Dynamic algorithm selection for machine learning on time series

Love Dahlberg
Dynamic algorithm selection for machine learning on time series

Love Dahlberg
This report is submitted in partial fulfillment of the requirements for the Bachelor’s degree in Computer Science. All material in this report which is not my own work has been identified and no material is included for which a degree has previously been conferred.

____________________________

Love Dahlberg

Approved, 2019-06-04

____________________________

Advisor: Tobias Pulls

____________________________

Examiner: Stefan Alfredsson
Abstract

We present a software that can dynamically determine what machine learning algorithm is best to use in a certain situation given predefined traits. The produced software uses ideal conditions to exemplify how such a solution could function. The software is designed to train a selection algorithm that can predict the behavior of the specified testing algorithms to derive which among them is the best. The software is used to summarize and evaluate a collection of selection algorithm predictions to determine which testing algorithm was the best during that entire period. The goal of this project is to provide a prediction evaluation software solution can lead towards a realistic implementation.
# Contents

Abstract

1 Introduction .................................................................................................................. 1  
1.1 Overview .................................................................................................................. 1  
1.2 Disposition ............................................................................................................... 2  

2 Background .................................................................................................................. 4  
2.1 Machine learning ..................................................................................................... 4  
  2.1.1 Training and testing of the algorithms  
  2.1.2 Relevant algorithms  
2.2 Data and statistics ................................................................................................... 6  
2.3 Tools ......................................................................................................................... 7  
2.4 Summary .................................................................................................................... 8  

3 Design .......................................................................................................................... 9  
3.1 Basic concept .......................................................................................................... 9  
  3.1.1 Naive design  
  3.1.2 The dataset and time series  
3.2 Training .................................................................................................................. 12  
  3.2.1 Teaching the selection algorithm  
  3.2.2 The time-window  
3.3 Testing .................................................................................................................... 15  
3.4 Summary ................................................................................................................... 16  

4 Implementation .......................................................................................................... 17  
4.1 Variables explanation ............................................................................................. 17  
4.2 The main body ........................................................................................................ 18  
4.3 Initializing the testing algorithms ............................................................................ 19  
4.4 The selection algorithm function ............................................................................ 21  
  4.4.1 Training  
  4.4.2 Testing  
4.5 Summary ................................................................................................................ 23
5 Results ................................................................................................................................. 24
  5.1 How to analyze the data................................................................................................. 24
  5.2 Evaluation ................................................................................................................... 25
  5.3 Summary ..................................................................................................................... 30

6 Discussion .......................................................................................................................... 31
  6.1 Result discussion ......................................................................................................... 31
  6.2 Feedback loop ............................................................................................................. 33
    6.2.1 Feedback delay
    6.2.2 Possible solutions
  6.3 Condition parameter ................................................................................................... 35
    6.3.1 Time-window
    6.3.2 Advantageous metrics
    6.3.3 Feedback loop
  6.4 Algorithm and dataset choice .................................................................................... 37
  6.5 Summary ..................................................................................................................... 38

7 Conclusion .......................................................................................................................... 39
  7.1 Future work ................................................................................................................ 40
  7.2 Concluding remarks ................................................................................................... 41

References ............................................................................................................................. 42
List of Figures

Figure 1: The basic machine learning principle: training and testing a number of algorithms. $A = \{A_1, A_2 \ldots A_n\}$ is the algorithms set. $D_1X$ is used as the features and $D_1y$ is the value to be predicted in the training phase. Testing phase: $D_2X$ is used as the features, $R$ is the prediction, $D_2X$ is the ground truth and $PX$ is the score. The resulting prediction in the testing phase is compared to the ground truth. The process would give a list of mean square error scores upon which one algorithm can be selected to be the best.

Figure 2: The sliding time-window. The window “slides” or “shifts” a set interval each prediction.

Figure 3: The basic concept of algorithm selection. $A = \{A_1, A_2 \ldots A_n\}$ is the algorithms set. $DX$ is used as the features and $R$ is the resulting prediction. The machine $M$ is fed predictions to eventually produce a machine prediction $PX$.

Figure 4: Machine training. $A = \{A_1, A_2 \ldots A_n\}$ is the algorithms set. $DX$ is used as the features and $R$ is the resulting prediction. The figure shows how the model will work by modifying the picture shown in Figure 3. The figure expands the machine $M$ and separates it into the different modules $m$ and $C$. $Dy$ is the ground truth and $V$ is the value to be predicted by the machine algorithm.

Figure 5: The machine-window and algorithm-window. Each window “slides” or “shifts” a set interval each prediction.

Figure 6: Machine prediction. $A = \{A_1, A_2 \ldots A_n\}$ is the algorithms set. $DX$ is used as the features and $R$ is the resulting prediction. The figure shows the module $m$ producing the prediction based on the training received by the data in machine-window. The prediction is outputted from the machine and stored for the evaluation phase together with respective ground truth. $Dy$ is the ground truth and $PX$ is the machine prediction.

Figure 7: Code illustrating the initialization step of the main body of the program.

Figure 8: Code illustrating how the algorithms are represented.
Figure 9: Code illustrating the execution step of the main body of the program.

Figure 10: Code illustrating the main body of the function init_algorithms().

Figure 11: Code illustrating the algorithm training and testing function algorithm_generator(). This function is used by init_algorithms().

Figure 12: Code illustrating the first part of the selection algorithm function machine().

The figure shows the training phase of the function.

Figure 13: Code illustrating the second part of the selection algorithm function machine().

The figure shows the prediction phase of the function.

Figure 14: Code illustrating the last part of the selection algorithm function machine().

The figure shows the values that are stored in the results frame.

Figure 15: An example of the data stored by the machine.

Figure 16: The Beijing dataset between Jan 1st, 2010 to Dec 31st, 2015. The temperature is in Celsius and is compared to time. The time value in the x-axis is interpreted as for example: 1.275 * 10^9 → 22:40, 27/5-2010 using UNIX time.

Figure 17: The numeric result produced using the Beijing dataset. The first two lines are statistics regarding the selection algorithm. The following two segments are statistics regarding the testing algorithms compared to the selection algorithm and the ground truth respectively.

Figure 18: Graphs showing the difference between the prediction and the ground truth. The temperature is in Celsius and is compared to time. The time value in the x-axis is interpreted as for example: 1.275 * 10^9 → 22:40, 27/5-2010 using UNIX time.

Figure 19: Graphs displaying the difference between each testing algorithm compared to the machine prediction. The temperature is in Celsius and is compared to time. The time value in the x-axis is interpreted as for example: 1.275 * 10^9 → 22:40, 27/5-2010 using UNIX time.

Figure 20: Graphs displaying the difference between each testing algorithm compared to the ground truth. The temperature is in Celsius and is compared to time. The time value in the x-axis is interpreted as for example: 1.275 * 10^9 → 22:40, 27/5-2010 using UNIX time.
1 Introduction

Machine learning is a concept that have received a lot of attention the last couple of years [13]. This rise in its popularity is mainly because of the enormous amount of computing power being available in today’s world and the large investments being made by companies such as Google, Amazon, Facebook and Microsoft. Machine learning has applications such as image/voice recognition, personal phone assistances, chatbots, weather forecasts, game AI such as AlphaGo [2] and future prospects such as human-like robots and self-driving cars. Although these applications use and implements the same broad concepts, the processes under the hood are different. There are a couple of different ways a machine can learn and a lot of different ways to do it. Choosing which method to use in what situation is a tedious task. Not only is it time-consuming, it can also be very hard. Machine learning applications base their behaviors on everchanging datasets, meaning that different methods or algorithms used within can be good at different times. This project will present a solution to this issue called the algorithm selection problem. The goal of the project is to design and implement an ideal and partly theoretical algorithm selection model to exemplify how a realistic model implementation could be evaluated.

The project takes place at the analytics department of the consulting company CGI in Sweden Karlstad City. CGI is a company that uses a lot of machine learning and artificial intelligence in their production with customers. By solving the algorithm selection problem CGI would be able to predict the current and future behavior of their products to their customers. This project is going to create a limited evaluation model using ideal conditions. Because of some assumptions made in this project, the model created should not be interpreted as a blueprint for a realistic implementation. We hope that CGI will consider the concepts discussed in this project as ground work for future research.

1.1 Overview

The model designed and implemented in this project uses a selection algorithm to rank which among the testing algorithms are the best at given points in time. The algorithm chosen for the selection was linear regression because of its general efficiency and consistency when used for
algorithm selection [6]. The algorithms chosen for testing were random forest regressor, decision tree regressor and KNeighbor regression.

The selection algorithm is trained to predict the value of the best testing algorithm based on a number of previous predictions. The prediction produced by the selection algorithm is compared to the predictions produced by the testing algorithms. The testing algorithm with the lowest mean square error deviation from the selection algorithm prediction is chosen as being the best at that point in time. If this process is repeated over a large dataset, we can summarize the choices made by the selection algorithm and compare it to reality or also called the ground truth.

The results produced in this project uses meteorology data from Beijing between Jan 1st, 2010 to Dec 31st, 2015 where the value to be predicted was the temperature in Celsius based on a time-stamp. We predicted a total of 52,339 rows from the dataset, where one row represents one hour. The result showed the selection algorithm prediction mean square error deviation being 1.0 °C. The model chose the correct test algorithm 63.2% of the times, called the prediction accuracy score. The algorithm chosen as the best was the decision tree regressor. It was picked 66% of the times and had a mean squared error deviation of 0.82 °C. The runner ups were random forest regressor, picked 27.5% of the times with 0.88 °C deviation, and KNeighbor regression which was picked 6.46% of the times with 7.91 °C deviation.

We are satisfied with everything in this result except the comparably low prediction accuracy score. We expected the prediction accuracy score to be higher, maybe around 70-80% when accompanied by a 1.0 °C selection algorithm prediction mean square error deviation.

1.2 Disposition

Chapter 2 introduces a couple basic concepts in machine learning and data analysis and presents some useful implementation tools. The topics explored in this chapter should be enough for a beginner to appreciate the project. We present the design of the model in Chapter 3. The chapter introduces the algorithm selection method, explains the training and testing phase of the selection algorithm and shows how to handle the dataset in a useful way. The implemented code of the model is presented in Chapter 4. The chapter explains and motivates the use of the two main functions init_algorithms() and machine(). Chapter 5 introduces ways of analyzing
the data and displays the results produced by applying the dataset to the model. The problems that arise when going from an ideal to a realistic model is discussed in Chapter 6. Lastly, Chapter 7 concludes the project by summarizing the content of Chapter 5 and Chapter 6 and discusses the prospects of future work.
2 Background

This chapter presents the background knowledge necessary for understanding the machine learning theory and the data manipulation used in this project. Section 2.1 presents the concept of machine learning and explains basic concepts such as training and testing. Section 2.2 presents ways of handling and evaluating the data. Section 2.3 introduces the software and tools necessary for the implementation of this project.

2.1 Machine learning

Machine learning is a broad field that can be described in many ways, one of which is the following:

"A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E." [3]

Machine learning is the process of imitation based on previous events. Machine learning can be used to teach a machine to act in a certain way by being fed data and information. The machine will continue to learn and improve over time without the need of human correction or rule based programing [3]. The models inside of a machine learning process creates and draws its own patterns and conclusions from the information it has been fed. The method by which the model learns is dependent on which model is used. This section will describe different way this process can work and introduce necessary concepts such as supervised learning, regression, classification, training and testing.

2.1.1 Training and testing of the algorithms

An algorithm is a set of rules typically defining how something should be calculated or processed. A machine learning algorithm is a set of rules describing how the computer should interpret, categorize and filter data to fulfill its ultimate purpose. There are several categories that separate different machine learning algorithms dependent on learning methodology, the dataset and area of application. This project will focus solely on algorithms in the supervised learning category using the regression models.
The machine learning algorithm needs to be taught what to predict. The method used in this project is called Supervised learning. Supervised learning is a method of learning by imitation [19]. The learning process can be viewed as if a conceptual teacher is supervising the algorithm. This learning process is called the training phase. The teacher gives the algorithm a subset of the data and the correct answers, or the value to be predicted that go with it. The algorithm takes the data and generates the features, also called X, from it. The algorithm is taught, while being observed by the supervisor, what features should result in what prediction. The prediction can also be called score or y. The variable y can also be called the dependent variable, a variable that depends on the variable X that is called the independent variable [15].

When the algorithm reaches an acceptable level of performance, the teacher stops the leaning process. The algorithm can now enter the testing phase. The algorithm will now, unsupervised, take a new subset of the same data and predict the correct answer. The correct answer is also called the ground truth.

The training and testing phase are different subsets of data of the same dataset. A normal data-split approach in supervised learning is usually between 70-90% of the original dataset for the training set and between 10-30% for the testing set. Note that these percentages usually consists of data points evenly split over the entire set. The training data is generally allocated noticeable more data compared to the testing data, since more training for the algorithm usually result in higher accuracy.

Supervised learning can be implemented in namely two sub-categories: classification and regression models. Classification sorts its predictions into distinct and discrete classes. An example of classification is the task of categorizing email into “spam” and “not spam” with the content and sender of the email as features [8]. Regression predicts continuous values or integer quantities. An example of regression is the task of predicting the price of a house with the size and location of the house as features [8]. Regression and classification are different methods but can be used interchangeably in some situations. An example of this is the use of a regression algorithm to predict a discrete value. The discrete value can be derived from the continuous valued prediction generated by the regression algorithm [8].

2.1.2 Relevant algorithms

The following algorithms are used during the project.
• Linear regression. It uses the dependent variable and one or more independent variables from a given datasets to create a linear function. The linear function is used to predict the dependent variable as a function of the independent variables. The model is called multiple variable linear regression if there is more than one independent variable [16].

• Decision tree regressor. It uses a tree model where the observation about the independent values are the branches and the conclusions or the dependent variable is in the leaf nodes [14].

• Random forest regressor. It creates multiple decision trees and outputs the mean prediction from all the trees [18].

• KNeighbor regression. It stores all available cases and uses a similarity function to determine how closely the testing independent variable resemble the stored training independent variable to do a prediction [1].

2.2 Data and statistics

This section is briefly going to discuss how to properly use the dataset and a useful way of analyzing the test data.

The data usage is arguably the most important part of machine learning. The machine learning model is worthless if the data is not processed and handled properly. There are plenty of different types of datasets that can be used for this purpose and the one chosen in this project is in a time series. A time series is a dataset with its data points indexed in time order [4]. When used in machine learning the time is used as one of the features to for a prediction. Analyzing a time series can give wisdom over how a given asset or variable changes over time or within seasons [4].

One important aspect of time series usage is to preserve the underlying time dependency, how the datapoints depend or associate to each other. If a prediction is to be made for a season the training phase of the algorithm cannot use a subset of data from a different time period compared to the testing phase. When predicting an upcoming season, the subset used in the training phase should, for example, be an adjacent season or a similar season further back in time. There are no set rules that has to be followed, but the performance will generally go up if the datapoint dependency is preserved.
After the algorithm has gone through the training and the testing phase it is time to enter the evaluation phase. The evaluation phase uses the mean squared error approach to calculate the accuracy of the prediction from the testing phase. The mean squared error approach calculates how much the algorithm prediction deviates from the ground truth. It is done by measuring the average squared difference between its parameters [17]. It produces a non-negative measure of quality, with a value close to 0 as optimal [17]. Since this method uses the average of squared distances, predictions with extreme single outliers will be easier to notice compared to the results created using a non-squared average. This is because larger squared distances grows notably faster than larger non-squared distances. Another advantage of the squared approach is the elimination of the sign. The non-squared average of two points that are equally far apart from the origin, but in difference directions, will be zero. The squared approach will on the other hand eliminate the sign by squaring the distances, making the deviation noticeable.

2.3 Tools

This section will present the tools such as the programming language and libraries used in the implementation of this project.

The program is written using the programming language Python. Python is a dynamically typed, garbage-collecting, interpreted and multi-paradigm (including object orientation) programming language [11]. Python is a good language for us because of its vast machine learning libraries. The machine learning library used in this project is Scikit learn [12]. Scikit learn is among the most popular machine learning libraries for Python and is a good choice for beginners because of its relative simplicity and good documentation.

The libraries Numpy, Pandas and Matplotlib are also used within this project. Numpy makes it so that Python can handle multidimensional arrays and matrices in addition to some mathematical functions [9]. Pandas makes the Numpy arrays more efficient to manage by adding table manipulation tools, which makes time series easier to control [10]. The arrays used by Pandas are called frames in this project. The library Matplotlib is used to display graphs in a proper way. The project uses UNIX time to represent time. UNIX time is a format which represent the number of seconds that have elapsed since 00:00:00 Thursday, 1 January 1970 [20]. Lastly, the project is written entirely using Jupyter notebook, a web-based notebook editor. Jupyter notebook is a tool that allows the user to write code as one would write a paper.
This is done by structuring an ordered list of input/output cells that can contain compiled code, text, mathematic formulas and plots [5].

2.4 Summary

This chapter has presented the basic background necessary for understanding this project. We have defined what machine learning and a machine learning algorithm is. Supervised learning was introduced with concepts such as training, testing and regression. How to properly manage and evaluate the data has been presented. Lastly, the tools used for the implementation was presented.
3 Design

This chapter explains the design of our solution, how we intend to solve the problem of selecting the most appropriate machine learning algorithm in a given setting. Section 3.1 describes a couple of basic concepts and some design choices. Section 3.2 shows the first phase of the machine structure: training the selection algorithm. Section 3.3 describes the second phase of the machine structure: testing the selection algorithm.

3.1 Basic concept

The purpose of this project is to design an algorithm selection model. We call the model the selection machine or just the machine. The machine should tell us what algorithm we give it is the “best”. The “best algorithm” is a relative statement that depend on what traits we value as beneficial. This project has chosen one trait for simplicity. Note that any other trait would work just as well. The trait can be described as: a desired algorithm prediction is one that has low mean square error deviation compared to the ground truth.

3.1.1 Naive design

Figure 1 shows the first intuitive solution to this project. The algorithms in set $A = \{A1, A2 \ldots An\}$ are trained on the dataset $D$. The subset $D_{1X}$ is used as the features and $D_{1Y}$ as the value to be predicted in the training phase. The same algorithms are used in the testing phase with the same dataset $D$ but with the new subset $D_{2X}$ as features. The produced prediction set $R = \{P1, P2 \ldots Pn\}$ is used in a comparison to the ground truth $D_{2Y}$. The algorithm prediction with the lowest deviation from the ground truth, $P_X$, is selected as the best algorithm.

The solution shown in Figure 1 can only select the best algorithm for that exact situation. The problem is that different algorithms can be good in different situations and points in time. A way of capturing this is to use a selection algorithm that will be taught to select which of the testing algorithms are the best. The selection algorithm would be trained on previous testing algorithm predictions sets, as denoted as $R$ in Figure 1. The comparison between the testing algorithm prediction set $R$ and the ground truth in Figure 1 would instead be a comparison between $R$ and the generated selection algorithm prediction henceforth called the machine prediction.
Figure 1: The basic machine learning principle: training and testing a number of algorithms. 

A = \{A1,A2 ... An\} is the algorithms set. \(D_{1X}\) is used as the features and \(D_{1y}\) is the value to be predicted in the training phase. Testing phase: \(D_{2X}\) is used as the features, \(R\) is the prediction, \(D_{2X}\) is the ground truth and \(P_X\) is the score. The resulting prediction in the testing phase is compared to the ground truth. The process would give a list of mean square error scores upon which one algorithm can be selected to be the best.

A problem to consider when using a selection algorithm is the algorithm choice. The algorithm used for selection has to be efficient in predicting the behavior of other algorithms. Ideally, we would need an algorithm selection machine to plug our different options into. The machine necessary to fulfill this would be the *machine* being designed and we do not have access to a similar selection instrument until the project is finished. The selection algorithm used in this process will be chosen to be multiple variable linear regression. Linear regression is regarded as being one of the more effective and consistent algorithms to be used in algorithm selection [6]. It is worth noting that algorithm selection typically is classified as a classification problem, not a regression problem. The linear regression model will be used in a similar fashion as a classifier would be used. The selection algorithm will predict a floating-point value and the testing algorithm with the comparably lowest mean square error deviation will be selected.

### 3.1.2 The dataset and time series

The testing algorithms used for this project will be random forest regressor, decision tree regressor and KNeighbor regression. The dataset used to exemplify this project is the time series “Data of Five Chinese Cities Data Set” between Jan 1st, 2010 to Dec 31st, 2015 [7]. The set contains time, weather conditions and other related meteorological data for each city. The
project is only going to use information from one city: Beijing, with the parameter’s year, month, day, hour and temperature in Celsius. These parameters imply that the time: year, month, day and hour are used as features by the testing algorithms and the temperature is used as the value to be predicted.

The amount of data and how it is used needs to be decided. The first alternative is to use a set amount of data for training and testing, for example an evenly distributed 70-30% split of the entire dataset respectively. But since the datapoints in the dataset is in a time series a 70-30% split would undermine the underlying time symmetry. The second method is the preferred way of using such data: make predictions close in time to the trained data. The training data subset used to make a prediction is called the time-window. Using the second method means that before each prediction we need to move the time-window and re-train the algorithms on the new time-window as demonstrated in Figure 2. The figure has a time axis going from \( t_1 \) to \( t_n \) showing how the sliding time-window shifts the chosen training interval a given amount each time a new prediction is made. Note that each cell in Figure 2 can represent one point in time or a time interval.

![Figure 2: The sliding time-window. The window “slides” or “shifts” a set interval each prediction.](image)

The sliding time-window is the most intuitive when used as a forecasting tool at the end of a dataset that expands at set periods. The value to be predicted would always be shifted each period to always be kept in the future. The results produced by the machine in this case would update each period, always showing the most up to date result. The forecasting method can be
used with the Beijing dataset even though it has a set size. The time-window will be placed at the start of the dataset, just as it would be placed when using a model such as the one shown in Figure 1. After each prediction the time-window will shift one interval forward, making the machine re-train the testing algorithms on the new time-window. The time-window will continue to shift until a set limit or until the end of the dataset has been reached. At this point, the results from each prediction can be summarized to produce interesting statistics.

3.2 Training

When using a selection algorithm, just like any other algorithm, it needs to be trained and tested. This section is going to present solutions covering the training phase of the selection algorithm.

3.2.1 Teaching the selection algorithm

The first concept to consider is what the linear regression model is going to be taught. The purpose is to artificially create a value to compare the testing algorithms predictions to. Figure 3 is an extension of the testing part of Figure 1. The features $D_x$ are given to the algorithm set $A = \{A1,A2 ... An\}$ and the prediction set $R = \{P1,P2 ... Pn\}$ is created. The machine $M$ takes the prediction set $R$ and selects one of its members as the prediction $P_x$.

![Diagram](image)

*Figure 3: The basic concept of algorithm selection. $A = \{A1,A2 ... An\}$ is the algorithms set. $D_x$ is used as the features and $R$ is the resulting prediction. The machine $M$ is fed predictions to eventually produce a machine prediction $P_x$."

The testing algorithms, in the context of the Beijing dataset, predicts the temperature based on the time. The selection algorithm would take the predicted temperatures and the time as
features. The testing algorithm prediction with the lowest mean square error deviation from the ground truth would be used as the value to be predicted in the selection algorithm.

Figure 4 presents an expanded view of the machine M. The algorithm set A is given the features $D_X$ and produces the prediction set $R = \{P1, P2..., Pn\}$. The module $m$ consists of the selection algorithm linear regression that is to be trained. The module $m$ takes the set $R$ with the respective time as features and the value $V$ as the value to be predicted. The purpose of module $C$ is to create the value $V$. The module $C$ takes the ground truth $D_y$, the prediction set $R$ and performs a mean square error calculation on each value in $R$ compared to $D_y$. The value in the prediction set $R$ that has the lowest deviation from the ground truth is selected as the value $V$.

Figure 4: Machine training. $A = \{A1, A2 ..., An\}$ is the algorithms set. $D_X$ is used as the features and $R$ is the resulting prediction. The figure shows how the model will work by modifying the picture shown in Figure 3. The figure expands the machine $M$ and separates it into the different modules $m$ and $C$. $D_y$ is the ground truth and $V$ is the value to be predicted by the machine algorithm.

Figure 4 is not a direct expansion of Figure 3 since Figure 4 does not produce a final selection prediction. Figure 4 is the representation of the training phase of the selection algorithm. To produce a prediction as a product of this training, a sufficient number of samples are needed.

3.2.2 The time-window

The time-window is conceptually easy to use but becomes complex when more than one time-window is needed. Preferably we would want a separate time-window for each testing algorithm and one for the selection algorithm, since different algorithms perform better or worse on
different sized data. To decrease the complexity, the number of time-windows used are reduced to one for the testing algorithms called the *algorithm-window* and one for the selection algorithm called the *machine-window*.

The *algorithm-window* takes a set amount of data points directly from the dataset while the *machine-window* takes a set amount of predictions from the previous *algorithm-windows* with respective timestamp.

Figure 5 shows both windows in the same figure. The figure presents a example where the time axis goes from $t_1$ to $t_{29}$, demonstrating how the time-windows shifts and work together. The *algorithm-window* is denoted by the black squares or the blue lines followed by the predicted value denoted by a circle. The *machine-window* is denoted by the red lines, which are the previous predictions from the *machine-window*. Following the *machine-window* is the value predicted by the *machine*, denoted by a purple dot. The black circles in the figure represent points where the *machine* has made a prediction. The example demonstrated in Figure 5 set the *algorithm-window* to 24 units and the *machine-window* to 2 units. These values imply that the *machine* can produce the first prediction at unit 27 based on the *machine-window*. At this point, the *machine-window* is completely filled by the predictions made by the algorithms based on the *algorithm-window*.

*Figure 5: The machine-window and algorithm-window. Each window “slides” or “shifts” a set interval each prediction.*
3.3 Testing

After training the selection algorithm model on an appropriate amount of testing algorithm predictions the testing phase can commence. This section is going to present solutions regarding the testing phase of the machine and its algorithm.

The testing phase of the machine is fairly similar to the training phase. The process in Figure 6 describes how the selection algorithm produces its prediction. The algorithm set \( A \) is given the features \( D_X \) and produces the prediction set \( R = \{P_1, P_2..., P_n\} \). The module \( m \) consists of the trained linear regression model. The module \( m \) takes the prediction set \( R \) with the respective time as features and produces the machine prediction \( P_X \). The prediction \( P_X \) is outputted from the machine, and later on, paired up with the ground truth \( D_Y \).

![Figure 6: Machine prediction. \( A = \{A_1,A_2...A_n\} \) is the algorithms set. \( D_X \) is used as the features and \( R \) is the resulting prediction. The figure shows the module \( m \) producing the prediction based on the training received by the data in machine-window. The prediction is outputted from the machine and stored for the evaluation phase together with respective ground truth. \( D_Y \) is the ground truth and \( P_X \) is the machine prediction.](image)

After the testing phase is finished the entire process is reset, starting at the training phase once again. The only difference this time around is the shift in respective time-windows: the algorithm-window and machine-window. The shift makes it so that a new point in the dataset is predicted, as seen in the previous Figure 5.
3.4 Summary

In this chapter, the design of each part of the *machine* has been presented. Basic concepts such as how to handle the selection algorithm and the dataset with the different time windows has been discussed. Lastly, the general design choices regarding the training and testing phase of the *machine* has been presented.
4 Implementation

This chapter is going to show the implementation of the model in this project. Section 4.1 introduces a few variables used in the program. Section 4.2 shows the outer most layer of the program. Section 4.3 explores the function responsible of handling the testing algorithms. Section 4.4 explores the function responsible of handling the selection algorithm.

4.1 Variables explanation

The variables interval, window_size, M_window_size is used to control the behavior of the machine. The variable interval is the fundamental base value: it sets what interval is going to be predicted by the algorithms. The value of interval corresponds to the number of rows in the dataset that is considered. The variable window_size represents the size of the algorithm-window and sets how many intervals each algorithm is going to base their prediction on. The variable M_window_size represents the machine-window and sets how many testing algorithm predictions the selection algorithm needs to produce a new prediction.

There are three different Pandas data frames used to keep track of the data. The frame called df stores the entire dataset. The frame called algorithms is used to store the testing algorithm predictions together with the ground truth and the time. The last frame called results store the testing algorithm predictions, corresponding machine prediction, time, ground truth and some additional information.

The operations of the machine are based on the proportions given by the variables and the communication between the different functions is done through the different data frames.
4.2 The main body

The main body of the *machine* consists of frame and variable initializers followed by two operation loops.

```python
def init():
    names = ['Random forest regressor', 'Decision Tree regressor', 'K Neighbors Regression']
    function_list = [RandomForestRegressor(max_depth=100, random_state=0, n_estimators=100),
                     DecisionTreeRegressor(max_depth=7, random_state=0),
                     KNeighborsRegressor(n_neighbors=5)]
```

*Figure 7: Code illustrating the initialization step of the main body of the program.*

As seen in Figure 7, the function *import_data()* reads and returns the dataset into the frame *df* and sets and returns the *interval*, *window_size*, *M_window_size* variables. The function *init()* creates and returns the data frames *algorithms* and *results* and sets and returns a couple of arrays and variables. The function *init()* is partly displayed in Figure 8, showing the content of the algorithm arrays *names* and *function_list*. The arrays represent the algorithms used in this project, *names* is a simple string array and *function_list* is a class pointer array.

*Figure 8: Code illustrating how the algorithms are represented.*

Figure 9 includes parts of the code from Figure 7 and the section following it. This section uses the two for-loops to make up the functionality of the program. The variable *loop* is set to a value dependent on the size of the dataset or *df*, so that each row of *df* is used in the process. The functions used within the two loops, *init_algorithms()* and *machine()*, is the bulk of the program. These functions will be described in section 4.3 and 4.4 respectively.
4.3 Initializing the testing algorithms

The function `init_algorithm()` takes the variables `pointer` and `iterator` as parameters. The `pointer` is a global offset variable that keeps track of which segments of the dataset `df` has been processed. The `iterator` variable is an offset variable that is used for writing to the `algorithms` frame. The main body of this function consist of two nested for-loops as seen in Figure 10.

The purpose of the `init_algorithms()` function is to use the local function `algorithm_generator()` to fill the `algorithms` frame up to the limit set by `machine-window` and the variable `M_window_size + 1`. When the loop is terminated, and the `pointer` value is returned to be used for the next iteration, the `algorithms` frame can be used by the function `machine()` described in section 4.4. The `algorithms` frame will consist of the testing algorithms predictions necessary for the selection algorithm to start producing predictions.

The function `algorithm_generator()` is partly displayed in Figure 11. Its purpose is to produce the testing algorithm predictions based on the `algorithm-window`, which size is defined by the variable `window_size` and `interval`.
Each testing algorithm is trained and tested using the class pointer array function. The local offset variable `train_window` keeps track of which portion of the original dataset `df` the testing algorithms should use for its respective training and testing phase. The dataset `df` is split into different portions: the frames `X_train` and `y_train` for training and `X_test` and `y_test` for testing. The `X_train` frame is the features of the prediction and the `y_train` frame is the value to be predicted. Both of them contain the `df` values defined by the `algorithm-window`. The `X_test` frame is the features of the test and the `y_test` frame is the ground truth. Both of them contain the `df` values that should be predicted with the help of the `algorithm-window`. In other words: the split is done in accordance with the sliding time-window concept.

The training frames are applied as parameters to the `fit` method for the algorithm defined by `function` as seen in Figure 11. The variable `model` stores a pointer to the new trained algorithm. The test frame `X_test` is applied as a parameter to the `predict` method for the newly trained algorithm defined by `model`. The result of the test is stored as a frame in `y_pred`.

The code segment followed by Figure 11 consists of plugging the algorithm prediction `y_pred` into the `algorithms` frame with the respective time-stamp. The function `algorithm_generator()` is called once for every testing algorithm for each time-stamp being processed. When all the testing algorithms have been processed for a given time-stamp, the ground truth from the dataset `df` is inserted into the `algorithms` frame before the next process of the upcoming time-stamp starts.
4.4 The selection algorithm function

The function called \texttt{machine()} takes the variable \textit{machine iterator} as a parameter. The variable \textit{machine iterator} is an offset variable that is used to read from the correct positions in the \textit{algorithms} frame.

The purpose of the \texttt{machine()} function is to train and test the selection algorithm. Figure 12 show the training part of the process. The local offset variable \textit{step} and \textit{upper_step} keeps track of which portion of the \textit{algorithms} frame the \texttt{machine()} function should use for its respective training and testing phase.

4.4.1 Training

The training phase of the function \texttt{machine()} can be seen in Figure 12. The \texttt{X\_train} frame, the \textit{features} of the \textit{machine} prediction, is filled with the testing algorithm predictions with respective time-stamps from the \textit{algorithms} frame as described by the \textit{machine-window}. The \texttt{y\_train} frame, the value \textit{to be predicted}, is given the testing algorithm prediction with the lowest mean square error deviation from the \textit{ground truth} for each given time-stamp. To clarify; the \texttt{y\_train} frame contains the best algorithm prediction for each given time-stamp. The \texttt{X\_train} and \texttt{y\_train} frames are applied as parameters to the \texttt{fit} method for the selection algorithm linear regression. A pointer to the trained linear regression model is stored in \texttt{model}.

```python
def machine(machine_iterator):
    step = machine_iterator*interval
    upper_step = N\_window\_size*interval
    X_train = pd.DataFrame(np.zeros((1,len(names))*len(time_format))), columns = names + time_format
    for i in range(upper_step):
        X_train.loc[i,names] = algorithms.loc[step + i, names]
        X_train.loc[i,time_format] = algorithms.loc[step + i,time_format]
    y_train = pd.DataFrame(np.zeros((1,1)))
    tmp = pd.DataFrame(np.zeros((1,len(names))), columns = names)
    for i in range(upper_step):
        for j in range(len(names)):
            tmp.loc[i,names[j]] = mean\_squared\_error(algorithms.loc[step + i,['Ground truth']],
            np.array([algorithms.loc[step + i,names[j]]]))
        y_train.loc[i,1] = algorithms.loc[step + i,tmp.loc[0,names].astype('float64').idxmin()]
    model = LinearRegression().fit(X_train,y_train)
```

\textit{Figure 12: Code illustrating the first part of the selection algorithm function \texttt{machine()}. The figure shows the training phase of the function.}
4.4.2 Testing

The testing phase of the function `machine()` can be seen in Figure 13. The `X_test` frame, the features of the `machine` prediction, is filled with the testing algorithm predictions with respective time-stamps from the `algorithms` frame as described by the `machine window`. The `y_test` frame, the algorithm prediction to be predicted, is given the algorithm prediction with the lowest mean square error deviation from the ground truth each given time-stamp. The `X_test` frame is applied as a parameter to the `predict` method for the trained linear regression model pointed to by `model`. The result of the `machine` prediction is stored as a frame in `y_pred`. If the result stored in `y_pred` is unexpectedly large, for example 100 or 1000 times larger than the average, the result is discarded and logged.

```python
X_test = pd.DataFrame(np.zeros((1,len(names)+len(time_format))),columns = names + time_format)
for i in range(len(interval)):
    X_test.loc[i,names] = algorithms.loc[step + upper_step + i, names]
    X_test.loc[i,time_format] = algorithms.loc[step + upper_step + i,time_format]

y_test = pd.DataFrame(np.zeros((1,1)))
for i in range(len(interval)):
    y_test.loc[i,] = algorithms.loc[step + upper_step + i,0]
    y_test.loc[i,] = algorithms.loc[step + upper_step + i,0,][0]

y_pred = model.predict(X_test)
```

*Figure 13: Code illustrating the second part of the selection algorithm function `machine()`.*

The figure shows the prediction phase of the function.

The last part of the `machine()` function is partly shown in Figure 14. The frames `X_test`, `y_pred`, `y_test` and the time format are added to the `results` frame. A mean square error deviation calculation between the frames `y_test` and `y_pred`, the testing algorithm with lowest mean square error deviation compared to the `machine` prediction, among other interesting values is stored in `results`. Discarded predictions are not stored in `results`. 
for i in range(interval):
    results.loc[step + i, 'names'] = X_test_loc[i, names]
    results.loc[step + i, 'Prediction'] = y_pred[1]
    results.loc[step + i, 'Ground truth'] = y_test_loc[i, :].values
    results.loc[step + i, 'Error of prediction'] = round(mean_squared_error(          y_test.loc[i, :].values, np.array([y_pred[1]])), 4)
    results.loc[step + i, 'time_format'] = X_test_loc[i, 'time_format'].values

    tmp = pd.DataFrame(np.zeros((1, len(names))), columns = names)
    for j in range(len(names)):
        tmp.loc[:, names[j]] = mean_squared_error(results.loc[step + i, 'Prediction'].values, np.array([results.loc[step + i, names[j]]]))
    results.loc[step + i, 'Best'] = tmp.loc[0, names].astype('float64').min()  

Figure 14: Code illustrating the last part of the selection algorithm function machine().

The figure shows the values that are stored in the results frame.

4.5 Summary

In this chapter the code used in the implementation has been displayed and explained. The variables interval, window_size, M_window_size has been introduced and explained. Code details for the functions that handle the training and testing of the testing algorithms as well as the functions that handle the training and testing of the selection algorithm has been displayed.
5 Results

This chapter shows the result of the project. Section 5.1 is going to introduce how to derive useful information from the result. Section 5.2 is going to present the result with a numeric figure and respective graphs.

5.1 How to analyze the data

The *machine* iterates through the dataset, shifting the respective time windows accordingly, until the manual set limit is reached or the dataset ends. After reaching the end of every test phase, the prediction is outputted from the *machine* and stored together with the respective *ground truth*. In the evaluation phase, all the stored values are processed and used to produce the results.

There are a couple of useful ways to compare and present data. The first couple of ways are presented in Figure 15. The first three columns from the left in the figure represent the respective testing algorithm predictions. The “Prediction” column shows the selection algorithm’s prediction. The “Best” column shows the name of the testing algorithm with the best match compared to the selection algorithm’s prediction. The “Ground truth” column shows the value the selection algorithm is predicting, which is the value of the testing algorithm with lowest deviation from the *ground truth*. The “Correct” column shows the name of the testing algorithm with the best match compared to *ground truth*. The “Error of prediction” column shows the mean square error deviation when comparing the “Prediction” column to the “Ground truth” column. The last four columns show the time format for each row of predictions.

The first interesting statistic displayed in Figure 15 is the “Best” and “Correct” columns. After summarizing the data from all *machine* iterations, these columns can be used to produce a “proportion of correct guesses by the *machine*” percentage. The statistic produced by the “Best” column can also be used to count how many times the *machine* chose each algorithm. The amount of times the *machine* chose each algorithm can be compared to the corresponding values produced by the “Correct” column for the *ground truth*. 

24
Another interesting statistic found in Figure 15 is the “Error of prediction” column. As explained previously, this column is the result of a mean square error comparison between the machine prediction and the ground truth.

The error of prediction approach can also be extended to each individual algorithm. The prediction of each algorithm would be compared to the ground truth and the machine prediction respectively.

5.2 Evaluation

This section is going to present and discuss the different results generated by the tests in the project. The dataset used for these tests are, as mentioned in Chapter 3, the time series set called “Data of Five Chinese Cities Data Set” [7]. Out of the five Chinese cities, Beijing was chosen for this project as displayed in Figure 16. The graphs displayed in this section use the UNIX time as x-axis.
The following test uses the following variable combination:

- *Interval* = 1
- *Window_size* = 12
- *M_window_size* = 12

As explained previously in chapter 4, the *interval* value is the number of rows predicted by all algorithms. The *Window_size* variable determines the size of the *algorithm window*. The *M_window_size* variable determines the size of the *machine window*. The combination was chosen as such to show a good example of the *machine* performance.

---

Figure 17: The numeric result produced using the Beijing dataset. The first two lines are statistics regarding the selection algorithm. The following two segments are statistics regarding the testing algorithms compared to the selection algorithm and the ground truth respectively.
The first test produced the result shown in Figure 17. The test used 52,339 rows of the Beijing dataset, where one row represents one hour. The test used 99.6% of the entire set and the code took around four hours to execute. The last 0.4% of the set is unintentionally left out because of the size symmetry of machine and algorithm windows and because of the way the code is implemented.

The proportion of correct predictions is shown as 63.2%. This is a statistic showing each time the machine chose the same algorithm as would have been chosen by the ground truth. Although this proportion looks like a measure of efficiency, it should not be interpreted that way. The proportion of correct prediction statistic should be viewed as a first look, not a representation of the result as a whole. Figure 17 also shows the mean squared error of all predictions as 1.0 °C. This statistic shows how many Celsius the machine prediction deviates from the ground truth.

The decision tree regressor algorithm was chosen as the best algorithm by the machine 66.1% of the times followed up by random forest regressor at 27.5% and KNeighbor regression at 6.5%. The decision tree regressor algorithm was also chosen as being the best algorithm when compared to the ground truth 77.5% of the times followed by random forest regressor at 11.7% and KNeighbor regression at 10.8%.

The algorithm that had the lowest mean square error deviation from the prediction was the decision tree regressor with 0.82 °C. This statistic shows how many Celsius the algorithm prediction deviates from the machine prediction and is another way of showing that the decision tree regressor is the best algorithm. The second best algorithm is the random forest regressor followed by the KNeighbor regression in last place. This ranking order corresponds with the ranking created by mean square error deviation compared to the ground truth. The difference between the machine prediction and the ground truth represents the cost of using a prediction as a shortcut to the ground truth.
Figure 18: Graphs showing the difference between the prediction and the ground truth. The temperature is in Celsius and is compared to time. The time value in the x-axis is interpreted as for example: $1.275 \times 10^9 \rightarrow 22:40, 27/5-2010$ using UNIX time.

Figure 18 shows the first set of graphs produced. The figure shows the difference between the machine predictions and the ground truth. The difference is fairly low, as seen by the mean squared error deviation at 1.0 °C in Figure 17. Both graphs displayed show the same graph but with different overlapping order, as to create a better perspective.

Figure 19: Graphs displaying the difference between each testing algorithm compared to the machine prediction. The temperature is in Celsius and is compared to time. The time value in the x-axis is interpreted as for example: $1.275 \times 10^9 \rightarrow 22:40, 27/5-2010$ using UNIX time.
Figure 19 shows the second set of graphs produced. All the graphs displayed show the difference between respective algorithm compared to the *machine* prediction. This figure visualizes the 0.88 °C average difference for the random forest regressor, the 0.82 °C average difference for decision tree regressor and the 7.92 °C average difference KNeighbor regression as seen in Figure 17.

![Graphs showing difference between respective algorithm and machine prediction.](image)

*Figure 19: Graphs showing the difference between respective algorithm and machine prediction. The temperature is in Celsius and is compared to time. The time value in the x-axis is interpreted as for example: 1.275 * 10^9 → 22:40, 27/5-2010 using UNIX time.*

Figure 20 shows the third set of graphs produced. All the graphs displayed show the difference between respective algorithm compared to the *ground truth*. This figure visualizes the 0.33 °C average difference for the random forest regressor, the 0.2 °C average difference for decision tree regressor and the 7.52 °C average difference for KNeighbor regression as seen in Figure 17.

![Graphs showing difference between respective algorithm and ground truth.](image)

*Figure 20: Graphs displaying the difference between each testing algorithm compared to the ground truth. The temperature is in Celsius and is compared to time. The time value in the x-axis is interpreted as for example: 1.275 * 10^9 → 22:40, 27/5-2010 using UNIX time.*
5.3 Summary

This chapter has presented the result of the project via graphs and numeric statics. How to interpret the data and *machine* efficiency and reliability has been introduced. Numeric statistics such as mean squared error deviation and proportion of correct predictions of the selection algorithm and each individual testing algorithm has been presented and discussed.
6 Discussion

This project presented a version of the algorithm selection model also called the *machine*, a version that fits the scope of this assignment. The version presented does not work in a real and practical situation without design and implementation expansions. The *realistic version* would fill a different purpose compared to the *machine* produced in this project.

This project focuses on creating a model that works as a proof of concept to support the creation of the *realistic version*. The *realistic version* would have two distinct functions: a forecasting tool for predicting the future and an analyzer of past behavior. This project has implemented a combination of these two functions, a summary of forecasting predictions. The implemented model does therefore not fulfill the same purpose as the *realistic version* would. The *realistic version* would be able to produce a statistical summary of all past data and a future prediction, with different time window configurations, for an upcoming time interval.

This chapter is going to discuss the produced results as well as design expansions necessary for the creation of a *realistic version*. This chapter is also going to introduce and discuss new potential problems, limits and shortcomings. Section 6.1 presents a discussion regarding the result produced in Chapter 5. Section 6.2 discusses the first problem that appears when going from this project to the *realistic version*, the feedback loop. Section 6.3 discusses potentially necessary additions. Lastly, section 6.4 discusses possible changes regarding the algorithms and dataset used.

6.1 Result discussion

The 1.0 °C mean squared error deviation of all 52 339 *machine* predictions seen in Figure 17 was considerably better than we expected, since some individual deviation could range up in the 600 °C range (a squared distance). The 63.2% proportion of correct predictions as seen in Figure 17 is however disproportionally low compared to the mean squared error deviation. The graphs shown in Figure 18 seems to show the 1.0 °C mean squared error deviation as expected.

We expected the *machine* to do a higher number of correct guesses with a comparably low deviation. It seems that the *machine* prediction can be within very close proximity to the *ground truth* and still end up choosing the wrong algorithm. The problem seems to occur when the
testing algorithms produce close to identical values, meaning that they are practically the same at that point. This is problematic since it can result in the machine choosing the wrong algorithm, even when the wrong algorithm is not similar to the best algorithm derived from the ground truth. Perhaps this problem can be solved by giving the best algorithm a weighted score each situation instead of a discrete win or loss. The weighted score would reflect the similarities between the different algorithm predictions, giving the winner a higher score the more certain the machine is of its answer. The remaining algorithms would also receive a weighted score, a score that increases the lower the perceived validity of the winner is.

The prediction accuracy score statistics, as shown in Figure 17 is about what we expect to see with 1.0 °C mean squared error deviation and 63.2% proportion of correct predictions. The algorithm chosen as being the best by the machine, decision tree regressor, is considerably better than the runner up, random forest regressor, when compared to the ground truth accuracy score. The difference between the random forest regressor and the worst algorithm, KNeighor regression, is notably worse compared to the ground truth. These comparisons can also be seen in Figure 19 for the machine prediction and in Figure 20 for the ground truth. Meaning that the difference between the decision tree regressor and the other algorithms are not as big as the prediction and ground truth accuracy score statistic show. This fact is also supported when comparing Figure 19 to Figure 20 each other.

The machine has seemingly no problem when the best algorithm is significantly better than the worse algorithms. However, just as in the case of proportion of correct predictions statistic, the problem arise when the algorithms are within very close proximity to one another. The solution of weighted score could be applied here as well. The machine would reward the algorithm it consider the winner a higher score the more certain the machine is of its answer. The remaining algorithms would receive a score that reflect the machines uncertainty of the winner.

The mean squared error of each algorithm compared to the machine prediction and the ground truth, as seen in Figure 17 also reflect the expectations from the 1.0 °C mean squared error deviation and 63.2% proportion of correct predictions. The algorithm chosen as the best by the machine prediction is the decision tree regressor at a 0.82 °C deviation, which is the same as the algorithm chosen when comparing to the ground truth at 0.2 °C. The ground truth in the mean squared error of each algorithm statistic create the same ranking order as the machine prediction does.
The mean squared error of each algorithm statistic is comparably better than the prediction and ground truth accuracy score statistic. In the machine prediction part of the mean squared error of each algorithm statistic we see that the random forest regressor deviation at 0.88 °C is really close to the decision tree regressor deviation at 0.82 °C. This fact can be seen in Figure 19 which show the similarities between the two algorithms. The KNeighbor regression has a larger deviation at 7.91 °C which also can be seen in Figure 19. The same pattern can be seen in Figure 20. These facts are not represented in the prediction and ground truth accuracy score statistic, which showed the decision tree winning by a large margin. If the we consider the prediction and ground truth accuracy score statistic as truth, we would expect a lot higher deviation than what we actually see in Figure 19 and Figure 20.

The machine discards and logs large unexpected values that are for example 100 or 1000 times larger than the average as previously mentioned in chapter 4. The log for this result show a single discarded machine prediction at 7.69 * 10^{12} °C at 19:00, 25/6-2013. This prediction is absurdly big and, if it were not discarded, would have changed the mean squared error deviation statistic to a drastic value of 1.13 * 10^{21} °C. The problem causing this lies with the selection algorithm linear regression. We do not know why the algorithm behaved so strangely at that single point out of 52 339 non-extreme predictions. Perhaps the problem lies within the original library we used and a change in the algorithm presets and parameters could solve it. Perhaps the linear regression model itself has tendencies of occasional producing larger outlying predictions and we should have considered using something else in its place.

6.2 Feedback loop

The first concept that needs examining when thinking about a realistic version is the so called feedback loop. The feedback loop is the channel in which the ground truth arrives to the machine as seen in Figure 6 as Dy.

6.2.1 Feedback delay

In this project, we can interact with the entire dataset at any point in time. We could see it as if we had instantaneous access to the future at all times. In a realistic version, the feedback values would not come from a locally kept dataset. The feedback values would, in the case of meteorological data, come from a separate weather module or station. The delay between making a machine prediction and receiving the ground truth to be used in the next machine
prediction is the feedback delay. The feedback delay in this project essentially is \( t = 0 \) and would in a realistic version be \( t > 0 \).

The concept of feedback delay reshapes the design considerable. Not only does the machine need to wait for the feedback to evaluate its own predictions, it needs the feedback to produce the next prediction. A serious problem occurs when the feedback does not arrive when it is expected. There are a couple of solutions to this, solutions which create even more problems.

6.2.2 Possible solutions

6.2.2.1 Waiting
The first solution, and the most intuitive one, to lost feedback would be for the machine to wait. The machine would set a timer and wait until it expires, upon which the machine would have to assume the machine prediction either did not arrive at the receiving module or that the feedback value got lost. The machine would have to resend the prediction and reset the timer, resetting the process. The timer could be changed dynamically based on previous trends. A lower timer would be optimal when the machine is expecting a time-out, so that the message can be re-sent faster.

6.2.2.2 Estimation
What if the feedback never arrives? The machine would either be stuck in an infinite waiting loop or be forced to estimate the ground truth value so it can move on to the next prediction. The machine could for example select the corresponding machine prediction, the mean of the machine window or some other interpolated value in place of the ground truth. Each machine prediction that has been sent to the receiving module would be labeled, so that the machine knows which values corresponds to what predictions. If the feedback or the ground truth for a prediction arrives later in time, the machine would be able to edit its database and update the current time windows. The ground truth estimation solution could be used together with or separately to the waiting method.

The next question we have to ask is: how often should the machine be allowed to estimate the ground truth? The machine cannot use this estimation method too often since it would corrupt the data, making future predictions less reliable. The machine could be allowed to make estimations to a certain degree. When a certain threshold is met, the threshold could perhaps be a percentual estimation limit of the machine window, considerable actions are needed. At this
point, the data would be considered too unreliable and the machine have to stop, go back in time and redo selected predictions and concurrently update its database.

6.2.2.3 Skipping
An alternative to estimating the ground truth when the feedback does not arrive is to skip it and do the upcoming machine predictions based on a smaller window. If the labeled feedback eventually arrives, the machine would be able to edit its database and update the current time windows. The skipping method could be used together with or separately to the waiting and the estimation methods. The skipping method is a better solution for certain machine learning algorithms which are less disrupted by shrinking window sizes, while others would perform worse.

Using the estimation and the waiting method interchangeably, dependent on the algorithm used, with the skipping method is potentially the wisest.

6.3 Condition parameter
Another important aspect of a realistic version is the condition parameter. The condition parameter is an umbrella term for the conditional options needed to control certain aspects of the machine. This section is going to discuss a few conditions needed to create a functional machine.

6.3.1 Time-window
The first useful condition is the option to tailor the different time windows. Chapter 3 introduces the idea of multiple time-windows, one for each algorithm. The chapter dismisses the idea for this project because of the added complexity. The realistic version would need to consider taking this idea for added algorithm efficiency. The condition parameter would ask the user for a desired time-window size each time a new algorithm is added, with a fitting default value if left empty.

One problem that arises when each algorithm has different sized time windows is regarding the machine prediction interval. The selection algorithm would need the same amount of predictions from each testing algorithm to produce its prediction, even though the different testing algorithms produce different amount of predictions on the same set of data. The machine would need to slice the data for each testing algorithm at certain points to control how many
predictions they produce. This problem is not as big as we initially thought, but it is still worth bringing up for discussion.

There is a potential problem how we use the time-window in this project. Each prediction done with the help of the time-window is done close to the training data. The selection algorithm is retrained on a new time window each time it makes a prediction. This approach works good with our dataset and time-window settings as seen in chapter 5, where each prediction interval is one hour. The problem arises if we want a much smaller interval, for example, 15 predictions each second. This time-window configuration would be theoretically allowed by our model but would cause the machine to retrain far too often. The retraining speed required to keep up with 15 predictions each second would probably be too high for a realistic version. A solution to this problem is to set a minimum limit on how many times the selection algorithm can be retrained each second. If the user wants to go below the threshold, the machine should change the retraining interval. Instead of retraining every new prediction, the machine could retrain every other, forth or eight prediction instead. The retraining interval should be defined by a function that depend on how far below the retraining limit the user is.

6.3.2 Advantageous metrics

This project defined one trait the machine should look for as low mean square error deviation compared to the ground truth. The realistic version would be able to select a number of traits via the condition parameter depending on what the user deems valuable, with a fitting default value selected if left empty. The machine would adjust the module denoted as C in Figure 4 to accommodate the behavior selected by the user. The format of the value outputted from the module C would be the same as seen in this project, the only difference would be the reasoning behind that choice. The result would need to be altered to reflect the selected behavior. The user could, in addition to low deviation from the ground truth, for example select if the algorithm prediction should prefer accuracy over a longer time span compared to a short time span and vice versa.

6.3.3 Feedback loop

The conditional parameter could be used to select among the options discussed in the previous section 6.2.2. The user could decide timer values, deterioration rate, the potential percentual estimation threshold of the machine window and the skipping rate. This option could be skipped entirely if further research conclude that a single setting is the most advantageous.
6.4 Algorithm and dataset choice

The algorithms used in this project were all supervised learning regression models and the dataset used was based on that choice. In practice, we would want to put any kind of algorithm and dataset in the \textit{machine}. This project has not explored nor discovered any possible restrictions regarding algorithm and dataset choice. Perhaps our implementation too heavily rely on the techniques used by for example supervised learning. Further research into this subject is necessary for a realistic version to be plausible.

Whether or not any possible limitations are found, the user should be able to add its own algorithms and datasets for examination. The user should not be able to add mismatching algorithm and dataset types, a regression algorithm should for example not be able to be paired up with a classification dataset. The user should also be able to dynamically give the \textit{machine} new algorithms to evaluate. These new algorithms would be trained and tested on the past history in the database automatically for the past history evaluation. These new algorithms would also be added to be used with the forecasting pool.

The selection algorithm used in this project was the linear regression model. We realize that choosing one selection algorithm approach is not ideal for every dataset and algorithm. The model choice was merely used to demonstrate how a realistic version could work. Perhaps there is another single algorithm that performs better under certain circumstances compared to the linear regression model. The realistic version could have a dynamic selection system, where the \textit{machine} changes selection algorithm dependent on which algorithm is expected to perform the best at that time, although this could be hard to implement. There could be a solution which uses another selection system entirely. The solution could perhaps use a sophisticated case based reasoning, which would figure out the best selection method for each iteration.

An interesting way of using the \textit{machine} would be to only give it a dataset accompanied by the type of the set and desired traits. This would result in the \textit{machine} choosing a number of default algorithms for that type and, if the option would exist, a default selection algorithm. The \textit{machine} would tell the user which one of its own basic algorithms are best to use on the given dataset based on the traits specified. This would be a simple way of producing results for someone less experienced with machine learning.
6.5 Summary

This chapter has discussed different design choices needed when considering going from the model implemented in this project to the potential realistic version. Problems regarding the feedback delay has been presented with potential solutions such as waiting, estimation, skipping and a combination of the three. Design solutions regarding the different time-windows and the evaluation metrics has been presented. Lastly, a discussion regarding selection algorithm and dataset choice has been discussed.
7 Conclusion

This chapter is going to conclude the project and its results, present important notes and suggestions for future work and finish of with some closing remarks.

The goal of this project was to create an evaluation model that would lay potential ground work for the consulting company CGI to do future work on. The project has designed and implemented a limited version of the evaluation model, a version that works as a forecasting summary tool using ideal conditions. The algorithms used to create the evaluation were random forest regressor, decision tree regressor and KNeighbor regression. The selection algorithm used was linear regression or more precisely multiple variable linear regression. The result was produced using meteorology data from Beijing between Jan 1st, 2010 to Dec 31st, 2015.

From the results presented in this project, it seems that the conceptual evaluation model and its use of linear regression as a selection algorithm worked out as we had intended. The results concluded that the mean square error deviation approach was an efficient evaluation tool. The only problem with the evaluation was the prediction accuracy score, a score that show how often the model is correct. In cases where the algorithms produced close to identical predictions it seemed like the prediction accuracy score produced the wrong answer more than it was expected to, resulting in a less valid final score. We discussed that the use of a weighted score in place of the prediction accuracy score would have potential to significantly improve the evaluation. The weighted score would prioritize non identical predictions by giving them a higher score compared to near identical predictions, which would hopefully result in a higher score overall.

If we were to redo the project we would want to implement the weighted score, a different selection algorithm in addition to the linear regression model and run more tests. Although we are satisfied with the result created by the linear regression model, we have no way of conforming its validity since there is nothing to compare it to. If we had more time to complete and run more tests, we could delve deeper into the subject and have a better discussion about our results.
7.1 Future work

The discussion presented in this project has described a couple of problems that arise when going from a model using ideal conditions to a model in a realistic setting. The first note that needs consideration is an important design difference between the versions. The realistic version would need to have two distinct functions, one being a forecasting tool for future predictions and another being an analyzer of past behavior. The model presented in this project uses a combination of these functions to exemplify how the realistic implementation could evaluate the result and should therefore not be used as a blueprint for future work.

The problem of feedback delay needs consideration as we remove the ideal conditions from the model. The feedback delay causes us to run the possibility of corrupting parts of the time-window and database. We discuss possible solutions to this problem such as waiting for lost values, estimating or skipping them entirely. We think that the solution with greatest potential is a combined version that uses the estimation and waiting method interchangeably and occasionally uses the skipping method, dependent on the algorithm used.

Additional alterations that could be made to a realistic version is the addition of multiple time windows and algorithm traits. From the discussion regarding the addition of multiple time windows we can conclude that it most likely would increase the overall performance of the model at the cost of slight data manipulation. The discussion regarding multiple algorithm traits shares the same conclusion as the multiple time window concept, but at a higher cost. The addition of multiple algorithm traits would force a redesign of the decision module denoted as $C$ in Figure 4. Although the redesign would be considerable we believe that the added analytic functionality would be worth the effort.

A discussion is needed regarding the choices made for different algorithms and dataset in this project. The selection algorithm approach used for this project is not guaranteed to be the ideal approach for every dataset and testing algorithm choice. The realistic approach could use a dynamic system, where the model could change selection algorithm dependent on what algorithm is expected to perform the best at that time.

Lastly, this project has not explored nor discovered any possible restrictions regarding algorithm or dataset choice. We acknowledge that, the designed model, could rely on unknown
conditions or prerequisites that hinder any possibility of expansion. To ensure that this is not the case, the need for further research into this subject becomes significant.

7.2 Concluding remarks

The evaluation model presented in this project hopefully serves as a suitable starting solution and illustration of the algorithm selection problem. The model presented performed on a satisfactory level according to our evaluation. By our own estimation, the ideal model can still be improved by for example testing different selection algorithms and by exploring the evaluation tool weighted score. We have presented and discussed the steps necessary to go from our ideal model to a realistic implementation. The realistic model will need to consider concepts such as function design, feedback delay, multiple time-windows, trait choice and potential algorithm and dataset restrictions. It is our hope that this project will work as a catalyst for further research into this subject which, eventually, could lead to the implementation of a working realistic non-ideal model.
References


