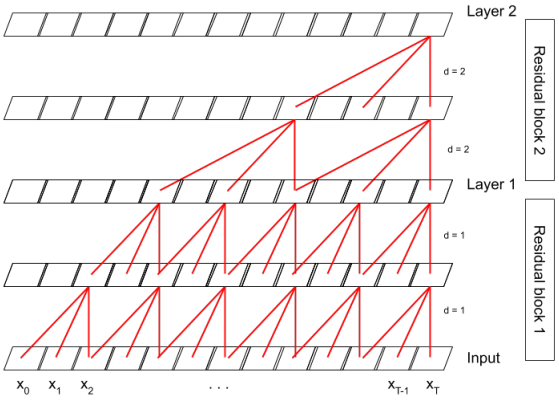


Figure 2.11: An example of the receptive field of two stacked residual blocks with $k = 3$ and dilation increasing exponentially with a base of 2.

2.2.5 Recurrent Neural Networks

A *Recurrent Neural Network* (RNN) is another sequence modeling extension of the ANN in which information from previous inputs x are incorporated in a hidden state h which is passed forward in time. This allows the network to retain a memory of previously observed inputs for each produced output [39]. Figure 2.12 displays the computational graph for a basic RNN with a single input unit.

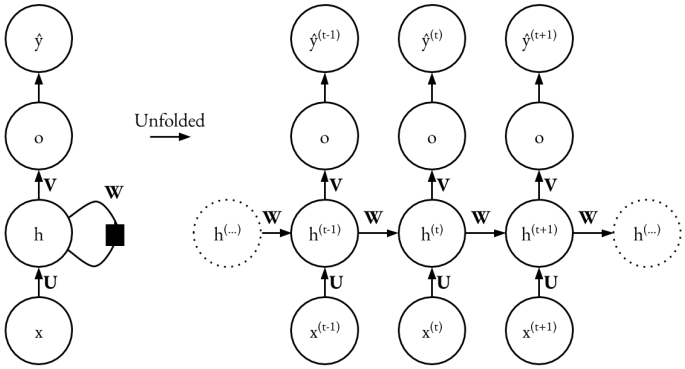


Figure 2.12: The computational graph of a basic RNN. **U** **W** and **V** are weight matrices, x is the input, h is the hidden state which is passed forward in time, o is the output, and $\hat{y}$ is the output after the activation function. The black box represents a delay of one time-step.

The process of producing outputs $\hat{y}^{(t)}$ (known as the *forward propagation*),

begins with an initial state $h^{(0)}$ and then for each time-step t in the sequence the following equations are applied:

$$\begin{aligned} a^{(t)} &= b + \mathbf{W}h^{(t-1)} + \mathbf{U}x^{(t)} \\ h^{(t)} &= \text{Activation}(a^{(t)}) \\ o^{(t)} &= c + \mathbf{V}h^{(t)} \\ \hat{y}^{(t)} &= \text{Activation}(o^{(t)}) \end{aligned} \tag{2.8}$$

where b, c are bias vectors [39].

2.2.6 Long Short-Term Memory (LSTM)

An LSTM [16] is an extension of the RNN that has an additional *memory cell*. The long-term memory comes from the weights, and the short-term memory comes from said memory cells. Another important feature added to the LSTM is its *gates*. A gate is a sigmoidal neuron that takes the current input $x^{(t)}$ and the hidden state $h^{(t-1)}$ as input [45]. It is called a gate because its result is multiplied with the result from another node. Therefore, if its value is close to zero the gate is closed, and vice versa. The LSTM unit has an *input gate* connected to the input node, a *forget gate* connected to the memory node, and an *output gate* connected to the output node [45]. Figure 2.13 displays the computational graph of a basic LSTM unit.

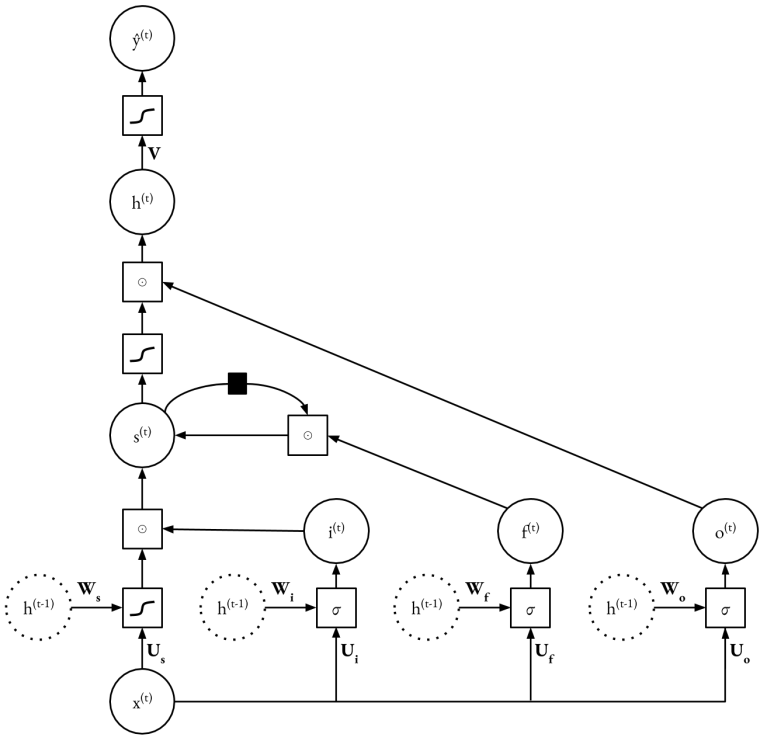


Figure 2.13: A computational graph of a basic LSTM unit. **U** **W** and **V** are weight matrices, x is the input, h is the hidden state which is passed forward in time, s is the memory node also passed forward in time, and $\hat{y}$ is the output after the activation function. i, f and o are the input, forget and output-gates. The black box represents a delay of one time-step, the flattened *S* represents an arbitrary activation function, and σ is the sigmoid activation function.

Similar to the RNN, the process of producing outputs $\hat{y}^{(t)}$, i.e. forward propagation, for an LSTM unit starts with a initial hidden state $h^{(0)}$, and then the equations in Equation 2.9 are applied for each time-step t of the sequence. Here $i^{(t)}$ denotes the input gate, $f^{(t)}$ the forget gate, $o^{(t)}$ the output gate, and $s^{(t)}$ the memory node.

$$
\begin{aligned}
i^{(t)} &= \sigma(\mathbf{W}_i h^{(t-1)} + \mathbf{U}_i x^{(t)}) \\
f^{(t)} &= \sigma(\mathbf{W}_f h^{(t-1)} + \mathbf{U}_f x^{(t)}) \\
o^{(t)} &= \sigma(\mathbf{W}_o h^{(t-1)} + \mathbf{U}_o x^{(t)}) \\
s^{(t)} &= f^{(t)} \odot s^{(t-1)} + i^{(t)} \odot \text{Activation}(\mathbf{W}_s h^{(t-1)} + \mathbf{U}_s x^{(t)}) \\
h^{(t)} &= o^{(t)} \odot \text{Activation}(s^{(t)}) \\
\hat{y}^{(t)} &= \text{Activation}(\mathbf{V} h^{(t)})
\end{aligned}
\tag{2.9}
$$

where $\sigma(\cdot)$ is the sigmoid activation function and $\odot$ is element wise multiplication [39]. All bias vectors have been excluded from the equations for simplicity. The size of all matrices $\mathbf{W}$ and $\mathbf{U}$ is a hyperparameter to the LSTM. Much like the TCN, the modern LSTM utilizes dropout [44], both between layers, and between the recurrent units.

2.2.7 Regression Tree

A Regression Tree [46] is a series of questions about the features in the dataset built up in a tree-like structure. Each node in the tree contains a question or test on a specific feature, thus, each branch represents an answer or outcome of the test. The leafs in the tree hold the prediction values [47]. Figure 2.14 displays a simple constructed example of a regression tree modeling the wait-time at a hospital. A regression tree divides the prediction space into non-overlapping regions, each region being a leaf node in the tree. This is illustrated further by Figure 2.15.

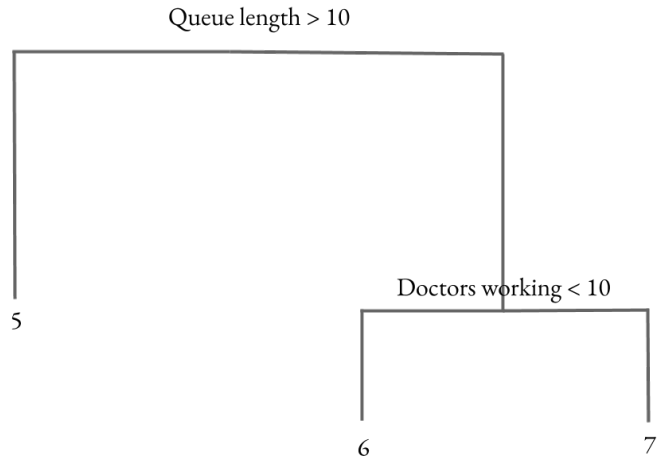


Figure 2.14: An example of a Regression Tree modeling the wait-time in a hospital queue. The leaf nodes show the predicted wait-time in minutes. If the outcome of the test is true, the right branch is selected.

When constructing a tree, an approach called *recursive binary splitting* is commonly used. This is a greedy top-down approach where you start with one region containing all data points. The region is then, successively, split into two smaller regions, where each split is optimal considering only this step, optimal meaning that all predictors and cutpoints are considered. The chosen

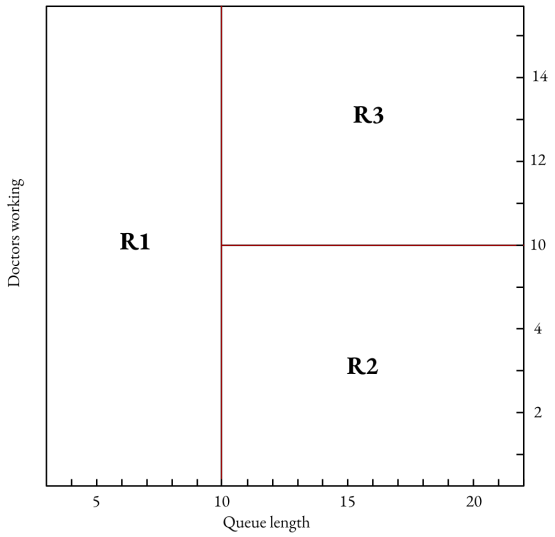


Figure 2.15: The prediction space divided into regions based on the Regression Tree in Figure 2.14.

predictor and cutpoint is the one that minimizes the Residual Sum of Squares (RSS) given by Equation 2.10, where J is the number of regions (R), and $\hat{y}_{R_j}$ is the mean prediction value of the points within region R_j. [47]

$$RSS = \sum_{j=1}^{J} \sum_{i \in R_j} (y_i - \hat{y}_{R_j})^2 \tag{2.10}$$

2.2.8 Random Forest Regressor (RF)

As the Regression Tree model described in subsection 2.2.7 has very high variance, meaning that two models trained on two partitions of the same data tend to differ, the ensemble learning technique known as Random Forest [14] greatly improves its accuracy. A RF model is, as suggested by the name, a collection of trees. The final prediction is simply the average of all the trees. Before constructing a single tree in the forest, a unique training set is first created by repeatedly sampling from the original training set, a process known as *bootstrapping*. The tree is then built by considering only a fixed number of predictors m during each split. Those predictors are chosen at random for each split, and usually $m = \sqrt{p}$ where p is the total number of predictors (size of input dimension). Because of the random selection of predictors and

bootstrapped training data, each tree in the forest become less correlated to the others, and the variance of the model is reduced. The number of trees, the maximum depth of each tree and the number of predictors m are hyperparameters to the model. [47]

2.2.9 Support Vector Regression (SVR)

Support Vector Classifier

A Support Vector Classifier (SVC) is a hyperplane that separates 2 classes of points with the widest possible margin M to the plane. The formula for a hyperplane is given by $\vec{w} \cdot \vec{x} - b = 0$, where $\vec{w}$ is a weight vector, b is a constant and $\cdot$ is the scalar product. All points on one side on the plane will have $\vec{w} \cdot \vec{x} - b > 0$, and all points on the other side will have $\vec{w} \cdot \vec{x} - b < 0$. Thus $\forall i$, the point x_i and its corresponding class $y_i \in \{-1, 1\}$ should satisfy Equation 2.11 [47].

$$y_i \times (\vec{w} \cdot \vec{x_i} - b) \geq M(1 - \epsilon_i) \tag{2.11}$$

Every point should be on the correct side of the plane, with margin, but allowing for some (ϵ_i) slack. The slack relaxes the problem so that a solution can be found for more cases. The size of the margin M can be calculated to $\frac{2}{||\vec{w}||}$. Maximizing the margin is therefore equivalent to minimizing the length of $\vec{w}$. The optimization problem for the SVC is

$$\begin{gathered} \text{Minimize } ||\vec{w}||^2 + C \sum_i \epsilon_i \\ \text{Under the constraints } y_i \times (\vec{w} \cdot \vec{x_i} - b) \geq 1 - \epsilon_i, \forall i \\ \epsilon_i \geq 0, \forall i \end{gathered} \tag{2.12}$$

where C is a hyperparameter that tunes the amount of slack allowed [48]. Figure 2.16 shows an example of a SVC in 2D space.

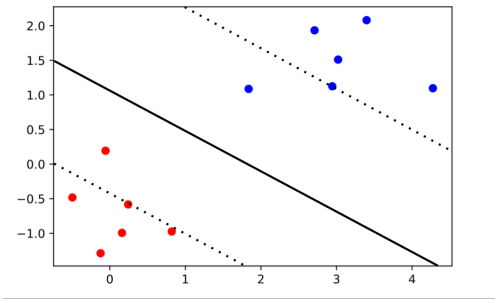


Figure 2.16: An example of a SVC in 2D space. The solid line represents the hyperplane, the dashed lines are the margins. The red and blue point being inside the margin is an effect of the their respective slack variables being larger than 0.

An interesting property of the SVC is that it is only the points that lay on or within the margin that affect the position of the hyperplane. These are known as the *support vectors* [47]. For example, if you were to move any of the top four blue points in Figure 2.16 to any other location on the correct side of the margin, it would not change the classifier at all.

In 2015 the authors in [49] proved that the linear regression model known as the Elastic Net [50] could be reduced to a SVC. For every Elastic Net instance, there exists a solution to a constructed binary classification problem in which the hyperplane is identical to the Elastic Net solution. The practical implications of this is that it enables the use of optimized SVC solvers for Elastic Net problems [49].

Support Vector Machine

A *Support Vector Machine* (SVM) is an extension of a SVC that can handle data with non-linear boundaries between classes by transforming the feature space to a higher dimension and solving the (linear) problem there. This is done with *kernels* [47] (not to be confused with convolutional kernels). The idea behind kernels is to use the advantage of a high-dimensional space without actually moving to a higher dimension. The only operation done between two points in the high-dimension space is the scalar product, this operation can be computed using the low-dimensional representation of the points. An example of how this is done is shown in Equation 2.14.

$$\vec{x} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \qquad \phi(\vec{x}) = \begin{bmatrix} x_1^2 \\ \sqrt{2}x_1x_2 \\ x_2^2 \end{bmatrix} \tag{2.13}$$

$$\begin{aligned} \phi(\vec{x}) \cdot \phi(\vec{y}) &= x_1^2y_1^2 + 2x_1x_2y_1y_2 + x_2^2y_2^2 = \\ &= (x_1y_1 + x_2y_2)^2 = (\vec{x} \cdot \vec{y})^2 = K(\vec{x}, \vec{y}) \end{aligned} \tag{2.14}$$

Equation 2.13 shows a transformation ϕ of a 2D vector into a 3D space, and Equation 2.14 shows how you can calculate the dot product in the 3D space by using the 2D space representations of the two vectors $\vec{x}$ and $\vec{y}$. This is an example of the common polynomial kernel

$$K(\vec{x}, \vec{y}) = (\vec{x} \cdot \vec{y} + d)^p \tag{2.15}$$

where $d = 0$ and $p = 2$. Another common kernel is the radial basis function (rbf):

$$K(\vec{x}, \vec{y}) = e^{-\gamma||\vec{x}-\vec{y}||^2} \tag{2.16}$$

where γ is a hyperparameter. [47]

The optimization problem for the SVM with the transformations is given by Equation 2.17 [48]. However, in practice, the optimization problem is converted into a corresponding dual problem using the Lagrangian approach [51]. It is in this dual problem, which is out of scope for this paper, the kernel functions are utilized.

$$\begin{aligned} &\text{Minimize } \frac{1}{2}||\vec{w}||^2 + C\sum_i \epsilon_i \\ &\text{Under the constraints } y_i \times (\vec{w} \cdot \phi(\vec{x_i}) - b) \geq 1 - \epsilon_i, \forall i \\ &\qquad\qquad \epsilon_i \geq 0, \forall i \end{aligned} \tag{2.17}$$

Support Vector Regression

The SVR model uses all the same techniques as the SVM, except that the problem has been converted in to a regression problem which focuses on finding a function $f(x)$ that has an error $|f(x_i) - y_i|$ at most ε [52]. The hyperparameter ε is an upper bound on the error tolerated, and much like the support vectors of the SVM, the SVR does not care about errors less than ε [53]. The optimization problem for the SVR, given training vectors $\vec{x_i}$ and their corresponding target values y_i, is:

$$\begin{aligned} &\text{Minimize } \frac{1}{2}||\vec{w}||^2 + C\sum_i(\zeta_i + \zeta_i^*) \\ &\text{Under the constraints } y_i - \vec{w}\cdot\phi(x_i) - b \leq \varepsilon + \zeta_i, \\ &\qquad \vec{w}\cdot\phi(x_i) + b - y_i \leq \varepsilon + \zeta_i^*, \\ &\qquad \zeta_i, \zeta_i^* \geq 0, \forall i \end{aligned} \tag{2.18}$$

where ζ_i^* denotes training errors above ε and ζ_i denotes training errors below ε [48], [54]. The ζ variables, analogous to the slack variables in the SVM optimization problem, allow for some pairs (x_i, y_i) to be approximated with worse precision than ε and thus relaxes the constraints of the optimization problem [53]. Figure 2.17 illustrates a linear SVR model.

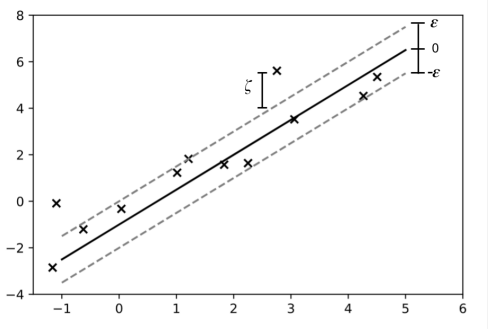


Figure 2.17: An example of a linear SVR in 2D space.

2.3 Related Work

In this section some of the identified related work is presented. Most of the work presented is focused on the ML approach. This is due to the fact that the choices of ML models and input features in said work were used as a basis for the choice of models and features in this thesis.

2.3.1 Wait-time Prediction

The research done in [11] is focused on producing new valuable predictors for a multi-server queue with time-varying arrival rate, service rate and customer abandonment, much like the queue in place at the company KRY. This study suggests that the predictors Queue-Length (QL) and Head of Line (HOL) are significant. HOL is the elapsed waiting time for the consumer in the front of

the line. Although [11] focuses on the queuing theory research area, the result of their study can be applied in this thesis.

In a study from 2017 by Mourao, Carvalho, Carvalho, and Ramos [12] several different models to predict (classify) wait-time overflows in banking queues were compared. Banks in Brazil can be served fines if their customers have to stand in line over a certain amount of time. Such a case was considered an overflow. The models compared were 3 ML models (Deep Learning, Gradient Boost Machine and RF) as well as a model based on queuing theory. The ML models were trained on historical data about the queue state for each costumer. The features of this data included, but were not limited to: Arrival Rate, Service Rate, Arrival Time, Average Service Time, Queue Length, Head of Line (HOL). All ML models outperformed the queuing theory model, both in accuracy and F1-score. The GBM achived the highest accuracy of 97%, and the RF model had an accuracy of 96%. The work done in [12] differs from this paper because of the single-server queuing system used and the classification approach.

The ML approach was proven to be viable for wait-time prediction by Kyritsis and Deriaz in their study from 2019 [10]. Kyritsis and Deriaz implemented a ANN model with 2 hidden layers and trained it with an Adam Optimizer on historical data from banking queues. The dataset used is publicly available and consisted of, for each person that joined the queue, the 5 features: queue length, day of week, hour of day, minute of hour and the wait-time. The data had a mean wait-time of 13.18 minutes, a median of 12 minutes and a standard deviation of 5.95 minutes. The final model produced and Mean Absolute Error (MAE) of 3.35 minutes, compared to a naive model which always predicted the mean with a MAE of 4.71 minutes. The work in [10] differs from this paper by the fact that the number of servers in the multi-server queue system was an unknown variable, and by the choice in ML model.

In a study by Zhang, Nguyen, and Zhang in 2013, several algorithms, including some ML models, were implemented to predict the wait-time on a time series dataset from the Department of Motor Vehicles (DMV) in San Jose, California [17]. The data consisted of the wait-time for every 10 minutes of the day. The simple linear regression model achieved the lowest average error, followed closely by the SVM. The ML models were trained on the features: latest wait-time, time of the day and day of the week. Much like the work in this thesis, [17] compares the performance of multiple ML models on wait-time prediction. However, the work differs by the fact that the data used as well as the algorithms chosen were much simpler.

2.3.2 Wait-time Prediction in Healthcare

As early as 1987 attempts at modeling queues in healthcare were made in [55]. Accurate wait-time predictions are especially important in healthcare, because the patients are more likely to already be in a worried state of mind when they are seeking help [5], [56].

In the study from 2017, performed by Joseph, Hijal, Kildea, Hendren, and Herrera the treatment durations in a Radiation Oncology clinic were predicted and compared by four different ML regression models [5]. The predicted treatment durations were used to produce waiting time estimations. The ML models were trained on historical data of treated patients. The features of the data included general information about the patient, which physician the patient had, at which time the appointment was scheduled as well as actually held, general information about the treatment received and more. The 4 different ML regression models compared were a linear regression model, a SVM, a Decision tree and a RF. The RF model was found to be the best with a MAE of 4.6 minutes and an R^2-score of 0.47. Even though the study done in [5] does not directly predict the wait-time as is done in this thesis, a different approach is proposed, and the authors raise the importance of doing research regarding wait-time prediction within a healthcare setting.

In 2018, Curtis, Liu, Bollerman, and Pianykh examined the possibility to predict patient wait-time using ML in the Massachusetts General Hospital Department of Radiology [9]. Four different examination types were considered: CT, MRI, ultrasound and radiography. Only the radiography examination modality had a drop-in queue system with 9 devices that could be used in parallel. From the existing information system at the hospital, the authors were able to extract 40 different predictors for each patient that visited the hospital during the span of half a year. Multiple ML models (such as GBM, Elastic Net, RF, SVM, Bagging, Neural Network and more) were trained and evaluated on the collected historical data. For the radiography examination modality, the RF model achieved the lowest RMSE of 8.816, and had the 3:rd highest R^2 score of 0.4582. The Elastic Net followed by the Linear Regression model had the 2 highest R^2 scores, both slightly above 0.46. The most important predictors (applicable to the drop-in radiography) were identified as the queue length (current and most recent) and the median examination time of the five most recent examinations. The work done in [9] provides further indication that RF is an good choice in model for wait-time prediction in patient queues. Moreover, it provides a new potential predictor which can be adopted in the work done in this paper.

Sanit-in and Saikaew made a comparative study of three wait-time prediction approaches in 2019 [13]. The models considered were a queuing theory model, a naive model predicting the mean wait-time and a RF model. The study used two datasets collected from two different queues. The first dataset had 3480 records from a post office queue, and the second had 1348 records from an ear, nose and throat clinic from a hospital. Both datasets had the same features. These features included, but were not limited to: the number on the queue ticket, hour of the day, day of the week, period of day (morning, afternoon), queue length, arrival rate, service rate and wait-time. The wait-time was divided in to 5 classes ranging from very short to very long in order to convert the problem in to a classification problem. The RF model achieved the highest accuracy on both datasets. For the post office dataset it had an accuracy of 81.70%, while the queuing theory model had an accuracy of only 65.23%. The naive baseline model achieved an accuracy of 64.94%. The RF model had an accuracy of 85.76% on the ear nose and throat clinic dataset, where the naive model had an accuracy of 68.89%. The queuing theory model could not be applied on the ear, nose and throat clinic, since it did not satisfy all constraints that the model requires. In addition to the comparative study, [13] also explored the feature importance for the RF model. Three features that proved to affect the outcome of the predictions the most were the queue length, the arrival rate and the service rate. The arrival rate was measured as the number of consumers arriving per minute at the time, and the service rate was measured as the number of consumers receiving the service per minute at the time. Compared to this thesis, the work in [13] had a lot less data. Moreover, they approached the wait-time prediction problem as a classification problem rather than as a regression problem.

Chapter 3

Methodology

This chapter describes the methods used to carry out the experiments. A description of the queuing system in place at KRY (hereby referred to as the company), and the historical data used for training is given. A detailed description of the implementation of all the prediction models is given (including the simulation algorithm implemented by the company), as well as the hyperparameter optimization executed for the ML models. Finally, the evaluation metrics are defined.

All hyperparameter optimization, training and evaluation were executed on a Deep Learning Amazon Machine Instance (DLAMI) of type g4dn.xlarge (4x vCPU, 16GB RAM, 1x NVIDIA T4 GPU), running Ubuntu 16.04.

3.1 The Queuing System and its Data

3.1.1 The Queuing System

The queue system in place at the company is a single-queue multi-server system, as described in subsection 2.1.2. The consumers are the patients and the servers are clinicians offering the consultation resource. However, there are some differences, making this system slightly more complex.

One difference in this system, compared to the system described earlier, is that patients can not be received by just any clinician. Patients seek help for a certain symptom and belong to certain age groups, and some symptoms or age groups require specialization to treat. One example is child care. A child can only receive help from a clinician specialized in child care. To solve this, patients are assigned labels when joining the queue, which are based on variables such as age and symptom. The patients can only be matched with

clinicians carrying the same labels, indicating that they can treat those patients.

Another difference is the fact that there exist some special cases of patients, which are given different priorities in the queue. Such cases include: infants, prescription renewals, and internal referrals. In the case of prescriptions renewals, patients do not have to have a video consultation but can do this via a text meeting. These meetings require much less time compared to video consultations. Internal referrals is the case where a clinician at the company has referred a patient to another clinician at the company. The patients in the queue are sorted firstly by priority, and secondly by arrival time.

The last difference in the queue system is that patients are unable to join the queue if the wait-time estimation by the simulation model exceeds 4 hours. This is a feature implemented to avoid overcrowding.

3.1.2 Data

Data from 767246 patients between the period of 2018-04-22 to 2020-02-17 was collected. In the data cleaning process patients with missing values and patients with a recorded wait-time exceeding 6 hours were considered anomalies and were therefore dropped. Next, patients with a difference in wait-time compared to the previous patient of more than 3 times the standard deviation ($\sigma = 9.3214$ minutes) were considered outliers and were also dropped. This was done due to the fact that admins can access the queue system and manually assign patients to clinicians, resulting in a very small wait-time. After the cleaning process, data from a total of 757086 patients remained.

The data collected from the queue is not uniformly sampled. Sequence modeling applications implicitly require that the data is uniformly sampled to be effective. The solution applied to this problem was to append the time intervals Δt (difference in time between each patients) to the input vector as suggested in [57]. A description of each feature included in an observation is displayed in Table 3.1.

For the sequence modeling models, the input features were normalized by removing the mean and scaling to unit variance. This was done using Scikit-learns StandardScaler [48]. The sequence length was set to 110, thus, each multivariate input sequence consisted of 110 inputs, $x_0, ..., x_{110}$, where each vector x_t contained information about all input features described in Table 3.1. The output target y_{110} was the actual wait-time feature for patient x_{110}, as described in Equation 2.2.

The technique of *Exponential Smoothing*, as described in subsection 2.2.1, was applied in order to incorporate the temporal dimension in the data for the

non-sequence modeling models. A smoothened representation for each input feature (except for year, month, day and minute) was calculated by Equation 2.3. The smoothened representations of the input features thus contained information about all previous input features, decreasing exponentially with a factor α. The value of α was left as a hyperparameter. Whether to use only the smoothened representations or to append them to the input vector was also left as a hyperparameter.

The data was divided into 3 different sets: a *training, validation* and *test-set*. This division was done in chronological order, in order to preserve the temporal dimension of the time series. 80% of the data was used for the train-set, 10% was used for the validation-set, and the remaining 10% for the test-set.

Feature	Description
Year	Year of the visit.
Month	Month of the year.
Day	Day of the month.
Hour	Hour of the day.
Minute	Minute of the hour.
Δt	Seconds elapsed since the latest patient joined the queue.
HoL	How long the patient currently in the front of the queue has waited.
Queue length	Number of patients currently in the queue.
Arrival rate	Number of patients arriving per second, measured over the latest 15 patients.
Gender	Gender of the patient. 0, 1 or 2, where 0 means not specified.
Age	Age of the patient.
Symptom	The symptom the patient is seeking help for. This is selected by the patient from a list of 61 symptoms, and was represented by the index in said list.
Latest meeting length	The length of the latest meeting that ended.
Latest wait-time	The wait-time of the latest patient that received help.
Doctors working	Number of doctors currently working.
Receiving service	Number of patients in meetings.
Mean average meeting length	The mean of the currently working doctors average meeting length.
Service rate	Mean number of patients in meetings measured over the latest 15 patients.
Simulation wait-time	The wait-time estimation given by the simulation model.
Actual wait-time	The actual wait-time in minutes for the patient.

Table 3.1: A description of all the features in the data collected from the queue. The first group consists of all input features used for the ML models. The second group consists of the simulation estimation and the target feature.

3.2 Models

3.2.1 Naive

Two naive models were implemented. The first model, Naive (previous), was defined as:

$$\hat{y}^{(t+1)} = y^{(t)} \tag{3.1}$$

This model is not possible to apply in the real world application, because the wait-time for the previous patient is most often not known when the next patient arrives. The previous patient is most likely still in the queue, thus, this prediction model uses information from the future. However, the model was used merely as a benchmark for the other models.

The second naive model, Naive (latest known), predicts the latest known wait-time. This is one of the input features as described in Table 3.1. More formally, the model was defined as:

$$\hat{y}^{(t+1)} = x^{(t)}_{\text{latest\_wait\_time}} \tag{3.2}$$

3.2.2 The Simulation Algorithm (Sim)

The simulation algorithm (Sim) implemented by the company executes periodically for continuous predictions, as well as on certain triggers, for instance, when a new patient joins the queue. The initial value that the simulation algorithm predicts for each new patient is saved. The algorithm was therefore not re-implemented, instead, the existing saved values were used. The simulation algorithm has access to all schedules for all clinicians, and thus knows when each clinician is working and when each clinician is on a break. Moreover, it uses the information in the labels for both patients and clinicians during the matching, to be able to mimic the queue as perfectly as possible. In order to calculate an expected end time for meetings which have started, the algorithm uses hard coded meeting length estimations. There is one estimation for each type of meeting. The estimation for text meetings is much shorter than the one for video meetings. These estimations are based on the average meeting length for meetings of the same type, but are used as a management tool in the company. For example, if the estimation is set to x minutes, all clinicians will be informed that they are expected to spend on average x minutes per consultation.

When the simulation runs, it moves time forward by iterating over all future events and adds to said event list continuously as more events are scheduled.

Initially, the events consist of all the start and end times of the clinician shifts, including their breaks, up to 7 days in to the future. Additionally, if any shift has an ongoing meeting, the expected end time of said meeting is added as an event. This end time is calculated based on the meeting length estimations explained above. At the time for each event in the ordered event list, a check is done if any ongoing meeting has ended, signifying that a clinician has become available for matching. Furthermore, all scheduled shifts are searched for shifts that have either ended or started at this point in time. Clinicians whose shifts just ended are no longer considered for matching, and vice versa for clinicians whose shifts just started. Next, the patients waiting in the queue are matched with the currently available clinicians based on their labeling information. An event is created for each matched patient, signifying the end time of their meeting. The matched patients are assigned wait-time estimations calculated by taking the difference of the simulated matching time and the actual current time. The algorithm is presented in pseudo code below in Algorithm 1.

Algorithm 1 Pseudo code for the simulation algorithm.

Input: $queue$ - the queue of patients
$shifts$ - all scheduled clinician shifts
now - the current time
Output: $waitTimes$ - a list of all patients from $queue$ with assigned wait-time estimations

```
1:  function SIMULATION(queue, shifts, now)
2:      – initialization –
3:      events ← empty ordered collection          ▷ Timestamps of events
4:      ongoing ← empty collection                 ▷ Shifts with ongoing meetings
5:      available ← empty collection               ▷ Active shifts with no meeting
6:      waitTimes ← empty collection               ▷ Patients with assigned wait-times
7:      for all shifts s in shifts do
8:          Add start and end time of s to events
9:          Add all start and end times of breaks in s to events
10:         if s has ongoing meeting then
11:             Add s to ongoing
12:             Add expected end time of meeting to events
13:         else if s is active now then
14:             Add s to available
15:         end if
16:     end for
17:     – main loop –
18:     while events is not empty do
19:         tick ← events.getAndRemoveFirst()
20:         for all shifts s in ongoing do
21:             m ← s.getOngoingMeeting()
22:             if m expected end time ≤ tick then          ▷ m has ended
23:                 Move s to available
24:             end if
25:         end for
26:         for all shifts s in shifts do
27:             if s's end time ≤ tick then                ▷ s has ended
28:                 Remove from available
29:             else if s started at tick then
30:                 Add s to available
31:             end if
32:         end for
33:         Match patients from queue to applicable clinicians in available
34:         Assign each matched patient a wait-time estimation (tick − now)
35:         Add their expected meeting end times to events
36:         Move matched patients from queue to waitTimes
37:         Move matched clinician shifts from available to ongoing
38:     end while
39:     return waitTimes
40: end function
```

3.2.3 TCN

The TCN model was implemented using TensorFlow2 [58] and the Keras library [59]. The implementation of the TCN layer in [60] was used, together with a final dense layer with linear activation. The TCN layer in [60] was made following the specification from [15]. The sole difference is that the *weight normalization* in the residual block was swapped for an optional *batch normalization* [61]. Whether to use batch normalization or not was left as a hyperparameter. The other hyperparameters were *kernel size*, number of *filters*, number of residual *blocks* (with exponentially increasing dilation), *dropout rate*, and *kernel initializer*. The Adam Optimizer [62] from Keras was used during training, with the default learning rate of 0.001. The model was trained for a maximum of 100 epochs with a *batch size* of 64 and a *sequence length* of 110. Early stopping was used with a patience of 20, i.e. the training was terminated if the validation MSE did not decrease for 20 consecutive epochs.

3.2.4 LSTM

The LSTM model was implemented using TensorFlow2 [58] and the Keras library [59]. The LSTM layer from Keras was used together with a dense layer with linear activation. All hyperparameters to the layer were set to the default values except for the number of *units*, *kernel initializer*, *dropout rate*, and *recurrent dropout rate*. Identically to the TCN, the LSTM was trained for a maximum of 100 epochs using: the Adam Optimizer [62] with the default learning rate of 0.001, early stopping with a patience of 20, a *batch size* of 64, and a *sequence length* of 110.

3.2.5 RF

The Scikit-learn implementation RandomForestRegressor [48] was used for the RF model. All hyperparameters were left to their default values except *n_estimators* (for the number of trees), *max depth*, and *max features* (maximum features considered at each split).

3.2.6 SVR

The GPU accelerated implementation of the SVR in the cuML library from the open source project Rapids [63] was used. The hyperparameters were left to default except for the type of *kernel*, which *degree* for polynomial kernels, amount of slack *C* allowed, and *epsilon*.

3.2.7 Combination Model (Comb)

A final combination model (Comb) was created which was a combination of the simulation model and the best performing ML model. This was done by simply appending the simulation predictions to the input vector of the ML model as an additional feature.

3.3 Hyperparameter Optimization

For each ML model a hyperparameter optimization was made. This was done using the library *Hyperopt* [64]. This library applies *Bayesian Optimization*, a probabilistic model, in order to find the hyperparameters that minimizes the validation loss by using a Tree-structured Parzen Estimator algorithm [65]. This has proved to be better than a random search, and it reduces the number of search iterations required by taking past iterations into account when starting new ones [64]. The details of this algorithm are described in [65]. The *search spaces* for all hyperparameters for all models are defined in Table 3.2, Table 3.3, Table 3.4, and in Table 3.5. For each model 100 trials (search iterations) were executed.

In order to speed up the hyperparameter optimization process, all models where trained on 40% percent of the training data (the most recent, i.e. the last in chronological order). The TCN and LSTM were trained for 4 epochs each trial, and the SVR had a limit on the maximum iterations made in the fitting algorithm of 20000.

Parameter	Search Space
kernel size	[2, 6]
filters	[32, 192]
blocks	[2, 6]
dropout rate	[0.0, 0.5]
kernel initializer	[he_normal, he_uniform]
use batch normalization	[False, True]

Table 3.2: TCN hyperparameter search space. The maximum amount of trainable parameters is 2845633, and the maximum receptive field is 631.

Parameter	Search Space
units	[32, 832]
kernel initializer	[glorot_normal, glorot_uniform]
dropout rate	[0.0, 0.5]
recurrent dropout rate	[0.0, 0.5]

Table 3.3: LSTM hyperparameter search space. The maximum amount of trainable parameters is 2832961.

Parameter	Search Space
n_estimators	[15, 512]
max depth	[0, 128]
max features	[1, n]
alpha	[0.2, 1]
only smooth	[False, True]

Table 3.4: RF hyperparameter search space. If **only smooth** was set to false, the smoothened representations of all features would be appended to the input vector instead of replacing it. Therefore, $n = 18$ or $n = 31$, depending on the value of **only smooth**.

Parameter	Search Space
kernel	[linear, poly, rbf]
degree	[2, 4]
C	$e^{\mathcal{N}(0,1)}$
epsilon	[0.0, 1.0]
alpha	[0.2, 1]
only smooth	[False, True]

Table 3.5: SVR hyperparameter search space, where $\mathcal{N}(0, 1)$ denotes the normal distribution with $\mu = 0$ and $\sigma = 1$. The **degree** parameter was only used for the polynomial kernel. If **only smooth** was set to false, the smoothened representations of all features would be appended to the input vector instead of replacing it.

3.4 Evaluation

In order to reduce the variance and effect of randomness for each ML model, 5 separate models were trained and combined into a final model by averaging their separate predictions. This does not apply to the SVR model, since its training process is deterministic, i.e. does not include randomness.

To determine if there was a statistically significant difference in the errors of any model, a Friedman test [66] was done.. The Friedman test is the non-parametric version of a analysis of variance test (ANOVA). The non-parametric version was used because the errors can not be assumed to be normally distributed. The null hypothesis was that all errors had the same distribution. The hypothesis was tested by picking a random sample of $N = 25000$ patients from the test data, which total size was 75600. The squared errors for the predictions from each final model were calculated for each patient in the sample. The Friedman test implementation from the SciPy library [67] was then used for testing said error samples. The significance level was set to 0.01.

The Friedman test was followed by a post-hoc pairwise comparison using the non-parametric version on the dependent (paired) t-test which is the Wilcoxon signed-rank test [68]. This was done to determine between which models there was a statistical significant difference in the errors. The null hypothesis tested was that the two paired samples come from the same distribution. Similar to the Friedman test, this was done by sampling $N = 25000$ patients from the test data, calculating the square errors in the predictions for each models, and then using the Wilcoxon singed-rank test implementation from the SciPy library [67] with a significance level of 0.01.

To measure the performance of each final model their MSE and R^2 was calculated on the test data. The MSE is the most commonly used loss function for regression, and it is calculated as:

$$\text{MSE} = \frac{1}{n}\sum_{i}^{n}(\hat{y}_i - y_i)^2 \tag{3.3}$$

The R^2 score ranges between 0 and 1, and is calculated as:

$$R^2 = \frac{\sum_i^n(\hat{y}_i - \bar{y})^2}{\sum_i^n(y_i - \bar{y})^2} \tag{3.4}$$

where $\bar{y}$ is the mean of the observed data.

Chapter 4

Results

This chapter starts by presenting the results of the hyperparameter optimization for each ML model. Secondly, it presents the result of the evaluation of all models. This includes the MSE, R^2, training times and the results of the statistical tests. Lastly, a selection of forecast values are displayed for each model.

4.1 Hyperparameter Optimization

The following sections present the hyperparameters used for the final ML models referenced in the rest of the paper. The hyperparameters chosen were the ones from the model that achieved the lowest validation loss out of the 100 different trial models.

4.1.1 TCN

The hyperparameters with the lowest validation loss are displayed in Table 4.1. The lowest validation loss was 52.502 and the average s/trial were 295.363. This yields a receptive field of 33 according to Equation 2.6.

kernel size	filters	blocks	dropout rate	kernel initializer	batch norm
3	64	3	0.0637	he_normal	False

Table 4.1: The hyperparameters used for the TCN model.

4.1.2 LSTM

Table 4.2 presents the hyperparameters for the LSTM model which resulted in the lowest validation loss of 50.470. The average s/trial were 2391.482.

dropout rate	kernel initializer	recurrent dropout rate	units
0.0457	glorot_normal	0.235	384

Table 4.2: The hyperparameters used for the LSTM model.

4.1.3 RF

The hyperparameters that received the lowest validation loss of 51.227 are displayed in Table 4.3. The RF model took on average 140.498 s/trial.

n_estimators	max depth	max features	alpha	only smooth
448	16	12	0.364	False

Table 4.3: The hyperparameters used for the RF model.

4.1.4 SVR

The SVR model with the lowest validation loss of 59.021 had the parameters displayed in Table 4.4. The SVR model took on average 216.340 s/trial.

kernel	C	epsilon	alpha	only smooth
rbf	339.705	0.968	0.201	False

Table 4.4: The hyperparameters used for the SVR model.

4.2 Evaluation

The p-value produced by the Friedman test on a sample of the test data with a size of 25000 was 0.0, signifying that at least one model does not have the same distribution in errors compared to the rest on a significance level of 0.01.

The resulting p-values of the post-hoc pairwise comparison using the Wilcoxon singed-rank test is displayed in Table 4.5. The table shows that there is a significant difference in the error distribution between all pairs of models with a significance level of 0.01.

Model	LSTM	TCN	RF	SVR	Comb
TCN	1.3×10^{-5}	-	-	-	-
RF	2.2×10^{-80}	7.9×10^{-69}	-	-	-
SVR	1.9×10^{-272}	1.5×10^{-255}	1.8×10^{-84}	-	-
Comb	4.1×10^{-17}	1.2×10^{-23}	6.7×10^{-95}	2.0×10^{-260}	-
Sim	1.3×10^{-3}	6.7×10^{-4}	1.0×10^{-17}	7.4×10^{-55}	5.2×10^{-63}

Table 4.5: The p-value results from the post-hoc pairwise Wilcoxon signed-rank test. Sample size $N = 25000$.

The MSE and R^2 for each of the final models are displayed in Table 4.6, and the training times are displayed in Table 4.7. The TCN model achieved the lowest MSE out of all standard models. Therefore, this model was used for the combination model (Comb).

Model	**MSE**	**R^2**
Naive (previous)	28.8	0.935
Comb	38.3	0.914
TCN	56.2	0.873
Simulation	56.6	0.872
LSTM	61.1	0.862
RF	70.3	0.841
SVR	78.6	0.823
Naive (latest known)	102.7	0.768

Table 4.6: MSE and R^2 for each model sorted on MSE. The Naive (previous) model is not possible to implement in a real scenario, because the previous patient has most likely not been assigned a clinician when the next patient arrives.

The Naive models were implemented as baselines to get a better understanding of the performance of the other models. The Naive (previous) model is not possible to implement in a real scenario due to the fact that it uses information from the future. However, it provides a benchmark MSE to the other models. Table 4.6 therefore shows that a model achieving an MSE close to 28.8 can be considered a high performing model. The Naive (latest known) model provides another benchmark MSE of 102.7. Models achieving MSEs close to this value have practically not learned anything, since the latest known patient wait-time is one of the input features.

The Comb model achieved the lowest MSE of all models, not including

the Naive (previous) model. All ML models and the Simulation model outperformed the Naive (latest known) baseline model.

The models in the non-sequence modeling ML approach (RF and SVR) achieved substantially higher MSEs and lower R^2 scores compared to the models in the sequence modeling approach (LSTM and TCN), as can be seen in Table 4.6. Since the statistical tests showed that there was a statistically significant difference between all said models, this provides evidence that the sequence modeling approach is superior to the non-sequence modeling approach at predicting wait-time in this specific digital healthcare setting.

The TCN is significantly faster at training compared to the LSTM. The non-sequence modeling ML models were significantly faster at training compared to the Neural Network models.

Model	training time (s)	s/epoch
SVR	225	-
RF	801	-
TCN	1920	52
Comb	1947	52
LSTM	48900	1407

Table 4.7: The average training time and seconds per epoch form all ML models. The RF and SVR models are not trained in epochs.

4.3 Forecasting

Figure 4.1 and Figure 4.2 display the forecasts from each model from 2 randomly selected days in the test data (not including the Naive (previous) model). It should be noted that these days only consist of a small part of the test set, which contained a total of 55 days, and might not represent the whole data set. More forecast graphs can be found in Appendix A.

As one can see in the figures, the ML models tend to react slower to changes in the wait-time. The simulation algorithm however seem to overestimate the changes and predict a higher wait-time than necessary during the peak hours. In Figure 4.1, the combination model do not seem to suffer from any of the problems mentioned. Whilst, in Figure 4.2, it seems to have both problems, but in a much smaller scale.

The simulation seems to be able to handle special cases better than the ML models. An example of this can be seen at approximately 17:45 in Figure 4.1, where the wait-time has 4 distinct dips. This happens to be 3 child patients

and one patient renewing his/her prescription. These are all considered special cases as explained in subsection 3.1.1. The simulation is able to assign a much lower wait-time to these patients, but the ML models does not seem to have learned this feature in the queue system, even though the input features include both age and symptom (prescription renewal is considered a symptom).

The combination model seems to have the benefit of fast reactions and special case detection from the simulation outputs and the benefit of not overestimating spikes in the wait-time from the ML model. This can be seen clearly in both Figure 4.1 and Figure 4.2.

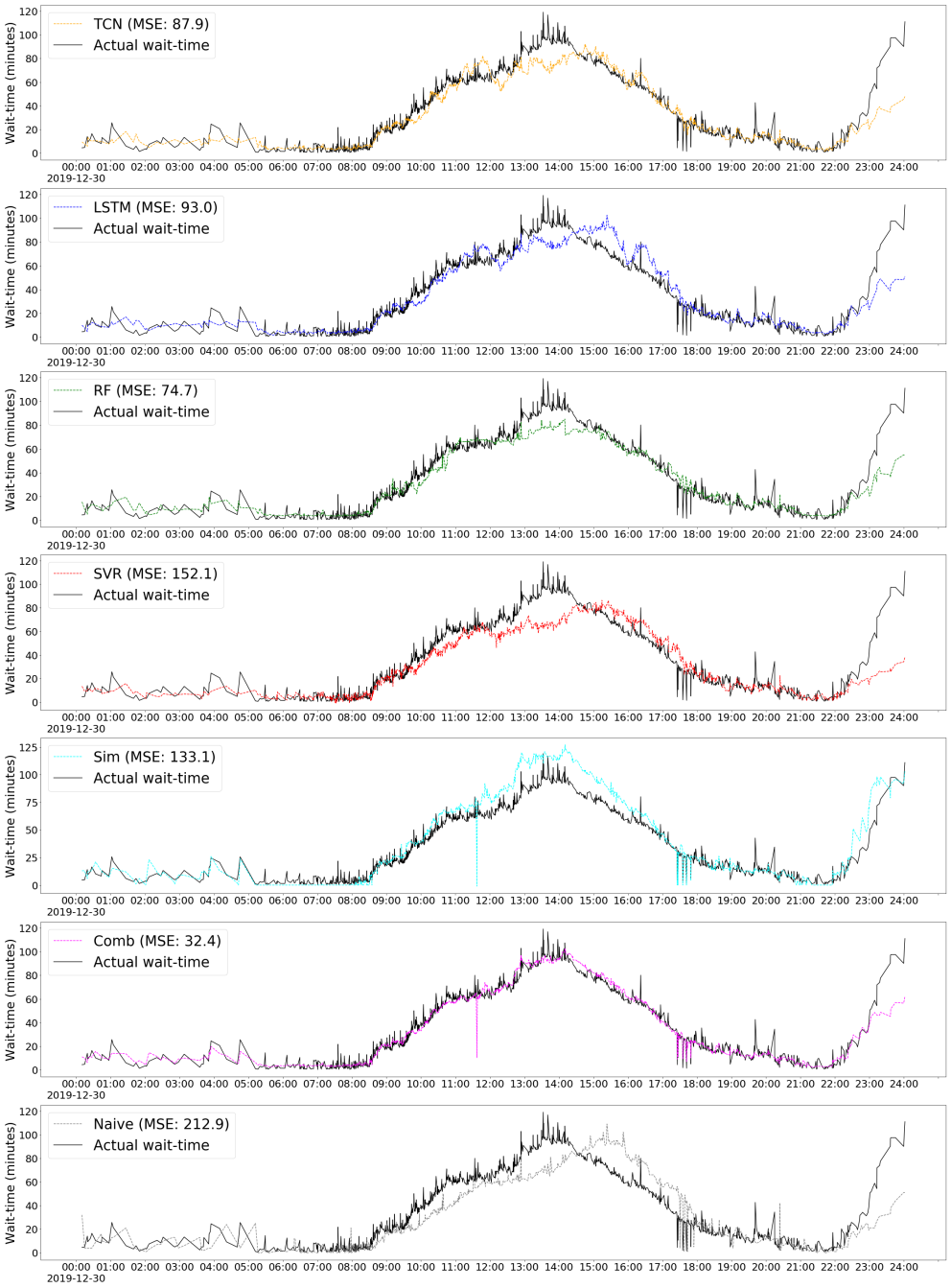


Figure 4.1: The actual wait-time and the predicted wait-time for each model for single day in the test data (2019-12-30).

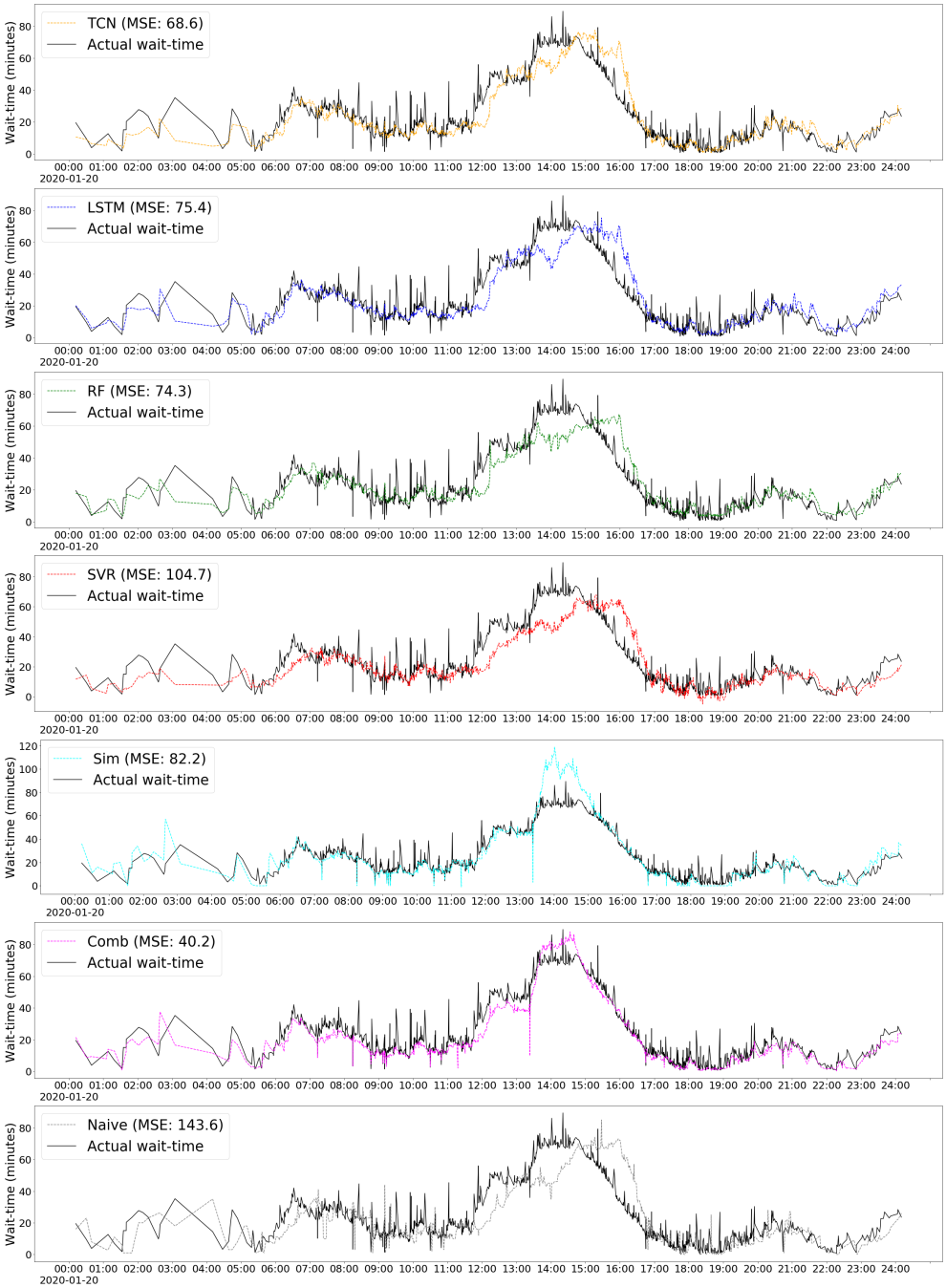


Figure 4.2: The actual wait-time and the predicted wait-time for each model for single day in the test data (2020-01-20).

Chapter 5

Discussion

In this chapter a discussion regarding the results and their implications is held. The performances of the models are compared according to the measured metrics, and other aspects of the different models are brought to light. The forecast graphs are analyzed and evaluated in detail. The limitations in the data used and the methodology are underlined, the impact of the work is discussed and finally future research is suggested.

5.1 Model Comparison

The sequence modeling approach was superior to the non-sequence modeling approach at predicting wait-time in this specific digital healthcare setting. The reason for this is believed to be that introducing the temporal dimension with sequences provides a better structure to the data which is easier to learn, compared to the exponential smoothing technique. While utilizing exponential smoothing did improve the performance of the models compared to not introducing the temporal dimension at all, it did not improve the performance as much as the sequencing. This raises the question regarding why no other research applying the more complex models from sequence modeling to the wait-time problem can be identified. One theory is that the implicit requirement of uniformed sampled data has been considered a bigger hindrance than it seems to be. Another reason could be the complexity which is introduced with neural networks.

The TCN achieved a lower MSE compared to the simulation algorithm, but the LSTM model did not. Therefore, no claim of superiority can be made for either approach. In order to outperform the simulation algorithm, the ML models have to learn everything the simulation algorithm is given, and ad-

ditionally learn or pick up on some other pattern unknown to the simulation algorithm. This is believed to be the reason as to why not all ML models did outperformed the simulation algorithm. The simulation algorithm has access to the schedules of all clinicians and the labels of said clinicians and all patients. This labeling information is very specific to the queuing system of the company. Therefore, the ML models were not given this information in an attempt to make the research conducted in this thesis more widely applicable. Yet, the TCN managed to achieve a MSE lower than the simulation algorithm, which is considered to be a notable result. The TCN having better performance compared to the LSTM is aligned with the research in [15], [20], [21].

The combination model achieved the lowest MSE and highest R^2 out of all models. This is believed to be due to the fact that this model does not have to learn to predict wait-time, rather it only have to learn to predict and prevent the mistakes the simulation algorithm makes. This model has access to more information compared to the other ML models, and it is therefore no surprise it has the best performance. There is only a difference of 9.4 in the MSE between the combination model and the Naive (previous) baseline model, which indicates that the performance of the combination model is impressive. A downside to consider for the combination model is that it will require the most maintenance. Both the simulation and the ML model (TCN) will have to be updated in the event of any major changes to the queue system.

The RF model outperformed the SVR model. This was expected, as it has proven to be the best model for wait-time prediction in numerous other studies [5], [9], [13].

The non-sequence modeling ML models required considerably less training time compared to the sequence modeling models. Lower training time means easier and faster maintenance of the models, which can be of importance if they were to be commercially used. The TCN was also significantly faster to train compared to the LSTM. This was expected because of the parallelizable structure of the TCN, and aligned with the results in other research [15], [20]. However, computational power has become cheaper and cheaper with the years, which might mitigate or even eliminate the problem of long training times.

Interpretability can be an important factor when using any prediction model commercially. This is a problem for all ML models in this paper, except the statistical RF model, and it is known as the *Black Box* problem. The models have virtually no interpretability and can therefore be regarded as black boxes, which receive an input and produces an output with no explanation as to why said output was produced. If a customer receives an unexpected wait-time

prediction, it might be of interest to the company to figure out the reason as to why, both to be able to provide an explanation to a potentially angry customer, and to be able to prevent such a scenario from re-occurring. Therefore, there is an advantage to using a simulation algorithm or the statistical RF model. The simulation algorithm can be re-run with the same inputs, and be analyzed step by step in order to exactly explain the reason behind the given wait-time estimation. The RF model is less interpretable compared to the simulation since it consists of several trees. However, the trees can still be unraveled and the questions they contain reveled in order to gain some understanding as to why a certain wait-time prediction was given.

5.1.1 Forecasts

A pattern can be noticed in the forecast behaviors of all ML models compared to the simulation. The ML models tend to have slower reactions to changes in the wait-time, which causes them to lag behind the curve a bit. This is believed to be due to the latest known wait-time feature, which essentially is the wait-time curve with a time delay. The ML models have to learn how the wait-time has changed since the latest known wait-time, and if they are unable to predict any change, the resulting forecasts will follow this time delayed curve to the point, which is exactly what the Naive (latest known) model does. The simulation, on the other hand, has faster reactions but tend to overestimate the spikes in the wait-time. The wait-time is affected by several different factors, for instance, at peak hours the working clinicians can potentially take the long queue into account and start working more effectively as a countermeasure. This is something the simulation is not able to consider, and might be the reason for the overestimations. The ML models can potentially learn the effect of such factors, if they are represented in the training data.

The combination model adopted the best traits from both of the approaches from which it is combined. This is believed to be one of the reasons behind superior performance of the combination model.

5.2 Limitations

5.2.1 Model Selection

It's important to note that the research conducted in this paper has its limitations. First and foremost only two different models were compared for each

ML approach, two sequence modeling models and two non-sequence modeling ML models. These were chosen based on their performance according to the literature study. There exist several other models which might have better performance on this specific problem. The linear regression / Elastic Net models also showed some potential in the literature study, but was left out of this thesis. As the hyperparameter optimization of the SVR model deemed the rbf kernel the best one, not the linear kernel, and due to the fact that any Elastic Net can be reduced to a linear SVC, an argument can be made that a linear model would not be better than the two non-linear models implemented in this thesis.

Due to time and computational limitations the experiment conducted had to be limited to four ML models, but to be able to conclude with more certainty that sequence modeling is the superior approach of the two more investigation might be required.

5.2.2 Temporal Dimension

Due to limitation in computing power and time, the sequence lengths of the sequence modeling models was set to a fixed number of 110. This thesis did not investigate the impact sequence length has on model performance. Since the receptive field of the final TCN model was 33, it could be interesting to examine what amount of previous patients actually determine the wait-time of a new patient. With a lower sequence length less information is provided, however, the training time will also be reduced.

Moreover, this thesis implemented a single method of introducing the temporal dimension to the non-sequence modeling models, namely the exponential smoothing method. Other related research identified have not attempted to introduce the temporal dimension at all. There exist other, more sophisticated, methods of doing this, for example Kalman Filters [69]. However, due to time limitations other methods were not explored. This is considered a limitation of the work conducted in this thesis, and the several ways in which the non-sequence modeling models can be improved further are left as future research.

5.2.3 Data

In order to make the data collected useful for the ML models, some cleaning had to be done. This cleaning removed certain patients due to missing or unrealistic data values. Said patients might have been crucial in explaining the

behavior of the wait-time, and their removal can thus have affected the performance of all ML models in a negative way.

The ML models require a lot of data, the more the better. However, due to the pandemic caused by the virus Covid-19 in the beginning of 2020, some data was deemed non usable. The company experienced a disproportionate increase of patients per day compared to earlier years, which meant that no data from patients seeking help past the date of 2020-02-17 could be used. Moreover, data from a long time ago might not reflect the behavior of the queue and wait-time more recently. For that reason and because of a change in the structure of the data, no data from earlier than 2018-04-22 could be used.

The chronological split of the data into training, validation and test sets could be a source of interference for the ML models as well. This splitting caused both Christmas (2019-12-24) and new years (2020-01-01), two significant holidays, to be within the test set. The consumer behavior on said days differs a lot from other regular days. This could have had an impact on the ML models performance on the test set.

Another significant limitation in the data is that it is collected from a single digital healthcare source. The trends in the data, and therefore the results of this thesis, might differ for other digital healthcare providers, or in other markets.

The features used to represent the patients in the data were chosen based on previous research and availability, but also on reproducibility. That means that features that were too specific to the queuing system in place at the company were not used, for instance, clinician and patient labels. However, the work in this thesis does not contain any analysis of said features. It could be interesting to investigate how much each feature contributes to the predictions. Some of the features used may only be unwanted noise. This is something that was not considered due to the time limitation of the study.

5.2.4 Hyperparameter Optimization

The hyperparameter optimization process of the methodology in this thesis was affected by the time and computational limitations quite heavily. Because of the time required to train the LSTM model, only a small part of the training data (40%) could be used during training for each trial. This might have caused the final hyperparameters to have a bias towards this small part of the data which might not represent the rest of the data accurately. Moreover, the models were only trained for 4 epochs for each trial. This was due to the fact that, as presented in subsection 4.1.2, the average number of seconds for a single trial

for the LSTM was 2391. As 100 trials were done, the total time required for the hyperparameter optimization for the LSTM was around 66 hours.

Not only was the amount of data limited but also the actual hyperparameters and their search space as well. The batch size and, as mentioned previously, the sequence length was set to a fixed number even though they might have a considerable impact on the training and performance of the models. The range for all hyperparameters in the search space were also chosen with the time limitations in consideration. This might have affected the result of the hyperparameter optimization in a way that if a larger search space was chosen, another minima might have been found. The current search space might have caused the search to end up in a local minima, when a better global minima could exist. In an attempt to make the hyperparameter optimization as fair as possible between the TCN and LSTM models, the search space was constructed in a way where the maximum possible size of the two models, measured in number of trainable weights, was roughly the same.

The hyperparameters deemed to be the best for the TCN were also used for the TCN in the combination model. Since the combination model has a new input (the simulation output), the best hyperparameters for the regular TCN might not be best in this case. But due to time limitations, the assumption that the hyperparameters would also be a good choice for the TCN in the combination model had to be made.

5.3 Social Aspects and Usability

As mentioned previously, the accuracy of wait-time prediction is especially important in healthcare, since the patients are likely to already be in a worried state of mind when seeking help. An accurate wait-time could calm the patients down. Therefore, the research conducted in this paper will hopefully have a positive societal impact, since it takes steps towards increasing said accuracy. Furthermore, with an accurate wait-time prediction model, steps can potentially be taken to reduce the wait-time when the predictions indicate high values. For instance, to schedule more clinicians. This is left as future research, but could lead to a better and more accessible healthcare.

More precise predictions can be crucial for patients in a critical condition. The patient might weigh the option of calling for an ambulance against the option of digital healthcare, and might not be in a state of mind to make the best decision. If presented with an accurate wait-time prediction the patient can make a better educated decision. On the other hand, the predictions are just predictions. Meaning, the wait-time can be affected by several unforeseen

variables. If a ML model is used and marketed as a model with high accuracy, the patient in a critical condition might place too much trust in the model, and suffer more damages than if he/she would have called the emergency services immediately. This, as well as the Black Box problem mentioned last in section 5.1, should be taken into consideration when using a ML model commercially in healthcare.

Based on the findings in this thesis, the sequence modeling approach using time series forecasting should be considered a viable approach to the wait-time prediction problem. Since the features used to describe the queue were relatively general, the time series forecasting technique is assumed to be viable not only for healthcare queues, but for any type of queue. Therefore, a sequence modeling approach could potentially replace a traditional ML approach applied to any queue, for an improved performance. Additionally, a sequence modeling approach could potentially be combined with any existing simulation algorithm to boost its performance.

5.4 Sustainability and Ethics

The data used in this study could be considered highly sensitive, as it includes personal information about patients. However, the company has anonymized the data before it was accessed in this study. This was done in such a way that any data point could not be tied back to a specific patient. Even though this was the case, the data is still considered sensitive. This has been taken into account, and an effort has been made to not disclose any potentially damaging information, while still striving to have as much transparency as possible in regards to the research conducted.

Worth mentioning is the recent increase in research regarding the energy consumption of deep learning models. The training process for deep learning models can be costly, both in regards to hardware and computational power, but also in regards to the environment [70]. The primary focus in ML is usually the accuracy and performance of the model, but recent research raises the importance of environmental sustainability, and having energy consumption as an evaluation metric as well [71].

5.5 Future Research

As mentioned in subsection 3.1.2, the sequence modeling models implicitly require the data to be uniformly sampled. The simplest solution to this problem

was applied in this thesis, appending the time differences between each patient Δt to the input vector. However, research has shown that there are other, better, solutions to this problem. More specifically [72] introduced the Phased-LSTM which extends the regular LSTM by adding a new time gate, which results in a greatly improved performance in standard RNN applications. Very recently, similar research has been done for the TCN in [57], which introduces a Continuous Convolutional Neural Network for non-uniform time series showing promising results. Future research is needed in order to investigate whether the models mentioned above could further improve the performance of the sequence modeling approach on the wait-time prediction problem.

The behavior and trends in the wait-time changes with time. As mentioned in subsection 5.2.3, data from a long time ago might not contribute as much to the performance of the ML models as more recent. It might just add unwanted noise. Future research is required to evaluate if any attempt to mitigate this problem will provide an increase in performance for the ML models. For instance, one could use sample weights in order to weight the data from more recent patients more heavily so that they contribute more during training. Furthermore, investigating other, more representative, evaluation techniques for the ML models could be of interest. Splitting the data in chronological order will cause the ML models to be evaluated on the latest trend which might not be included in the training data. In order to alleviate this affect one could, for instance, evaluate the ML models on one week at a time, allowing them to train on the data from the previous week before evaluating the next.

Chapter 6

Conclusion

To summarize, the result of the comparison between a sequence modeling approach, a non-sequence modeling ML approach utilizing exponential smoothing, and a simulation algorithm, was that the sequence modeling TCN model and the simulation algorithm had almost equal performances on wait-time prediction in digital healthcare. The two models had an MSE around 56, and they had their respective advantages and disadvantages. When combined together, the best performing model was attained. The combination model adopted the advantages from both approaches. The non-sequence modeling approach utilizing exponential smoothing had a considerably worse performance compared to the other approaches, but excelled in other aspects such as interpretability. No single approach could be concluded as optimal, due to the limitations in the conducted research. However, it can be concluded that the sequence modeling approach is a viable option, and it should therefore be considered in future wait-time prediction research.

Several other suggestions for future research are proposed. For instance, investigating whether alternatives to exponential smoothing are better at introducing the temporal dimension to the non-sequence modeling ML approach. Alternatively, investigating whether extensions of the sequence modeling models, designed specifically for non-uniformly sampled data, will have increased performance.

Bibliography

[1] D. H. Maister *et al.*, *The psychology of waiting lines*. Harvard Business School Boston, MA, 1984.

[2] M. K. Brady and J. J. Cronin Jr, "Some new thoughts on conceptualizing perceived service quality: A hierarchical approach", *Journal of marketing*, vol. 65, no. 3, pp. 34–49, 2001.

[3] R. Davis, T. Rogers, and Y. Huang, "A survey of recent developments in queue wait time forecasting methods", in *Proceedings of the International Conference on Foundations of Computer Science (FCS)*, The Steering Committee of The World Congress in Computer Science, Computer ..., 2016, p. 84.

[4] A. Komashie, A. Mousavi, P. J. Clarkson, and T. Young, "An integrated model of patient and staff satisfaction using queuing theory", *IEEE Journal of Translational Engineering in Health and Medicine*, vol. 3, pp. 1–10, 2015, ISSN: 2168-2372. DOI: 10.1109/JTEHM.2015.2400436.

[5] A. Joseph, T. Hijal, J. Kildea, L. Hendren, and D. Herrera, "Predicting waiting times in radiation oncology using machine learning", eng, in *2017 16th IEEE International Conference on Machine Learning and Applications (ICMLA)*, vol. 2018-, IEEE, 2017, pp. 1024–1029, ISBN: 9781538614174.

[6] *Kry*, https://www.kry.se/, Accessed: 2020-03-30.

[7] G. Arha, "Reducing wait time prediction in hospital emergency room: Lean analysis using a random forest model", Master's thesis, University of Tennessee, Knoxville, 2017.

[8] M. Kembe, E. Onah, and S. Iorkegh, "A study of waiting and service costs of a multi-server queuing model in a specialist hospital", *International Journal of Scientific & Technology Research*, vol. 1, no. 8, pp. 19–23, 2012.

[9] C. Curtis, C. Liu, T. J. Bollerman, and O. S. Pianykh, "Machine learning for predicting patient wait times and appointment delays", *Journal of the American College of Radiology*, vol. 15, no. 9, pp. 1310–1316, 2018.

[10] A. Kyritsis and M. Deriaz, "A machine learning approach to waiting time prediction in queueing scenarios", Sep. 2019, pp. 17–21. DOI: 10.1109/AI4I46381.2019.00013.

[11] "Wait-time predictors for customer service systems with time-varying demand and capacity", eng, *Operations Research*, vol. 59, no. 5, pp. 1106–1118, 2011, ISSN: 0030-364X.

[12] R. Mourao, R. Carvalho, R. Carvalho, and G. Ramos, "Predicting waiting time overflow on bank teller queues", eng, in *2017 16th IEEE International Conference on Machine Learning and Applications (ICMLA)*, vol. 2018-, IEEE, 2017, pp. 842–847, ISBN: 9781538614174.

[13] Y. Sanit-in and K. R. Saikaew, "Prediction of waiting time in one-stop service", *International Journal of Machine Learning and Computing*, vol. 9, no. 3, 2019.

[14] L. Breiman, "Random forests", eng, *Machine Learning*, vol. 45, no. 1, pp. 5–32, 2001, ISSN: 0885-6125.

[15] S. Bai, J. Z. Kolter, and V. Koltun, "An empirical evaluation of generic convolutional and recurrent networks for sequence modeling", *CoRR*, vol. abs/1803.01271, 2018. arXiv: 1803.01271. [Online]. Available: http://arxiv.org/abs/1803.01271.

[16] S. Hochreiter and J. Schmidhuber, "Long short-term memory", eng, *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997, ISSN: 0899-7667.

[17] Y. Zhang, L. T. Nguyen, and J. Zhang, "Wait time prediction: How to avoid waiting in lines?", in *Proceedings of the 2013 ACM conference on Pervasive and ubiquitous computing adjunct publication*, 2013, pp. 481–490.

[18] R. G. Brown, *Statistical forecasting for inventory control*. McGraw-Hill, 1959. [Online]. Available: http://hdl.handle.net/2027/mdp.39015001311771.

[19] J. Gehring, M. Auli, D. Grangier, D. Yarats, and Y. N. Dauphin, "Convolutional sequence to sequence learning", *CoRR*, vol. abs/1705.03122, 2017. arXiv: 1705.03122. [Online]. Available: http://arxiv.org/abs/1705.03122.

[20] J. Berglind, *Temporal convolutional networks for forecasting patient volumes in digital healthcare*, eng, KTH, Skolan för elektroteknik och datavetenskap (EECS), 2019.

[21] R. Wan, S. Mei, J. Wang, M. Liu, and F. Yang, "Multivariate temporal convolutional network: A deep neural networks approach for multivariate time series forecasting", eng, *Electronics*, vol. 8, no. 8, p. 876, 2019, ISSN: 2079-9292. [Online]. Available: https://doaj.org/article/5d25208427d54aa1ba99c368c8f6d6e1.

[22] M. Pickup, *Introduction to time series analysis*, eng, ser. Quantitative applications Introduction to time series analysis. 2016, ISBN: 1-4833-9085-3.

[23] S. Siami-Namini and A. S. Namin, "Forecasting economics and financial time series: ARIMA vs. LSTM", *CoRR*, vol. abs/1803.06386, 2018. arXiv: 1803.06386. [Online]. Available: http://arxiv.org/abs/1803.06386.

[24] R. Sen, H.-F. Yu, and I. Dhillon, "Think globally, act locally: A deep neural network approach to high-dimensional time series forecasting", *arXiv preprint arXiv:1905.03806*, 2019.

[25] S. d. O. Domingos, J. F. de Oliveira, and P. S. de Mattos Neto, "An intelligent hybridization of arima with machine learning models for time series forecasting", *Knowledge-Based Systems*, vol. 175, pp. 72–86, 2019.

[26] A. Rubio, J. Bermúdez, and E. Vercher, "Improving stock index forecasts by using a new weighted fuzzy-trend time series method", eng, *Expert Systems with Applications*, vol. 76, p. 12, 2017, ISSN: 0957-4174. [Online]. Available: http://search.proquest.com/docview/1932176199/.

[27] C. Ng and B. Soong, "Single-queue markovian systems", eng, in *Queueing Modelling Fundamentals*, Chichester, UK: John Wiley Sons, Ltd, 2008, pp. 103–140, ISBN: 9780470519578.

[28] G.-S. Hu and F.-Q. Deng, "The analysis of queuing system based on support vector machine", eng, in *ICARCV 2004 8th Control, Automation, Robotics and Vision Conference, 2004*, vol. 3, IEEE, 2004, 2320–2325 Vol. 3, ISBN: 0780386531.

[29] R. G. Brown, *Exponential smoothing for predicting demand*, INST OPERATIONS RESEARCH MANAGEMENT SCIENCES, 1957. [Online]. Available: https://www.industrydocuments.ucsf.edu/docs/jzlc0130.

[30] A. Azadeh, M. Sheikhalishahi, M. Tabesh, and A. Negahban, "The effects of pre-processing methods on forecasting improvement of artificial neural networks", *Australian Journal of Basic and Applied Sciences*, vol. 5, no. 6, pp. 570–580, 2011.

[31] R. Hyndman, *Forecasting with Exponential Smoothing The State Space Approach*, eng, 1st ed. 2008.., ser. Springer Series in Statistics. 2008, ISBN: 1-281-49162-4.

[32] W. McCulloch and W. Pitts, "A logical calculus of the ideas immanent in nervous activity", eng, *The bulletin of mathematical biophysics*, vol. 5, no. 4, pp. 115–133, 1943, ISSN: 0007-4985.

[33] R. Rojas, *Neural networks: a systematic introduction*. Springer Science & Business Media, 2013.

[34] F. Rosenblatt, "The perceptron: A probabilistic model for information storage and organization in the brain.", *Psychological review*, vol. 65, no. 6, p. 386, 1958.

[35] H. J. Kelley, "Gradient theory of optimal flight paths", *Ars Journal*, vol. 30, no. 10, pp. 947–954, 1960.

[36] P. Werbos, "Beyond regression: New tools for prediction and analysis in the behavioral science. thesis (ph. d.). appl. math. harvard university", PhD thesis, Jan. 1974.

[37] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors", *Nature*, vol. 323, no. 6088, p. 533, 1986, ISSN: 0028-0836.

[38] Y. LeCun *et al.*, "Generalization and network design strategies", *Connectionism in perspective*, vol. 19, pp. 143–155, 1989.

[39] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, `http://www.deeplearningbook.org`.

[40] A. v. d. Oord, S. Dieleman, H. Zen, *et al.*, "Wavenet: A generative model for raw audio", *arXiv preprint arXiv:1609.03499*, 2016.

[41] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition", *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun. 2016. DOI: `10.1109/cvpr.2016.90`. [Online]. Available: `http://dx.doi.org/10.1109/CVPR.2016.90`.

[42] T. Salimans and D. P. Kingma, *Weight normalization: A simple reparameterization to accelerate training of deep neural networks*, 2016. arXiv: 1602.07868 [cs.LG].

[43] V. Nair and G. E. Hinton, "Rectified linear units improve restricted boltzmann machines", in *Proceedings of the 27th international conference on machine learning (ICML-10)*, 2010, pp. 807–814.

[44] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting", *The journal of machine learning research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[45] Z. C. Lipton, "A critical review of recurrent neural networks for sequence learning", *CoRR*, vol. abs/1506.00019, 2015. arXiv: 1506.00019. [Online]. Available: http://arxiv.org/abs/1506.00019.

[46] L. Breiman, J. Friedman, C. J. Stone, and R. A. Olshen, *Classification and regression trees*. CRC press, 1984.

[47] G. James, D. Witten, T. Hastie, and R. Tibshirani, *An introduction to statistical learning*. Springer, 2013, vol. 112.

[48] F. Pedregosa, G. Varoquaux, A. Gramfort, *et al.*, "Scikit-learn: Machine learning in Python", *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[49] Q. Zhou, W. Chen, S. Song, J. R. Gardner, K. Q. Weinberger, and Y. Chen, "A reduction of the elastic net to support vector machines with an application to gpu computing", *ArXiv*, vol. abs/1409.1976, 2015.

[50] H. Zou and T. Hastie, "Regularization and variable selection via the elastic net", *Journal of the royal statistical society: series B (statistical methodology)*, vol. 67, no. 2, pp. 301–320, 2005.

[51] I. Steinwart, *Support Vector Machines*, eng, 1st ed. 2008.., ser. Information science and statistics. 2008, ISBN: 1-281-92704-X.

[52] C. Cortes and V. Vapnik, "Support-vector networks", eng, *Machine Learning*, vol. 20, no. 3, pp. 273–297, 1995, ISSN: 0885-6125.

[53] A. Smola and B. Schölkopf, "A tutorial on support vector regression", eng, *Statistics and Computing*, vol. 14, no. 3, pp. 199–222, 2004, ISSN: 0960-3174.

[54] F.-K. Wang and T. Mamo, "A hybrid model based on support vector regression and differential evolution for remaining useful lifetime prediction of lithium-ion batteries", eng, *Journal of Power Sources*, vol. 401, pp. 49–54, 2018, ISSN: 0378-7753.

[55] D. Worthington, "Queueing models for hospital waiting lists", *Journal of the Operational Research Society*, vol. 38, no. 5, pp. 413–422, 1987.

[56] R. A. Nosek and J. P. Wilson, "Queuing theory and customer satisfaction: A review of terminology, trends, and applications to pharmacy practice", eng, *Hospital Pharmacy*, vol. 36, no. 3, pp. 275–279, 2001, ISSN: 0018-5787.

[57] H. Shi, Y. Zhang, H. Wu, S. Chang, K. Qian, M. Hasegawa-Johnson, and J. Zhao, *Continuous convolutional neural network fornonuniform time series*, 2020. [Online]. Available: https://openreview.net/forum?id=r1e4MkSFDr.

[58] Martın Abadi, Ashish Agarwal, Paul Barham, *et al.*, *TensorFlow: Large-scale machine learning on heterogeneous systems*, Software available from tensorflow.org, 2015. [Online]. Available: https://www.tensorflow.org/.

[59] F. Chollet *et al.*, *Keras*, https://keras.io, 2015.

[60] P. Rémy, *Keras tcn*, https://github.com/philipperemy/keras-tcn, 2020.

[61] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift", *arXiv preprint arXiv:1502.03167*, 2015.

[62] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization", *arXiv preprint arXiv:1412.6980*, 2014.

[63] *Rapids*, https://rapids.ai, 2020.

[64] J. Bergstra, D. Yamins, and D. D. Cox, "Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures", eng, JMLR, 2013, ISBN: 1938-7228.

[65] J. S. Bergstra, R. Bardenet, Y. Bengio, and B. Kégl, "Algorithms for hyper-parameter optimization", in *Advances in Neural Information Processing Systems 24*, J. Shawe-Taylor, R. S. Zemel, P. L. Bartlett, F. Pereira, and K. Q. Weinberger, Eds., Curran Associates, Inc., 2011, pp. 2546–2554. [Online]. Available: http://papers.nips.

cc/paper/4443-algorithms-for-hyper-parameter-optimization.pdf.

[66] M. Friedman, "The use of ranks to avoid the assumption of normality implicit in the analysis of variance", *Journal of the american statistical association*, vol. 32, no. 200, pp. 675–701, 1937.

[67] P. Virtanen, R. Gommers, T. E. Oliphant, *et al.*, "SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python", *Nature Methods*, vol. 17, pp. 261–272, 2020. DOI: https://doi.org/10.1038/s41592-019-0686-2.

[68] F. Wilcoxon, "Individual comparisons by ranking methods", eng, *Biometrics Bulletin*, vol. 1, no. 6, pp. 80–83, 1945, ISSN: 00994987.

[69] R. E. Kalman, "A New Approach to Linear Filtering and Prediction Problems", *Journal of Basic Engineering*, vol. 82, no. 1, pp. 35–45, Mar. 1960, ISSN: 0021-9223. DOI: 10.1115/1.3662552. [Online]. Available: https://doi.org/10.1115/1.3662552.

[70] E. Strubell, A. Ganesh, and A. McCallum, "Energy and policy considerations for deep learning in NLP", *CoRR*, vol. abs/1906.02243, 2019. arXiv: 1906.02243. [Online]. Available: http://arxiv.org/abs/1906.02243.

[71] E. Garcıa-Martın, C. F. Rodrigues, G. Riley, and H. Grahn, "Estimation of energy consumption in machine learning", *Journal of Parallel and Distributed Computing*, vol. 134, pp. 75–88, 2019.

[72] D. Neil, M. Pfeiffer, and S.-C. Liu, "Phased lstm: Accelerating recurrent network training for long or event-based sequences", in *Advances in neural information processing systems*, 2016, pp. 3882–3890.

Appendix A

Forecasts

This section presents the predicted and actual wait-time for all models for 4 randomly selected days in the test-data.

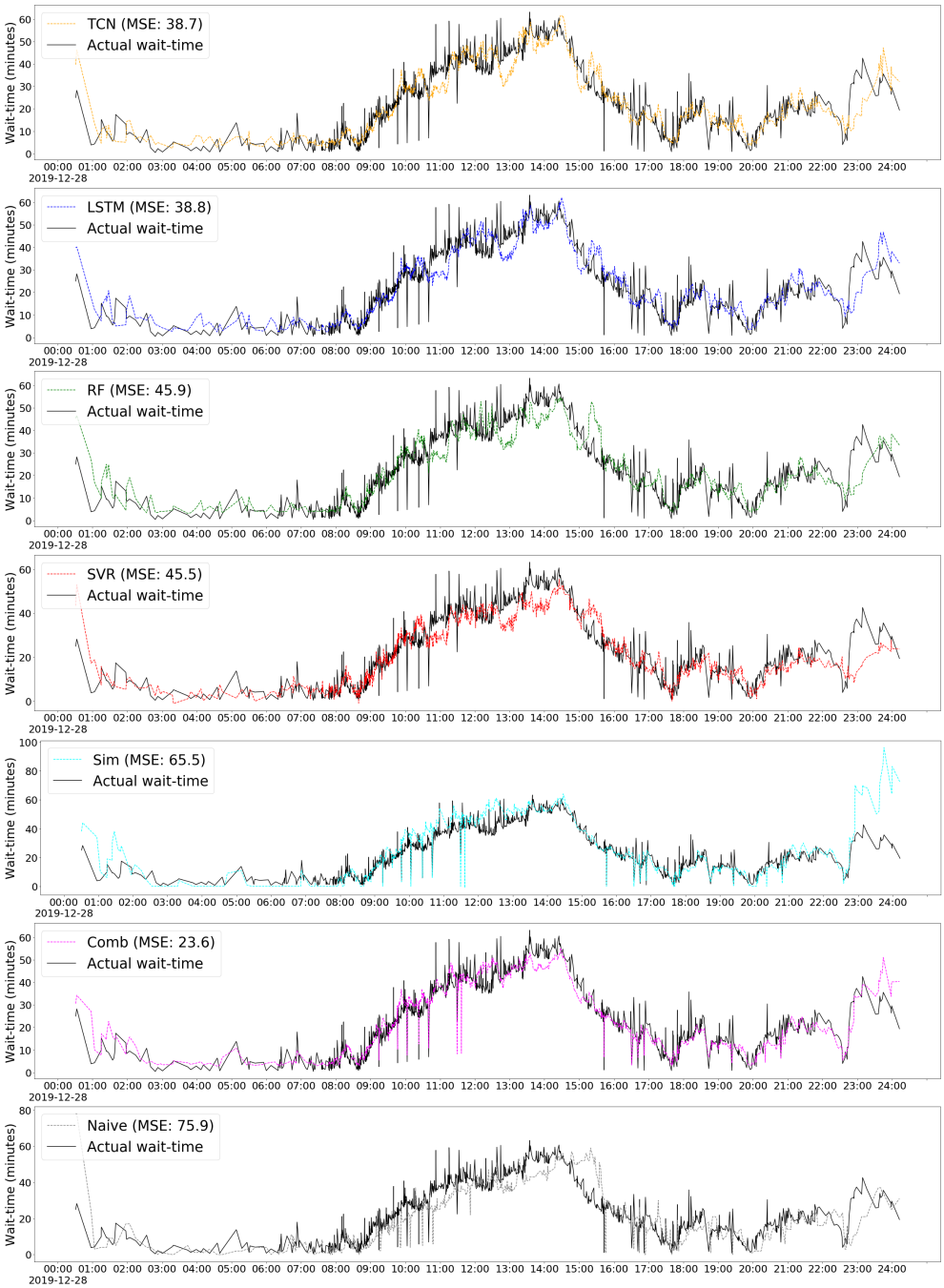


Figure A.1: The actual wait-time and the predicted wait-time for each model for single day in the test data (2019-12-28).

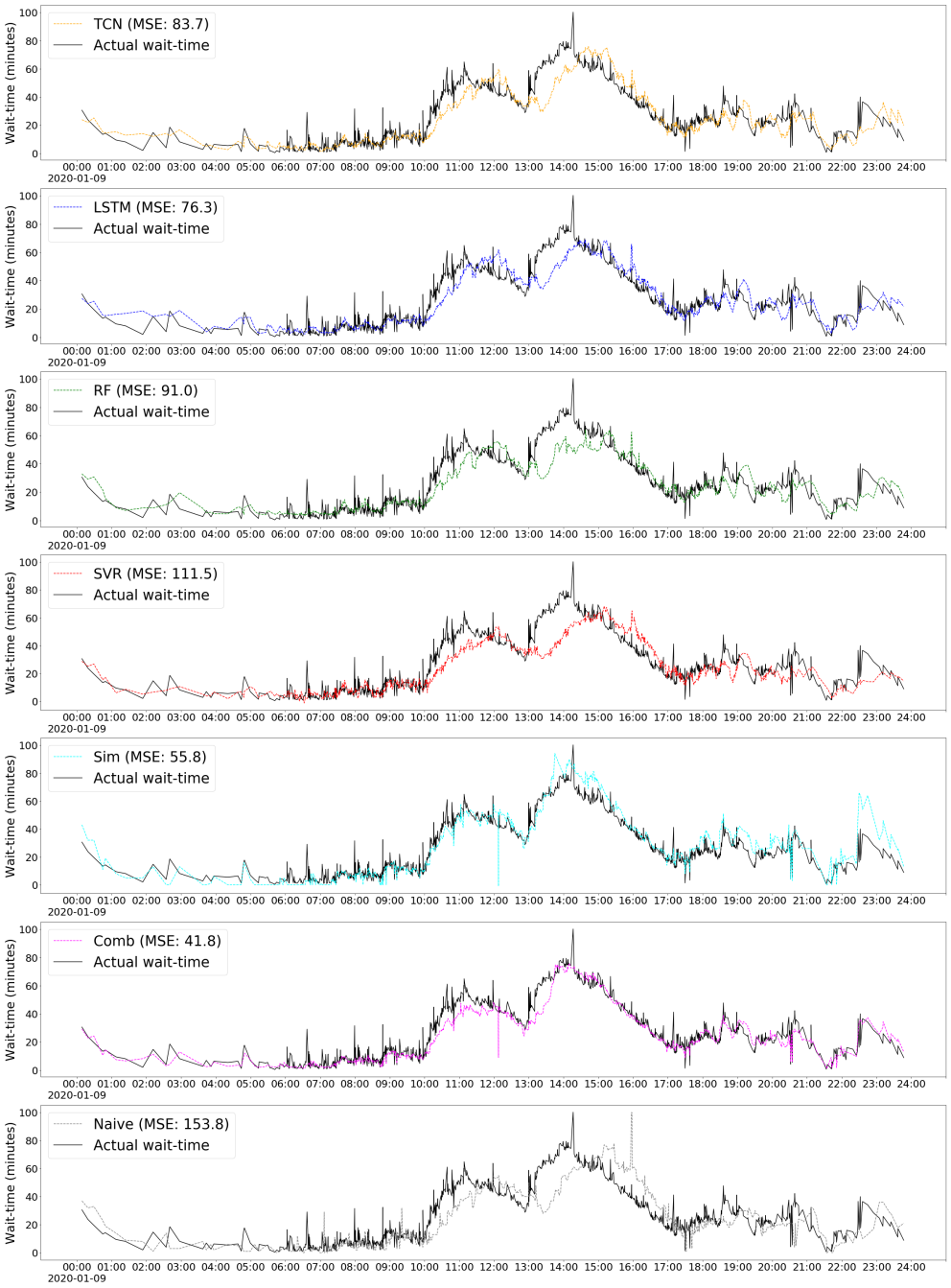


Figure A.2: The actual wait-time and the predicted wait-time for each model for single day in the test data (2020-01-09).

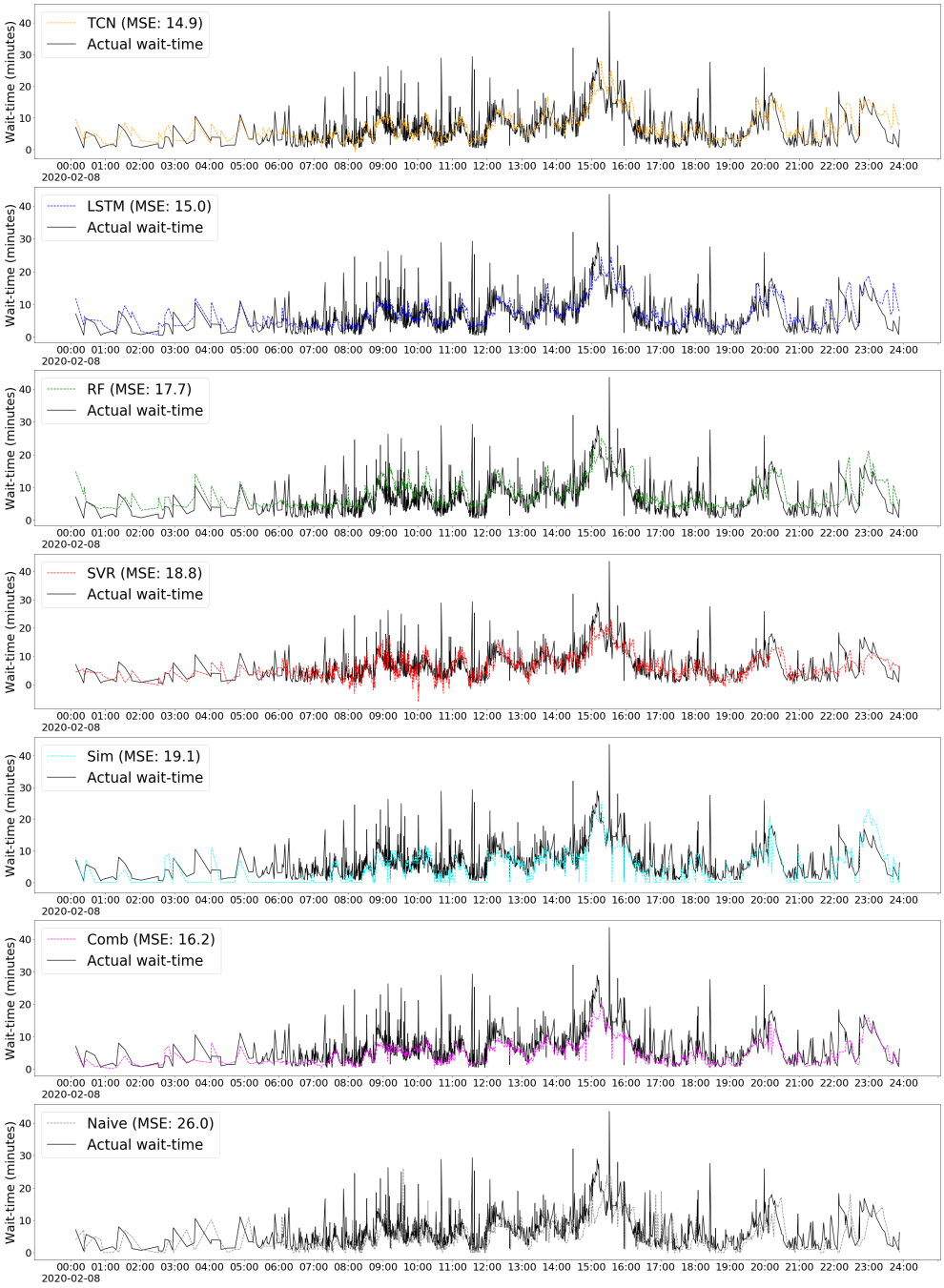


Figure A.3: The actual wait-time and the predicted wait-time for each model for single day in the test data (2020-02-08).

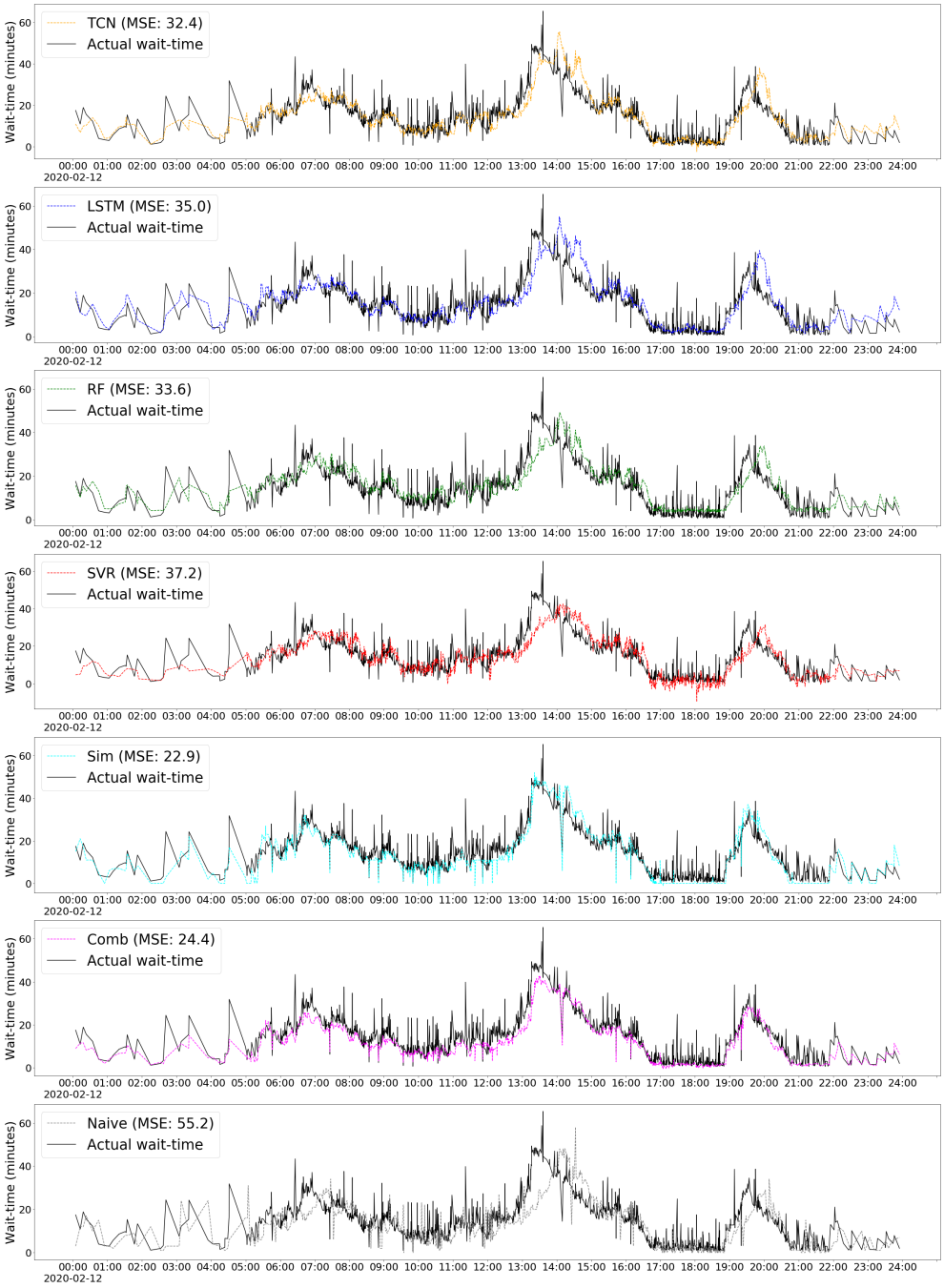


Figure A.4: The actual wait-time and the predicted wait-time for each model for single day in the test data (2020-02-10).

TRITA EECS-EX-2020:241