On the potential for machine learning in prediction of insurance policy sales
helping insurance intermediaries get insights in their clients' insurance needs

Ampt, A.B.F.

Award date:
2017

Technische Universiteit Eindhoven University of Technology

Department of Mathematics and Computer Science
Data Mining Group

On the potential for machine learning in prediction of insurance policy sales

Helping insurance intermediaries get insights in their clients' insurance needs

Adrian B.F. Ampt

	Supervisors:
TU/e:	prof. dr. M. Pechenizkiy
C-Profile:	drs. W. Broeders
TU/e:	dr. ing. M. Hassani

Public version

Eindhoven, June 2017

Abstract

C-Profile is currently experimenting with statistical analysis to give insurance intermediaries insights in the insurance needs of their clients. This means that they analyze a client dataset and create a model to predict if a client is interested in an insurance product. The statistical analysis has two problems. It is too slow to integrate into the C-Profile platform, and the analysis is prone to mistakes. Therefore, we conduct this study to investigate if we can use machine learning to make these predictions. We have implemented ten classification algorithms and designed an experimental environment to run six experiments to answer the question: *Which machine learning technique has the highest potential for predicting insurance product interest?* Various algorithms have potential to create models quickly to make accurate predictions. Ultimately the Decision Tree algorithm and Logistic regression could generate model quickly enough and make accurate predictions. The results were not as accurate as statistical analysis in some cases, but the results were competitive and consistent. The models could also be generated much faster than statistical analysis which takes weeks to setup for a new dataset, whereas machine learning could make predictions within the eight hour window, although often within a minute and even seconds. Finally, machine learning was able to handle irrelevant features very well, which means that the a data scientist does not necessarily have to comb through the data to pick relevant features. With these results in mind we have concluded that C-Profile should pursue machine learning as a tool for making predictions of clients' insurance product interest. Mainly focusing on the two algorithms; Decision Tree and Logistic regression,

Preface

Before you lies my thesis: ”On the potential for machine learning in prediction of insurance policy sales.” This thesis has been written as part of the graduation requirements for the Computer Science & Engineering Master program at the University of Technology Eindhoven (TU/e). The project was undertaken at request of C-Profile for their product C-Profile Advisory Portals. During the course of ten months, I have conducted research to answer the following question: Which machine learning technique has the highest potential for predicting insurance product interest?

Hereby, I would like to thank C-Profile for giving me space and time to properly execute my research. I especially like to thank my daily supervisor Ward Broeders. You provided me with guidance and support for the entire duration of the project. I always enjoyed our debates about the content and results of the project. I would also like to show my gratitude towards my supervisor from the TU/e, Mykola Pechenizkiy; without your feedback and insight, I would have never been able to deliver this level of quality in my work.

Furthermore, I would like to thank my colleagues at C-Profile. Your widespread enthusiasm and interest in the project helped me a lot during moments of doubt. I also want to thank my parents Marie-Therse and Kees for supporting me during the entirety of my academic career. Additionally, I like to thank my siblings Sofie, Kirsten, Kevin and Eamon and their spouses Thomas, Erwin and Emmanuelle for believing in me and helping me when needed. I would like to give thanks to my nieces and nephew Eline, Ella and Ruben for cheering me up with their child like attitude. Finally, I would like to thank my friends and in particular my girlfriend Sandra, for supporting me during these last ten months with their positive outlook.

I hope you enjoy your reading, and I would like to thank you for your interest.

Adrian B.F. Ampt
Eindhoven, Friday 30th June, 2017

Contents

Chapter 1

Introduction

This Chapter gives an introduction to this master thesis project. First, in Section 1.1 we give a brief introduction about the company at which this project has been executed. Next, we give the motivation for this project in Section 1.2. In Section 1.3 we will describe our research goals and in Section 1.4 the problem is translated into research question to be solved. Finally, the methodology to realize the goals is elaborated in Section 1.5.

1.1 C-Profile

C-Profile is a small startup located in Waalre, the Netherlands, developing solutions for insurance intermediaries. The company is a collaboration between two companies; Clascon and LD Software. Clascon is an auditing consultancy, while LD Software designs and develops software. C-Profile has a team of twelve members, divided over content, sales, data analysis and software, developing an online advice platform. The goal of the platform is to deliver low budget, high quality, personal advice to clients of insurance intermediaries.

Such advice is requested for instance if we want to buy a new house. We could research ourselves what the consequences are. However, the advice we need depends on our income, whether we currently rent a house or not, our family situation, i.e. if we have a partner? If we have kids? How old are those kids? and a ton of other characteristics of our life situation. We might be able to factor some of those characteristics in our research, but we surely will not be able to factor them all in within reasonable time, because we are not experts on the subject. Hence we would normally go to our intermediary, to get it sorted out. However, getting an intermediary to get the information is expensive, and for a lot of people it is not affordable. This problem is known as the advice gap.[7]

The C-Profile platform is introduced to tackle this problem, by delivering digitally personalized advice in the platform. The advice of C-Profile is roughly divided in four categories.

- Articles

- Personal articles
- Simulations
- Contact with intermediary

Every client has access to a set of articles. Articles are written around current events and developments that affect a client's daily life. They are not personalized. Besides generic articles, the portal contains personal articles. Personal articles are customized to a person's profile. This means the client only sees what is relevant for their current situation. A more in depth form of advice, is given in simulations. Simulations run a scenario in a person's live given by their current situation. For instance, if a client wants to start a new business. The simulation takes the client through the whole process and shows advice that is only relevant to their situation in a compact and concise manner. If a client wants even more advice, the intermediary can also be contacted. Either via message/email by web cam conversation or in the most extreme case by making an appointment. See Figure 1.1 for the four advice categories.

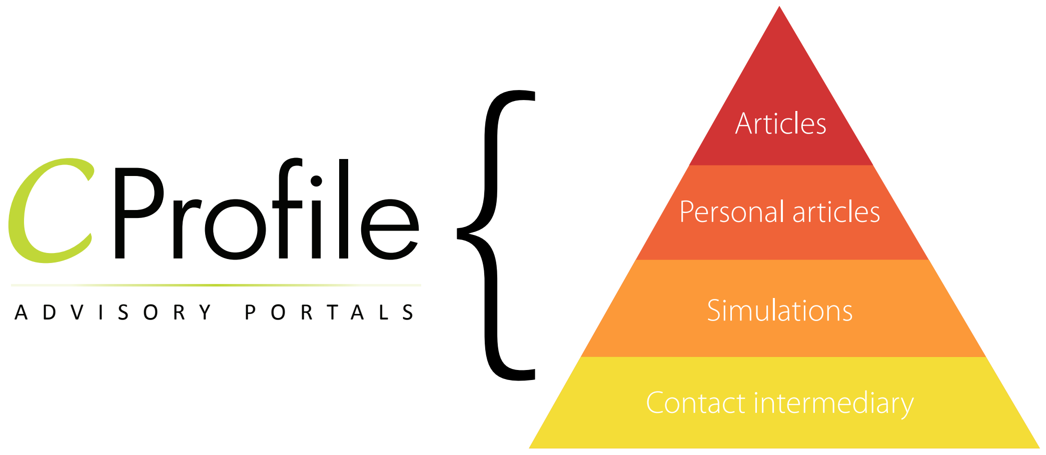
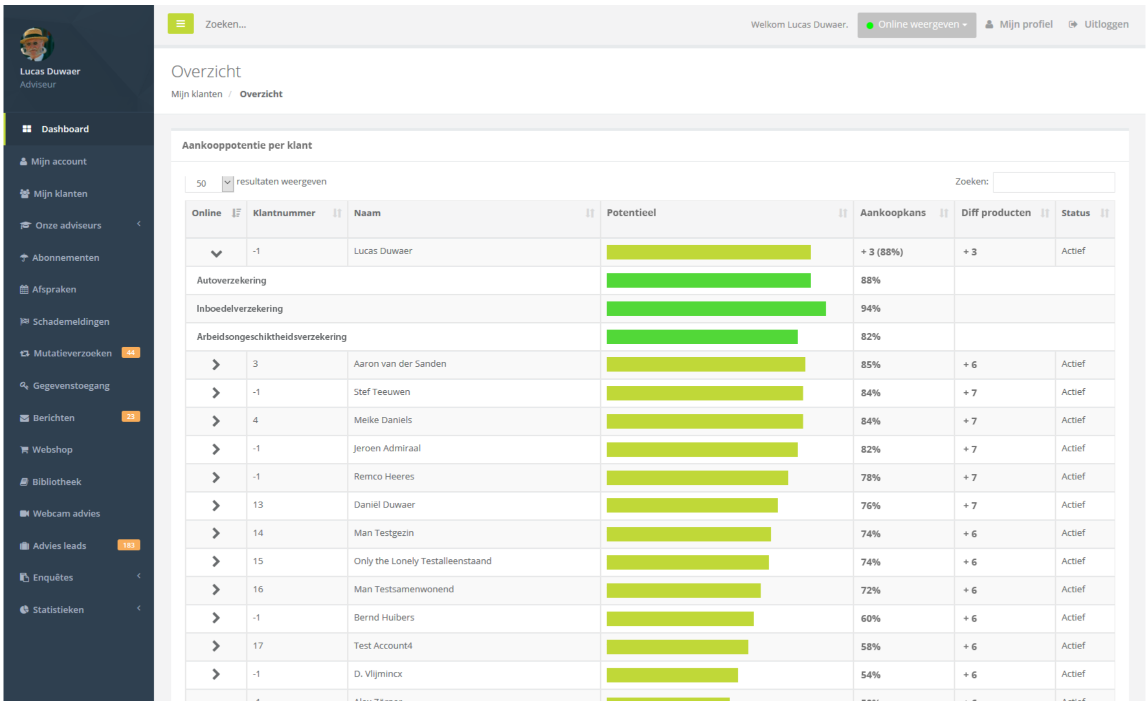


Figure 1.1: The advice type pyramid. Higher on the pyramid means less personal. Higher on the pyramid also means more accessible (less costly).

Besides giving advice to the clients, the platform tries to help insurance intermediaries to get better insights into their clients' needs, by providing advice leads. Leads are indicators of either mutations in clients' situations or life events that require adjustments and advice from the intermediary. So, a lead is a particular situation that requires a product or advice from the intermediary to the client and therefore leads to conversion. The leads result from mutations of client profiles on the one hand and properties of their profiles (i.e. almost eighteen years old) on the other hand.[5]

Leads can also be generated by aggregating client data. With aggregated client data, statistics can be calculated about the population of clients. Using these statistics, we can derive some conclusions. A silly example would be calculating the minimum age of clients who have a car insurance. If it turns out this is 17 years old, it likely means that a 16-year-old will not ever buy a car insurance, which means we do not have to target this demographic. This is very likely due to Dutch law that states you cannot get a (car) drivers license before the age of 17. The statistics used to generate leads can of course be much more complex than the example given.

See Figure 1.2 for a concept of the new leads page, where product sales potentials are laid out for each client.



Online	Klantnummer	Naam	Potentieel	Aankoopkans	Diff producten	Status
⌄	-1	Lucas Duwaer	🟩	+ 3 (88%)	+ 3	Actief
Autoverzekering			🟩	88%		
Inboedelverzekering			🟩	94%		
Arbeidsongeschiktheidsverzekering			🟩	82%		
›	3	Aaron van der Sanden	🟩	85%	+ 6	Actief
›	-1	Stef Teeuwen	🟩	84%	+ 7	Actief
›	4	Meike Daniels	🟩	84%	+ 7	Actief
›	-1	Jeroen Admiraal	🟩	82%	+ 7	Actief
›	-1	Remco Heeres	🟩	78%	+ 7	Actief
›	13	Daniël Duwaer	🟩	76%	+ 7	Actief
›	14	Man Testgezin	🟩	74%	+ 6	Actief
›	15	Only the Lonely Testalleenstaand	🟩	74%	+ 6	Actief
›	16	Man Testsamenwonend	🟩	72%	+ 6	Actief
›	-1	Bernd Huibers	🟩	60%	+ 6	Actief
›	17	Test Account4	🟩	58%	+ 6	Actief
›	-1	D. Vlijmincx	🟩	54%	+ 6	Actief

Figure 1.2: Various product sales potentials are shown for each client.

1.2 Motivation

Current advice leads, are relatively simple and straightforward. They are generated based on the client's own profile. This does not give any new insights for the insurance company. The massive amounts of data the insurance companies have is not aggregated, to help generate leads. From a statistical point of view this data is very powerful, because this data is fairly rich which means a lot of client data is available. Furthermore, data of a lot of clients is available, which allows us to make stronger statistical conclusions. Hence, we want to generate new leads, by aggregating this data.

The data science department of C-Profile is experimenting with statistical analysis on the data from insurance intermediaries to predict which client to target for which insurance product. The statistical analysis outputs a predicted probability that a client will buy a product, for each client and each product.

Currently the models on which statistical analysis is performed are handpicked. This means that domain knowledge of the data is required. A data scientist needs to comb through the data and determine which features have which type of effect on the sales of an insurance product. This means that the data scientist needs to know exactly what every feature describes and in which way each feature contributes to the likelihood of a product sale. For small datasets this can be done quite well, but as the datasets grow in (mostly) the feature size, (i.e. 100+ features) it becomes much harder to hand pick the features and their effect. Furthermore, the data analyst might have too much of a preconception on the data. This means that new types of effects of various features might stay underexposed.

Consider a dataset with 100 features detailing properties about 50K clients, and a classification if each of these clients bought a 12 Month Travel Insurance Policy. A data scientist (with domain knowledge) wants to create a model based on this data to make predictions. The data scientist first needs to inventorize the features and their effect on the data. This is done using various statistical tests, for example by simply executing a Linear Regression prediction and looking at the correlation of certain parameters on the prediction. A correlation that might be found is, for instance, with age. Young people do not have the time to travel a lot, while old people might not have the health to travel a lot. However, middle aged people might have the time and the health to travel and hence are more inclined to buy a 12 Month Travel Insurance Policy. Another correlation might exist for income. Clients with high income have more means to travel and hence are more inclined to buy a 12 Month Travel Insurance Policy.

However, it is difficult for the current statistical analist, to detect correlations on prediction by specific combinations of features. In the example above, the statistical annalist might miss a specific category of clients. For instance, students. They are young (age between 18 and 24) and have low income. However, students might travel a lot and hence buy a 12 Month Travel Insurance Policy.

This example illustrates the two main problems C-Profile is currently having with the generation of leads using statistical analysis.

1. The statistical analysis might be prone to mistakes, which means the accuracy will decrease.
2. And more importantly, it takes a data scientist too long (weeks/months) to setup a good model for statistical analysis. This will reoccur every time a new dataset becomes available, or if the datasets changes often.

To counter these problems, C-Profile wants to investigate if it is possible to setup a model automatically, using machine learning. The reason for choosing machine learning, is that machine learning has less of a preconception of the features of the data, and hence has less bias towards such features. Secondly it is believed by C-Profile that machine learning is capable of generating such models quickly, i.e. within minutes or hours, without interference from the data scientists.

The research will focus on investigating various machine learning techniques to verify if they have the potential to replace statistical analysis in predicting insurance policy sales opportunities. In order to do so, the machine learning techniques must be able to generate models quickly, with high accuracy.

There are several potential issues that arise when using machine learning, though. One of these issues is scalability. The notion of time complexity states that the runtime of algorithms increases if the size of dataset upon which they act increases. This means that we must ensure that the algorithms we use are scalable in the runtime necessary to generate models. C-Profile aims to regenerate the model on a weekly basis. The time available to generate new models is roughly eight hours (overnight).

Furthermore, we set out to find algorithms that are capable of generating a good model on any dataset. Often it is so that tailor made (statistical) models outperform generic models. However, it costs a lot of time and effort to setup such a tailor made model. Hence we must aim to get similar accuracy with the machine learning techniques (or optimistically better) compared to the current statistical analysis methods (at a fraction of the time).

Finally, we have to investigate if the machine learning techniques can handle irrelevant features. With irrelevant features, we mean features that do not contribute (much) to the chance if a person is interested in a product. If the algorithms are able to handle irrelevant features, we do not have to preprocess the data by hand, thereby possibly losing information we think is irrelevant. This will both save time and (possible) loss of information.

1.3 Goals

Our main research goal is to verify if we can use machine learning for predicting insurance product interest for individual clients, based on a dataset of clients. Currently manual model selection is used in a statistical approach for predicting insurance product interest. We want to investigate if machine learning has the potential to replace statistical analysis for making these predictions.

The second goal of this study is to find out which type of machine learning technique has the highest potential of creating models quickly, which can make accurate predictions on insurance product interest for clients. We want to answer the question: Which technique should we develop further to make our predictions?

The final goal is to create a framework in which; data can be fed to algorithms, algorithms can be trained, algorithm models can be retained, and prediction results can be retrieved. This framework should be independent of the algorithms, such that algorithms can be easily changed or enhanced. We will use this framework for performing experiments, but this framework should be setup in such a way that it can be incorporated in the C-Profile platform.

1.4 Problem statement

The main research question is formulated as follows:

Which machine learning technique has the highest potential to replace statistical analysis for predicting insurance product interest?

We will answer this question using various sub research questions. As discussed in Section 1.2 we have three potential issues by using machine learning techniques. The questions that arise are

1. How accurate can machine learning algorithms predict insurance product interest for different datasets? What we consider accurate will be covered in Chapter 2.
2. How fast can machine learning algorithms create models for prediction of insurance product interest for different datasets? How does the size of the datasets influence the performance in terms of speed for various techniques?
3. How well can machine learning algorithms handle irrelevant features? I.e. features that do not add value to the prediction. Are the techniques able to filter out the important features to make accurate predictions?

1.5 Methodology

In order to answer these questions, we will first give some insight in the type of data on which predictions will be made, and what the desired form of these predictions is. We need to know this to find out which type of machine learning techniques we should focus on. Once we know the type of predictions we want, we can focus on the type of machine learning techniques we can utilize to solve the problem (Chapter 2).

Using this knowledge, we can implement and verify various machine learning techniques to make predictions. Because this is a feasibility study, we will not go in too much depth for each of the techniques. We will create a high level theoretical comparison for the techniques in our application domain (Chapter 3).

To analyze the performance and ultimately answer our research questions, we will detail the experimental framework which allows us to test the algorithms on different datasets, measuring various performance measures. Using various experiments, which will be executed using the experimental framework, we will answer the research questions. (Chapter 4) The experiments will test various aspects of the algorithms such as:

1. The ability to make accurate predictions
2. The ability to create models quickly
3. The ability to filter relevant information

The research questions defined in Section 1.4 will be answered using these experiments. (Chapter 5)

Chapter 2

Background

In this Chapter we will give some background for the problem. In Section 2.1 we will detail the data available. In Section 2.2 we will describe our stakeholders and their requirements. In Section 2.3 we will discuss the type of machine learning techniques are available and in Section 2.4 we detail how we will model the problem, and which type of machine learning techniques we will investigate. Finally, in Section 2.7 we will discuss how to measure the accuracy of predictions and what is considered accurate.

2.1 Data

In order to decide which algorithms we should investigate, we first have to take a look at the available data. The data consists out of rows for each client containing features with properties about the client. These properties range from the date of birth, (estimated) income, employer, family situation, geo-location of living, etc. For each client, we also have binary values stating whether they have bought an insurance product from a category in question. Product categories are for example: health insurance, car insurance, burglary insurance, etc.

In practice the data used should be extracted from the information recorded in the web portal. The data in the portal will have approximately $50 \sim 100$ features and consists out of $100K \sim 1M$ clients (most likely growing in the future). Currently we are recording 32 different categories of insurance products.

However, the statistical analysis performed at C-Profile is still in its experimental phase. This means that analysis has only been performed for companies who were interested enough to fund the experiments. Unfortunately, they were reluctant to hand over all the data they have, which means we have fairly limited data sets (in terms of feature size) to our disposal. This also means that individual data sets have different features. In order to also test the techniques with a richer data set, we will use the TIC Benchmark / CoiL Challenge 2000 dataset.[37] This dataset records features of clients of an insurance intermediary, and a boolean value indicating that they bought a caravan insurance. This is the same type of data we have in other datasets. This dataset however, is far more enriched than the other datasets at our disposal. Unfortunately, the number of clients is fairly low with approximately 10K rows only.

For the experiments we will use a subset of the datasets in Table 2.1. Statistical analysis is performed for datasets C, D and E

Name	Number of clients (m)	Number of features (n)
Dataset A	9822	86
Dataset B	180024	11
Dataset C	74768	17
Dataset D	74768	17
Dataset E	74768	17

Table 2.1: Statistics about the five datasets used during the experiments

2.2 Stakeholders

In order to determine the requirements we must meet, we will identify our stakeholders. This project has three main stakeholders:

- C-Profile
- Insurance intermediaries
- Insurance intermediary clients

Each of these stakeholders have their own requirements and preferences.

C-Profile, the company for which we will develop this solution, wants to be able to provide its insurance intermediaries with leads about insurance products for clients in their portal. For each client, the interest for each product should be calculated. They want to give the advisor a list of probabilities for sales of different products. See Figure 1.2 for a concept of the leads page. The prediction model should be created every week, overnight (i.e. within eight hours) preferably without any interference of a data scientist (so no preprocessing of the data nor parameter selection). Furthermore, a (future) request is the ability to extract the reasoning behind a client's interest in a product, so a story can be created to help intermediaries with their sales.

Insurance intermediaries, the companies who target their clients with the information they gain, want to have insights in which clients to pursue. The problem they have is that they have limited capacity to target every client. Hence, they want to decide who to target. Targeting a client that is not interested costs man hours, but a bigger concern is not targeting a client that is interested, because this is missed revenue. A secondary problem they have is investment in new clients. If they register a new client, they want to be able to quickly determine which insurance products are relevant for this new client. Hence they want to be able to quickly create an overview of the products that should appeal to them.

Clients, the people who will be targeted by insurance intermediaries, only want to be approached for the products they actually want or need. This means that the client does not want to be approached for products the client does not care about.

Each of these stakeholders have different priorities. We are developing this solution for C-Profile. Hence, this is our main stakeholder. Their criteria have the highest priority. The secondary stakeholders are insurance intermediaries and their clients.

The main criteria from the different stakeholders can be summarized in three main categories.

- Speed of model creation, C-Profile demands that at most eight hours are needed to create a model.
- Accuracy of the predictions, intermediaries and clients do not want incorrect predictions. They want neither false negatives (intermediaries) nor false positives (clients).
- Autonomy, C-Profile requires that interference by data scientists is not required. Hence the techniques should be able to filter irrelevant features.

these three criteria result in trade-offs between each other. Speed affects accuracy, while autonomy affects speed and accuracy. It should be noted that for speed we have a hard criteria of eight hours. As long as we meet that criteria, we can focus on accuracy and autonomy. The requirement for autonomy by C-Profile is also quite strict. Hence we will ensure that the techniques will be able to filter irrelevant features, and that no data scientist has to interfere. So the trade-off between these criteria becomes:

1. Autonomy
2. Speed (ensure model can be created within eight hours, as long as that criteria is met, low priority)
3. Accuracy

2.3 Machine learning techniques

Machine learning algorithms can be divided in three categories

- Supervised learning
- Unsupervised learning
- Reinforcement learning

[26]

The distinction is made in the way the algorithms learn their objective function. Supervised learning uses a training set with labeled instances. In other words, it is given a set of instances with their classification. It uses these classified instances, to estimate an underlying function to classify new or unknown instances. Unsupervised learning does not consider the given classification when learning its objective function. It learns completely on its own, the structure in the data. Reinforcement learning on the other hand, interacts with a dynamic environment. It learns its objective function by using feedback it receives from its environment and reacting on it.

In Section 2.1 we state that the datasets used have labels indicating whether a client bought a certain product or not. This means we can quickly focus on supervised learning techniques. However, there are various generalizations in which a supervised learning problem can be modeled. Some of these generalizations are:

- Classification techniques
- Regression techniques
- Learning to rank
- Structured prediction

Classification is a form of machine learning that identifies to which category of instances an unknown instance (a client in our case) belongs. In our case, we could generate two classes per product, one class (usually 1) of clients that are interested in the product, and one class (usually 0) of clients who are not interested in the product. For each product type, we run classification algorithms to determine which client is interested in which product. This type of classification is known as binary classification. To handle ambiguous cases, it is desirable to return a probability.[26] Classification can often be extended with a probability model. Instead of only classifying an instance to belong to a certain class. We can return a score stating the probability the instance belongs to the class in question.

Regression is a technique that tries to explain a relationship between variables. Where classification has a discrete set of possible outputs, i.e. 0 and 1, regression has a real valued output. [26] In case we try to predict a price for a house based on a set of features, a real value makes sense. However, in predicting whether someone wants to buy an insurance product, these values are a bit more difficult to interpret. We could interpret the output as a probability, but the output is not restricted to a domain of $[0, 1]$ which might make it unsuitable for this interpretation. [16]

Learning to rank is a technique that focuses on creating an optimal ordering of items. It is often used in document retrieval.[14] We could use learning to rank to order product categories in predicted interest for every client. In other words, we can create a ranking of products for each client. For products at the start of the list the interest is higher than for products at the end of the list. Similarly, we can create a ranking of clients for each product category. The drawback of ranking is that we do not explicitly predict how much interest exists for each product or for each client. Instead instances are ordered, for instance, using pair wise comparison.[34] This means that the cutoff point for which clients and which products to pursue is difficult to determine.

Structured prediction is a technique that is fairly different from the previous techniques. Instead of predicting a single value or order, structured prediction can predict any arbitrary object. Structured prediction often deals with large (but finite) output spaces.[35] Instead of using classification on a large set of classes, we can use the structure of the output space to make better predictions. Using structured prediction, we could predict the classifications for all product categories at once. Structured prediction is often used in speech to text recognition.

2.4 Approach

Using the information gathered in Sections 2.2 and 2.3, we can determine which type of techniques we will pursue. We will discuss all four techniques and how capable they are to comply with the wishes of the stakeholders.

The advantage of structured prediction is that the algorithms only have to generate one model for prediction of all products. However, we believe that the structure between products does not give enough of an advantage to predict the interest of all products at once. Learning to rank seems like a technique that is usable at first. However, learning to rank only states which products or which clients have the most potential. They do not give insights in how much absolute potential they have. Therefore, it is still difficult to determine which clients and products to pursue. Regression on the other hand does give insights in the interest per product for each client. However, the score regression gives to a product for a client has in infinite range. Therefore, it is difficult to interpret this result, we could cut off the results outside of our range $[0, 1]$ or we could normalize the results, but these methods are not flawless. Classification on the other hand does give a result that is easy to interpret. There are only two categories; 1 (interested) or 0 (not interested). The problem with classic classification is that we do not get an indication about how much interest a client has in a product. Hence we will adapt classification using a probabilistic model. A probabilistic method assigns probabilities that a client belongs to a certain class. In our case, we can simply focus on the probability a client belongs to the class 1 (interested).

Because we will use classification with a probabilistic model, we will have to create a model for each product category. Our research will be focused on fast model creation, resulting in accurate predictions. This means we will not focus too much on prediction speed.

Lets formalize our classification problem as follows. We define a training set $\mathcal{T}$, with m examples $\{\mathbf{x}, y\}$, where $\mathbf{x}$ is a vector of n features about a client and y is the classification of $\mathbf{x}$, with $y \in \{0, 1\}$. $y = 1$ indicates the client is interested in the product and $y = 0$ indicates the client is not interested in the product. We can now define a classifier as a function $h(\mathbf{x}) \in [0, 1]$ where $h(\mathbf{x})$ is interpreted as $p(y = 1 \mid \mathbf{x})$. This indicates the probability that a client with features $\mathbf{x}$ is interested in the product. The function $h(\mathbf{x})$ is refined by *training* the classifier with the training set $\mathcal{T}$.

2.5 Feature normalization

The data available comes in various forms. Features may consist of numbers, dates, text, or as a defined set of values, for instance $\{true, false\}$. Different algorithms have different capabilities of handling input data. However, most algorithms are capable of handling continuous numerical values. Hence we choose to transform where possible our data into a numerical range automatically without interference from a data scientist. For features that have a finite set of values they can attain, we simple number each of the values and assign it to the feature. For text values, we can often do something similar. We can check if the values in the training set are unique or if they occur in multiple instances. If they do occur multiple times, we can treat them as if they are a set of possible values. If strings are mostly unique, this is not a good idea, because no new information is extracted from the feature. There are several techniques to get meaningful information from such a type after all. These are based on distance metrics between a string A and B. Techniques include; editdistance metrics, fast heuristic string comparators, tokenbased distance metrics, and hybrid methods. However, domain knowledge is required to decide which metric should be used and string distances are not suitable for comparing entities with non-trivial structures.[8] Therefore we simple ignore text features with mostly unique values. Examples of such features are: names and description fields.

After converting our dataset to numerical values, only one problem remains. This is the problem of different feature scales. Consider two features, age and $annual\ income$. Age is on the scale $[0, 120]$, whereas $annual\ income$ is roughly on the scale $[0, 1000000]$. Because these scales are so very different, adjusting the parameters in algorithms might become biased. This is especially the case if (Euclidean) distance metrics are used. The feature $annual\ income$ has a much larger impact on the distance between two instances. To counter this effect, we will scale all features to the same range. It is shown that for several algorithms feature scaling obtains a better quality, efficient and more accurate result.[25] Therefore we will scale all features in the $[-1, 1]$ scale.

There are various techniques to perform this transformation. [33] Arguably the simplest technique is Min-Max scaling. [1] For a feature x_j and training set $\mathcal{T}$, $(\mathbf{x}, y) \in \mathcal{T}$ we scale as follows:

$$x'_j = \frac{x_j - min(x_j)}{max(x_j) - min(x_j)} \tag{2.1}$$

Where $min(x_j)$ and $max(x_j)$ are the minimum and maximum value of feature x_j for $\{x, y\} \in \mathcal{T}$ respectively. Scaling all features x_j for $x, y \in \mathcal{T}$ using Min-Max scaling ensures $\forall j \mid (x, y) \in \mathcal{T} : 0 \leq x'_j \leq 1$, which is certainly in the scale $[-1, 1]$. Note that if $min(x_j) = max(x_j)$ it means that all values are equal. Hence this feature does not provide any new information and we can just as well skip the feature entirely.

Another technique is standardization (also known as Z-score normalization)[1] which scales features such that they will have the properties of a standard normal distribution with $\mu = 0$ and $\sigma = 1$, where μ is the mean and σ the standard deviation of x_i for all $(x, y) \in \mathcal{T}$. Hence again we have that this is in the scale $[-1, 1]$. Note that this does not mean we stay within the range $[-1, 1]$ because technically any value is still possible. However, according to the three-sigma-rule, 99.7% of all values lay within three standard deviations of the mean. [30] In our case this means 99.7% lies in the range $[-3, 3]$, which is arguably in the scale $[-1, 1]$.

These two feature scaling techniques are amongst the most common techniques used for feature scaling.[33] This means they are very broadly used and hence are most likely to work well in general.

2.6 Parameter tuning

Most algorithms have certain parameters that influence the quality of their predictions. These parameters are often dependent on the data they are fed. However, as mentioned in Section 1.2, we like to remove the requirement for domain knowledge. Hence we require a method to pick the best parameters for each algorithm. The method we will use is known as parameter tuning (cross validation). Another method is k-fold cross validation, but this is an expensive method (expensive as in it takes a long time).[17]

In parameter tuning, a finite predefined set of parameter settings are tested. The best combination of parameters is chosen for training. We test parameters using cross validation. In cross validation, the training set is divided into two parts. The (new) training set, and the validation set. For all parameter settings, the algorithm is trained using the training set, and tested using the validation set. The settings with the lowest error rate, according to an error measurement, are chosen for training. In training, the entire training set (training set and validation set) will be used again. How to measure errors will be discussed in Section 2.7

2.7 Measuring accuracy

Classification learning normally has two phases; the training phase and the execution phase. In the training phase, the algorithm is fed data with labels. It uses this sample to get some understanding of the data, usually by learning an objective function and minimizing the cost of mispredictions. After the training phase, the algorithm goes through the execution phase. In the execution phase the data is provided without labels. The algorithm predicts the classification of these unclassified instances.

When investigating which algorithms work best, the execution phase is often replaced with a testing phase. In the testing phase, data with labels are available. However, the data fed to the algorithm does not contain these labels, just like in the execution phase. The predictions of the algorithm can be compared to the actual classifications. Then a metric of accuracy can be applied to determine which algorithm is the best. What type of metric to use depends on what is considered accurate.

Accuracy can be defined and measured in different ways. In binary classification, there are four categories a prediction can belong to. These are shown in Figure 2.1. Accuracy is then defined as the ratio between true predictions (TP + TN) and false predictions (FP + FN). The most straightforward way to measure this definition of accuracy is by using the Percental Error (PE) measurement. This measurement measures the percentage of instances that were wrongly classified. PE is calculated as follows:

$$PE = \frac{1}{m}\sum_{j=1}^{m} \mid \lfloor h(x_j) \rceil - y_j \mid \tag{2.2}$$

where $h(x_j)$ is the estimated value of y_j, $\lfloor x \rceil$ is the nearest integer to x (also known as the round function), and $\mid x \mid$, the absolute value of x.

However, we are using a probabilistic model on top of the classification. Hence we might not only want to penalize wrong predictions, but also take the probabilities into account. To take probabilities into account we can use the Mean Squared Error (MSE) measurement. The MSE takes into account how close the prediction was to the actual classification. MSE is calculated as follows:

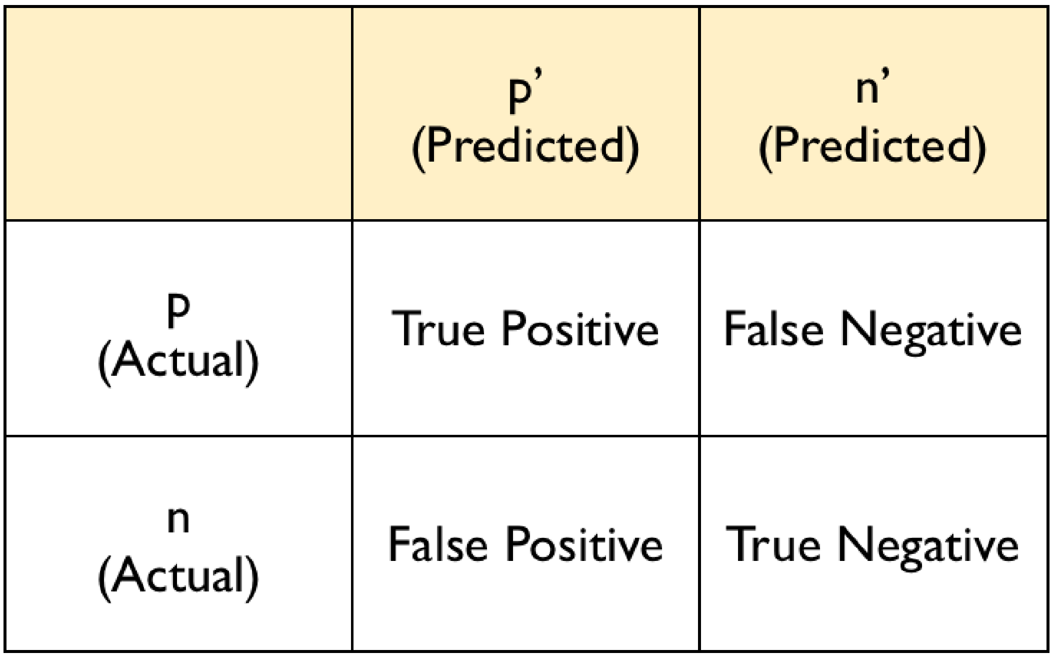
$$MSE = \frac{1}{m}\sum_{j=1}^{m}(h(x_j) - y_j)^2 \tag{2.3}$$

This measurement takes into account the probabilities the algorithms output. If an algorithm gave a low probability of its classification, the error is less, than when a high probability was given.

PE is an ideal measurement for communication to executives and insurance intermediaries, because it is easy to understand. While MSE is a bit more sophisticated as it takes the probabilities into account. This measurement effectively gives more information about how far off prediction were.

Accuracy however, is not always a good measure of the quality of classification models. [9] This is especially true if the amount of observed training examples do not have a similar number of instances belonging to each class. Consider a dataset containing 10000 clients of which only 500 bought a product. A classifier predicting a client is not interested for every client, has an accuracy of 0.95 (or a PE of 0.05). This is a fairly high accuracy even though the classifier is useless.

Hence we will use two more measures; precision and recall. Precision measures what percentage of positive predictions were correct. This measurement is a good fit from the client perspective. They do not want to



	p' (Predicted)	n' (Predicted)
p (Actual)	True Positive	False Negative
n (Actual)	False Positive	True Negative

Figure 2.1: The four categories a prediction can belong to.

be bothered for products they are not interested in. Clients prefer a high Precision. Precision is defined as follows:

$$P = \frac{TP}{TP + FP} \tag{2.4}$$

[9]

Recall on the other hand measures what percentage of the positive cases we predict correctly. This measurement is a good fit from the insurance intermediary perspective. They mostly care about targeting every client that is interested. Insurance intermediaries prefer a high Recall. Recall is defined as follows:

$$R = \frac{TP}{TP + FN} \tag{2.5}$$

[9]

In our experiments, we will mostly focus on the PE and MSE measurements, but we will take precision and recall into consideration.

Chapter 3

Comparison of techniques

Several machine learning techniques exist that could be applied to our problem. In Section 3.1 we will briefly discuss some of the techniques available and which ones we will use. In Appendix A a more comprehensive explanation of each algorithm is given. In Section 3.2 we will make a theoretical comparison between the various techniques and algorithms.

3.1 Techniques

Various classification techniques exist. One categorization of techniques is:

1. Statistical learning algorithms
2. Regression techniques
3. Instance based learning
4. Clustering
5. Perceptron based techniques
6. Support Vector Machines
7. Logic based algorithms

[21]

These seven techniques have different methods of finding their classifications, but they can all be used in the same framework, because they all have a training and execution phase. This makes it easy to compare the techniques. Most of them also have various parameters to set, hence parameter tuning can be applied as well. See Figure 3.1 for the categories and algorithms that belong to those categories.

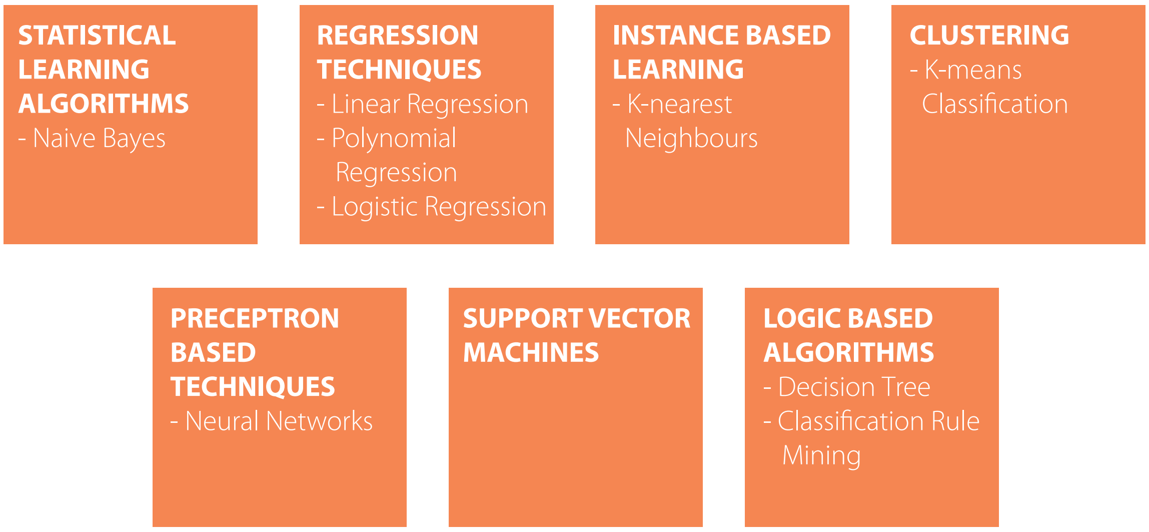


Figure 3.1: Seven classification techniques and some algorithms that belong to those categories.

3.1.1 Statistical learning algorithms

Statistical learning algorithms are based on an underlying statistical model. One of the algorithms in this category is the Naive Bayes algorithm. Naive Bayes is based on Bayes' rule. This rule is defined as follows:

$$p(Y \mid X) = \frac{p(X \mid Y)\,p(Y)}{p(X)} \tag{3.1}$$

This rule states that the probability of product interest ($Y = 1$) given a certain person (features X) can be calculated using the inverse $p(X \mid Y)$. $p(X \mid Y)$, $P(Y)$ and $P(X)$ are calculated using the training set.

Naive Bayes uses Bayes' rule to classify instances. If $P(Y = 1 \mid X = \mathbf{x}) \geq 0.5$ the instance $\mathbf{x}$ is classified as $Y = 1$ (i.e. client with features x is interested in the product), otherwise the instance $\mathbf{x}$ is classified as $Y = 0$.

3.1.2 Regression techniques

Regression is a method to fit a (linear) equation to the data of the form:

$$h(\mathbf{x}) = \boldsymbol{\beta}^T\mathbf{x} \tag{3.2}$$

$\mathbf{x}$ are the features of an instance. $\boldsymbol{\beta}$ is a vector of parameters. This vector shows the dependency of each feature on the classification. The parameters are found during the training phase by optimizing a cost function. The exact cost function depends on the type of regression, but usually it is a form of minimizing the amount of mispredictions.

The most straightforward method is Linear Least Squares Regression. This method fits a linear equation to the data. A feature x_0 is introduced such the line to fit the data does not have to go through the origin. The cost function can be defined as follows:

$$J(\boldsymbol{\beta}, \mathcal{T}) = \left(\sum_{i=1}^{m} (h_\beta(x^{(i)}) - y^{(i)})^2 + \lambda \sum_{j=1}^{n} \boldsymbol{\beta}_j^2 \right) \tag{3.3}$$

[16]

A bit more complicated method is Polynomial Least Squares Regression. This method has the same cost function as Linear Least Squares, but enriches the features before optimizing the cost function. Features can be combined to create polynomial features, on which linear least squares is performed. This allows for more complex models.[3] For instance for a polynomial degree of two the features generated can be defined as follows:

$$\forall i \in \{i \mid 0 \geq i \geq m\} \forall j \in \{j \mid i \geq j \geq m\} x_i x_j$$

(See Algorithm 2 for an algorithm to generate these polynomials for polynomial degree $\rho \geq 2$.)

A more complex method is Logistic Regression. Linear- and Polynomial Regression are technically regression techniques. Logistic Regression however, is a classification technique by design. Rather than modeling the response y directly, Logistic Regression models the probability that y belongs to a particular category.[16]

Logistic Regression has a more complex cost function, that penalizes mispredictions heavily. The cost function can be defined as follows:

$$J(\boldsymbol{\beta}, \mathcal{T}) = -\frac{1}{m} \left(\sum_{i=1}^{m} y^{(i)} \log h_\beta(x^{(i)}) + (1 - y^{(i)}) \log(1 - h_\beta(x^{(i)})) \right) + \frac{\lambda}{2m} \sum_{j=1}^{n} \boldsymbol{\beta}_j^2 \tag{3.4}$$

[28] Logistic Regression and Polynomial Regression can be combined to create an even more powerful method.

The statistical analysis currently used by the data analysis department of C-Profile is also a form of regression.

3.1.3 Instance based learning

Instance based learning algorithms are known as lazy learners. This because most of the work is delayed until the execution phase. K-nearest Neighbours (KNN) is a simple instance based learning algorithm. KNN classifies an instance by searching its k nearest neighbours based on a distance metric. The instance is then classified by a (weighted) average of the classifications of the neighbours. This can be formalized as follows. Given a set of k nearest neighbours K:

$$h(\mathbf{x}) = \frac{1}{\|K\|} \sum_{\{x_z, y_z\} \in K} y_z \tag{3.5}$$

A weighted version can also be considered if proximity of points to the instance $\mathbf{x}$ is considered important:

$$h(\mathbf{x}) = \frac{\sum_{\{x_z, y_z\} \in K} \frac{1}{d(x, x_z)^2} y_z}{\sum_{\{x_z, y_z\} \in K} \frac{1}{d(x, x_z)^2}} \tag{3.6}$$

[21] where $d(x, z)$ is a distance measure, Euclidean distance in our case.

3.1.4 Clustering

The fourth technique is clustering. Clustering is typically used as unsupervised learning algorithms.[6] Clustering algorithms group instances that are similar together. This means clustering techniques find structure in unstructured data. We can also use this result to classify instances, if we add a classification to these groups or clusters. The classification of a cluster of points C is defined as follows:

$$h(\mathbf{x}) = \frac{1}{\|C\|} \sum_{\{x, y\} \in C} y \tag{3.7}$$

K-means Classification is a clustering technique that can be used for classification. It is an algorithm that finds k clusters based on proximity of the points. It chooses k random points and tries to form clusters. If the clusters are formed it tries to find better clusters, until it cannot improve the quality of the clusters. It uses a distance metric to determine clusters. The distance used is usually Euclidean distance. The cost function (the function to optimize) for creating clusters is defined as follows:

$$J(\mathcal{T}, k) = \sum_{i=1}^{k} \sum_{x \in C_i} \|x - \mu_i\|^2 \tag{3.8}$$

[4]

3.1.5 Perceptron based techniques

Perceptron based techniques are similar to regression in that they try to fit an equation to the data and find their parameters $\boldsymbol{\beta}$ by optimizing an objective function. The functions used however, can be very different. A fairly new but by now common method is Neural Networks. Neural Networks is a technique that emulates how the human brain learns. Neurons, axons and dendrite are translated to computational units, input wires and output wires. The objective function searches the weights for these input wires.

a Neural Network often consists of a couple of layers with neurons. Neurons of layer 1 only have outgoing connections to neurons from layer 2, and neurons from layer 2 only have outgoing connection to layer 3, etc. The connections are known as axons. Each axon has a numerical value indicating the weight and each neuron has an activation function $a_\beta(x)$:

$$a_\beta(x) = tanh(\boldsymbol{\beta}^T \mathbf{x}) \tag{3.9}$$

$$tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}} \tag{3.10}$$

bishop1995neural In this case $\mathbf{x}$ indicate the value from the incoming neurons and $\boldsymbol{\beta}$ the weights from the incoming axions. A classification for a client $\mathbf{x}$ is computed by setting the values of the features in the first layer (also known as the input layer) and computing the activation function of the neuron in the last layer (also known as the output layer). This activation function is dependent on the calculations in the layers between.

In order to formalize the classification a Neural Network does we will introduce some notation. The number of layers in our neural network will be denoted by L. We will introduce matrices $\boldsymbol{\beta}^{(j)}$ as the matrix of parameters (or weights) controlling function mapping from layer j to layer $j+1$. Note that the size of $\boldsymbol{\beta}_j$ depends on the number of neurons in layers j and $j+1$. If s_j denotes the number of neurons in layer j. We get that $\boldsymbol{\beta}_j$ is a $s_{j+1} \times s_j + 1$ matrix. Furthermore we will use $a_i^{(j)}$ as the activation of neuron i in layer j. The activation is the result of the neuron's activation function. for layer 1, the input layer, we will set $a_i^{(1)} = x_i$ Finally we will introduce $z_i^{(j)}$ as the weighted linear combination of input values in a neuron i in layer j.

$$z_i^{(j)} = \sum_{k=0}^{s_j} \boldsymbol{\beta}_{i,k}^{(j-1)} a_k^{(j-1)} \tag{3.11}$$

The concrete calculation of $a_i^{(j)}$ becomes:

$$a_i^{(j)} = tanh(z_i^{(j)}) \tag{3.12}$$

[3]

The classifier is then simply defined as

$$h(\mathbf{x}) = a^{(L)} \tag{3.13}$$

3.1.6 Support Vector Machines

Another technique is Support Vector Machine. A Support Vector Machine is known as a large margin classifier. This means it tries to separate the data such that the line separating them is as far away from the nearest points as possible. The hyperplane seperating the data is used in the prediction:

$$h(x) = \begin{cases} 1 & \text{if } \boldsymbol{\beta}^T \mathbf{x} \geq 0 \\ 0 & \text{if } \boldsymbol{\beta}^T \mathbf{x} < 0 \end{cases} \tag{3.14}$$

The hyperplane separating the data is chosen such that the gap between the separating hyperplane and the closest points on either side of the hyperplane are as large as possible. This maximizes the likelihood that unseen points closer to the hyperplane will still be classified correctly. This also means that the hypeplane is defined by these two closest points. The points are known as support vectors. The classifier is trained using the following condition:

$$\forall i \in \{i \mid 1 \leq i \leq m\} y^{(i)} \boldsymbol{\beta}^T \mathbf{x}^{(i)} - (1 - y^{(i)}) \boldsymbol{\beta}^T \mathbf{x}^{(i)} \geq 1 \tag{3.15}$$

This condition can only be met if the data is linearly separable. This however, is not always the case. Hence a hinge loss function $L(\boldsymbol{\beta}, \mathbf{x}, y)$ is introduced. We want to minimize for all $\{x, y\} \in \mathcal{T}$:

$$L(\boldsymbol{\beta}, \mathbf{x}, y) = \max\left(0, 1 - (y\boldsymbol{\beta}^T \mathbf{x} - (1 - y)\boldsymbol{\beta}^T \mathbf{x})\right) \tag{3.16}$$

Using this hinge loss function we can define the cost function $J(\boldsymbol{\beta}, \mathcal{T})$ to optimize:

$$J(\boldsymbol{\beta}, \mathcal{T}) = \frac{C}{m} \left(\sum_{i=1}^{m} L(\boldsymbol{\beta}, \mathbf{x}^{(i)}, y^{(i)}) \right) + \frac{1}{2} \|\boldsymbol{\beta}\|^2 \tag{3.17}$$

[36]

Support Vector Machines also have a trick to transform the data such that a non-linear seperating hyperplane can be defined. This is done using a kernel. Using a kernel the points can be transformed to m dimensional points, hence raising the dimensionality of the problem. This means non-linear hyperplane (in n dimensions) can be defined. A commonly used kernel is the Gaussian kernel (also know as the Radial Bias Function)[2].

$$K(\mathbf{u}, \mathbf{v}) = \exp\left(\frac{\|u - v\|^2}{2\sigma^2}\right) \tag{3.18}$$

3.1.7 Logic based algorithms

Logic based algorithms are techniques that classify instances based on a set of logic based rules. The rules state exactly what values certain features of a client need to have in order to classify them as interested. Using a rule based system ensures we have flexibility. This because we can classify our instances by only using a subset of the features, or even using non-continuous ranges.

Decision Tree is a technique to generate such rules. Using the training set; a tree is build up. The tree is growing by recursively partitioning the data using binary tests.[32]. At every node, a decision is made in which direction the path should continue. At the end of a path, in the leaves, a classification is given. The rules are defined by all the paths in the tree. The set of rules of a Decision Tree T can be defined by:

$$\{\bigwedge n \in \mathit{ancestors}(l) \mid l \in T \wedge \mathit{children}(l) = \emptyset\}\} \tag{3.19}$$

where $\mathit{ancestors}(n)$ is the set of nodes on the path from the node n to the root (including the root itself) and $\mathit{children}(n)$ is the set of direct descendants of the node n.

Another technique is Classification Rule Mining. The Apriori algorithm is an algorithm to find such rules. It uses frequent item sets to generate rules. Frequent item sets are combinations of features that occur often together. If they often occur together with a given classification, it is likely that they have an influence on that classification. Rules are found using the anti-monotonicity of itemsets, ”if an itemset is not frequent, any of its superset is never frequent" [38]. If we only consider the set of positive rules R (i.e all rules such that $I \Rightarrow 1$) we can define our classifier as follows:

$$p(y \mid x) = \frac{\sum\limits_{\{I,y\} \in R : I \subseteq X} s_I \times c_{\{I,y\}}}{\sum\limits_{\{I,y\} \in R : I \subseteq X} s_I} \tag{3.20}$$

where s_I is the support of I and $c_{\{I,y\}}$ the confidence of $I \Rightarrow y$. See Appendix A.10 for definitions of support and confidence.

3.2 Comparison

The chosen algorithms belonging to one of the seven common techniques have various advantages and disadvantages. we will start with listing the advantages and disadvantages of the techniques themselves. Afterwards we judge all algorithms on four criteria in Table 3.1. The criteria the algorithms are judged upon are:

- Model complexity, can the algorithms create complex models or only make the most straight forward predictions.
- Conversion speed, are the algorithms able to create models quickly or does it take a long time?
- Probability model, do the algorithms have a probability model? Do we have to make some assumptions to get something resembling a prbabability? Or is a probability model not (yet) present.
- Parameters, do the algorithms have many parameters? And do these parameters have a big influence on prediction?

The advantage of Statistical learning algorithms is mostly their simplicity. This means the methods are often very fast. However, this simplicity also has some down sides. Simplistic methods often generalize too much, hence they often do not fit very well in a concrete dataset. This means that although the speed is very fast, the accuracy is most likely not good. Besides that, Naive Bayes assumes a normal distribution, which means it will have a hard time with biased datasets that are not normally distributed. The algorithms does not have any parameters to tune.

The advantages of Regression techniques are their mathematical soundness and broad experience with the method in the scientific community. The method (mostly Logistic Regression, but also other forms) is also used in statistical analysis. Hence its potential certainly exists. However, this also suggests that model selection is quite important. Furthermore, due to the nature of regression forms, they are not able to handle non-continuous ranges very well. Hence the ordering of categorized data makes a difference in the capability to classify (hence additional features might be required). This means a bias is introduced. The accuracy is most likely better than those of Bayesian methods. The speed of the algorithms probably varies a lot. Where Polynomial Regression will take a long time if the dataset grows in n (the number of instances). The probability model of Linear Regression and Logistic Regression is not completely sound, because their output range is not in $[0, 1]$. Hence assumptions are made, and perhaps errors are introduced.

Instance based learning methods can quickly generate a model and hence are very fast methods. The model created by K-nearest Neighbours is also very easy to understand. Its accuracy is hard to predict however as it likely strongly depends on the parameters used.

The main advantage of clustering techniques is that it does not have a bias towards classifications, because at its core its an unsupervised learning algorithm. Furthermore, the models generated can be complex, but they are still fairly easy to understand. The K-means algorithm is also fast if implemented correctly.

The perceptron based techniques are amongst the more complex techniques. Neural Networks have the capability to generate very complex methods. This is a major advantage, given that they should be able to handle various datasets with different properties. This however, also makes it fairly difficult to understand the model and which features impose certain classifications.

Support Vector Machines is one of the most complex methods for classification. Support Vector Machines (SVM) also have the capability to generate very complex methods. However due to nature of SVMs, the algorithm is fairly slow when using large datasets. This might be a deal breaker for real world usage. A second disadvantage, is that SVMs only predict classes and not probabilities. SVM also requires a lot of parameter tuning, this makes its model creation runtime even slower.

Logic based learning's main advantage is flexibility. It is perfectly capable of generating complex models, including non-continuous ranges. The models, however, are easy to understand. There are disadvantages using this technique as well though. Its decision-making process is strongly influenced by the first choice made. Furthermore, it is fairly difficult to determine which rules are meaningful or meaningless. Hence it is difficult to predict the accuracy of these techniques. Similarly, the speed of the algorithms is difficult to determine. Classification Rule Mining might become very slow if a lot of frequent item sets exist. Decision Trees on the other hand becomes slow, if a lot of decisions have to be made. Another disadvantage of the Decision Tree algorithm is that no probability model exists for the version used (C4.5)[15]. However theoretically it should be possible to introduce a probabilistic model.

Algorithm	Model complexity	Conversion speed	Probability model	Parameters
Naive Bayes	--	++	+	++
Linear Regression	-	+	o	+
Polynomial Regression	o	-	o	+
Logistic Regression	o	o	+	o
K-nearest Neighbours	o	+	+	-
K-means Classification	+	+	+	o
Neural Network	++	o	+	+
Support Vector Machine	++	--	--	-
Decision Tree	+	+	-	+
Classification Rule Mining	++	o	+	--

Table 3.1: Comparison of ten classification algorithms. The algorithms are judged on four criteria. Scores can take one of five values: --, -, o, +, ++ (worst to best)

Chapter 4

Experimental setup

To answer the research questions defined in Section 1.3 we have defined some experiments. In this chapter, we will first give an outline of our experimental setup in Section 4.1. After that we will give an outline of the experiments designed for each question. For each experiment, we will give a table showing the configuration. In this configuration, we show the algorithm(s), dataset, feature scalar and error measurement. The dataset will be split up using a 2 - 2 - 3 ratio in training set, validation set, and test set, unless we explicitly mention otherwise. Furthermore, we will limit the runtime of the algorithm to eight hours, which is considered reasonable time.

4.1 Experimental setup

To facilitate the execution of the experiments we have designed an experimental environment. The environment is setup in a generic way such that the algorithms itself do not have to worry about measurements of their performance. The environment follows a standard pipeline for supervised learning.[31]

4.1.1 Components

The experimental environment consists of five elements:

1. Input readers
2. Feature scalars
3. Algorithms
4. Error measurements
5. Output writers

For each of these elements one concrete implementation should be provided. Algorithms are the only exception. The experimental environment can handle multiple algorithms at the same time. Each of the five elements are self-explanatory.

Input readers read data from a file, database, or any other data source. For instance, a CSV file, Excel file or a MySQL database. Their only goal is to transform such a data source to a concrete training set.

Feature scalars scale the data to a $[-1, 1]$ scale. Examples are the MinMax scalar and the Standardization scalar as discussed in Section 2.5.

Algorithms are the actual algorithms discussed in Chapter 3 to be tested. The algorithms have four operations which will be called by the experimental environment.

1. **preProcess**, to alter the dataset, if needed.
2. **setParameters**, to set different values of parameters of the algorithm.
3. **train**, to train the model, using a training set.
4. **execute**, to predict the result of an unknown instance.

Error measurements, are measurements to determine the error an algorithm made during the test phase. They can be used to rank the algorithms (i.e. during the tuning phase) or to determine a confidence of the algorithm. Examples are the Percental Error (PE) measurement and the Mean squared Error (MSE) measurement as disused in Section 2.7.

Output writers simply output results to the console, a file or a database. This can either be classification results or meta data of the algorithm. For instance, the error measurements, running time, etc.

4.1.2 Phases

If a run has all its components, its experiment can be executed. The procedure consists of:

1. Data extraction
2. Pre-processing
3. Feature scaling
4. Tuning
5. Training
6. Testing
7. Output

as shown in Figure 4.1. For each algorithm, the entire procedure is repeated. For the first three phases (up to feature scaling), the entire dataset is used. After that the dataset is split into three equal parts, the training set, the validation set and the test set. The training set is used for training during the tuning phase. The

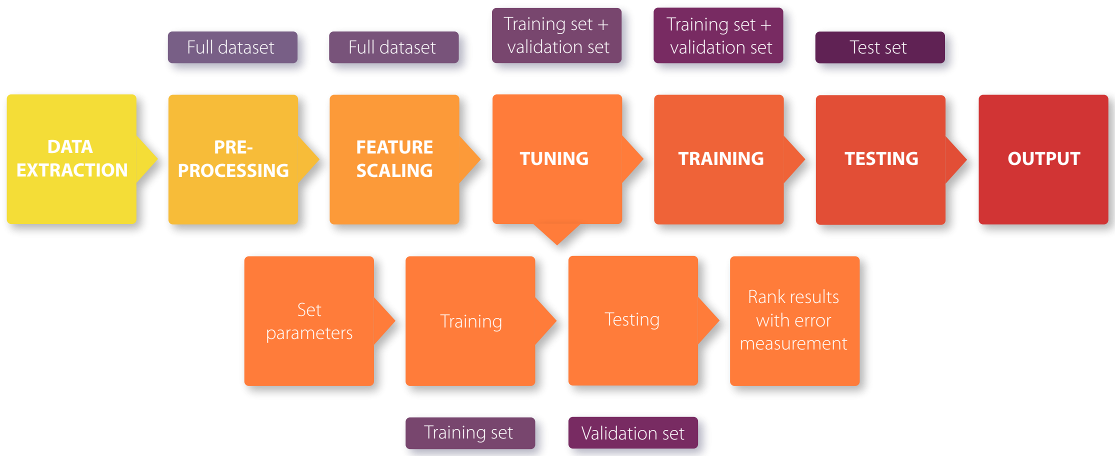


Figure 4.1: Graph showing the phases of the experimental procedure. The purple blocks show which part of the dataset is used in each phase.

validation set is used to test during the tuning phase. For each parameter, the results are ranked using the error measurement. The best ranked parameter setup is then chosen for the next phases.

After tuning, the training phase comes in. The algorithm is trained with both the training- and the validation set. After training the algorithm is tested using the test set, and its results are measured using the error measurement. Finally, the results are fed to the output writer and the procedure ends. During all phases (except feature scaling) the performance is measured. This includes running time and memory usage. These results are also fed to the output writer.

If the experimental procedure is executed for all algorithms, the experiment simply terminates.

4.2 How accurate can machine learning algorithms predict insurance product interest for different data-sets?

To answer this question, we have to determine which configurations give the best results. This means we have to determine what we should optimize in tuning phase. Concretely this means we have to determine which error measurement used in the tuning phase results in the best algorithm configuration. Similarly, we have to determine which feature scalar results in the best accuracy.

If we know which configurations to use, we can determine which algorithms have the best potential, by comparing the accuracy to each other. We will also compare the accuracy to statistical analysis already performed on various datasets. Hence we will design four experiments to answer the following questions.

1. Should we optimize for correct classifications or for good probabilities to get the best accuracy of predictions?

2. How does feature scaling influence the algorithms and their prediction accuracy? Which feature scalar results in the best accuracy of predictions?
3. Which algorithms have the highest potential for accurate predictions?
4. Can we predict insurance product interest for clients, with similar or even higher accuracy compared to current statistical models?

4.2.1 Should we optimize for correct classifications or for good probabilities to get the best accuracy of predictions?

As explained in Section 2.7, we have two error measurements implemented to rank algorithms. These measurements are the PE and MSE. Not only are they used to benchmark the algorithms, they are also used to determine the best parameters during the tuning phase. Therefore, they have a significant influence on the performance of the algorithms.

We propose an experiment to verify which error measurement gives the best results in the tuning phase. We will tune all algorithms with both error measurements. For each error measurement, we determine the best configuration, after which we test the algorithms with both error measurements again. Whichever configuration scores best on both error measurements in the testing phases determines which error measurement works best.

The experiment will test all algorithms on subsets of all four datasets with $m = 7000$, using the standardization feature scalar. See Table 4.1 for an outline of the experiment.

Expected results

We expect that the MSE measurement has a better influence in the prediction phase, because this measurement takes into account how far off the hypothesis is, compared to the expected outcome. Note that this is only useful if the algorithm outputs probabilities. Otherwise, the PE and MSE output exactly the same values.

Even though we expect that the MSE measurement is better for the accuracy during the tuning phase, we believe that the PE is not useless. This because the PE is an easy to interpret. Especially for communication to the insurance intermediary companies who will use the results.

4.2.2 How does feature scaling influence the algorithms and their prediction accuracy?

As mentioned in Section 2.5 we have two feature scalars to scale the data to a $[-1, 1]$ range. It is stated that feature scaling improves the results for various algorithms. [25] However, we do not know which scalar works best, and to which extend the results are improved. Feature scalars transform the data, and because the algorithms are data driven, it is crucial to determine the effects of data transformation, in effect choosing which feature scalar works best.

Algorithm	**Dataset**	**Feature scalar**	**Error measurement**
ALL	A (with $m = 7000$)	Standardization Scalar	• Percental Error • Mean Squared Error (during testing only)
ALL	B (with $m = 7000$)	Standardization Scalar	• Percental Error • Mean Squared Error (during testing only)
ALL	C (with $m = 7000$)	Standardization Scalar	• Percental Error • Mean Squared Error (during testing only)
ALL	D (with $m = 7000$)	Standardization Scalar	• Percental Error • Mean Squared Error (during testing only)
ALL	A (with $m = 7000$)	Standardization Scalar	• Percental Error (during testing only) • Mean Squared Error
ALL	B (with $m = 7000$)	Standardization Scalar	• Percental Error (during testing only) • Mean Squared Error
ALL	C (with $m = 7000$)	Standardization Scalar	• Percental Error (during testing only) • Mean Squared Error
ALL	D (with $m = 7000$)	Standardization Scalar	• Percental Error (during testing only) • Mean Squared Error

Table 4.1: The eight configurations run for the benchmark experiment. All ten algorithms are used, and an additional error measurement is used during the testing phase.

We propose an experiment to test which feature scalar works best. We will tune train and test, all algorithms with three possible feature scalars. The Min-max scalar, Standardization scalar, and a dummy scalar, which does not transform the data. The experiment will use all four datasets with $m = 7000$, using the MSE error measurement. See Table 4.2 for an outline of the experiment.

Algorithm	Dataset	Feature scalar	Error measurement
ALL	A (with $m = 7000$)	Min-max Scalar	Mean Squared Error
ALL	B (with $m = 7000$)	Min-max Scalar	Mean Squared Error
ALL	C (with $m = 7000$)	Min-max Scalar	Mean Squared Error
ALL	D (with $m = 7000$)	Min-max Scalar	Mean Squared Error
ALL	A (with $m = 7000$)	Standardization Scalar	Mean Squared Error
ALL	B (with $m = 7000$)	Standardization Scalar	Mean Squared Error
ALL	C (with $m = 7000$)	Standardization Scalar	Mean Squared Error
ALL	D (with $m = 7000$)	Standardization Scalar	Mean Squared Error
ALL	A (with $m = 7000$)	Dummy Scalar (no scaling)	Mean Squared Error
ALL	B (with $m = 7000$)	Dummy Scalar (no scaling)	Mean Squared Error
ALL	C (with $m = 7000$)	Dummy Scalar (no scaling)	Mean Squared Error
ALL	D (with $m = 7000$)	Dummy Scalar (no scaling)	Mean Squared Error

Table 4.2: The twelve configurations run for the feature scaling experiment. All ten algorithms are used.

Expected result

We expect that feature scaling only slightly influences the results. Even though the scales become easier to compare when scaling, we believe most algorithms will be capable handling different scalars. On the other hand, we do expect that the Standardization scalar has a better influence on the prediction accuracy of some of the algorithms. This because the standardization scalar scales to a normal distribution, which is assumed in various algorithms, such as Naive Bayes (Appendix A.1) and Logistic Regression (Appendix A.4).

4.2.3 Which algorithms have the highest potential for accurate predictions?

This is the main experiment to determine which techniques to pursue after this study is completed. The experiment is simple. Using the results of Experiments 4.2.1 and 4.2.2 to determine our configuration, we will run all algorithms on all four datasets. The accuracy of the algorithms will be used as metric. See Table 4.3 for an outline of the experiment.

Algorithm	Dataset	Feature scalar	Error measurement
ALL	A	TBD (Experiment 4.2.2)	TBD (Experiment 4.2.1)
ALL	B	TBD (Experiment 4.2.2)	TBD (Experiment 4.2.1)
ALL	C	TBD (Experiment 4.2.2)	TBD (Experiment 4.2.1)
ALL	D	TBD (Experiment 4.2.2)	TBD (Experiment 4.2.1)

Table 4.3: The four configurations run for the potentiality experiment. All ten algorithms are used on the full datasets.

Expected results

There are various algorithms that have a high potential to be very accurate. The most accurate algorithms are most likely the more complex algorithms, because these have been developed for a reason. Hence we have high expectations for Neural Networks (Appendix A.7) and SVM (Appendix A.8). On the other hand, we believe that KNN (Appendix A.5) and Naive Bayes (Appendix A.1) are too simple to result in very accurate predictions. The various regression forms (Appendices A.2 to A.4) will probably also perform well, with increasing accuracy. The other three algorithms: Decision Tree (Appendix A.9), Classification Rule Mining (Appendix A.10) and K-means Classification (Appendix A.6) are more difficult to predict. They could be really good or really disappointing.

4.2.4 Can we predict insurance product interest for clients, with similar or even higher accuracy compared to current statistical models?

This experiment is the most important experiment of this paper. It determines whether C-Profile should pursue machine learning or stick with statistical analysis as a technique to generate advice-leads about client interest in insurance products. We will test the algorithms ability to make predictions to the current statistical analysis performed at C-Profile. The experiment is similar to Experiment 4.2.3, with the exception that we will use the datasets C, D and E, because these are the only datasets for which statistical analysis is performed. The experiment is simply a comparison of the machine learning techniques and the statistical analysis. The configuration will be the same as in Experiment 4.2.3. The results of all techniques will be compared using the two error measurements. See Table 4.4 for an outline of the experiment.

Algorithm	Dataset	Feature scalar	Error measurement
ALL	C	TBD (Experiment 4.2.2)	TBD (Experiment 4.2.1)
ALL	D	TBD (Experiment 4.2.2)	TBD (Experiment 4.2.1)
ALL	E	TBD (Experiment 4.2.2)	TBD (Experiment 4.2.1)

Table 4.4: The three configurations run for the comparison experiment. All ten algorithms are used on the full datasets.

Expected results

We expect that it will be difficult to get better accuracy than tailor made statistical analysis on the datasets. However, we do believe that we can get close to the accuracy of statistical analysis (within 5%).

4.3 How fast can machine learning algorithms create models for prediction of insurance product interest for different datasets?

To answer this question, we have to determine which factors determine the speed of the algorithms. One of the components that can influence the speed of the algorithms is feature scaling.[25] So we need to answer the question: How does feature scaling influence the model creation speed of the algorithms? We can use the configuration of Experiment 4.2.2 to determine which feature scalar results in fast algorithms.

A huge factor in the time it takes algorithms to make predictions, is the tuning phase. If we can eliminate this phase, some of the algorithms become a factor 100 faster. However, we can only eliminate this phase if the parameters are always the same (or at least similar) for different datasets. Hence the following question is raised: How much do various parameters influence the prediction accuracy of the algorithms? Can we eliminate the tuning phase?

A third factor in speed is the size of the dataset. The notion of time complexity states that the algorithms running time is influenced by the size of the data. Hence we have to investigate how much this influences the speed of the algorithms. Hence the following question comes to mind: How well does the model creation time of algorithms scale to large datasets?

Hence we define the following sub questions.

1. How does feature scaling influence the model creation speed of the algorithms? Which feature scalars result in the fastest model creation times?
2. How much do various parameters influence the prediction accuracy of the algorithms? Can we eliminate the tuning phase?
3. How well does the model creation time of algorithms scale to large datasets?

To answer these three questions, we will reuse Experiment 4.2.2 to measure the effect of feature scalars on the speed of the algorithms. To answer the other two questions, we will design the following two experiments.

4.3.1 How much do various parameters influence the prediction accuracy of the algorithms? Can we eliminate the tuning phase?

Tuning is one of the largest factors in total runtime for most algorithms. Some algorithms have to estimate multiple parameters (and hence combinations of parameters) resulting in a large amount of parameter tuning rounds. SVM, for instance, has two parameters C and σ which are both tested for 13 values. This results in $13 \times 13 = 139$ tuning rounds. Hence it might be worth checking the influence of the various parameters, to verify if tuning is necessary. We also wish to verify if a certain value of a parameter is always the best value, or if it really does depend on the data.

We propose an experiment to record the errors during the tuning phase to verify if the effect of the various parameters. This experiment will run for all algorithms except Naive Bayes (Appendix A.1), because Naive Bayes does not have any parameters to tune. The experiment will run on all datasets with $m = 7000$ to verify the dependence of parameters on the data. See Table 4.5 for an outline of the experiment.

Algorithm	Dataset	Feature scalar	Error measurement
ALL (except Naive Bayes)	A (with $m = 7000$)	Standardization scalar	Mean Squared Error
ALL (except Naive Bayes)	B (with $m = 7000$)	Standardization scalar	Mean Squared Error
ALL (except Naive Bayes)	C (with $m = 7000$)	Standardization scalar	Mean Squared Error
ALL (except Naive Bayes)	D (with $m = 7000$)	Standardization scalar	Mean Squared Error

Table 4.5: The four configurations run for the tuning experiment. All ten algorithms are used except Naive Bayes. During the tuning phase, all errors are reported for analysis

Expected results

Unfortunately, we expect that tuning does have a significant influence on the accuracy of the various algorithms. Not every parameter will have the same amount of influence. It is difficult to predict which parameters will have a high influence. We also expect that the best parameter settings do depend on the provided data. We expect that the amount of reduction that can be done in the tuning phase is minimal.

4.3.2 How well does the model creation time of algorithms scale to large datasets?

Accuracy of the algorithms is very important, but the algorithms should also be able to run on large datasets in reasonable time. Datasets can be large in two dimensions. These are the number of instances (m) and the number of features (n).

We propose an experiment to test the influence of these two dataset parameters on the run time of the algorithms. In this experiment, we will not take accuracy into account. We will use dataset A to experiment with various values of n. We will use the dataset with $m = 7000$ and values of n in the set $\{11, 22, 44, 86\}$. On the other hand, we will use dataset B with various values of m in the set $\{7000, 21000, 63000, 180024\}$. See Table 4.6 for an outline of the experiment.

Expected results

Creating expectations for this experiment are difficult. However, we do expect that SVM (Appendix A.8) might have trouble with large values of m, because its kernel transforms the problem into an m dimensional problem. For large values of m this might blow up the runtime. On the other hand, we believe that KNN (Appendix A.5) might have problems with large values of n. This because the kd-tree becomes high dimensional and therefore complicated. The same holds for Classification Rule Mining (Appendix A.10), where a lot of itemsets can be generated. One almost certain algorithms having troubles is Polynomial Regression (Appendix A.3). This because of the creation of polynomial features means we get $n = 3741$.

Algorithm	Dataset	Feature scalar	Error measurement
ALL	A (with $m = 7000, n = 11$)	Standardization Scalar	Mean Squared Error
ALL	A (with $m = 7000, n = 22$)	Standardization Scalar	Mean Squared Error
ALL	A (with $m = 7000, n = 44$)	Standardization Scalar	Mean Squared Error
ALL	A (with $m = 7000, n = 86$)	Standardization Scalar	Mean Squared Error
ALL	B (with $m = 7000, n = 11$)	Standardization Scalar	Mean Squared Error
ALL	B (with $m = 21000, n = 11$)	Standardization Scalar	Mean Squared Error
ALL	B (with $m = 63000, n = 11$)	Standardization Scalar	Mean Squared Error
ALL	B (with $m = 180024, n = 11$)	Standardization Scalar	Mean Squared Error

Table 4.6: The eight configurations run for the scalability experiment. All ten algorithms are used.

4.4 How well can machine learning algorithms handle irrelevant features?

In order to answer this question, we will design an experiment. In the experiment, we will add randomized features in order to see if and how much it influences the results of the algorithms. We will add up to 50% randomized features. The randomized features will be evenly distributed random numbers. We will use dataset A for this experiment. See Table 4.7 for an outline of the experiment.

Algorithm	Dataset	Feature scalar	Error measurement
ALL	B (with $m = 7000$)	Standardization Scalar	Mean Squared Error
ALL	B + 10% randomized features (with $m = 7000$)	Standardization Scalar	Mean Squared Error
ALL	B + 30% randomized features (with $m = 7000$)	Standardization Scalar	Mean Squared Error
ALL	B + 50% randomized features (with $m = 7000$)	Standardization Scalar	Mean Squared Error

Table 4.7: The four configurations run for the bias experiment. All ten algorithms are used.

Expected results

We expect that most algorithms will be capable of ignoring (some of) the irrelevant features. However, KNN (Appendix A.5) and K-means Classification (Appendix A.6) might have trouble with excessive amounts of irrelevant features, because the dimensions of the data influences the capability of structuring the data. The same holds most likely for SVM (Appendix A.8). The other algorithms should have less trouble with irrelevant features.

Chapter 5

Experimental evaluation

In this chapter, we will evaluate the results of the experiments defined in Chapter 4. We will briefly discus the results and what it means. We will mostly take an aggregate result. Extended results of the experiments can be found in Appendix B.

5.1 How accurate can machine learning algorithms predict insurance product interest for different datasets?

To answer this question we had define four sub questions in Section 4.2:

1. Should we optimize for correct classifications or for good probabilities to get the best accuracy of predictions?
2. How does feature scaling influence the algorithms and their prediction accuracy? Which feature scalar results in the best accuracy of predictions?
3. Which algorithms have the highest potential for accurate predictions?
4. Can we predict insurance product interest for clients, with similar or even higher accuracy compared to current statistical models?

The question will ultimately be answered by question four: *Can we predict insurance product interest for clients, with similar or even higher accuracy compared to current statistical models?* (Experiment 4.2.4). In order to answer this question, we will first answer the other three sub questions. Once we have answered those, we will come back to answer our main question.

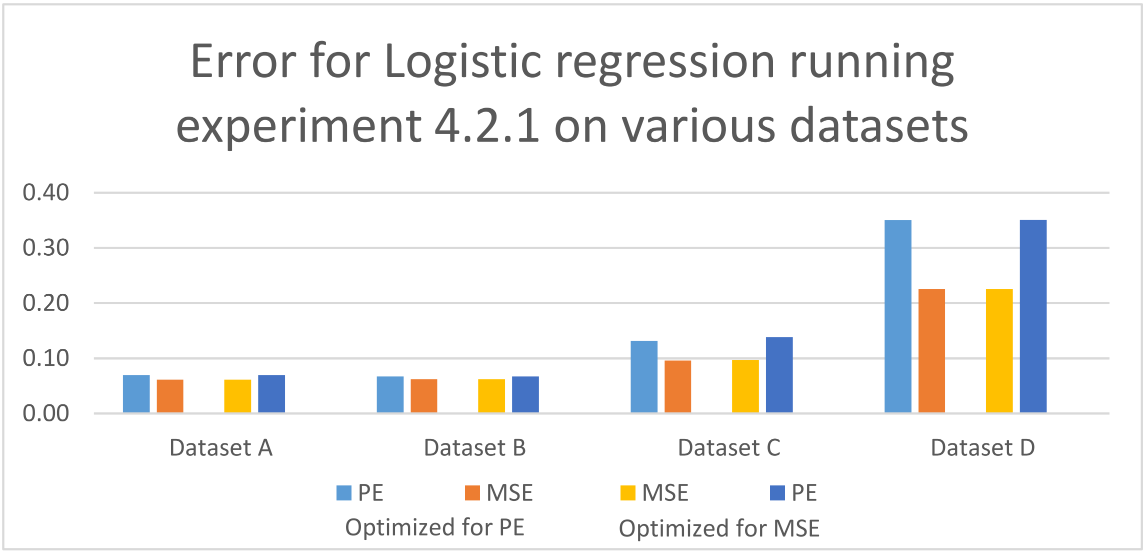


Figure 5.1: The results of Logistic Regression for Experiment 4.2.1.

5.1.1 Should we optimize for correct classifications or for good probabilities to get the best accuracy of predictions?

In Figures 5.1 and 5.2 the errors of Logistic Regression and Neural Networks are shown for Experiment 4.2.1. (See Figures B.1 to B.4 for the results of all algorithms) For each dataset we see four bars. The left two bars indicate the test error, when tuning was performed using the PE measurement. The right two bars indicate the test error, when tuning was performed using the MSE measurement. In most cases including Logistic Regression (Figure 5.1) we see that the error measurements perform (roughly) the same. That means that in both cases the PE values are similar and the MSE values are similar. This indicates that tuning with one or the other error measurement does not make a significant difference. In Figure 5.2 however, we see that this does not apply to the Neural Network algorithm. In some instances, in our case datasets A and C, the error significantly increases when using the MSE measurement in the tuning phase. This is most likely caused by over training. The penalty the MSE gives to slightly wrong predictions (i.e. predictions with low probabilities, say 0.6) might cause over training of the model. When the model is actually trained using these same parameters, the results are very different.

Since the results in most instances are very similar we can choose either measurement. However, for the Neural Network algorithms, the difference is noticeable in some cases, where the PE measurement comes on top. Hence we can conclude that the optimizing for correct classifications using the PE measurement results in the best accuracy of predictions. This is somewhat of a surprise since it was believed that the MSE measurement would yield better results.

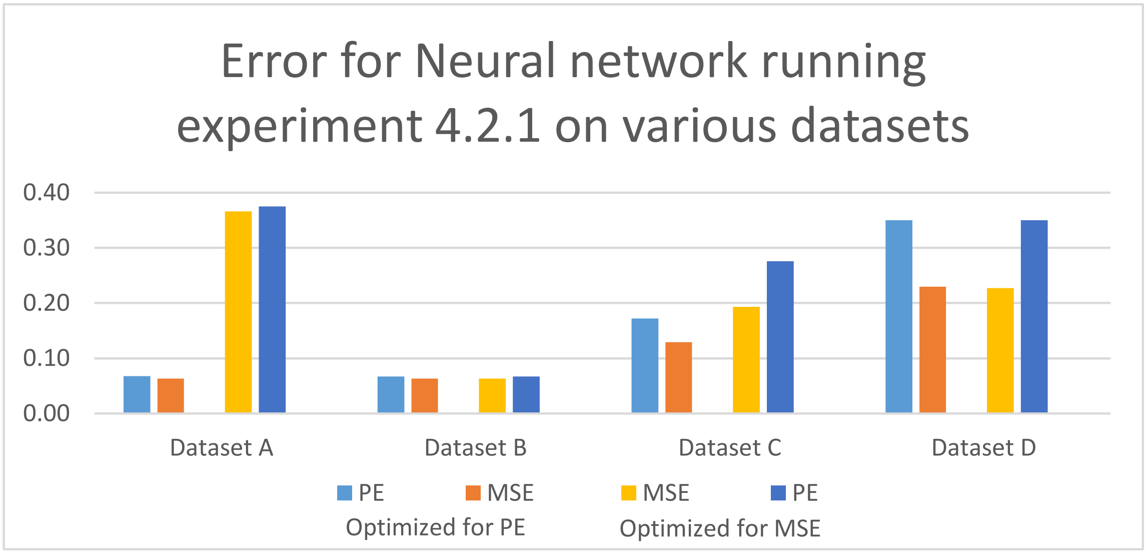


Figure 5.2: The results of Neural Network for Experiment 4.2.1.

5.1.2 How does feature scaling influence the algorithms and their prediction accuracy?

In Figures 5.3 to 5.5 the errors of K-means, Logistic Regression and Naive Bayes are shown for Experiment 4.2.2. (See Figures B.5 to B.8 for the results of all algorithms) For each algorithm we see the MSE using different feature scalars. It should be noted that Polynomial Regression (and in some cases Linear Regression and Logistic Regression where $^\circ = 2$) was not able to converge in reasonable time (within eight hours). Hence we do not have results for those instances.

There are some instances, including K-means (Figure 5.3) that do not seem to be significantly influenced by feature scaling. On the other hand, we can see that for quite some cases, including Logistic Regression (Figure 5.4) no scaling results in worse results.

Hence we have determined that feature scaling does in fact significantly influence the accuracy of algorithms. In the results, we can see that the results using the Min-max scalar are similar to those using the Standardization scalar. For some algorithms however, the results are improved by using the Standardization scalar, an example is Naive Bayes (Figure 5.5). This is somewhat expected due to the normality assumption (See Appendix A.1). However, we also expected that Logistic Regression would benefit more from the Standardization scalar.

Hence the results suggest that feature scaling certainly influences the algorithms and their prediction accuracy. In most cases, it increases the accuracy of predictions. The Standardization scalar improves the results the most of the scalars that were investigated.

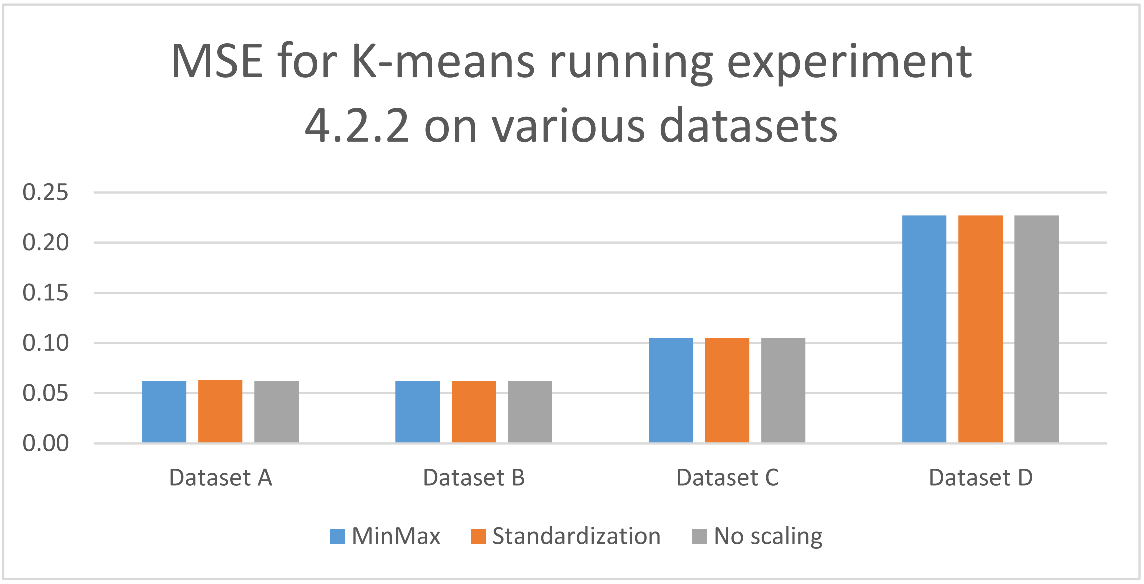


Figure 5.3: The results of K-means for Experiment 4.2.2.

5.1.3 Which algorithms have the highest potential for accurate predictions?

Using the results of Experiments 4.2.1 and 4.2.2 we can define the configuration that should result in the best predictions. As concluded in Section 5.1.1, the PE error measurement gives the best results. Similarly, we have concluded in Section 5.1.2 that the Standardization scalar results in the best predictions.

Hence we use these two components in our configuration to make predictions on the full datasets. See Figure 5.7 for the errors for each algorithm running Experiment 4.2.3 on dataset D. (See Figures B.9 to B.13 for the results of all algorithms on each dataset) We can see that the results between algorithms do not have a high deviation. In Figure 5.6 we can see the errors for each algorithm running Experiment 4.2.3 on dataset C.

We see that the results of Naive Bayes are worse than the other algorithms. We can also clearly see that according to the PE measure, Decision Tree, Support Vector Machine, Logistic Regression and Classification Rule Mining are the most promising algorithms to get a good accuracy. On the other hand, according to the MSE measure Support Vector Machine and Decision Tree are amongst the worst algorithms (including Naive Bayes). This is not entirely surprising since these are the only algorithms that classify only. they do not have a probabilistic model. This means that a misclassification gets penalized maximally. From these results, we can conclude two things. If we only value whether our predictions are correct, Decision Tree is the most promising algorithm to get the best predictions. On the other hand, if we value the probability behind predictions, Classification Rule Mining is the most promising algorithm.

In Section 2.7 we mentioned that we can also use Precision and Recall as a measurement of accuracy. In Figure 5.8 the Precision and Recall for all algorithms running Experiment 4.2.3 on dataset C are outlined. We can see that for Neural Networks the Precision and Recall are lower than for the other algorithms. We can see that the most promising algorithms Decision Tree and Classification Rule Mining also have high Precision and Recall. Hence they are still the most promising algorithms.

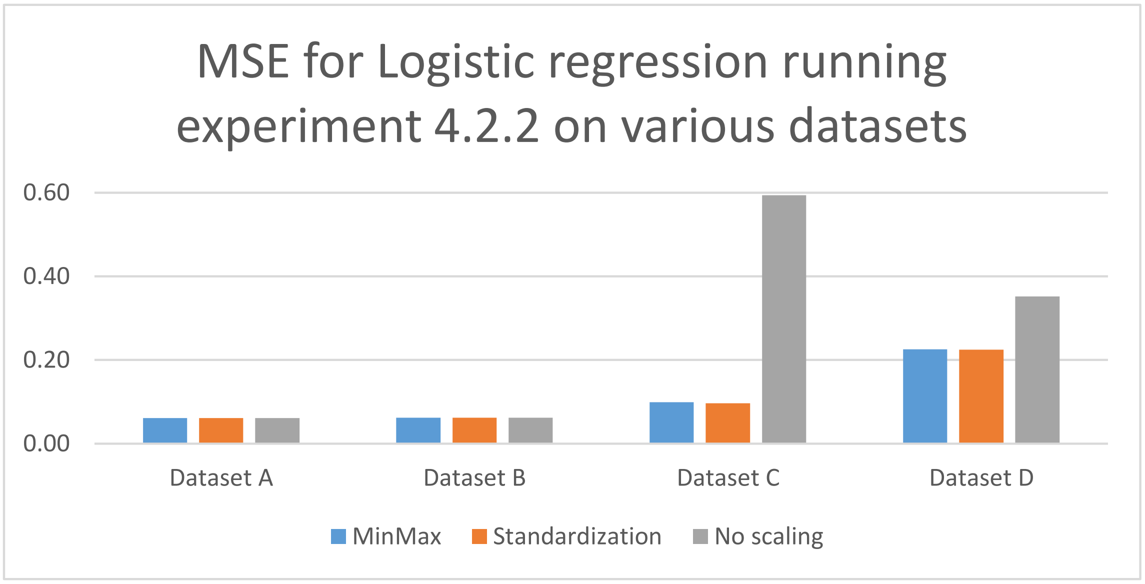


Figure 5.4: The results of Logistic Regression for Experiment 4.2.2.

5.1.4 Can we predict insurance product interest for clients, with similar or even higher accuracy compared to current statistical models?

As discussed in Section 4.2.4 statistical analysis is only performed on datasets C, D and E. Hence our results are limited to that set. The results might seem a bit surprising. The results of data set C (Figure 5.9) suggest that machine learning gives a better accuracy compared to statistical analysis, in both the PE and MSE measures. However, the results of dataset D and E (Figures 5.10 and 5.11) suggest that machine learning is less accurate than statistical analysis. This seems contradictory but it can be explained by bias from the data scientist. After reviewing the data we found out that a correlation exists between one of the features if it was transformed (birth date to age), and the classification. The data scientist left this feature out because this correlation suggested perfect prediction. This directly feeds the argument made in Section 1.2 that statistical analysis might be prone to mistakes.

If we only look at datasets D and E, we can conclude that statistical analysis results in better accuracy than machine learning without model selection (specific data transformation). However, the machine learning techniques can get close to the results of the statistical analysis. In Figure 5.10 we can see that the difference in the PE for dataset D is 0.11 (i.e. 11%). In Figure 5.11 we can even see that the difference is 0.01 (i.e. 1%) for dataset E. If we look at Recall and Precision for dataset E in Figure 5.12 we see that various machine learning techniques have competitive Precision and Recall. These facts are a clear indication that machine learning has the potential to get similar accuracy as statistical analysis. As of right now, machine learning cannot predict insurance product interest for client, with higher accuracy compared to current statistical models.

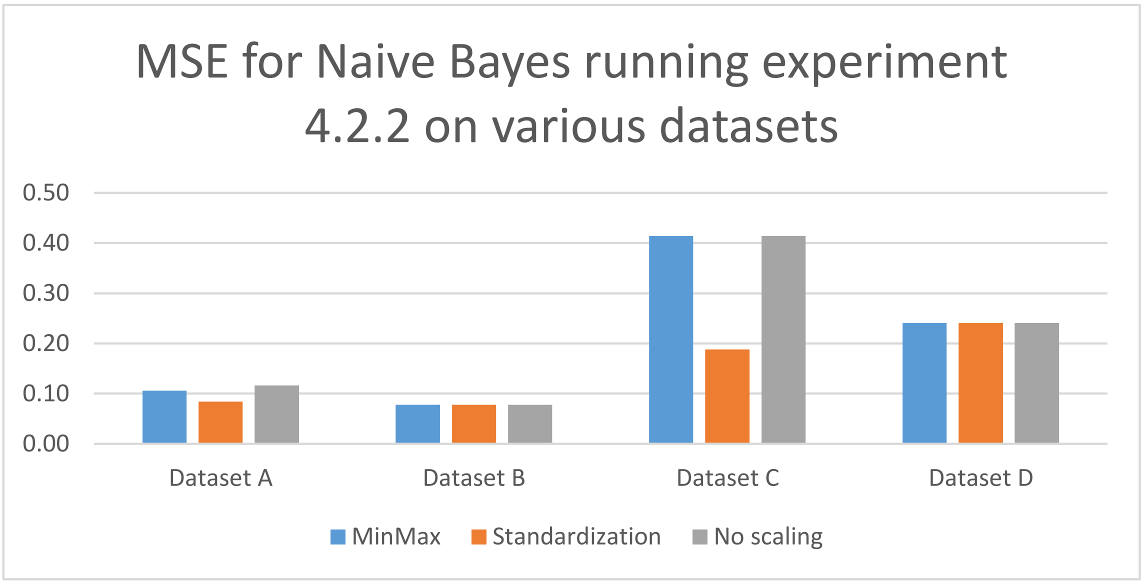


Figure 5.5: The results of Naive Bayes for Experiment 4.2.2.

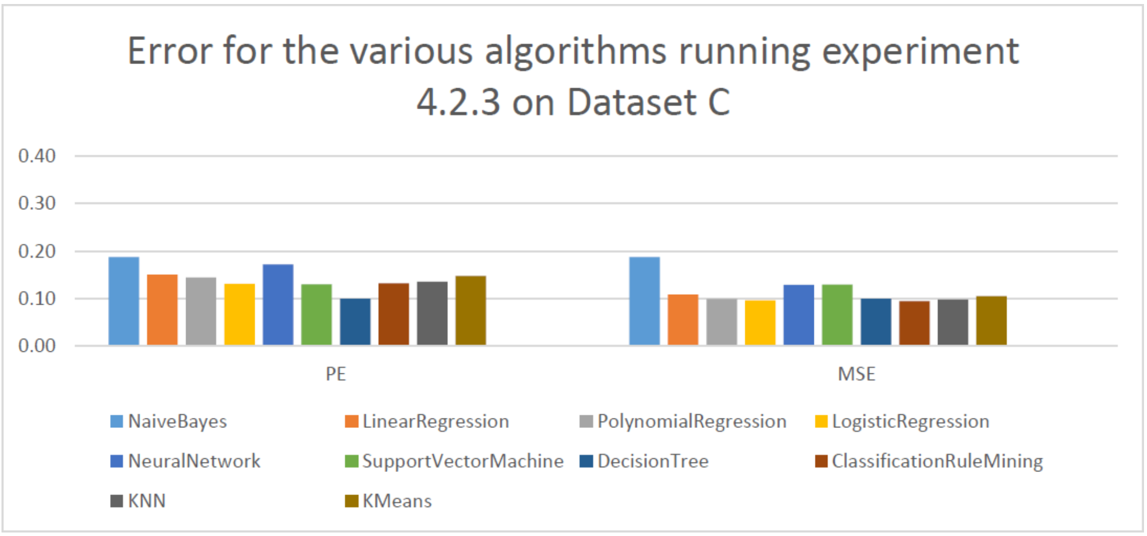


Figure 5.6: The results of all algorithms running Experiment 4.2.3 on dataset C.

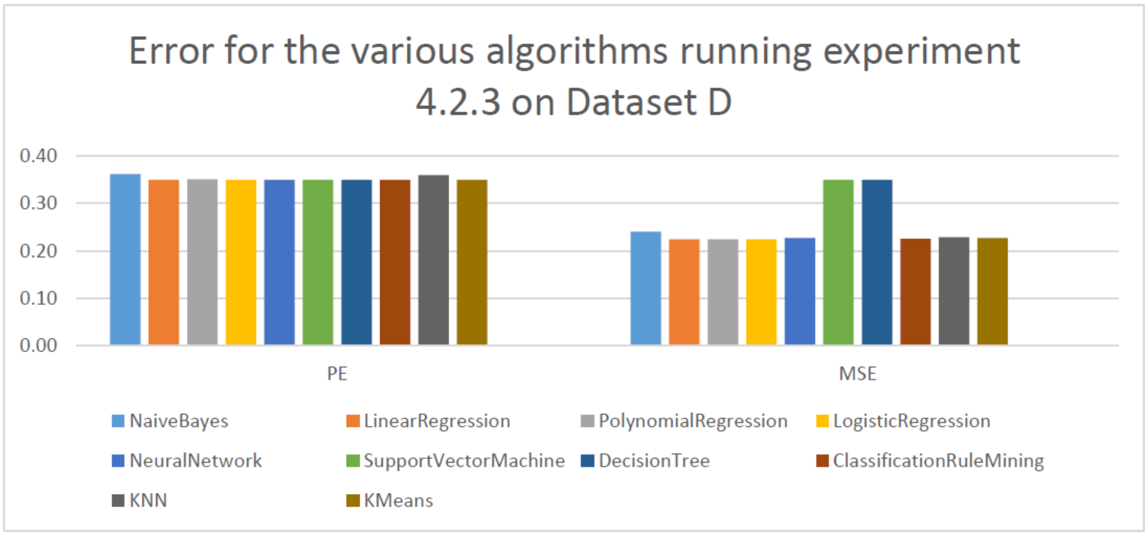


Figure 5.7: The results of all algorithms running Experiment 4.2.3 on dataset D.

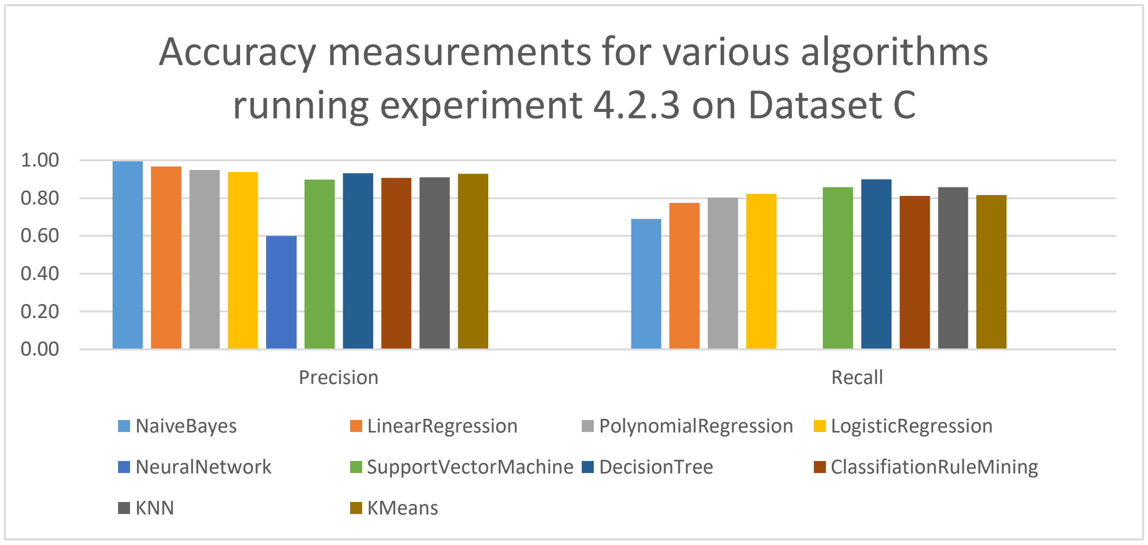


Figure 5.8: The results of all algorithms running Experiment 4.2.3 on dataset C.

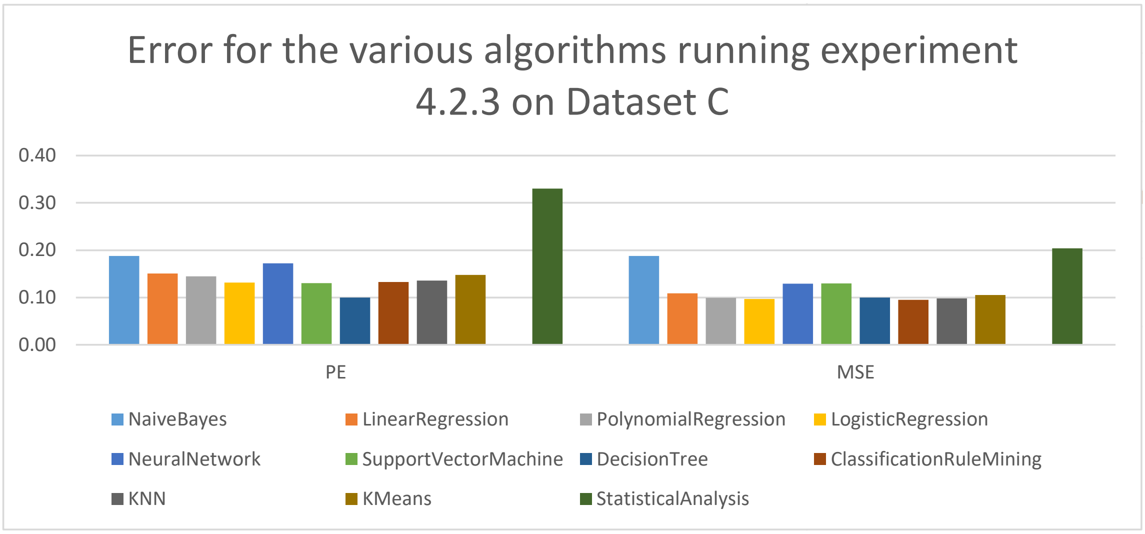


Figure 5.9: The results of Experiment 4.2.3 for dataset C.

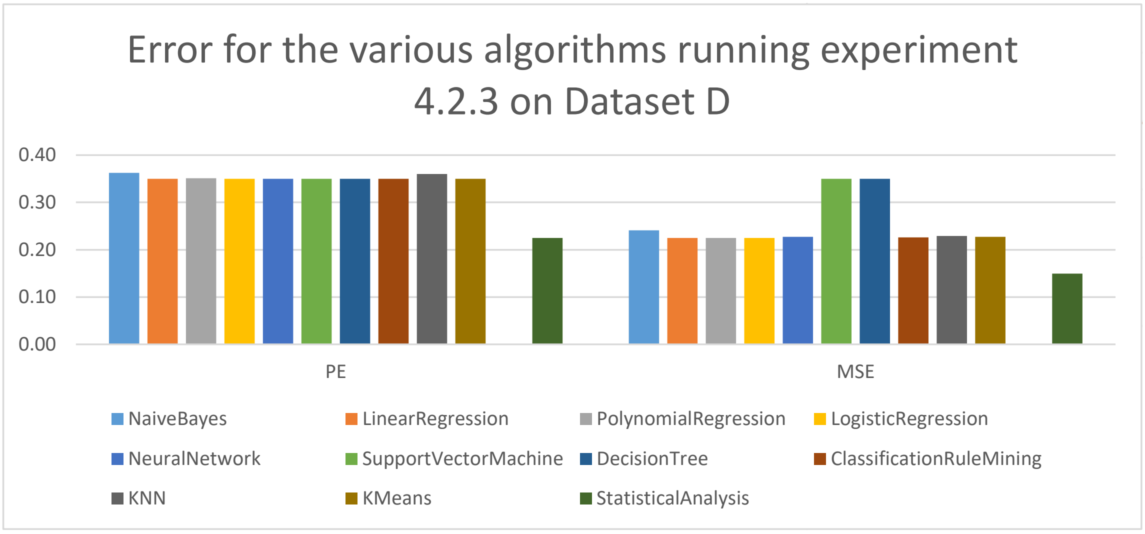


Figure 5.10: The results of Experiment 4.2.3 for dataset D.

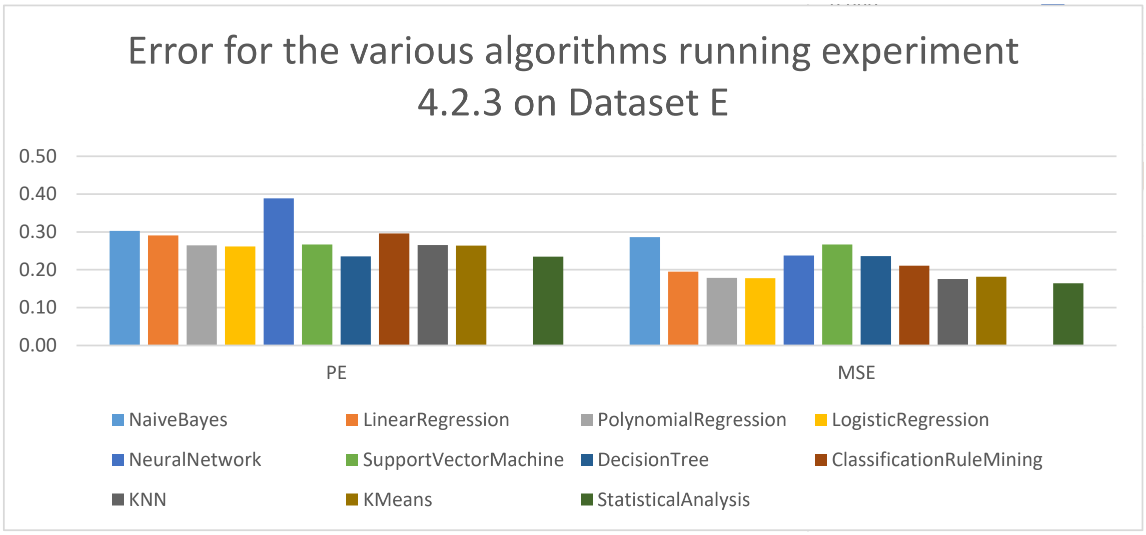


Figure 5.11: The results of Experiment 4.2.3 for dataset E.

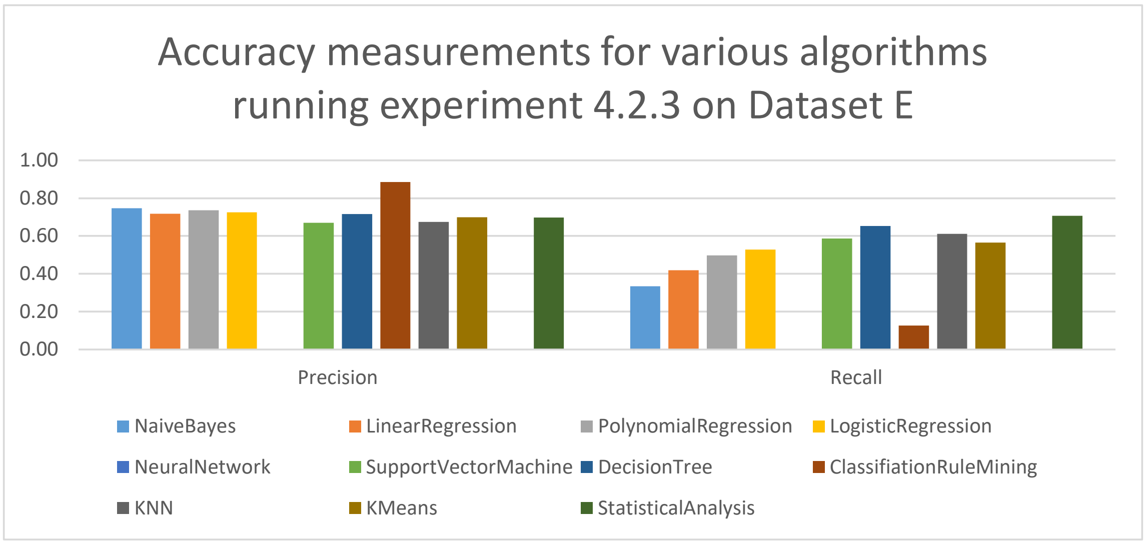


Figure 5.12: The results of Experiment 4.2.3 for dataset E.

5.2 How fast can machine learning algorithms create models for prediction of insurance product interest for different datasets?

This question will be answered by answering the three questions defined in Section 4.3:

1. How does feature scaling influence the model creation speed of the algorithms? Which feature scalars result in the fastest model creation times?
2. How much do various parameters influence the prediction accuracy of the algorithms? Can we eliminate the tuning phase?
3. How well does the model creation time of algorithms scale to large datasets?

5.2.1 How does feature scaling influence the model creation speed of the algorithms?

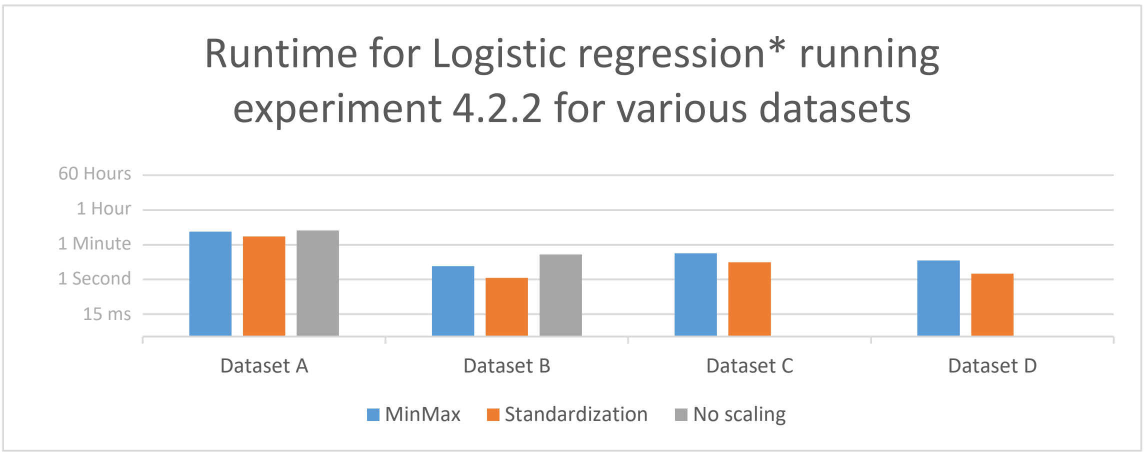


Figure 5.13: The runtime of Logistic Regression with $\circ = 1$ on all datasets for Experiment 4.2.2.

In Figure 5.13 the runtime for Logistic Regression with $\circ = 1$ for each dataset are summarized. (See Figures B.14 to B.17 for the results of all algorithms). For each dataset, we see the runtime using different feature scalars. It should be noted that in case of dataset C and D Logistic Regression did not converge in reasonable time (within eight hours) when using no scalar. When we look at the individual results (Figures B.15 to B.17) we can see that all regression forms have trouble, if no scaling is used. Hence a form of feature scaling is required.

If we look at the results of the runtime for the Min-max scalar and the Standardization scalar, we see that the results are similar, on average Standardization results in the fastest runtimes. Hence we are free to pick either of these two feature scalars, since they result in the bets model creation times.

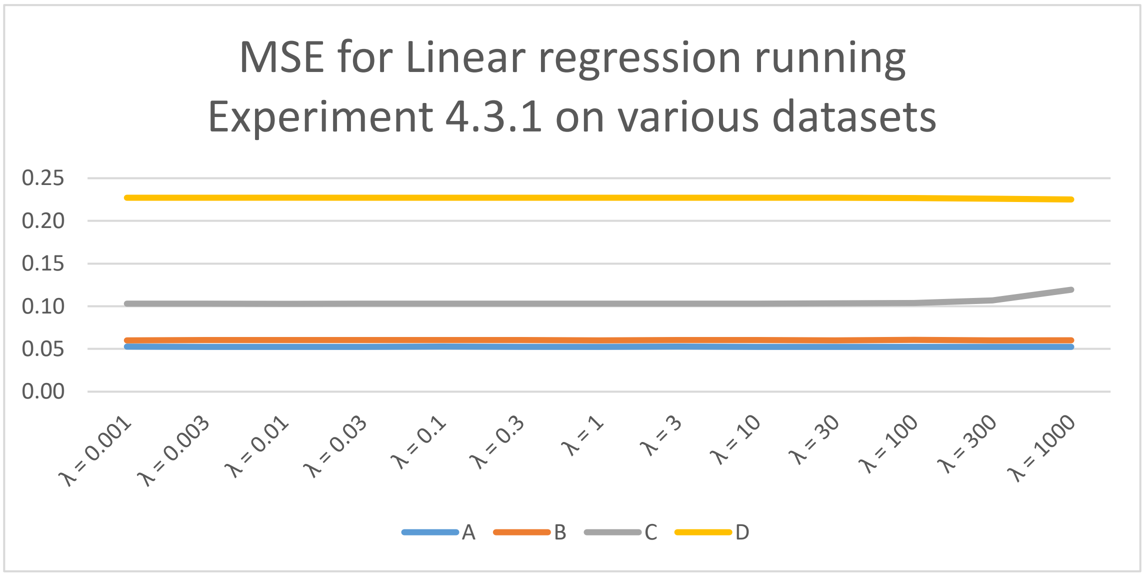


Figure 5.14: The results of Experiment 4.3.1 for Linear Regression.

5.2.2 How much do various parameters influence the prediction accuracy of the algorithms? Can we eliminate the tuning phase?

The results of this experiment are quite difficult to analyze. The various algorithms have different parameters, that have different effects on the results. Hence we must analyze the results separately. In Figures 5.14 and 5.15 we can see the results of both Linear and Polynomial Regression. Here we see that the results, although quite minimal, are influenced quite differently depending on the value of λ. Dataset B gives better results for low values of λ, whereas dataset C gives better results for high values of λ. The other datasets remain quite similar over different values of λ. For Logistic Regression (Figure 5.16) the impact of λ is more visible. The impact of the polynomial degree $^\circ$ is more subtle, but still varies over the datasets. Hence tuning is still required.

The results of Neural Network (Figure 5.17) and Support Vector Machine (Figure 5.18) have a far more complex relation with the parameters and their error. No real pattern can be found over the four datasets. For the results of Support Vector Machine some pattern still exists. However, the pattern is not consistent. Dataset B has several valleys, whereas datasets A, B, and C have peaks. Hence tuning is certainly required for the methods.

For the logic based learning techniques (Figures 5.19 and 5.20), some more patterns are visible. A confidence value of $c = 0.05$ or $c = 0.1$ consistently give the best results for the Decision Tree, whereas lower and higher values often give a higher error. This means that tuning might not be required for Decision Tree. For Classification Rule Mining (Figure 5.20) we also see that a lower values of the interval size (Is) give better results. This means that less discretization gives better accuracy. On the other hand, the minimum support (Ms) and minimum confidence (Mc) have an influence on the results that does not have a regular pattern between datasets. Hence tuning is still required but can be reduced to Mc and Ms only.

For the instance based and clustering methods (Figures 5.21 and 5.22) we also clearly see some patterns.

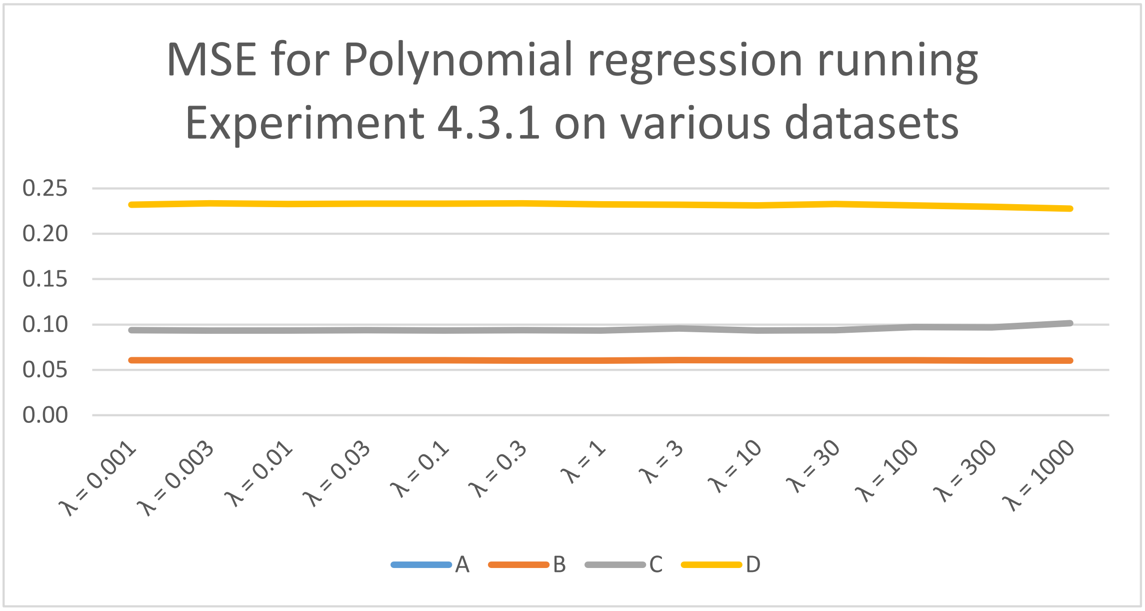


Figure 5.15: The results of Experiment 4.3.1 for Polynomial Regression.

First of, it is clear that unweighted ($w = f$) KNN gives far better results than weighted ($w = t$) KNN. Furthermore, it seems that higher values of k (while $w = f$) gives better results than low values. Most datasets give the best results at $k = 64$. The exception to this rule is Dataset B which gives the best result at $k = 16$. Hence, tuning cannot be fully eliminated, but it can be greatly reduced to the unweighted version and the higher values of k. Similarly, the range values of k for the K-means algorithm can be greatly reduced. High value give similar results which are worse on average. Hence, values larger than $k = 265$ are not worth investigating during the tuning phase. Similarly the low values of k do not seem to have much contribute much on low errors for this method. Values of $k = 8$ start being interesting. It should be noted that for datasets with larger values of m (number of instances) it might be worth to investigate larger values of k.

In conclusion, for some of the techniques tuning can be skipped or reduced. Unfortunately, the algorithm with the highest running time, Support Vector Machine, does not belong to this category. On the other hand, Classification Rule Mining, the algorithm with the second highest runtime, does belong to the category of algorithms for which the tuning phase can be reduced. The other algorithms that can reduce or skip tuning, already have fast running times. hence, skipping the tuning phase is not an option that will benefit a lot for the running time of the algorithms in practice.

5.2.3 How well does the model creation time of algorithms scale to large datasets?

In this experiment, we made a distinction between Logistic Regression with $^\circ = 1$ only, and Logistic Regression with either $^\circ = 1$ or $^\circ = 2$. This because Logistic Regression with $^\circ = 2$ did not always converge. In Figures 5.23 and 5.25 the Logistic Regression with $^\circ = 1$ only, is indicated as ”LogisticRegression *”.

In Figure 5.23 we can see that Classification Rule Mining does not scale well for large values of n (number of features). We can also see that Polynomial Regression and Logistic Regression with $^\circ = 2$ does not scale

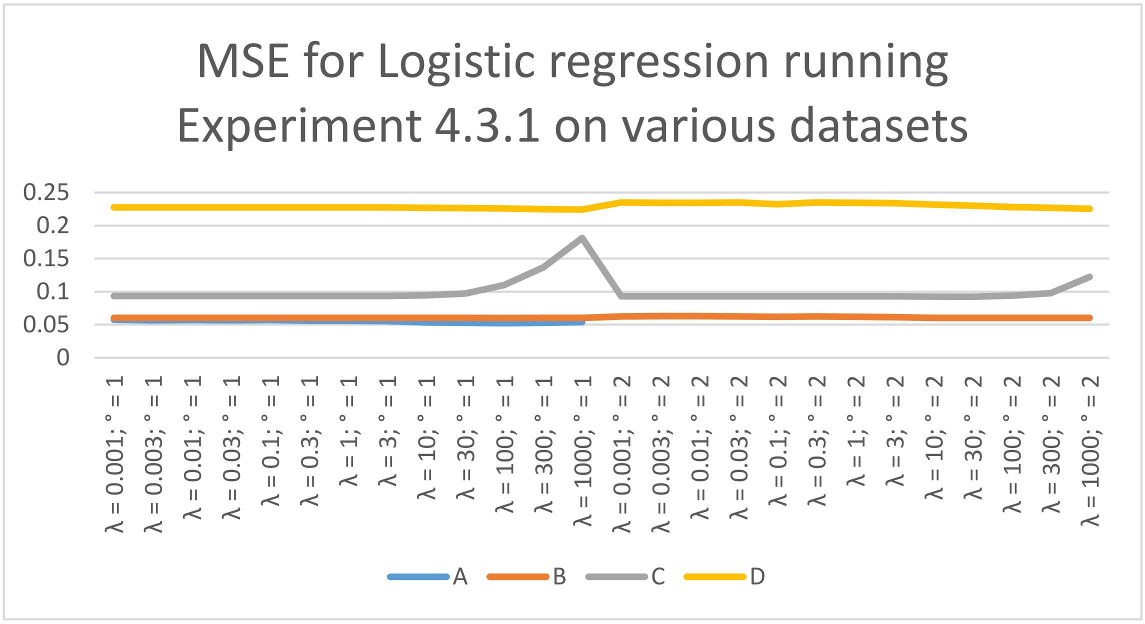


Figure 5.16: The results of Experiment 4.3.1 for Logistic Regression.

well for large values of n. All three algorithms were not able to converge within eight hours for larger values of n. In Figure 5.24 we can also see that the other algorithms have a run time dependency of approximately $O(\log n)$. This means that these algorithms do scale well to large values of n.

In Figure 5.25 we can see that Support Vector Machine does not scale well to large values of m (number of instances). Already for a value of $m = 21000$, Support Vector Machine is not able to converge within eight hours. Hence Support Vector Machine does not scale well to large values of m. This is expected, because Support Vector Machine transform the problem to an m dimensional problem. In Figure 5.26 we can also see that the other algorithms have a run time dependency of approximately $O(\log m)$. This means that these algorithms do scale well to large values of m.

In conclusion Classification Rule Mining, Polynomial Regression, Logistic Regression (with $° = 2$) and Support Vector Machine, do not scale well. This means that they might not be suitable for making predictions in practice. The other algorithms do scale well and might still be suitable for making predictions in practice.

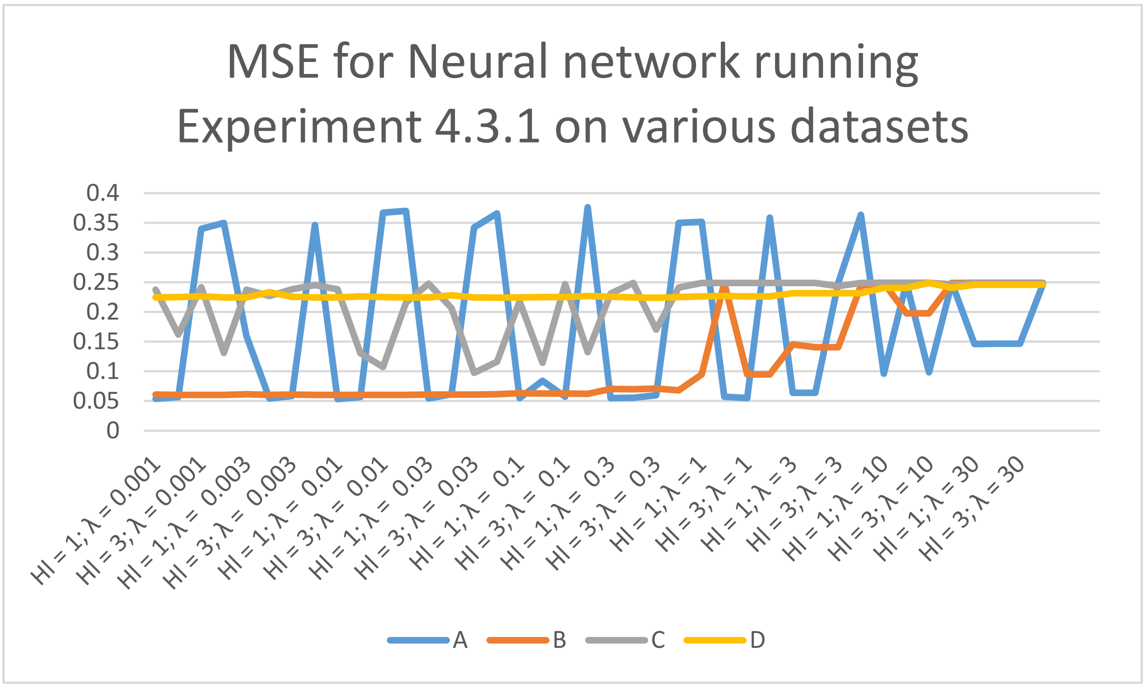


Figure 5.17: The results of Experiment 4.3.1 for Neural Network.

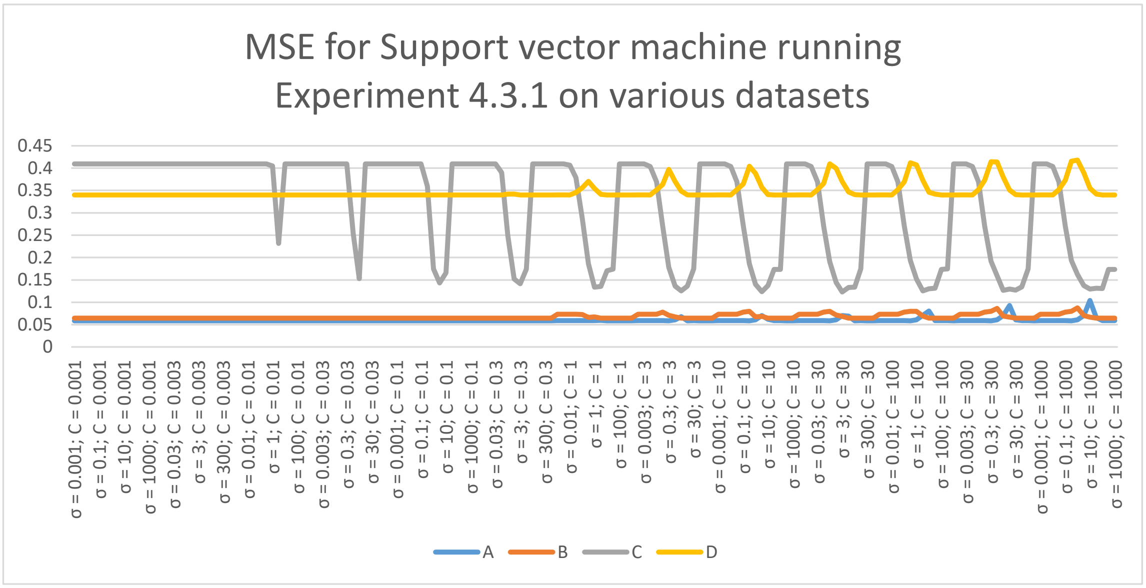


Figure 5.18: The results of Experiment 4.3.1 for Support Vector Machine.

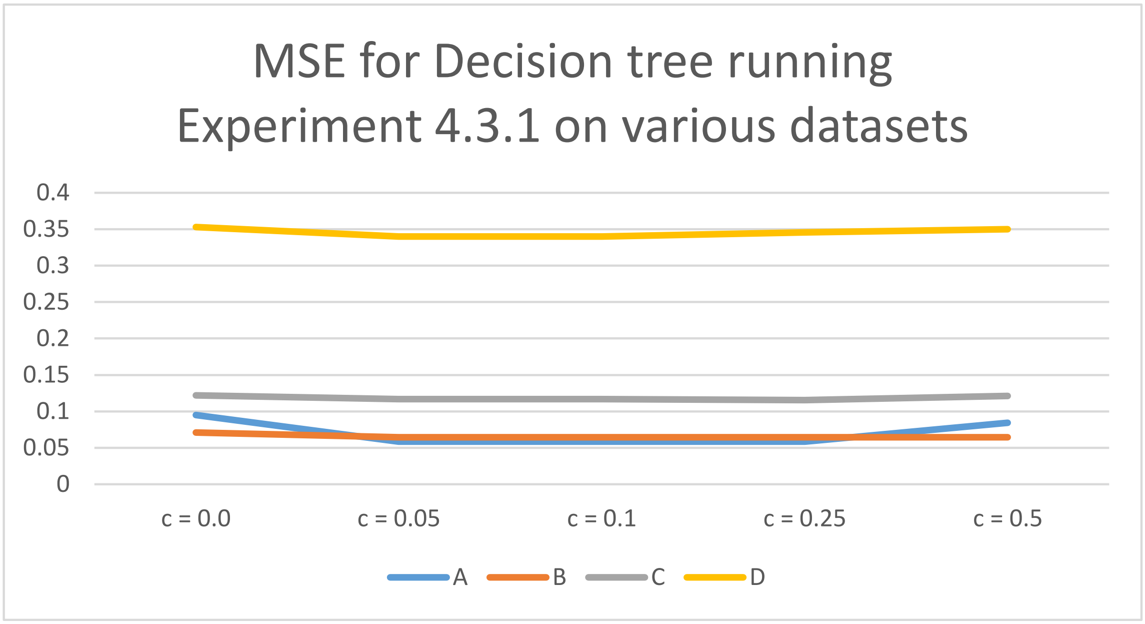


Figure 5.19: The results of Experiment 4.3.1 for Decision Tree.

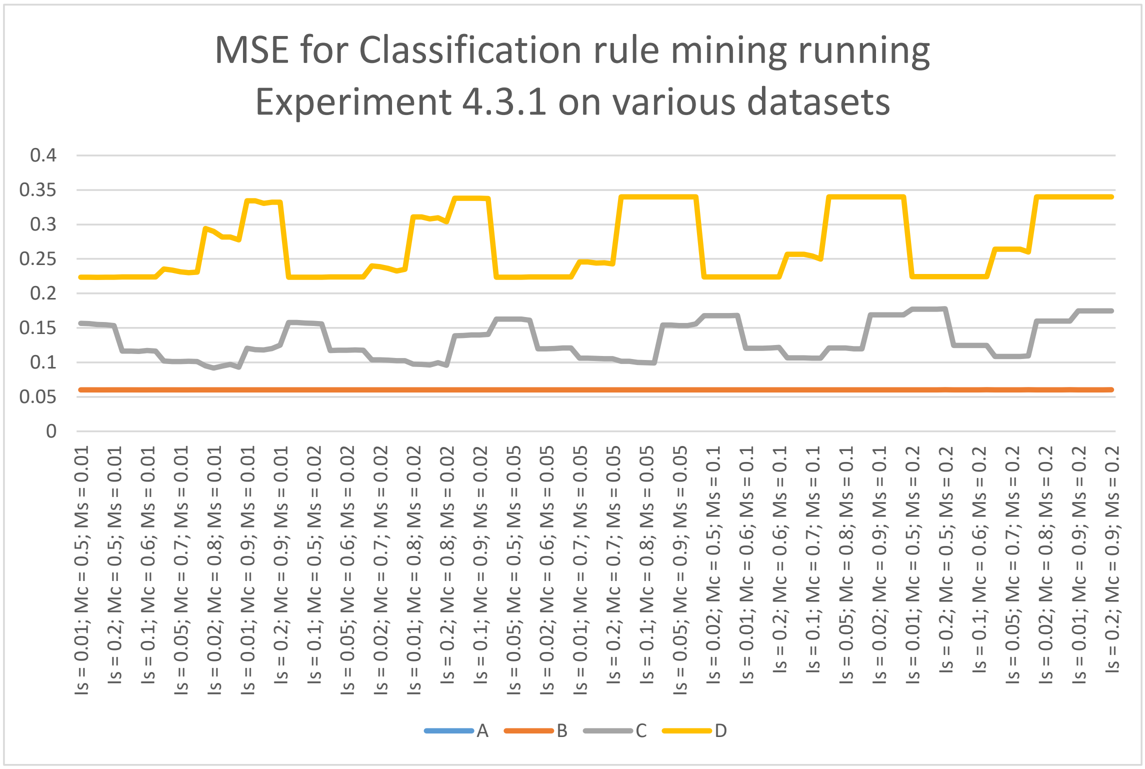


Figure 5.20: The results of Experiment 4.3.1 for Classification Rule Mining.

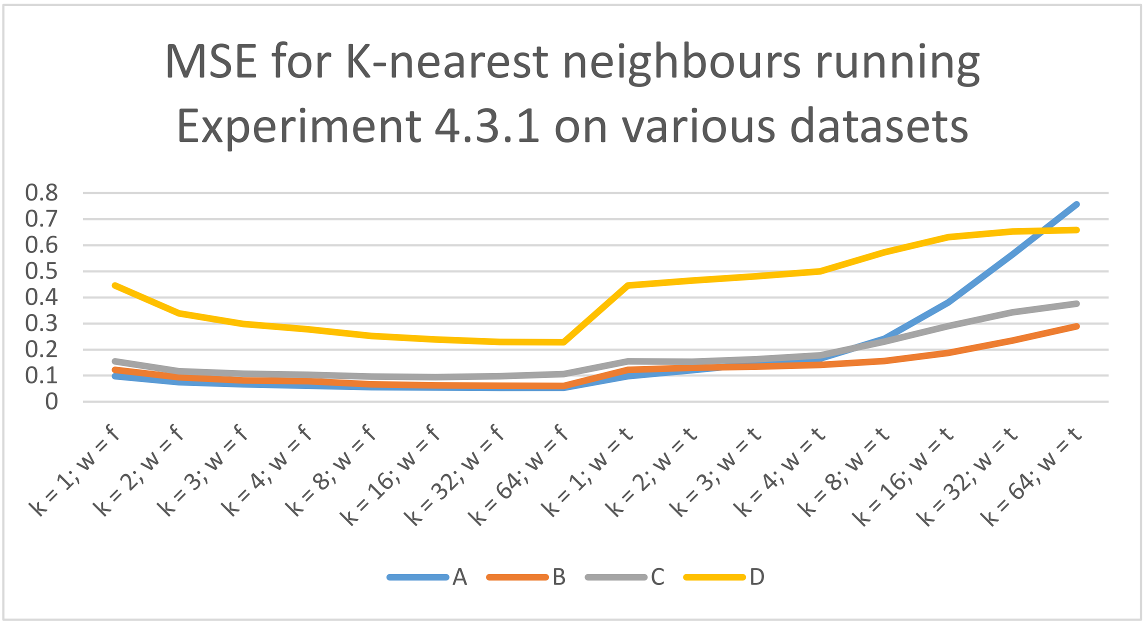


Figure 5.21: The results of Experiment 4.3.1 for K-nearest Neighbours.

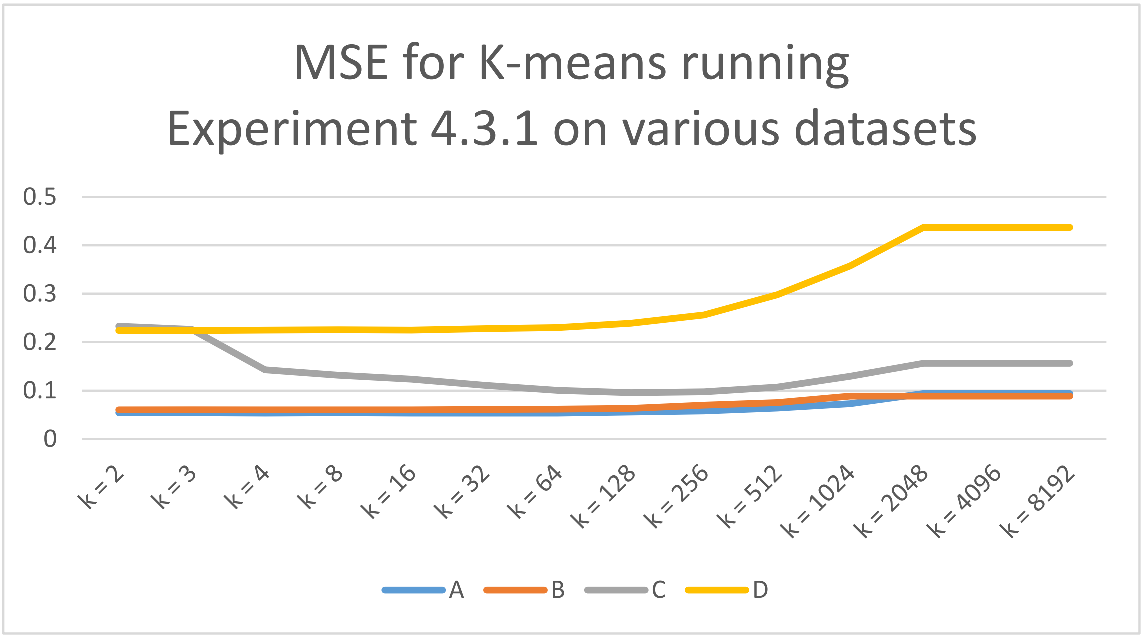


Figure 5.22: The results of Experiment 4.3.1 for K-means.

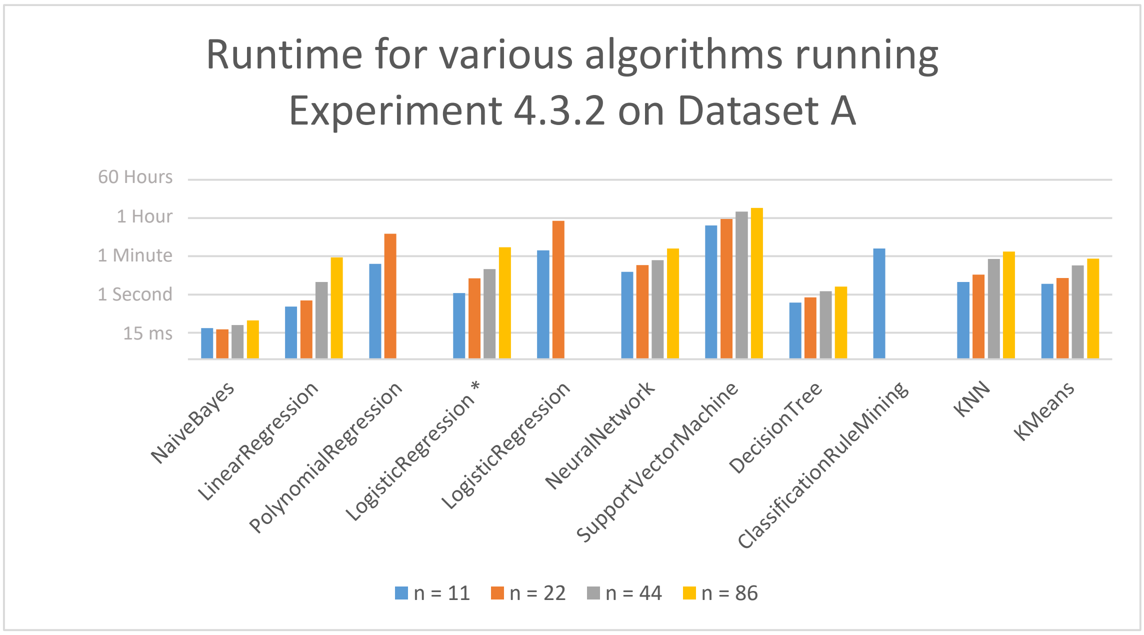


Figure 5.23: The results of Experiment 4.3.2 for variations of dataset A.

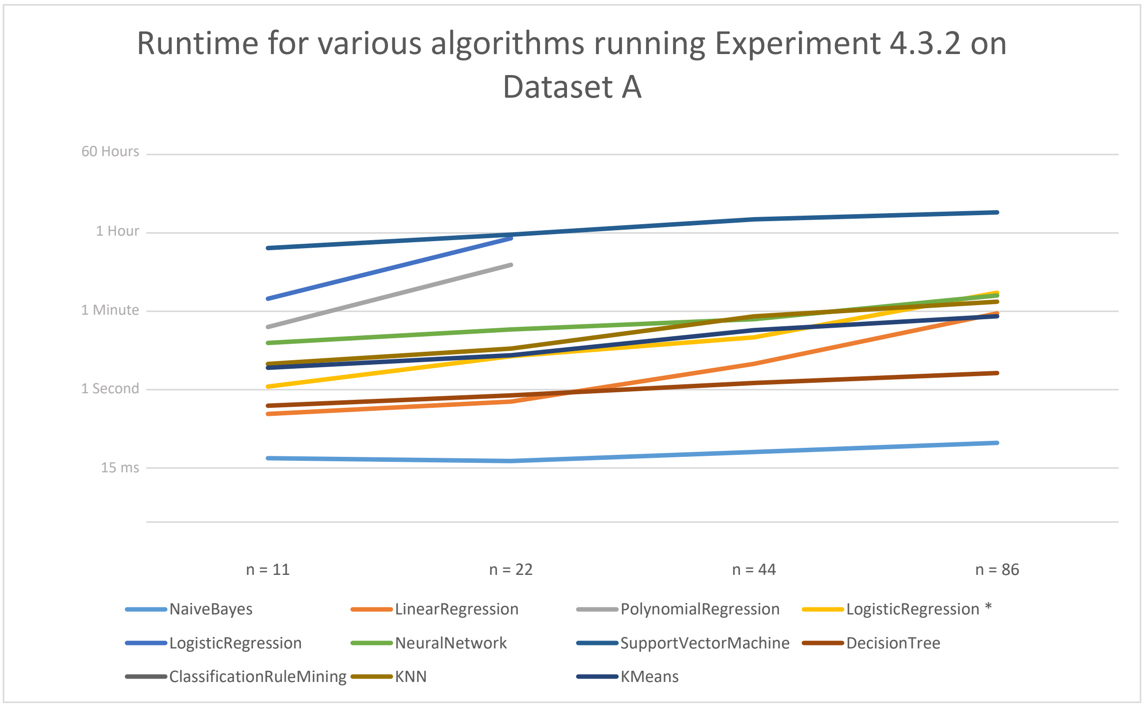


Figure 5.24: The results of Experiment 4.3.2 for variations of dataset A.

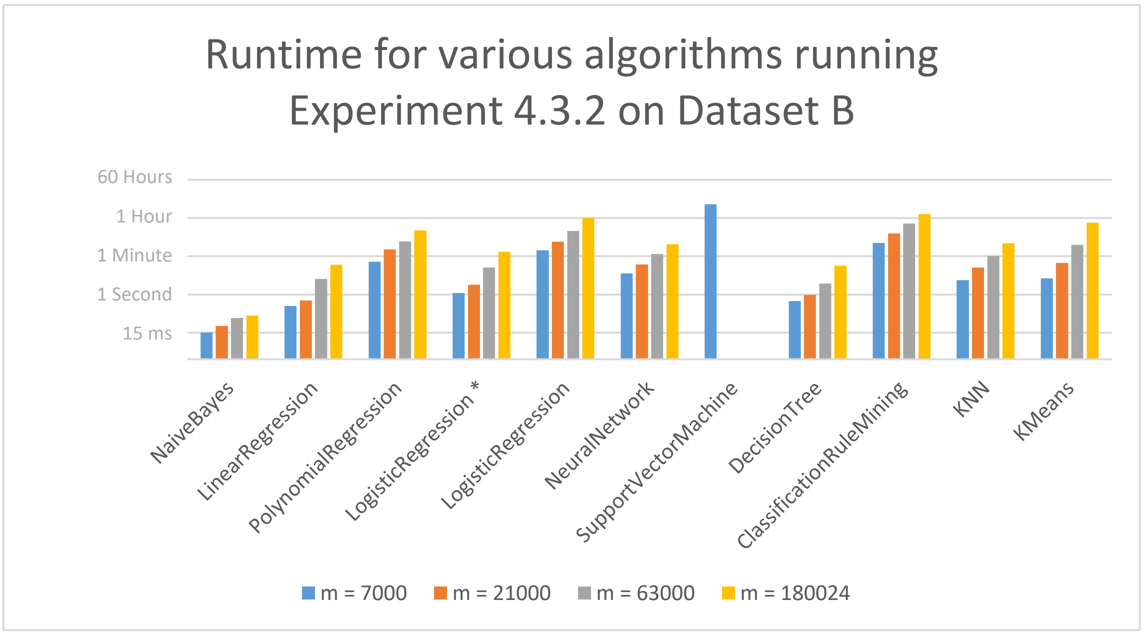


Figure 5.25: The results of Experiment 4.3.2 for variations of dataset B.

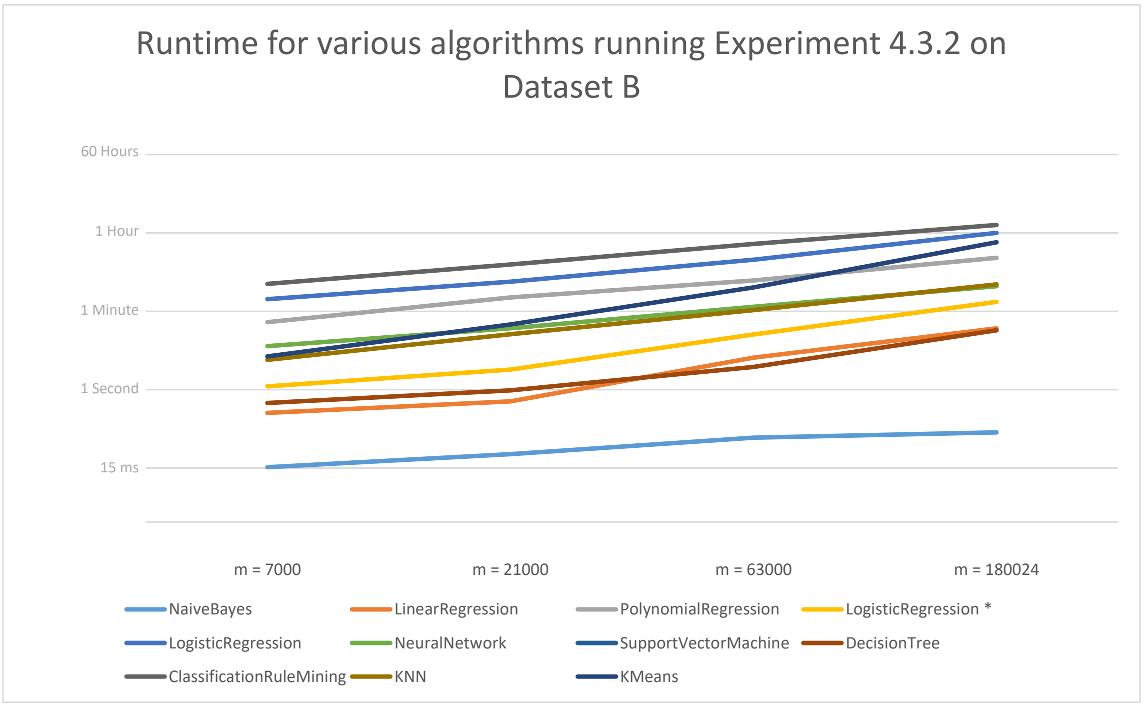


Figure 5.26: The results of Experiment 4.3.2 for variations of dataset B.

5.3 How well can machine learning algorithms handle irrelevant features?

The results of this experiment are very straight forward. In Figures 5.27 and 5.28 we can see that the number of random features (r) does not have influence on the results of the algorithms. This means that all algorithms are capable of separating irrelevant features from relevant features.

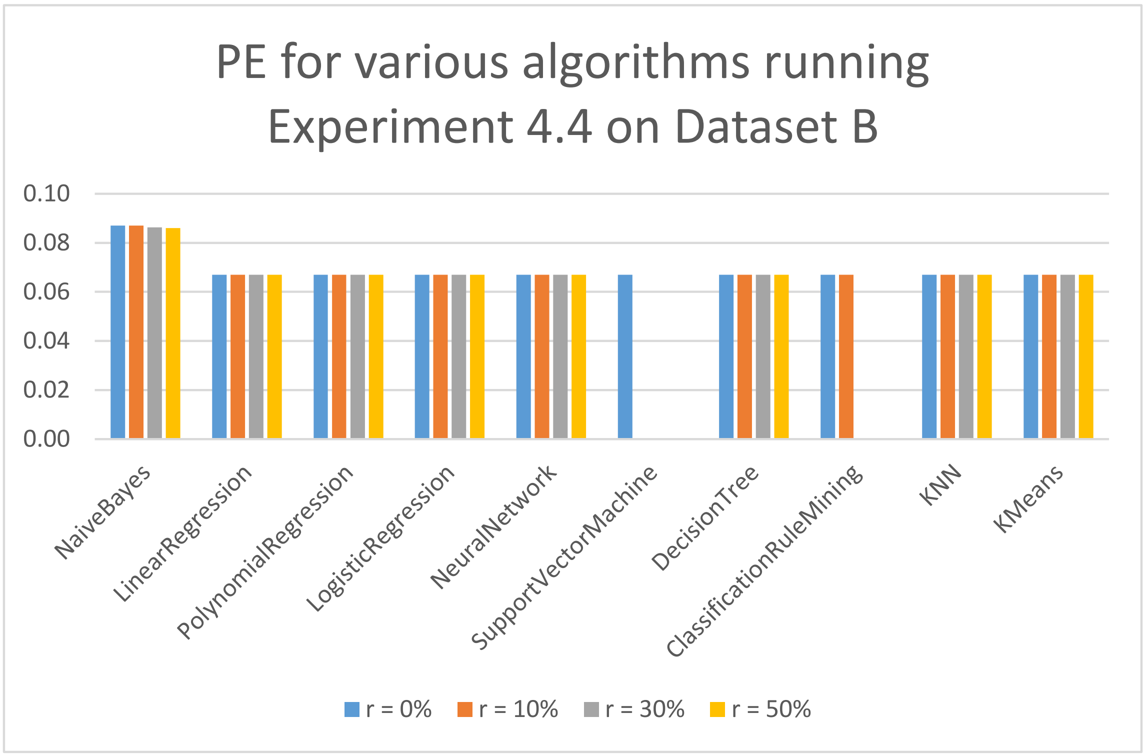


Figure 5.27: The results of Experiment 4.4 for dataset A and error measurement PE.

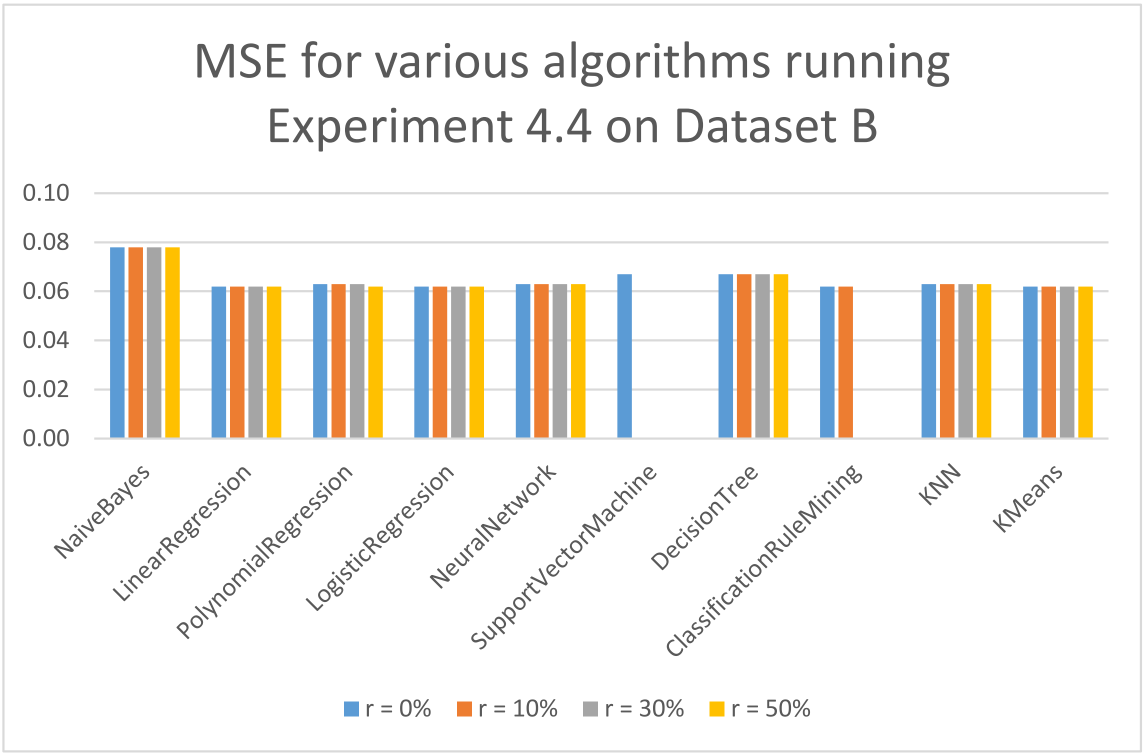


Figure 5.28: The results of Experiment 4.4 for dataset A and error measurement MSE.

Chapter 6

Conclusions

In Section 1.2 we have defined that C-Profile has two main problems with their current statistical analysis.

1. The statistical analysis might be prone to mistakes, which means the accuracy will decrease.
2. It takes a data scientist too long (weeks/months) to setup a good model for statistical analysis. This will reoccur every time a new dataset becomes available, or if the datasets changes often.

We have investigated if we can solve these two problems using machine learning, by implementing ten classification algorithms and running several experiments. In Section 6.1 we will answer our main research question by answering the sub questions. In Section 6.2 we will discuss opportunities that are of interest of future work.

6.1 Research questions

We have designed and executed experiments to answer the main research question defined in Section 1.4; *Which machine learning technique has the highest potential to replace statistical analysis for predicting insurance product interest?* We answer this question using three sub research questions in Sections 6.1.1 to 6.1.3 and finally answer our main research question in Section 6.1.4.

1. How accurate can machine learning algorithms predict insurance product interest for different datasets? What we consider accurate will be covered in Chapter 2.
2. How fast can machine learning algorithms create models for prediction of insurance product interest for different datasets? How does the size of the datasets influence the performance in terms of speed for various techniques?
3. How well can machine learning algorithms handle irrelevant features? I.e. features that do not add value to the prediction. Are the techniques able to filter out the important features to make accurate predictions?

6.1.1 How accurate can machine learning algorithms predict insurance product interest for different data-sets?

In Experiments 4.2.1 and 4.2.2 we have determined which configurations of error measurement and feature scaling result in the best accuracy. It turns out that the PE measurement in combination with the Standardization scalar gives the best accuracy, when measured for both the PE and MSE error measurement.

In Experiment 4.2.3 we have determined that the algorithm best suited for making predictions depends on what we value most. If we only value correctness, the algorithms to pursue are Decision Tree, Support Vector Machine and Logistic Regression. On the other hand, if we value the probability behind the predictions, the algorithms to pursue are Classification Rule Mining, Logistic Regression and Polynomial Regression.

In Experiment 4.2.4 we have compared the accuracy of machine learning techniques compared to statistical analysis. We have demonstrated that, although hand tailored statistical analysis gives better results, machine learning gives consistent results that can compete with the accuracy of statistical analysis. We even showed that statistical analysis is indeed prone to mistakes that reduces accuracy.

The accuracy of machine learning techniques is fairly high. We have shown that the PE scores of the best algorithms range from 0.06 to 0.34 for different datasets. This means that the algorithms are up to 94% accurate. An accuracy of up to 94% allows us the present the predictions to insurance intermediaries in practice with great confidence. This effectively allows insurance intermediaries to target clients to sell insurance products with a 94% success rate. This is much more efficient than targeting all clients, and hence lowing the success rate, but also the service level (clients do not like to be bothered for something they are not interested in).

If we compare the statistical analysis results with the machine learning techniques on the same datasets (excluding dataset C which turned out to be an unfair comparison in Section 5.1.4), we see that statistical analysis has accuracy up to 78%, whereas machine learning can achieve accuracy up to 77%. However, the lowest measures are lower for machine learning compared to statistical analysis with 78% versus 67%. In the MSE measures the differences are less extreme with a worst MSE of 0.22 for machine learning compared to a worst MSE of 0.15 for statistical analysis.

These results indicate that at this moment statistical analysis is still superior to machine learning regarding accuracy. However, machine learning does have competitive accuracy, with a margin of only 11%. This means that the most accurate machine learning techniques presented as of now can be used in practice for automated lead generation in C-Profile. If a company requires a more in depth analysis, statistical analysis would still be the better choice.

6.1.2 How fast can machine learning algorithms create models for prediction of insurance product interest for different datasets?

In Experiment 4.2.2 we have determined that feature scaling does indeed have influence on the runtime of model creation for the algorithms. Which particular scalar is used does not have much of an influence. Hence we can choose the scalar used based on other criteria.

In Experiment 4.3.1 we investigated the influence of parameter tuning on the accuracy, because parameter tuning has a big influence on the runtime of model creation for the algorithms. We have concluded that most algorithms do require some form of parameter tuning, with the exception of Decision Tree. Hence, in general we should not reduce model creation runtimes by eliminating parameter tuning.

In Experiment 4.3.2 we have determined that the runtime of model ceation does scale well for all algorithms considering either large values of n (number of features) or large values of m (number of instances). The algorithms in question are Support Vector Machine, Classification Rule Mining, Polynomial Regression and Logistic Regression with $^\circ = 2$. Hence these algorithms are not suitable in practice.

This means that for large datasets, the remaining algorithms, Naive Bayes, Linear Regression, Neural Network, Decision Tree, KNN, K-means and Logistic Regression with $^\circ = 1$, have capabilities of creating a model withing the required eight hours. These algorithms are able to create a model within one Hour for large datasets. Most algorithms can even generate models within a minute and a few even within a few seconds. This is more than fast enough for the application in C-Profile. Instead of waiting weeks for an analysis of the data and the resulting predictions, C-Profile can provide it within minutes. This has a huge impact on the scalability of the platform.

6.1.3 How well can machine learning algorithms handle irrelevant features?

We answered this question using Experiment 4.4. The results are very convincing. The irrelevant features do not have influence on the accuracy of the algorithms. Hence machine learning is excellent in separating relevant features from irrelevant features. This means that we can generate accurate leads without interference from a data scientist. This improves the speed in which we can give predictions for insurance intermediaries.

6.1.4 Verdict

The answers of the three questions above help us answer the main research question defined in Section 1.4: *Which machine learning technique has the highest potential to replace statistical analysis for predicting insurance product interest?*. The conclusion in Section 6.1.3 does not help us get closer to the answer, because all algorithms have no problem with irrelevant features. Hence, the result depends on the runtime and the accuracy of the algorithms.

In Section 6.1.2 we have determined that Support Vector Machine, Classification Rule Mining, Regression regression and Logistic Regression with $^\circ = 2$ are not scalable and are therefore not suitable in practice. The other six algorithms, and Logistic Regression with $^\circ = 1$ only, are suitable. They can make create a model for predictions within the specified eight hours with ease.

With that in mind we can filter the most accurate algorithms as concluded in Section 6.1.1. If we only take correctness in consideration, the Decision Tree algorithm has the highest potential, because it consistently generates the best PE scores. On the other hand, if we take convincing probabilities behind the predictions into account, the algorithm of choice would be Logistic Regression (with $^\circ = 1$), because it generates the best MSE scores.

Logistic Regressing has the highest potential considering the MSE measurement. We believe that this measurement is more important than PE as an indicator of the potential accuracy of the algorithm. We do however, believe that the Decision Tree algorithm has the highest potential if it can be enhanced with a probabilistic model. Furthermore, we also believe that the K-means algorithm has some potential, because it is consistently in the upper 50% in accuracy, for both PE and MSE measures, even though it is not a very complex method.

6.2 Future work

In this study, we have only implemented relatively simple versions of the algorithms. This means that the algorithms might still have potential to improve. It is worth investigating if the algorithms with the highest potential (Decision Tree, Logistic Regression and K-Means) can give better results. Similarly, it might be worth investigating if the runtime of Support Vector Machine and Classification Rule Mining can be improved, because their models did result in high accuracy.

Besides applying machine learning to predict which insurance products a client is interested in, C-Profile might want to investigate if it can use machine learning in other aspects of its platform. For instance in information generation. Machine learning might be able to give insights in the type of content clients are interested in.

This study has shown that C-Profile can certainly benefit from machine learning to generate Advice leads. They should pursue the Decision Tree and Logistic Regression algorithms and implement them into their platform.

Bibliography

[1] Luai Al Shalabi, Zyad Shaaban, and Basel Kasasbeh. Data mining: A preprocessing engine. *Journal of Computer Science*, 2(9):735–739, 2006. 12

[2] Shun-ichi Amari and Si Wu. Improving support vector machine classifiers by modifying kernel functions. *Neural Networks*, 12(6):783–789, 1999. 20, 82

[3] Christopher M Bishop. *Neural networks for pattern recognition*. Oxford university press, 1995. 17, 19

[4] Paul S Bradley and Usama M Fayyad. Refining initial points for k-means clustering. In *ICML*, volume 98, pages 91–99. Citeseer, 1998. 18, 75

[5] Ward Broeders. On the production efficiency of dutch insurance intermediaries. Master's thesis, Tilburg School of Economics and Management, 2016. 2

[6] M Emre Celebi and Kemal Aydin. *Unsupervised Learning Algorithms*. Springer, 2016. 18

[7] Urjan Claassen. *De toekomst van particulier advies*. Vakmedianet, 2016. 1

[8] William Cohen, Pradeep Ravikumar, and Stephen Fienberg. A comparison of string metrics for matching names and records. In *Kdd workshop on data cleaning and object consolidation*, volume 3, pages 73–78, 2003. 11

[9] Jesse Davis and Mark Goadrich. The relationship between precision-recall and roc curves. In *Proceedings of the 23rd international conference on Machine learning*, pages 233–240. ACM, 2006. 13, 14

[10] Sam Drazin and Matt Montag. Decision tree analysis using weka. *Machine Learning-Project II, University of Miami*, pages 1–3, 2012. 84

[11] J Friedman and Bogdan E Popescu. Gradient directed regularization for linear regression and classification. Technical report, Citeseer, 2003. 69

[12] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Aistats*, volume 9, pages 249–256, 2010. 76

[13] Gongde Guo, Hui Wang, David Bell, Yaxin Bi, and Kieran Greer. Knn model-based approach in classification. In *OTM Confederated International Conferences" On the Move to Meaningful Internet Systems"*, pages 986–996. Springer, 2003. 74

[14] LI Hang. A short introduction to learning to rank. *IEICE TRANSACTIONS on Information and Systems*, 94(10):1854–1862, 2011. 10

[15] Badr HSSINA, Abdelkarim MERBOUHA, Hanane EZZIKOURI, and Mohammed ERRITALI. A comparative study of decision tree id3 and c4.5. 22

[16] Gareth James, Daniela Witten, Trevor Hastie, and Robert Tibshirani. *An introduction to statistical learning*, volume 6. Springer, 2013. 10, 17

[17] Thorsten Joachims. Text categorization with support vector machines: Learning with many relevant features. In *European conference on machine learning*, pages 137–142. Springer, 1998. 12

[18] George H John and Pat Langley. Estimating continuous distributions in bayesian classifiers. In *Proceedings of the Eleventh conference on Uncertainty in artificial intelligence*, pages 338–345. Morgan Kaufmann Publishers Inc., 1995. 65, 66

[19] Bekir Karlik and A Vehbi Olgac. Performance analysis of various activation functions in generalized mlp architectures of neural networks. *International Journal of Artificial Intelligence and Expert Systems*, 1(4):111–122, 2011. 76

[20] Seung-Jean Kim, Kwangmoo Koh, Michael Lustig, Stephen Boyd, and Dimitry Gorinevsky. An interior-point method for large-scale ℓ_1-regularized least squares. *IEEE journal of selected topics in signal processing*, 1(4):606–617, 2007. 68

[21] Sotiris B Kotsiantis, I Zaharakis, and P Pintelas. Supervised machine learning: A review of classification techniques, 2007. 15, 18

[22] Ron Kupers and Maurice Ptito. ”seeing” through the tongue: cross-modal plasticity in the congenitally blind. In *International Congress Series*, volume 1270, pages 79–84. Elsevier, 2004. 76

[23] Wee Sun Lee and Bing Liu. Learning with positive and unlabeled examples using weighted logistic regression. In *ICML*, volume 3, pages 448–455, 2003. 72

[24] John Mingers. An empirical comparison of pruning methods for decision tree induction. *Machine learning*, 4(2):227–243, 1989. 83

[25] Ismail Bin Mohamad and Dauda Usman. Standardization and its effects on k-means clustering algorithm. *Res. J. Appl. Sci. Eng. Technol*, 6(17):3299–3303, 2013. 12, 28, 31

[26] Kevin P Murphy. *Machine learning: a probabilistic perspective*. MIT press, 2012. 9, 10

[27] Andrew Ng. Linear regression with multiple variables - gradient descent in practice ii: learning rate, 2015. 70

[28] Andrew Ng. Regularized logistic regression, 2015. 17

[29] Andreas Nuchter, Kai Lingemann, and Joachim Hertzberg. Cached kd tree search for icp algorithms. In *3-D Digital Imaging and Modeling, 2007. 3DIM'07. Sixth International Conference on*, pages 419–426. IEEE, 2007. 73

[30] John S Oakland. *Statistical process control*. Routledge, 2007. 12

[31] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. Scikit-learn: Machine learning in python. *Journal of Machine Learning Research*, 12(Oct):2825–2830, 2011. 25

[32] Anurag Srivastava, Eui-Hong Han, Vipin Kumar, and Vineet Singh. Parallel formulations of decision-tree classification algorithms. In *High Performance Data Mining*, pages 237–261. Springer, 1999. 20, 83

[33] Andreas Stolcke, Sachin Kajarekar, and Luciana Ferrer. Nonparametric feature normalization for svm-based speaker verification. In *Acoustics, Speech and Signal Processing, 2008. ICASSP 2008. IEEE International Conference on*, pages 1577–1580. IEEE, 2008. 12

[34] Krysta Marie Svore, Lucy Vanderwende, and Christopher JC Burges. Enhancing single-document summarization by combining ranknet and third-party sources. In *Emnlp-conll*, pages 448–457, 2007. 10

[35] Ben Taskar, Vassil Chatalbashev, Daphne Koller, and Carlos Guestrin. Learning structured prediction models: A large margin approach. In *Proceedings of the 22nd international conference on Machine learning*, pages 896–903. ACM, 2005. 10

[36] Ioannis Tsochantaridis, Thomas Hofmann, Thorsten Joachims, and Yasemin Altun. Support vector machine learning for interdependent and structured output spaces. In *Proceedings of the twenty-first international conference on Machine learning*, page 104. ACM, 2004. 20, 82

[37] P. van der Putten and M. van Someren (eds). Coil challenge 2000: The insurance company case. Technical report, Sentient Machine Research, Amsterdam, June 2000. 7

[38] Xindong Wu, Vipin Kumar, J Ross Quinlan, Joydeep Ghosh, Qiang Yang, Hiroshi Motoda, Geoffrey J McLachlan, Angus Ng, Bing Liu, S Yu Philip, et al. Top 10 algorithms in data mining. *Knowledge and information systems*, 14(1):1–37, 2008. 21, 85

[39] Harry Zhang. The optimality of naive bayes. *AA*, 1(2):3, 2004. 65, 66

Appendix A

Explanation of algorithms

A.1 Naive Bayes

Naive Bayes is a fairly simple model for classification. It is based on Bayes' rule:

$$p(Y \mid X) = \frac{p(X \mid Y)\, p(Y)}{p(X)} \tag{A.1}$$

We can adapt this to our classification problem for an example x by setting $X = x$ and $Y = y$ for a concrete class y. X is classified as class $Y = 1$ if

$$h_b(X) = \frac{p(Y = 1 \mid X)}{p(Y = 0 \mid X)} \geq 1 \tag{A.2}$$

X is classified as class $Y = 0$ otherwise. $h_b(X)$ is called a Bayesian classifier. [39] If we assume that all features are conditionally independent, we get

$$p(X = x \mid Y = y) = p(\bigwedge_{j=1}^{n} X_j = x_j | Y = y) = \prod_{j=1}^{n} p(X_j = x_j \mid Y = y) \tag{A.3}$$

[18] which results in the following classifier

$$h_{nb}(X) = \prod_{j=1}^{n} \frac{p(Y = 1 \mid X_j)}{p(Y = 0 \mid X_j)} \tag{A.4}$$

$h_{nb}(X)$ is called a naive Bayesian classifier.

If we are using discrete values it would be fairly easy to come up with the estimated probabilities $p(X)$ and $p(X \mid Y)$ for each value of X and Y, using $\mathcal{T}$. However when using continuous variables this would not work since then $p(X = x \mid Y = y)$ would be 0 for any value of x. If we assume that the data is a normal distribution, we can use the Gaussian probability density function

$$g(x, \mu, \sigma) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(x-\mu)^2}{2\sigma^2}} \tag{A.5}$$

$$p(X = x \mid Y = y) = g(x, \mu_{x|y}, \sigma_{x|y}) \tag{A.6}$$

[18] This means we only need to estimate the mean $\mu_{x|y}$ and standard deviation $\sigma_{x|y}$ from $\mathcal{T}$. This is fairly straight forward, where

$$\mu_{x|y} = \frac{\sum\limits_{\{X,Y\}\in\mathcal{T}|Y=y} X}{\sum\limits_{\{X,Y\}\in\mathcal{T}|Y=y} 1} \tag{A.7}$$

and

$$\sigma_{x|y} = \frac{\sum\limits_{\{X,Y\}\in\mathcal{T}|Y=y} (X - \mu_{x|y})^2}{\sum\limits_{\{X,Y\}\in\mathcal{T}|Y=y} 1} \tag{A.8}$$

Note that for strict classification purposes the denominator of Equation A.1 does not need to be calculated when using the naive Bayes classifier, because they cancel out in Equation A.2. See Equation A.9.

$$\frac{p(Y = 1 \mid X)}{p(Y = 0 \mid X)} = \frac{\frac{p(X|Y=1)\,p(Y=1)}{p(X)}}{\frac{p(X|Y=0)\,p(Y=0)}{p(X)}} = \frac{p(X \mid Y = 1)\,p(Y = 1)}{p(X \mid Y = 0)\,p(Y = 0)} \tag{A.9}$$

However, we prefer to get a probability as our result, hence we will also calculate the denominator. In practice this means we calculate Equation A.1 for both $Y = 0$ and $Y = 1$ and use the highest probability for classification.

A.1.1 Assumptions

As mentioned earlier naive Bayes makes two assumptions. Conditional independence of the features and the normal distribution of features values within a class. The conditional independence assumption, is a dangerous assumption to make because it is rarely true in real-world applications.[39] The normal distribution assumption is less dangerous, but should be treated with care. If the data is normalized using a Gaussian distribution (see Section 2.5) no harm is done, however if the data is not modified or modified in some other way, the results might be unexpected. Despite these two assumptions, naive Bayes has a surprisingly good performance.[39]

A.2 Linear Regression

Linear Regression (also known as linear least squares) is a statistical technique for modelling the relationship of a set of variables by a linear equation $h_\beta(x)$. The number of features determines the dimension of the equation. If we have two features x_1 and x_2, the linear equations looks as follows:

$$h_\beta(x) = \beta_2 x_2 + \beta_1 x_1 + \beta_0 \tag{A.10}$$

where β_0, $\beta_1$and β_2 are constants. We can generalize this equation by adding a constant feature $x_0 = 1$, to $h_\beta(x) = \beta_2 x_2 + \beta_1 x_1 + \beta_0 x_0$. If we vectorize this equation we get

$$h_\beta(x) = \boldsymbol{\beta}^T \mathbf{x} \tag{A.11}$$

where $\boldsymbol{\beta} = [\beta_0, \beta_1, \beta_2]$ and $\mathbf{x} = [x_0, x_1, x_2]$. This can be generalized for $n >= 1$.

Our goal is to find values for $\boldsymbol{\beta}$, to get the best equation $h_\beta(x)$ to predict values Y for instances of X. We do this by minimizing a cost function $J(\boldsymbol{\beta}, \boldsymbol{\mathcal{T}})$, which we will define as follows:

$$J(\boldsymbol{\beta}, \mathcal{T}) = \frac{1}{2m} \sum_{i=1}^{m} (h_\beta(x^{(i)}) - y^{(i)})^2 \tag{A.12}$$

This cost function is known as the squared error cost function. Note that because we are minimizing $J(\boldsymbol{\beta}, \mathcal{T})$ for a given $\mathcal{T}$, we have that $\frac{1}{2m}$ is constant, and therefore does not contribute to the equation. However, when solving the optimization problem, it is more convenient to use Equation A.12, because its partial derivative with regard to $\boldsymbol{\beta}$ is easier to calculate.

A.2.1 Overfitting

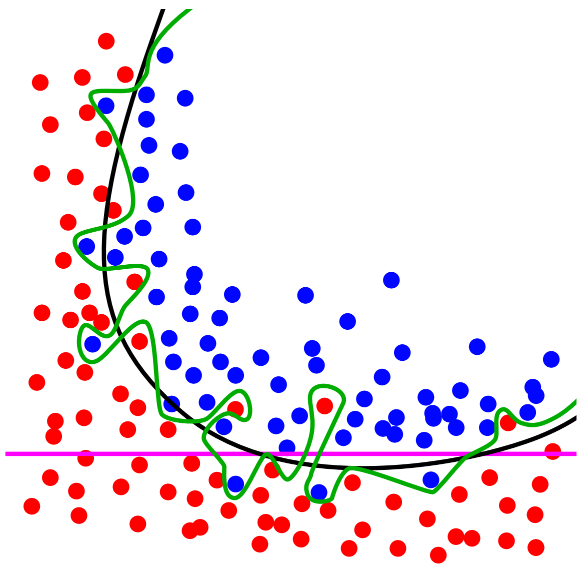


Figure A.1: The green line represents an overfitted model, the black line represents a regularised model, the magenta line represents a underfitted model. While the green line best follows the training data, it is too dependent on it and it is likely to have a higher error rate on new unseen data, compared to the black line. The magenta line is over regularized, resulting in a poor model.

The problem with this cost function is that it promotes overfitting of the model. Overfitting means that the model is catered too specifically to the training data. This means that the overfitted model would probably perform badly when it is presented with new data, because it is so depended on the data that was given. See

Figure A.1. There are various ways to counter this. The easiest method would be to decrease the number of features used. Because the complexity of the regression hypothesis is determined by the number of features, overfitting could easily occur, when a high number of features are used. However, this means that human interference is required, when hand picking the features. It is possible to use a model selection algorithm, however, loss of features means loss of data, which might be essential.

The most common method without losing features is regularization. This method reduces the magnitude of the parameters $\boldsymbol{\beta}$, by penalizing large values of these parameters. The cost function $J(\boldsymbol{\beta}, \mathcal{T})$ is altered as follows:

$$J(\boldsymbol{\beta}, \mathcal{T}) = \frac{1}{2m} \left(\sum_{i=1}^{m} (h_{\beta}(x^{(i)}) - y^{(i)})^2 + \lambda \sum_{j=1}^{n} \boldsymbol{\beta}_j^2 \right) \tag{A.13}$$

As can be seen a regularization term ($\lambda \sum_{j=1}^{n} \boldsymbol{\beta}_j^2$) is introduced. All parameters, except $\boldsymbol{\beta}_0$, are penalized. Because $\mathbf{x}_0$ is constant, it does not make a lot of sense to penalize its parameter. Furthermore, we can see that a regularization parameter λ is introduced to determine the amount of regularization needed. λ should be carefully chosen (or tuned in our case). A small value would promote overfitting, while a large value would set all values of $\boldsymbol{\beta}$ (except β_0) to zero, resulting in a horizontal line, which would be underfitted. See Figure A.1 for an example.

A.2.2 Solving minimization problem

Now that we have determined our cost function $J(\boldsymbol{\beta}, \mathcal{T})$, all we need to do is solve the minimization problem. There are two widely used methods to solve this. Gradient descent and normal equation.

Normal equation for linear least squares is defined as follows:

$$\boldsymbol{\beta} = (\mathbf{X}^T\mathbf{X} + \lambda\mathbf{I}_0)^{-1}\mathbf{X}^T\mathbf{y} \tag{A.14}$$

[20] Where $\mathbf{X}$ is a $m \times (n+1)$ matrix of training examples, while $\mathbf{y}$ is a $m \times 1$ vector of expected results and $\mathbf{I}_0$ is defined as a $(n+1) \times (n+1)$ identity matrix, with one altercation, namely: 0 at the first entry (row 1, column 1). This because we do not regularize $\boldsymbol{\beta}_0$. The normal equation method seems fairly convenient and simple. The only bottleneck is inverting $\mathbf{X}^T\mathbf{X}$, but for reasonable low number of features (n), this can be calculated fast enough with modern computer power. However, if $\mathbf{X}^T\mathbf{X}$ cannot be inverted, then the equation cannot be solved. This is mostly the case when we have redundant (linearly dependent) features, which can simply be removed because no additional information is gained.

While normal equation is an analytical solution specifically for linear least squares, gradient descent is a generic iterative solution. This means that gradient descent can be used for any problem where a function should be minimized for a number of parameters. Gradient descent tries to solve the minimization problem by descending down the slope of the function. See Algorithm 1 for reference.

As we can see in Algorithm 1, gradient descent, keeps descending down the slope until its gain is minimal (less than ε). One thing to note, is that a variable $tempResult$ is introduced. We do this to ensure that $\boldsymbol{\beta}$ is updated simultaneously. Otherwise we might get unexpected results. Furthermore we need to be able to

Algorithm 1 GrandientDescent

```
1: procedure GRANDIENTDESCENT(J, $\boldsymbol{\beta}$, $\mathcal{T}$)
2:     $newCost \leftarrow$ J$(\boldsymbol{\beta}, \mathcal{T})$
3:     repeat
4:         $oldCost \leftarrow newCost$
5:         $tempResult \leftarrow$ [LENGTH($\boldsymbol{\beta}$)]        ▷ Initialize a vector the size of $\boldsymbol{\beta}$
6:         for $i \leftarrow 1$ to LENGTH($\boldsymbol{\beta}$) do
7:             $tempResult_i \leftarrow \boldsymbol{\beta}_i - \alpha \frac{\partial}{\partial \boldsymbol{\beta}_i}$J$(\boldsymbol{\beta}, \mathcal{T})$
8:         $\boldsymbol{\beta} \leftarrow tempResult$
9:         $newCost \leftarrow$ J$(\boldsymbol{\beta}, \mathcal{T})$
10:    until $newCost - oldCost < \varepsilon$        ▷ Execute until convergence
```

calculate the partial derivative for J ($\frac{\partial}{\partial \boldsymbol{\beta}_i} J(\boldsymbol{\beta}, \mathcal{T})$) which in our case is:

$$\begin{aligned}\frac{\partial}{\partial \boldsymbol{\beta}_j} J(\boldsymbol{\beta}, \mathcal{T}) &= \frac{\partial}{\partial \boldsymbol{\beta}_j} \frac{1}{2m} \left(\sum_{i=1}^{m} (h_\beta(x^{(i)}) - y^{(i)})^2 + \lambda \sum_{j=1}^{n} \boldsymbol{\beta}_j^2 \right) \\ &= \begin{cases} \frac{1}{m} \sum_{i=1}^{m} (h_\beta(x^{(i)}) - y^{(i)}) x_j^{(i)} & \text{if } j = 0 \\ \frac{1}{m} \sum_{i=1}^{m} (h_\beta(x^{(i)}) - y^{(i)}) x_j^{(i)} + \frac{\lambda}{m} \boldsymbol{\beta}_j & \text{if } j > 0 \end{cases}\end{aligned} \tag{A.15}$$

We also have to choose a learning rate α, to determine the step size. Setting α is fairly important, because if we set α too large we might overshoot our objective, while a learning rate that is too small needs (too many) iterations. There are various ways to set α. One way is to adapt α after each iteration. If the error is decreased, α is increased, but if the error increases (i.e. we overshoot the optimum) α is decreased. Often though, $\alpha = \frac{1}{n}$ is chosen as the learning rate, which works well in practice. Finally we can see that gradient descent takes an initial $\boldsymbol{\beta}$ as its parameter. We will simply choose $\boldsymbol{\beta} = \mathbf{0}$ as our initial value.

One disadvantage of gradient descent is that it finds a local optimum which is not guaranteed to be the global optimum. However if the cost function is convex (i.e. it has only one local optimum) then gradient descent does find the global optimum. Luckily Equation A.13 is convex.[11]

A.2.3 Output

Now that we know how to determine our cost function, we have all the pieces to execute Linear Regression. However as explained in Section 2.4 we require an output in $[0, 1]$. Hence we will simply round our values back to this range. Hence we define our resulting Linear Regression function $f(x)$ as

$$f(x) = \begin{cases} 0 & \text{if } h(x) < 0 \\ 1 & \text{if } h(x) > 1 \\ h(x) & \text{otherwise} \end{cases} \tag{A.16}$$

A.2.4 Parameter tuning

Linear Regression has effectively one parameter to tune; the regularization parameter λ. For λ it would not make sense to include negative values, because this would promote infinitely large parameters. Hence only positive values makes sense. A common way to tune parameters is using a range from 10^{-3} to 10^{3} using regular increases of about 3. This means we get a range in the form $0.001, 0.003, 0.01, \ldots 300, 1000$. [27]

A.3 Polynomial Regression

Often, a linear hypothesis is not a good enough hypothesis for classification as shown in Figure A.2. Polynomial Regression allows us to use the mechanism of Linear Regression with more complicated (non-linear) functions. The process of solving the problem is exactly the same. The only difference is the model selection. Where Linear Regression has $n + 1$ features, Polynomial Regression can have any number of features. In our case we can construct new features from existing features. If, for example, we have a data set with the $length$ and $width$ of a lot, we can calculate the $area$ as $area = width \times length$. Now if we suspect that there is a cubic relation between the size of the house and the desire to buy a burglary insurance, we can use $area^3$ as one of our features.

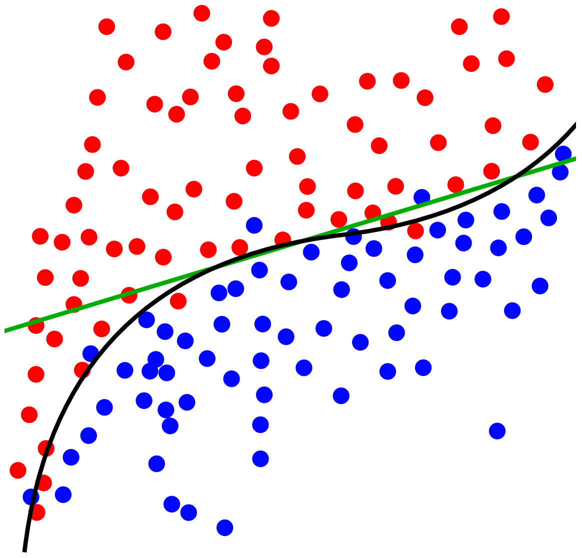


Figure A.2: The green line represents a linear model and the black line represents a cubic model. The black line does a better job separating the data, compared to the black line.

A.3.1 Picking features

New features can be generated by manual selection, but they can also be generated automatically for a certain polynomial degree ρ. See Algorithm 2. The algorithm creates every combination of multiplying ρ features exactly once. Note that $x_i \times x_j = x_j \times x_i$ for any $i, j \in \mathbb{N}$. For instance if $\rho = 3$ and $n = 4$, one of the combinations is $x_1 \times x_3 \times x_3$.

Because the algorithm creates every combination of ρ features, the algorithm should be called with $\mathbf{x}$ containing the constant feature $\mathbf{x}_0$. Including $\mathbf{x}_0 = 1$ ensures that actually all polynomials are generated, also the ones with fewer terms, since $1^i \times x_j = x_j$ for any $i, j \in \mathbb{N}$. An example of such a combination is x_2^2, which should also be included as a feature.

The disadvantage of this automated method is that the number of features increases significantly when n andρ increases at a rate of $O(n^\rho)$ which in turn makes the algorithm slower. Therefore in practice we will only use $\rho = 1$ and $\rho = 2$.

Algorithm 2 GeneratePolynomials

```
1:  procedure GENERATEPOLYNOMIALS($\rho$, $\mathbf{x}$)
2:      $indices \leftarrow []$
3:      for $i \leftarrow 1$ to $\rho$ do                          ▷ Initialize all indices to 1
4:          PUSH($indices, 1$)
5:      $result \leftarrow []$
6:      $startCounter = 1$
7:      while $indices[1] <$ LENGTH($\mathbf{x}$) do               ▷ We work from back to front so the first index determines when to stop
8:          for $i \leftarrow startCounter$ to LENGTH($\mathbf{x}$) do
9:              $indices[\rho] = i$                                 ▷ Set the index of the feature we are working on
10:             $newFeature \leftarrow 1$
11:             for $j \leftarrow 1$ to $\rho$ do
12:                 $newFeature \leftarrow newFeature \times \mathbf{x}_{indices[j]}$
13:             PUSH($result, newFeature$)
14:         $nextIndex = \rho$
15:         while $indices[nextIndex] >$ LENGTH($\mathbf{x}$) do    ▷ Check which index is next to increase
16:             $nextIndex \leftarrow nextIndex - 1$
17:         $startCounter = indices[nextIndex] + 1$
18:         for $i \leftarrow nextIndex$ to $\rho$ do               ▷ Set all later indices to $indices[nextIndex]$ so we do not get duplicates
19:             $indices[i] = startCounter$
20:     return $result$
```

A.4 Logistic Regression

Logistic Regression is a classification algorithm based on the mechanics of linear (or polynomial) regression. Where the Linear Regression function $h_\beta(x) \in \mathbb{R}$, we have that the logistic function $h_\beta(x) \in [0, 1]$. This means it outputs a value we will interpret as a probability suited for classification, without any further post processing. In order to get this output range, we introduce a new hypothesis defined as follows:

$$h_\beta(x) = g(\boldsymbol{\beta}^T \mathbf{x}) \tag{A.17}$$

$$g(z) = \frac{1}{1+e^{-z}} \tag{A.18}$$

Equation A.18 is known as the sigmoid function. The sigmoid function is asymptotically bound by $[0, 1]$.

Now that we have a new hypothesis, can we still use the same cost function for determining $\boldsymbol{\beta}$? Unfortunately $J(\boldsymbol{\beta}, \mathcal{T})$ as defined in Equation A.13 is non-convex for our new cost function. This means gradient descent would not be guaranteed to find the optimal solution. Hence we will define a new cost function $J(\boldsymbol{\beta}, \mathcal{T})$ as

$$J(\boldsymbol{\beta}, \mathcal{T}) = -\frac{1}{m}\left(\sum_{i=1}^{m} y^{(i)} \log h_\beta(x^{(i)}) + (1-y^{(i)})\log(1-h_\beta(x^{(i)}))\right) + \frac{\lambda}{2m}\sum_{j=1}^{n}\boldsymbol{\beta}_j^2 \tag{A.19}$$

This cost function is derived from the Maximum likelihood estimation principle.[23] This means that we have an assumption that the data has a normal distribution. But as discussed in Appendix A.1 this is not a big problem.

So how does this function work? The inner part of the sum is the most interesting. Lets define it as $i(x, y) = y \log h_\beta(x) + (1-y)\log(1-h_\beta(x))$. If $y = 1$ and $h(x) = 1$ we get that $i(x, y) = 1 \log 1 + 0 \log 0 = 1 \times 0 + 0 \times -\infty = 0$, so no penalty is given when the hypothesis is exactly right. However if $y = 1$ and $h(x) = 0$ we get that $i(x, y) = 1 \log 0 + 0 \log 1 = 1 \times -\infty + 0 \times 0 = -\infty$ which, due to the minus sign in $J(\boldsymbol{\beta}, \mathcal{T})$ is a huge error. This is justified, because the cost function was completely off.

As discussed in Appendix A.2.2 the update rule for gradient descent requires us to calculate $\frac{\partial}{\partial \boldsymbol{\beta}_j} J(\boldsymbol{\beta}, \mathcal{T})$. With our new cost function we get

$$\begin{aligned}\frac{\partial}{\partial \boldsymbol{\beta}_j} J(\boldsymbol{\beta}, \mathcal{T}) &= \frac{\partial}{\partial \boldsymbol{\beta}_j}\left(-\frac{1}{m}\left(\sum_{i=1}^{m} y^{(i)} \log h_\beta(x^{(i)}) + (1-y^{(i)})\log(1-h_\beta(x^{(i)}))\right) + \frac{\lambda}{2m}\sum_{j=1}^{n}\boldsymbol{\beta}_j^2\right) \\ &= \begin{cases} \frac{1}{m}\sum_{i=1}^{m}(h_\beta(x^{(i)}) - y^{(i)})x_j^{(i)} & \text{if } j = 0 \\ \frac{1}{m}\sum_{i=1}^{m}(h_\beta(x^{(i)}) - y^{(i)})x_j^{(i)} + \frac{\lambda}{m}\boldsymbol{\beta}_j & \text{if } j > 0 \end{cases}\end{aligned} \tag{A.20}$$

Which is the same as Equation A.15 except that $h_\beta(x)$ is defined differently. Hence gradient descent remains the same algorithm for both linear and Logistic Regression, and since the cost function is still convex [23], gradient descent finds the global optimum.

A.5 K-nearest neighbours

K-nearest neighbours, also known as KNN, is one of the simplest classification algorithms. Classification occurs by comparing a test instance $\{x_z, y_z\}$ to all training examples $\{x, y\}$, and choosing the k nearest neighbours. A (weighted) average of the classifications of these neighbours is used to calculate the classification of $\{x_z, y_z\}$.

KNN queries are performed using a kd-tree. A kd-tree (k dimensional tree) is a data structure specialized in multidimensional nearest neighbour searches. In this case the k is the number of dimensions of the values in the tree. In our case $k = n$, since $\mathbf{x}$ are n dimensional vectors. kd-trees are constructed, by splitting the values on one dimension in each node. To balance the tree, the median of the values in the particular dimension is used. This ensures that the kd-tree is balanced (all nodes are approximately the same distance from the root). See Figure A.3 for a kd tree with $k = 2$.

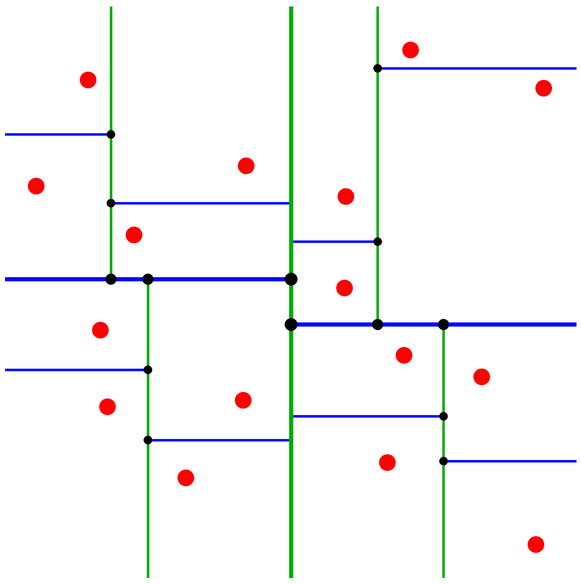


Figure A.3: A 2d-tree. Green lines separate the data on the first (x) dimension, blue lines separate the data on the second (y) dimension. Every line is a node, every black dot is a vertex and every cell is a leaf.

A.5.1 Classification

Nearest neighbours are recursively found by traversing the tree in a depth first search manner. At every node the best minimum distance so far is maintained and updated when necessary. if the hyper sphere of the minimum distance so far does not intersect any of the regions in the sub tree. The entire sub tree can be discarded, and the search can continue in a different part of the tree. This is done until all nodes are discarded, at which point the nearest neighbour is found. This process can be generalized by maintaining the k nearest neighbours and using the furthest neighbour (the k-th neighbour) as the minimum distance. See [29] for a more detailed explanation on kd-tree nearest neighbour queries.

As mentioned earlier, classification of an unknown instance x is assigned by the average of the classification of its k nearest neighbours. This however means that far neighbours are equally weighted as close neighbours. Intuitively this seems wrong. To counter this we can use a weighted average. The inverse of the distance is the weight. If the distance of one of the neighbours is 0, then that classification is used, since 0 cannot be inverted. for the set of k nearest neighbours K the formula becomes:

$$h(x) = \frac{\sum\limits_{\{x_z, y_z\} \in K} \frac{1}{d(x, x_z)^2} y_z}{\sum\limits_{\{x_z, y_z\} \in K} \frac{1}{d(x, x_z)^2}} \tag{A.21}$$

where $d(x, z)$ is a distance measure, Euclidean distance in our case.

$$d_{Euclidean}(x, z) = \sqrt{\sum_{i=1}^{n} (x_i - z_i)^2} \tag{A.22}$$

A.5.2 Parameter tuning

KNN's parameter to tune is k, the number of neighbours to consider for classification. Naturally we have $k \in \mathbb{N}$. Low values for k result in complex models, because outliers can have a high influence on the classification. Large values of k have a regularization effect. This because individual data points have less influence on the classification results. Too large values of k however, will favour the majority, resulting in underfitting the model.

There are not many guidelines on the values for k because it is largely data dependent. [13] Therefore we will use a wide range of values. For the small values we will use a step of 1, and after we will decrease the values with a factor 2. The resulting range becomes: $[1, 2, 3, 4, 8, 16, 32, 64]$.

Another (binary) parameter to tune is whether we use weighted KNN or unweighted KNN. Weighted KNN uses the classification as defined in Equation A.21. For unweighted KNN, the weights are set to 1. the resulting classifier then becomes

$$h(x) = \frac{1}{\|K\|} \sum_{\{x_z, y_z\} \in K} y_z \tag{A.23}$$

A.6 K-means classification

K-means classification is a clustering algorithm. Clustering is usually not associated with classification, however with some modification, clustering algorithms can be adopted to classification algorithms. The idea is to cluster the data, and assign classifications to the clusters, based on the data points in the cluster using the average of classifications. New points can be classified by identifying the cluster they belong to. K-means is one of the most well known clustering algorithms. It assigns each data point to one of the k clusters based on the minimum sum of squares to the mean of the cluster. This means that if we have sets C_i for cluster i, we get the following objective function to minimize:

$$J(\mathcal{T}, k) = \sum_{i=1}^{k} \sum_{x \in C_i} \|x - \mu_i\|^2 \tag{A.24}$$

[4]

A.6.1 Minimizing the cost function

This objective function is minimized using the k-means algorithms. The k-means algorithm is initialized using random initialization. k random points are assigned as cluster means μ_i for $1 \leq i \leq k$. After initialization, the algorithm has two steps; the assignment step, and the update step. The assignment step assigns each point to one of the cluster means, such that the sum of squares of the distance is minimized. This means for a point x_j is assigned to cluster C_i such that:

$$C_i = \{x_j \mid \|x_j - \mu_i\|^2 \leq \|x_i - \mu_l\|^2 \ \forall l \in \{l \mid 1 \leq l \leq k\} l \neq j \tag{A.25}$$

The update step, updates the means μ_i to the centroids (the mean of the cluster points) of each cluster C_i for $1 \leq i \leq k$.

$$\mu_i = \frac{1}{\mid S_i \mid} \sum_{x \in C_i} x \tag{A.26}$$

These steps are repeated until the means do not change anymore.

After the clusters are found, they receive a classification based on the average of the points inside the cluster. A new data point is then classified by finding the cluster it belongs to using the same process as in the assignment-step, see Equation A.25.

A.6.2 Parameter tuning

k in K-Means determines the number of clusters to use for classification. Large amounts of clusters means on average a small amount of data points are used in classification. Small amounts of clusters means on average a large amount of data points are used in classification. Because k is a value that determines the classifications of the entire data set, we need to consider very large values. We will use the same tactics as in Appendix A.5.2, but with a higher upper bound. The range becomes [2, 3, 4, 8, 16, 32, 64, 128, 256, 512, 1024, 2048, 4096, 8192]

A.7 Neural Network

Neural Network is a non-linear classification algorithm. Polynomial Regression is also able to learn non-linear hypotheses. However, as discussed in Appendix A.3, the algorithm run time blows up if the non linearity is increased. Neural Networks is better at handling such non-linearity.

Neural Networks are inspired by the learning capability of the human brain. The human brain is capable of sensory substitution, which refers to the capacity of the brain to replace the function of a lost sense by another sensory modality [22]. This means there exists one learning algorithm, which can learn how to see, how to hear, etc. based on the (sensory) data it receives. This is very powerful and precisely the inspiration behind Neural Networks.

As its name already suggests, Neural Networks consist of various connected neurons. Neurons consist of a computational body (the cell body compared to neurons in the human brain), input wires (the dendrite) and output wires (the axon). The computational body has an activation function $h_\beta(\mathbf{x})$ which uses the values of the input wires $(\mathbf{x})$ as input and outputs its value to all output wires. Often a form of the sigmoid function is used as the activation function. However, it is shown that the hyperbolic tangent (see Equation A.28) shows better results.[19] [12] The activation function becomes:

$$h_\beta(x) = tanh(\boldsymbol{\beta}^T\mathbf{x}) \tag{A.27}$$

$$tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}} \tag{A.28}$$

See Figure A.4 for an example of a neuron.

The networks often consist of several layers, where a layer only has input connections from the previous layer. Such a Neural Network is called a feed-forward Neural Network. We distinguish three types of layers. The input layer is the first layer in the network and coincides with the features used from the data set. The output layer is the last layer in the network and determines the classification. For binary classification this always consists of one node. The output value of this node is the predicted value. The third type of layer, is the hidden layer. These are all layers between the input and output layer. These layers transform the data between input and output layers. More hidden layers means the model becomes more complex. As, expected, the computation time also grows with more hidden layers. Therefore, it should carefully considered when a more complex model actually improves the results. Each layer, with the exception of the output layer, has a bias node as well. The bias node has no input connections, and always outputs one. It is similar to the x_0 feature in regression (see Appendix A.2).

A.7.1 Hypothesis

In order to explain the calculations of a Neural Network we will introduce some notation. The number of layers in our Neural Network will be denoted by L. We will introduce matrices $\boldsymbol{\beta}^{(j)}$ as the matrix of parameters (or weights) controlling function mapping from layer j to layer $j+1$. Note that the size of $\boldsymbol{\beta}_j$ depends on the number of neurons in layers j and $j+1$. If s_j denotes the number of neurons in layer j. We get that $\boldsymbol{\beta}_j$ is a $s_{j+1} \times s_j + 1$ matrix. Furthermore we will use $a_i^{(j)}$ as the activation of neuron i in layer j. The activation is the result of the neuron's activation function. for layer 1, the input layer, we will set $a_i^{(1)} = x_i$ Finally we will introduce $z_i^{(j)}$ as the weighted linear combination of input values in a neuron i in

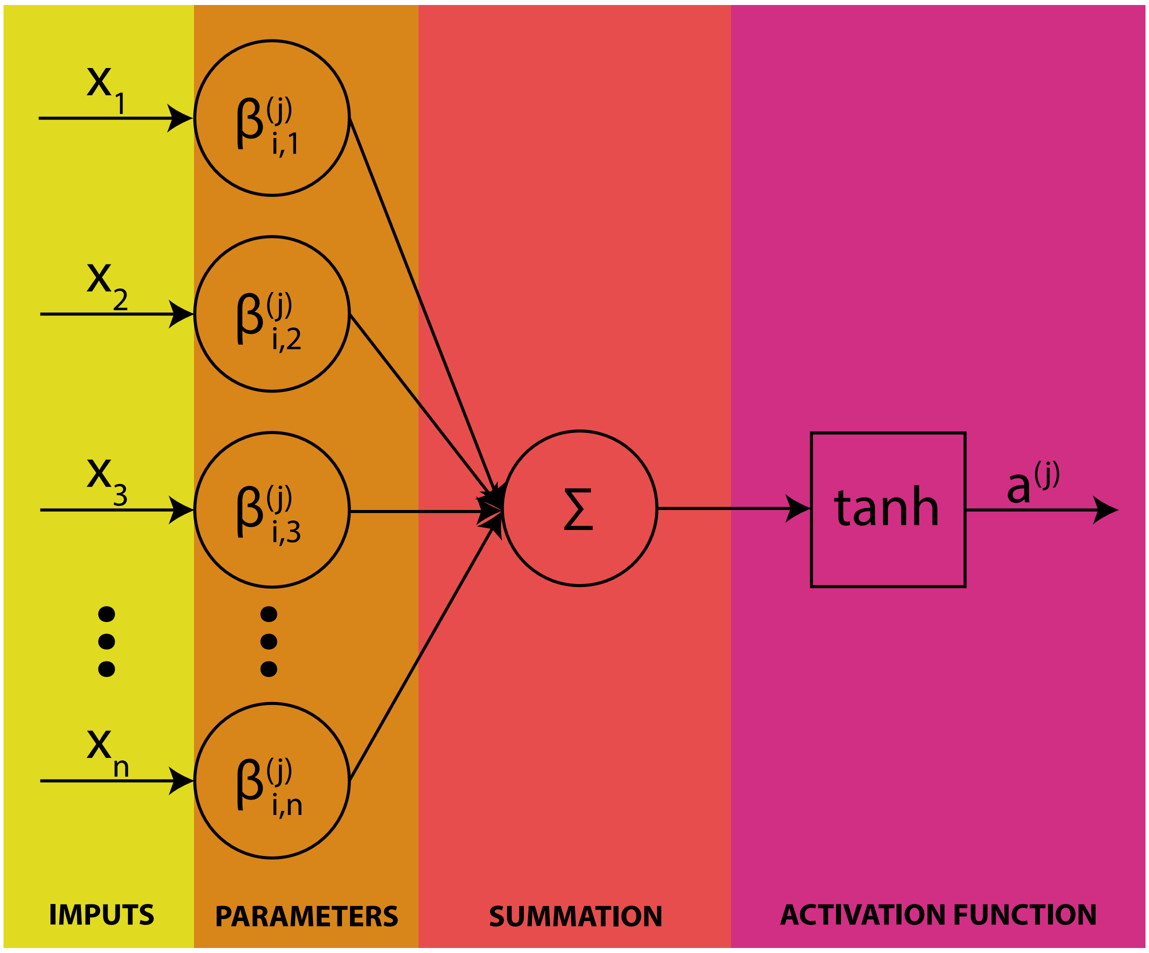


Figure A.4: A depiction of a neuron with the hyperbolic tangent as activation function

layer j.

$$z_i^{(j)} = \sum_{k=0}^{s_j} \beta_{i,k}^{(j-1)} a_k^{(j-1)} \tag{A.29}$$

The concrete calculation of $a_i^{(j)}$ becomes:

$$a_i^{(j)} = tanh(z_i^{(j)}) \tag{A.30}$$

See Figure A.5 for an example of a Neural Network.

Using these equations we can calculate $h_\beta(x)$ using an algorithm called forward propagation. See Algorithm 3.

Note that because we are using $tanh(x)$ as activation function we get that $h_\beta(x) \in [-1, 1]$. However as

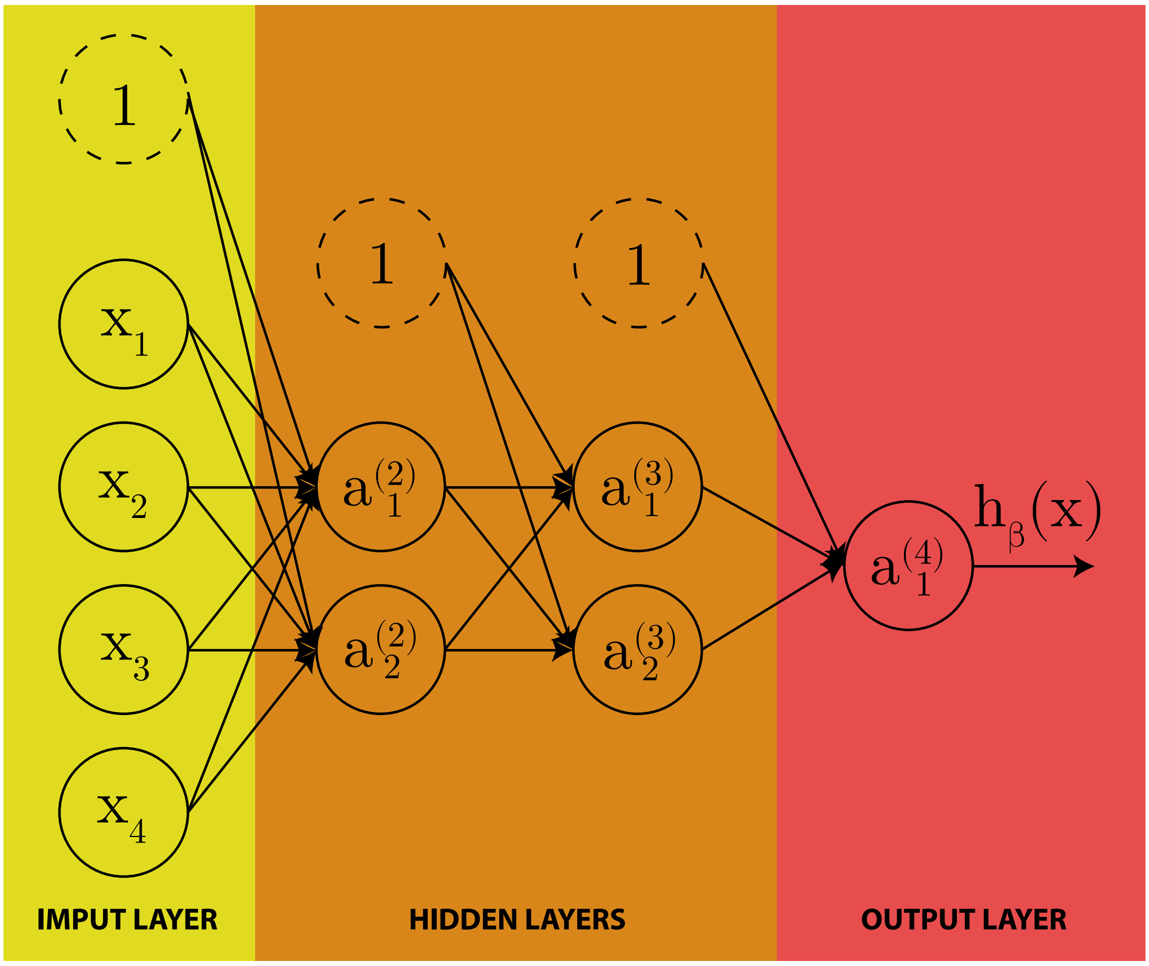


Figure A.5: A feed-forward Neural Network with 4 input features, 2 hidden layers, and one output feature.

explained in Section 2.4 we require an output in range $[0, 1]$. Hence we will post process this result as $p(y = 1|x) = \frac{h_\beta(x)+1}{2}$

A.7.2 Minimizing the cost function

Just like Logistic Regression, Neural Networks learn by determining the values of $\boldsymbol{\beta}$ by minimizing a cost function. Where Logistic Regression, has one vector $\boldsymbol{\beta}$, Neural Networks have a $\boldsymbol{\beta}$ matrix for each layer. However The cost function for Neural Networks is similar to the one for Logistic Regression. The only difference is the regularization part of the cost function. This because we need to take the extra parameters

Algorithm 3 ForwardPropagation

```
1: procedure FORWARDPROPAGATION($L$, $\boldsymbol{\beta}$, $\mathbf{x}$)
2:     $a \leftarrow \mathbf{0}$
3:     $z \leftarrow \mathbf{0}$
4:     $a^{(1)} \leftarrow x$
5:     for $j \leftarrow 1$ to $L-1$ do
6:         $z^{(j+1)} = \boldsymbol{\beta}^{(j)} a^{(j)}$
7:         $a^{(j+1)} = tanh(z^{(j+1)})$
8:     return $a^{(L)}$
```

β into account. See Equation A.31

$$J(\boldsymbol{\beta}, \mathcal{T}) = -\frac{1}{m}\left(\sum_{i=1}^{m} y^{(i)} \log h_\beta(x^{(i)}) + (1-y^{(i)}) \log(1-h_\beta(x^{(i)}))\right) + \frac{\lambda}{2m}\sum_{l=1}^{L-1}\sum_{j=1}^{s_l}\sum_{i=1}^{s_l+1}(\boldsymbol{\beta}_{i,j}^{(l)})^2 \tag{A.31}$$

Once again we will use gradient descent in order to minimize this cost function. Therefore we will need to be able to compute $\frac{\partial}{\partial \boldsymbol{\beta}_{i,j}^{(l)}} J(\boldsymbol{\beta}, \mathcal{T})$. This is where the back propagation algorithm comes in. The back propagation algorithm computes the partial derivative by computing error terms $\boldsymbol{\delta}^{(j)}$ for layer j. We compute error terms for $2 \geq j \geq L$, because we do not associate an error term to the input layer. We calculate these error terms from the last layer up to the second layer, hence the algorithm is called back propagation. The formula for the error terms $\boldsymbol{\delta}^{(j)}$ is as follows:

$$\boldsymbol{\delta}^{(j)} = \begin{cases} y - a^{(j)} & \text{if } j = L \\ (\boldsymbol{\beta}^{(k)})^T \boldsymbol{\delta}^{(j+1)} \circ \frac{\partial}{\partial \mathbf{z}^{(j)}} tanh(\mathbf{z}^{(j)}) & \text{if } j < L \end{cases} \tag{A.32}$$

where $\circ$ denotes the element wise multiplication of two vectors (also known as the Hadamard product) and where

$$\frac{\partial}{\partial \mathbf{z}^{(j)}} tanh(\mathbf{z}^{(j)}) = \mathbf{a}^{(j)} \circ (\mathbf{1} - \mathbf{a}^{(j)}) \tag{A.33}$$

Using Equation A.32 we can define the back propagation algorithm to calculate $\frac{\partial}{\partial \boldsymbol{\beta}_{i,j}^{(l)}} J(\boldsymbol{\beta}, \mathcal{T})$. See Algorithm 4 for the pseudo code.

A.7.3 Initialization

Now we have all the ingredients to perform gradient descent and minimize $J(\boldsymbol{\beta}, \mathcal{T})$. However as you might recall gradient descent takes an initial value of $\boldsymbol{\beta}$. Unfortunately we cannot choose $\boldsymbol{\beta} = \mathbf{0}$ as our initial β because all neurons in layer j for $1 > j \geq L$ will have the same activation. Which in turn means that gradient descent will update the weights of the inputs for each neuron in layer j for $1 > j \geq L$ the same.

Algorithm 4 BackPropagation

```
1:  procedure BACKPROPAGATION($\mathcal{T}$, $L$, $\boldsymbol{\beta}$)
2:      $\boldsymbol{\Delta}^{(l)}_{i,j} \leftarrow 0$                                        ▷ for all $i, j, l$
3:      for $i \leftarrow 1$ to $m$ do
4:          FORWARDPROPAGATION($L, \boldsymbol{\beta}, \mathbf{x}^{(i)}$)                   ▷ Calculate all $\mathbf{a}^{(l)}$
5:          $\boldsymbol{\delta} \leftarrow \mathbf{0}$
6:          for $j \leftarrow L$ to $2$ do
7:              $\boldsymbol{\delta}^{(j)} \leftarrow \ldots$                     ▷ Calculate $\boldsymbol{\delta}^{(j)}$ using Equation A.32
8:          $\boldsymbol{\Delta}^{(l)} \leftarrow \boldsymbol{\Delta}^{(l)} + \boldsymbol{\delta}^{(l+1)}(\mathbf{a}^{(l)})^T$                 ▷ for all $l$
9:      $\mathbf{D}^{(l)}_{i,j} \leftarrow \frac{1}{m}\boldsymbol{\Delta}^{(l)}_{i,j} + \lambda\boldsymbol{\beta}^{(l)}_{i,j}$              ▷ for all $i, j, l$ if $j > 0$
10:     $\mathbf{D}^{(l)}_{i,j} \leftarrow \frac{1}{m}\boldsymbol{\Delta}^{(l)}_{i,j}$                         ▷ for all $i, j, l$ if $j = 0$
11:     return $\mathbf{D}$
```

Therefore we will use random initialization we will initialize $\boldsymbol{\beta}^{(l)}_{i,j}$ to a random value in $[-\varepsilon, \varepsilon]$. This will insure symmetry breaking, meaning that the nodes are not likely to compute the same activation function. ε is a small value close to 0.

A.7.4 Parameter tuning

Just like Linear Regression, Neural Networks have a regularization parameter. We will tune this parameter the same way as described in Appendix A.2.4 for Linear Regression.

A.8 Support Vector Machine

Support Vector Machine also known as SVM is a large margin classifier. SVM tries to separate the data in such a way that the margin between any of the training examples is as large as possible. See Figure A.6.

The hyperplane separating the data is of the form $\boldsymbol{\beta}^T\mathbf{x} = 0$. This means that

$$h_\beta(x) = \begin{cases} 1 & \text{if } \boldsymbol{\beta}^T\mathbf{x} \geq 0 \\ 0 & \text{if } \boldsymbol{\beta}^T\mathbf{x} < 0 \end{cases} \tag{A.34}$$

Hence we have no probabilities, but only classifications. (i.e. values in $\{0, 1\}$)

Due to the formula of the hyperplane we have that $\boldsymbol{\beta}$ is normal to the hyperplane. We will use this to calculate the margin. The margin is nothing less than twice the distance from the plane to the nearest point. Hence we have to be able to calculate the distance of the hyper plane to each point. We do this by projecting each point $\mathbf{x}^{(i)}$ to $\boldsymbol{\beta}$. We will call the projection, $\mathbf{p}^{(i)}$. We define $\mathbf{p}^{(i)}$ as follows:

$$\mathbf{p}^{(i)} = (\mathbf{u} \cdot \mathbf{x}^{(i)})\mathbf{u} \tag{A.35}$$

where

$$\mathbf{u} = \frac{\boldsymbol{\beta}}{\|\boldsymbol{\beta}\|} \tag{A.36}$$

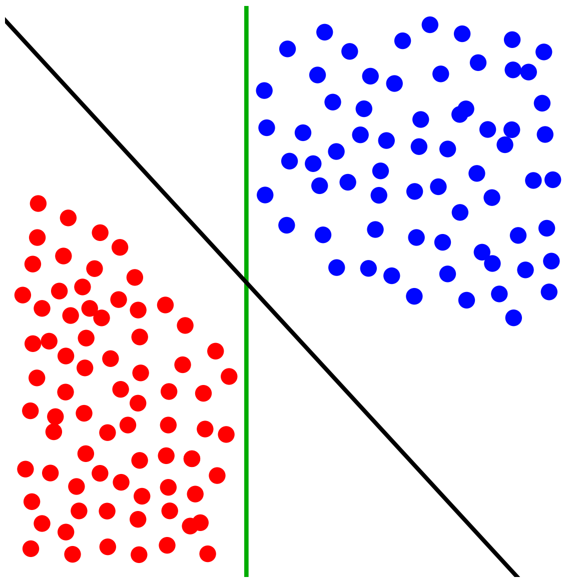


Figure A.6: The green line represents a model separating the data, but the margin is small, while black line does a far better job separating the data, because the margin is large.

The margin becomes

$$\min_{i=1}^{m} 2\|p^{(i)}\| \tag{A.37}$$

A.8.1 Cost function

Now the optimization problem is simply finding the hyperplane with the largest margin. Given a hyperplane H_0 that separates the data which satisfies $\boldsymbol{\beta}^T\mathbf{x} = 0$, We can define two additional hyper planes H_1, H_2 which also separate the data such that H_1 satisfies $\boldsymbol{\beta}^T\mathbf{x} = 1$ and H_2 satisfies $\boldsymbol{\beta}^T\mathbf{x} = -1$, where H_0 is of equal distance to both H_1 and H_2. The euclidean distance between these planes is $\frac{2}{\|\boldsymbol{\beta}\|}$. H_1 and H_2 define the margin. This means that no points should lie between these planes, so we will add the constraint:

$$\forall i \in \{i \mid 1 \leq i \leq m\} \begin{cases} \boldsymbol{\beta}^T\mathbf{x}^{(i)} \geq 1 & \text{if } y^{(i)} = 1 \\ \boldsymbol{\beta}^T\mathbf{x}^{(i)} \leq -1 & \text{if } y^{(i)} = 0 \end{cases} \tag{A.38}$$

we can rewrite this as

$$\forall i \in \{i \mid 1 \leq i \leq m\} y^{(i)}\boldsymbol{\beta}^T\mathbf{x}^{(i)} - (1 - y^{(i)})\boldsymbol{\beta}^T\mathbf{x}^{(i)} \geq 1 \tag{A.39}$$

Hence if we minimize $\|\boldsymbol{\beta}\|$ such that Equation A.39 holds, we find the hyperplane separating the data with the largest margin.

However, if the data is not linearly separable, Equation A.39 cannot be satisfied. Hence we will introduce a hinge loss function $L(\boldsymbol{\beta}, \mathbf{x}, y)$ we want to minimize for all $\{x, y\} \in \mathcal{T}$:

$$L(\boldsymbol{\beta}, \mathbf{x}, y) = \max\left(0, 1 - (y\boldsymbol{\beta}^T\mathbf{x} - (1-y)\boldsymbol{\beta}^T\mathbf{x})\right) \tag{A.40}$$

Note that $L(\boldsymbol{\beta}, \mathbf{x}, y) = 0$ if the constraint is met. Using this function we can define a cost function $J(\boldsymbol{\beta}, \mathcal{T})$ for the optimization problem. We define $J(\boldsymbol{\beta}, \mathcal{T})$ as follows:

$$J(\boldsymbol{\beta}, \mathcal{T}) = \frac{C}{m}\left(\sum_{i=1}^{m} L(\boldsymbol{\beta}, \mathbf{x}^{(i)}, y^{(i)})\right) + \frac{1}{2}\|\boldsymbol{\beta}\|^2 \tag{A.41}$$

As you can see, a regularization parameter C is introduced.[36] C determines the trade off between the constraint that all data points lie on the correct side and the size of the margin. Where large C favours the constraint Equation A.39 to be satisfied as much as possible.

A.8.2 Introducing non-linearity

The Support Vector Machine so far, is a linear classifier. However, we can also adapt SVM as a non linear classifier. This can be done with the use of a kernel function. A kernel function is a measure of similarity between to points. A commonly used kernel is the Gaussian kernel (also know as the Radial Bias Function)[2]. See Equation A.42

$$K(\mathbf{u}, \mathbf{v}) = \exp\left(\frac{\|u - v\|^2}{2\sigma^2}\right) \tag{A.42}$$

Note that this is a mapping from $[0, 1]$, where 1 means two points are equal and 0 means two points are infinitely far apart. σ determines the amount of decay in similarity. Large σ ensure items with large distance are still deemed similar.

We can use the kernel function to define new features for our SVM classifier.For each $1 \leq i \leq m, 1 \leq j \leq m$ we define a feature $f^{(j)}$ of length m such that $f_i^{(j)} = K(x^{(i)}, x^{(j)})$. Once again we will also add $f_0^{(j)} = 1$. Now we have transformed our feature vectors in $m + 1$ dimensional vectors. Which makes sure we can describe non linear (in $n + 1$ dimension) functions. naturally this will increase the training time for SVM.

A.8.3 Parameter tuning

Instead of a regularization parameter λ, SVM has a regularization parameter C. Instead of penalizing a complex model, C promotes reducing the error, and thereby satisfying the constraint. If we compare this with Linear Regression, we can state $C = \frac{1}{\lambda}$. Since λ determines the amount of regularization and C determines the margin. This means that we can use the same range as defined for λ in Appendix A.2.4, since $\frac{1}{10^3} = 10^{-3}$, and $\frac{1}{300} \approx 0.003$.

Besides the regularization parameter, another parameter σ is introduced with the Gaussian kernel. σ determines the decay for the measure of similarity. Because σ is squared in the kernel function, we can rule out negative values. Hence we will use the same tactics as used for C.

A.9 Decision Tree

A Decision Tree is based on the idea that you can make a big decision by making several small decisions. Using the training set, a tree is created where at each node a decision is made which path to follow based on a formula on the instance. The leaves of the three classify the instance.

A Decision Tree can handle two types of data, discrete values and continuous values. Discrete (or nominal) data is data where no particular order is stated, but where the number of values is limited. Continuous values on the other hand can take on an infinite number of values. In our case, because we generalize the data, we only have continuous values. Therefore we will use binary tests on the nodes of the form $t = (x_j >= value)$, where value is one of the values found in our training set for x_j. This means that we have a discrete number of tests we can conduct, because the training set has a finite number of training examples. If $t = true$ we will continue down the left branch, otherwise we will continue down the right branch. The tree is growing by recursively partitioning the data using these tests.[32] When all data in a node has the same expected value y, we can stop partitioning the data and make a leaf node, using y as the classification. If all training examples in a node have the exact same features x, we also create a leaf node and classify as the most common y.

Deciding with which test we use to start building the tree is one of the most important factors of building a Decision Tree. This because it influences how big the tree will grow. Hence we want to pick the decision that spilt the data most equally first. We will use the information gain for this. In order to define information gain we must first define entropy $H(Y)$ for a node:

$$H(Y) = -p(Y = 1)\log(p(Y = 1)) - p(Y = 0)\log(p(Y = 0)) \tag{A.43}$$

When considering a split we can calculate the $H(Y_{before})$ before the split, and we can calculate $H(Y_{after})$ after the split, which is simply

$$H(Y_{after}) = m_{left}H(Y_{left}) + m_{right}H(Y_{right}) \tag{A.44}$$

Together this forms the information gain:

$$I(Y) = H(Y_{before}) - H(Y_{after}) \tag{A.45}$$

Now we will simply calculate the information gain of all possible binary decisions defined above, and chose the one with maximum entropy. This process is repeated until the process terminates.

A.9.1 Regularization

The described process, just like most other algorithms, is prone to overfitting. To counter the overfitting problem, we will introduce the process of pruning. Pruning will decrease the size of the tree and remove unneeded nodes. The pruning method we will use is pessimistic sub-tree replacement. This because no additional validation set is required and its computationally simple. [24]

Pessimistic sub-tree replacement replaces a subtree with a leave classifying as the most common expected value, if we believe that this would increase the accuracy. The error is calculated as $E = \frac{e_{node}}{m_{node}}$, where e_{node} is the number of misclassifications, and m_{node} the number of training examples at the node. However because we only have a training set, we cannot calculate the accuracy exactly. Hence we will introduce a confidence parameter. With the confidence parameter, we calculate the confidence interval of our error, using the z-test (normal distribution). Since we are using pessimistic sub-tree replacement, we will replace the sub-tree with a node if the upper bound of the error confidence interval is higher than the weighted sum of the upper bounds of the error confidence interval of the nodes. [10] See Equation A.46.

$$\begin{aligned}&\text{Replace if}\\ &CI_u(E_s) > \sum_{i \in s} \left(\frac{m_i}{m_s} CI_u(E_n) \right)\end{aligned} \tag{A.46}$$

where CI_u is the upper bound of the confidence interval, s is a sub-tree of nodes and m_s the number of training examples in s.

A.9.2 Parameter tuning

The algorithm's only parameter to tune is the size of the confidence interval CI_Size. As mentioned before this has an influence on the amount of pruning. We define that $CI_Size = 0$ means, no pruning occurs. The maximum size of the confidence interval is 1, so our parameter tuning occurs within that range. Very small values of CI_Size do not make a lot of sense because the length of the confidence interval becomes too small. A lower bound value of 0.1 seems enough. Because this is a rather small range, instead of tripling the value each time, we can choose to (roughly) double it each step. Our parameter range becomes $[0, 0.1, 0.2, 0.5, 1]$.

A.10 Classification Rule Mining

Classification Rule Mining is a technique that derives association rules, or in our case classification rules, from frequent item sets found in the training set. That means if a subset of features $I \subseteq \mathbf{x}$ in a training set occurs often with a given classification y, we associate I with y. The classification rule $I \Rightarrow y$ is derived from this.

Because Classification Rule Mining uses frequent item sets, it assumes the data is discrete. However as explained in Section 2.5 our data might not be discrete. This means we have to discretize our data. We can do this by introducing intervals. Since all our data is in the [-1, 1] range, we can simply discretize the data using intervals. Every value is categorized in an interval and the intervals are used as the new discrete value.

Not every rule is equally reliable. This is where the confidence and support comes in. The support s_I of an item set I is the number of times it occurs in the training set. The support $s_{\{I,y\}}$ of a rule $I \Rightarrow y$ is the number of times the combination I and y occurs in the training set. Using this we can calculate the confidence of a rule $I \Rightarrow y$. The confidence is a measure for how reliable a rule is. It is calculated as follows:

$$c_{\{I,y\}} = \frac{s_{\{I,y\}}}{s_I} \tag{A.47}$$

Rules are found using the anti-monotonicity of itemsets, "if an itemset is not frequent, any of its superset is never frequent" [38]. An itemset is deemed frequent if a minimum support requirement is met. The minimum support is defined by the parameter min_s. Let C_k be the set of candidate item sets of size k and let F_k be the frequent item sets of length k that meet the requirement of minimum support. Using F_k we can generate C_{k+1} by taking the union of pairs of itemsets $P_k, Q_k \in F_k$ which have k elements in common. Then we can generate F_{k+1} from all elements $P_{k+1} \in C_{k+1}$ such that the minimum support requirement is met. To prevent overfitting of the training set we prune the set of rules afterwards using a minimum confidence requirement.

We can use the confidence of rules to classify instances with a probability $p(Y = 1 \mid x)$. Since we only have two types of classification, we can even classify instances with probability $p(Y = 1 \mid X)$ by transforming negative rules (of the form $I \Rightarrow 0$) into positive rules (of the form $I \Rightarrow 1$). Transforming negative rules into positive rules is fairly easy. Since $s_{\{I,1\}} = s_I - sI, 0$ and $c_{\{I,1\}} = 1 - c_{\{I,0\}}$.

Now that we only have positive rules, we can calculate the probability $p(Y = 1 \mid X)$ using the set of positive rules R as follows:

$$p(y \mid x) = \frac{\sum\limits_{\{I,y\} \in R : I \subseteq X} s_I \times c_{\{I,y\}}}{\sum\limits_{\{I,y\} \in R : I \subseteq X} s_I} \tag{A.48}$$

A.10.1 Parameter tuning

Classification Rule Mining has three parameters to tune. The first parameter is the interval size I_s for the discretization of the data. Since our data is in the scale $[-1, 1]$, we have that for $I_s = 0.2$, we have 10 different possible values for each feature. This also means that for $I_s = 0.01$ we have 200 possible values for each feature. This seems like an upper bound for the number of features Classification Rule Mining can handle in a usable runtime. We can use the same step size as defined in Appendix A.9.2. The range for I_s becomes $[0.01, 0.02, 0.05, 0.1, 0.2]$

The second parameter to tune is the minimum support parameter min_s. min_s determines when an itemset is frequent. min_s can have a value between [0, 1]. However, using this full range as possible values is not realistic. First of all, high values for min_s are unrealistic, because this means a high percentage of the training data should have the same (discretized) features, which is unlikely. So an upper bound of $min_c = 0.2$ is probably enough. On the other hand, too small values will ensure that we get too many rules. This means that the relevance of the rules declines, as well as that the run time increases enormously. Hence we will use a lower bound of $min_s = 0.01$. We can use the same step size as defined in Appendix A.9.2. The range for min_s becomes $[0.01, 0.02, 0.05, 0.1, 0.2]$

The third parameter to tune is the minimum confidence parameter min_c. Because we have two classes, we have that the lower bound becomes $min_c = 0.5$. This ensures we have only one rule for each itemset. On the upper bound of min_c, we have that $min_c = 1$ might be a bit unrealistic. Since this means that the rules are only taken into account if they always predict the same value. Hence we will pick an upper bound of $min_c = 0.9$. For this parameter we will use a regular interval with step size 0.1.

Appendix B

Results

B.1 How accurate can machine learning algorithms predict insurance product interest for different data-sets?

B.1.1 Should we optimize for correct classifications or for good probabilities to get the best accuracy of predictions?

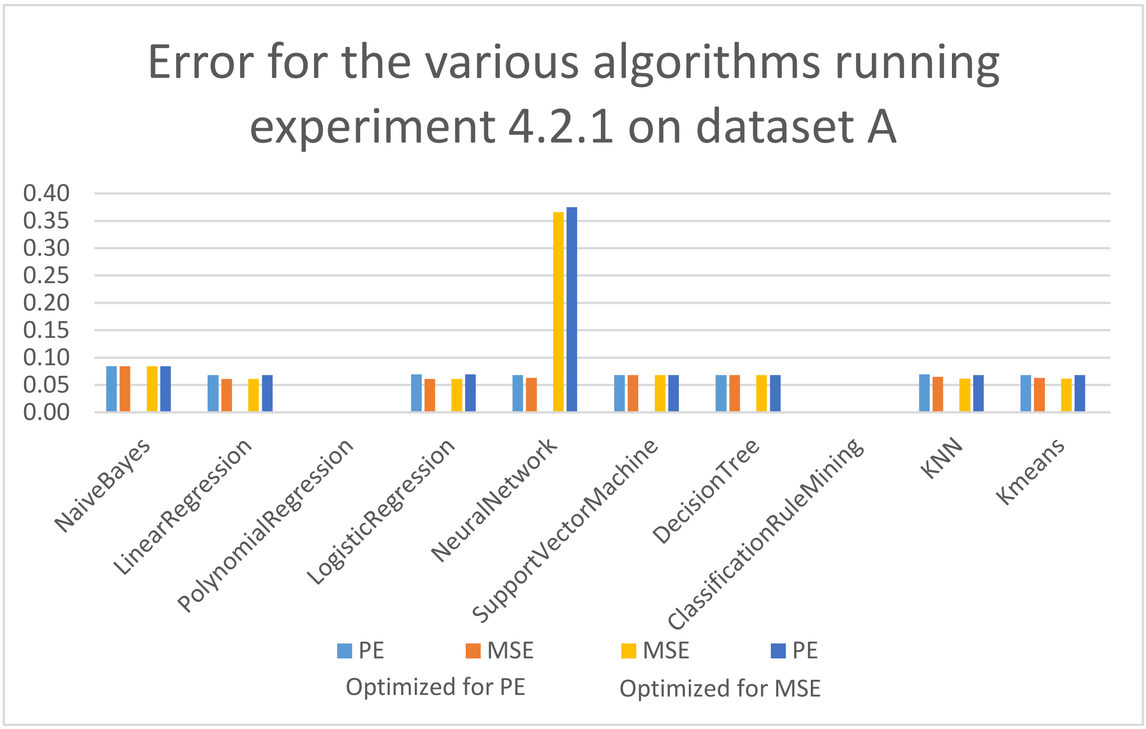


Figure B.1: The results of Experiment 4.2.1 for dataset A.

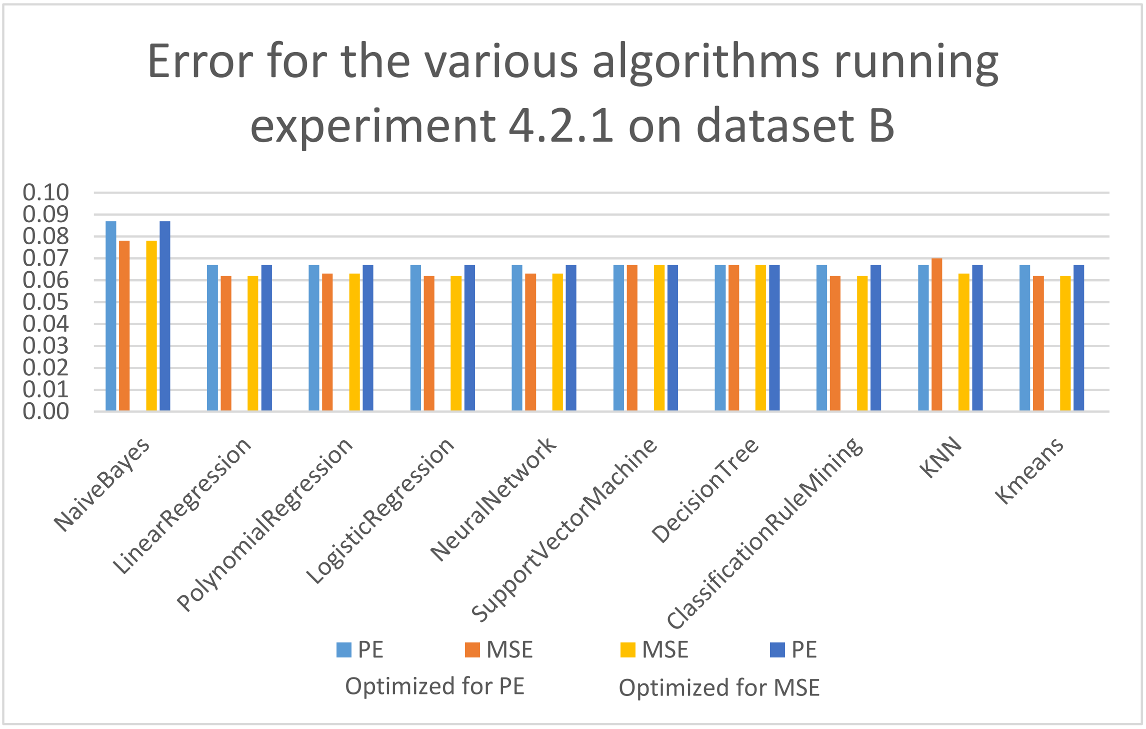


Figure B.2: The results of Experiment 4.2.1 for dataset B.

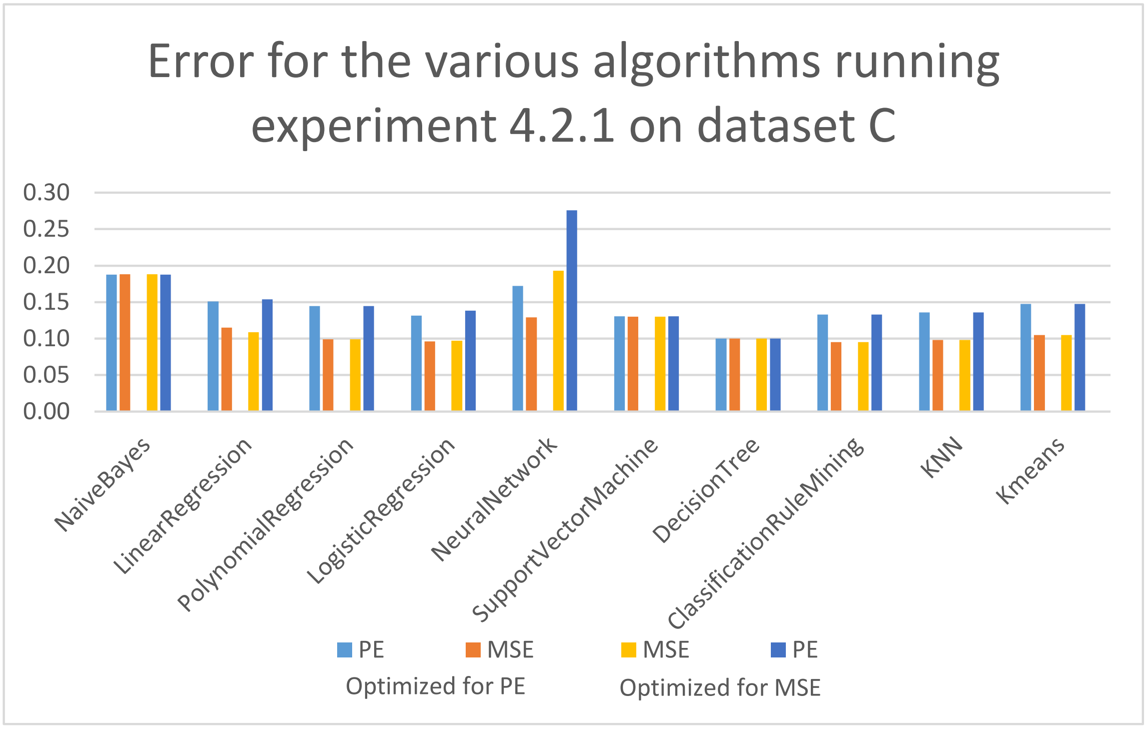


Figure B.3: The results of Experiment 4.2.1 for dataset C.

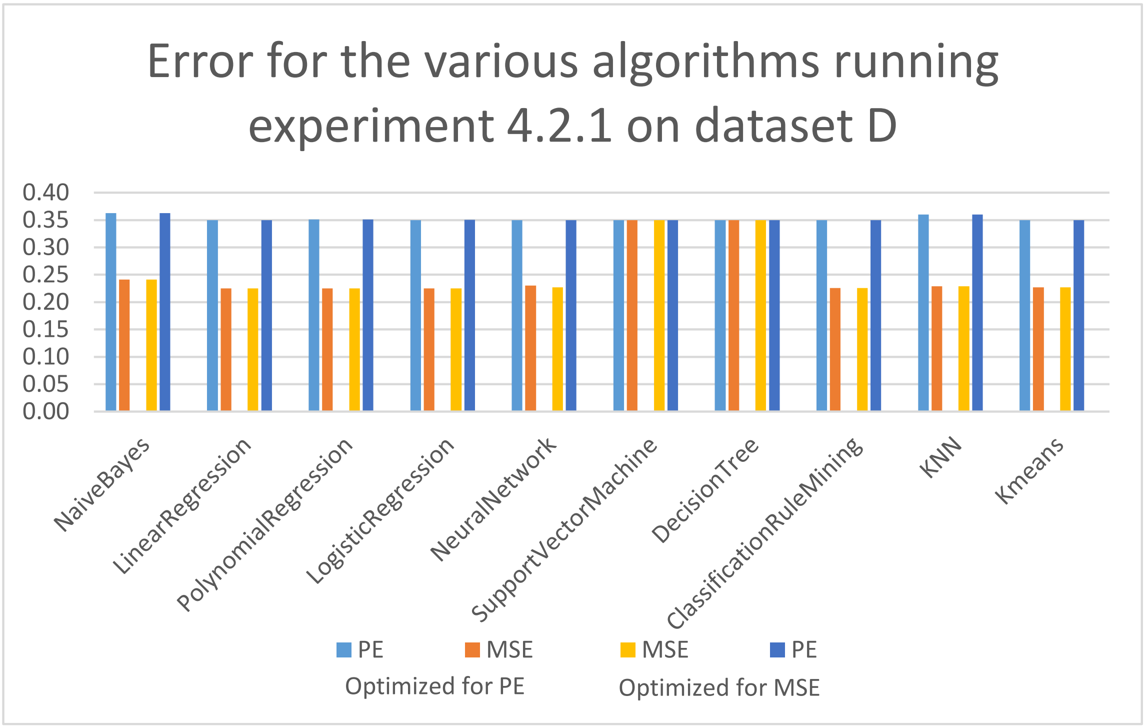


Figure B.4: The results of Experiment 4.2.1 for dataset D.

B.1.2 How does feature scaling influence the algorithms and their prediction accuracy?

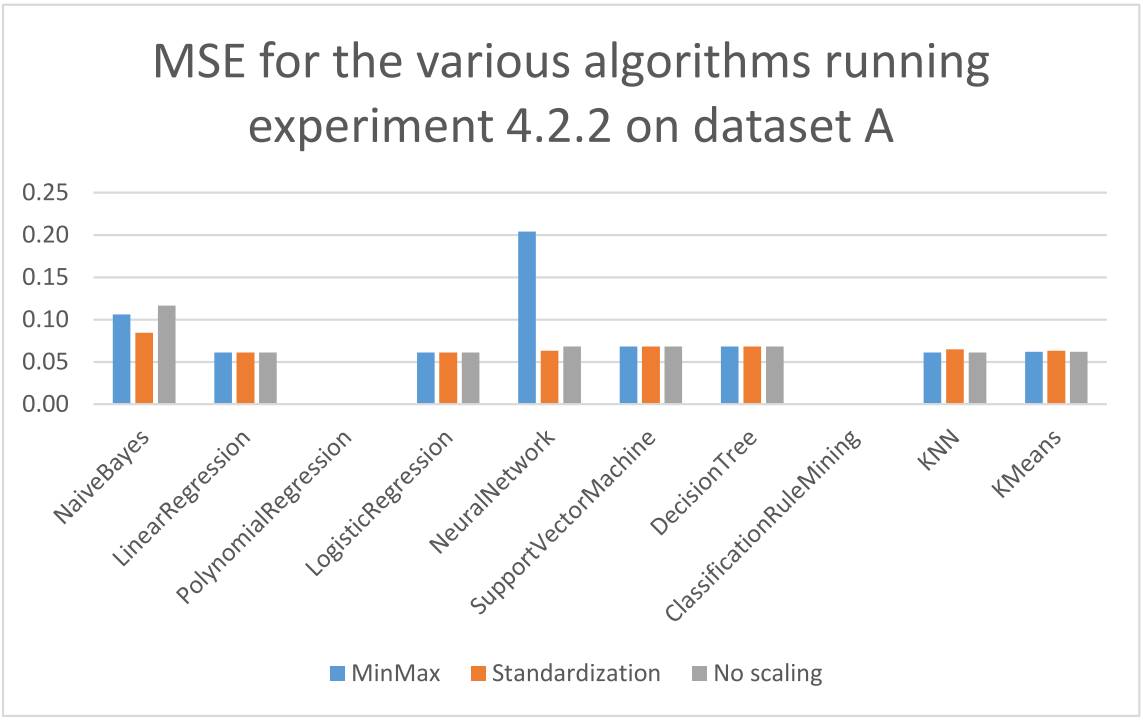


Figure B.5: The results of Experiment 4.2.2 for dataset A.

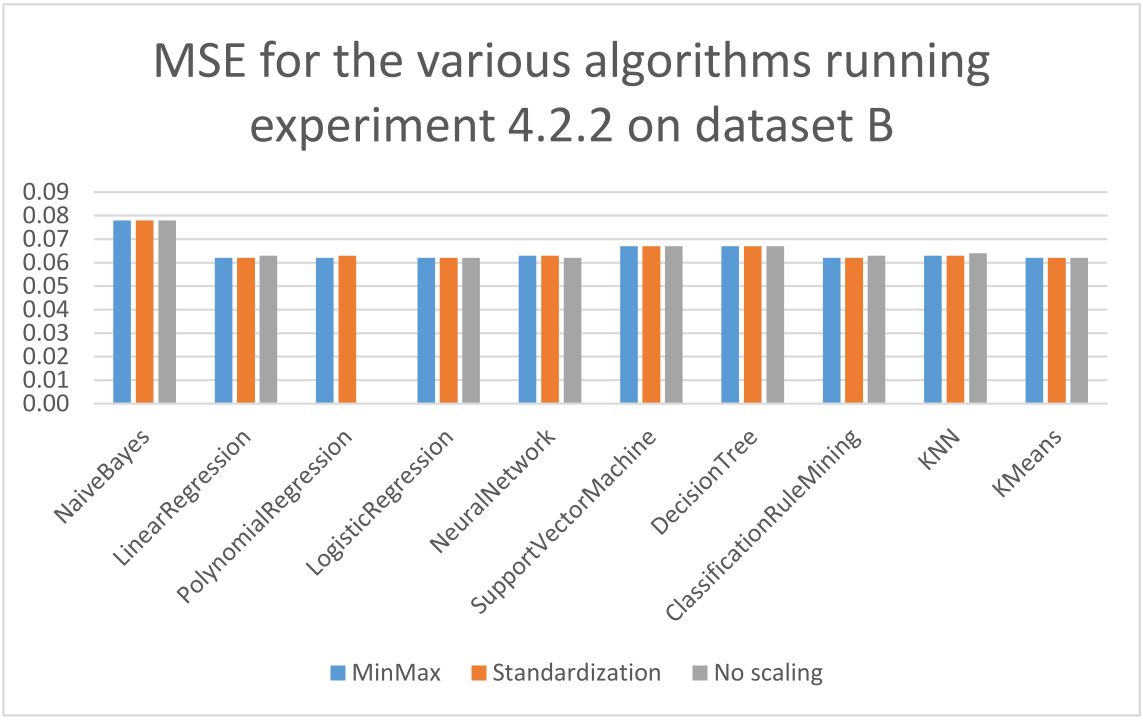


Figure B.6: The results of Experiment 4.2.2 for dataset B.

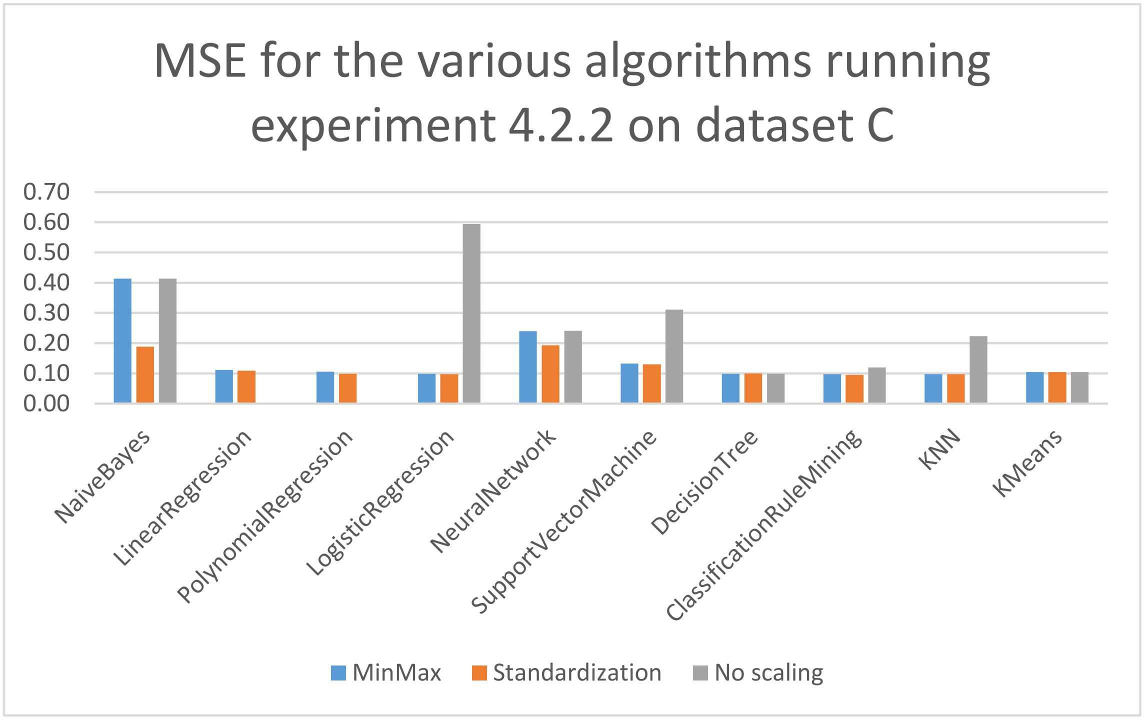


Figure B.7: The results of Experiment 4.2.2 for dataset C.

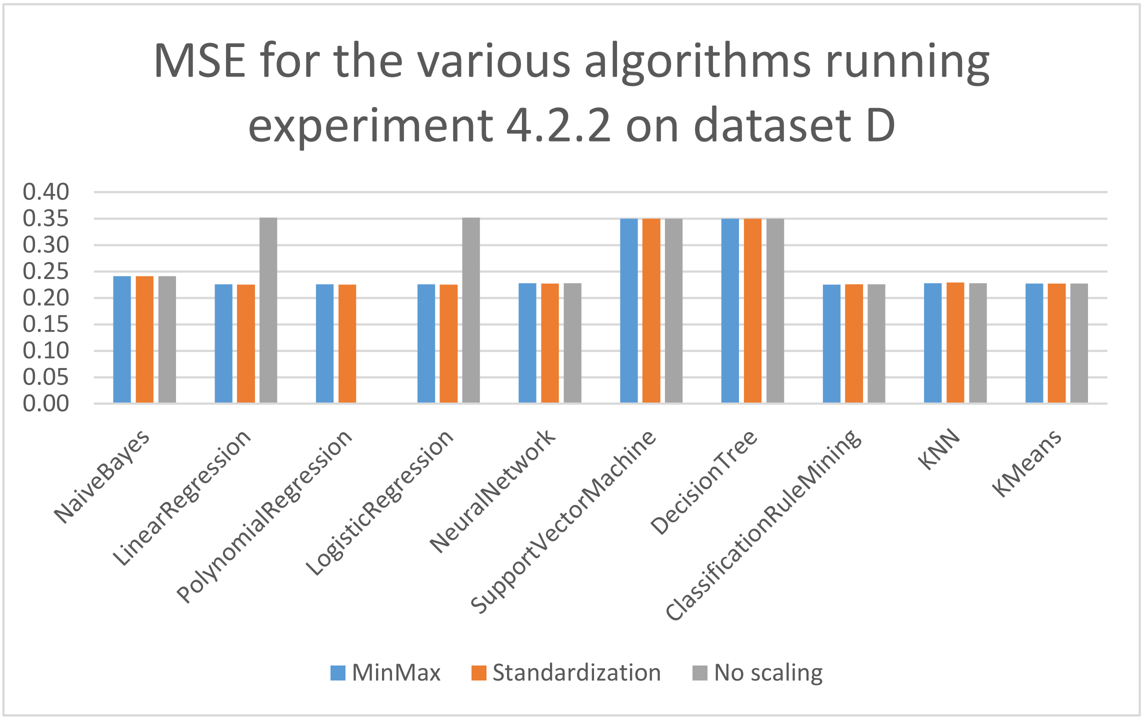


Figure B.8: The results of Experiment 4.2.2 for dataset D. Note that logistic regression is ran without $\circ = 2$

B.1.3 Which algorithms have the highest potential for accurate predictions?

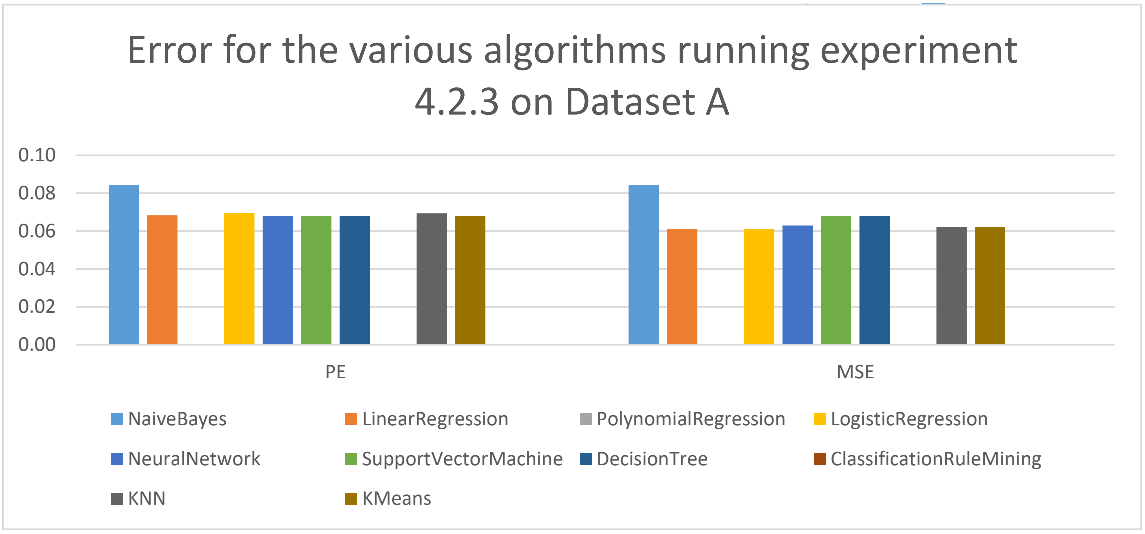


Figure B.9: The results of Experiment 4.2.3 for dataset A.

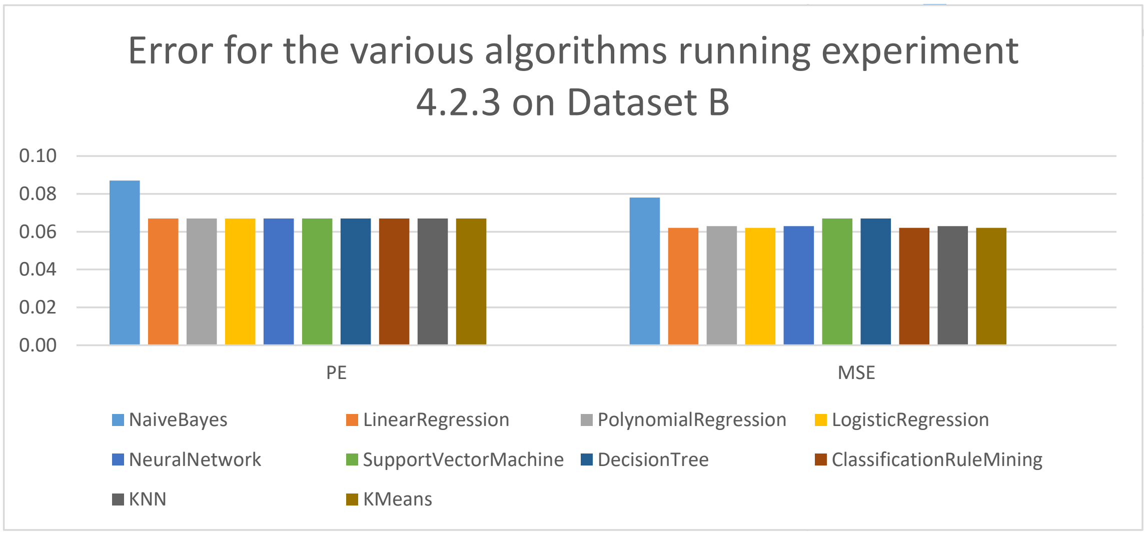


Figure B.10: The results of Experiment 4.2.3 for dataset B.

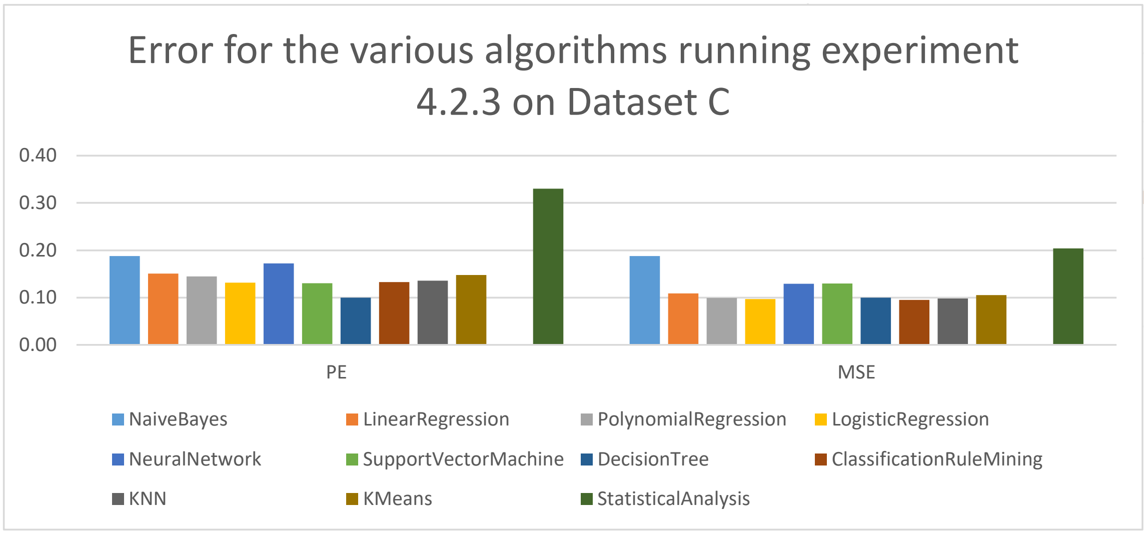


Figure B.11: The results of Experiment 4.2.3 for dataset C.

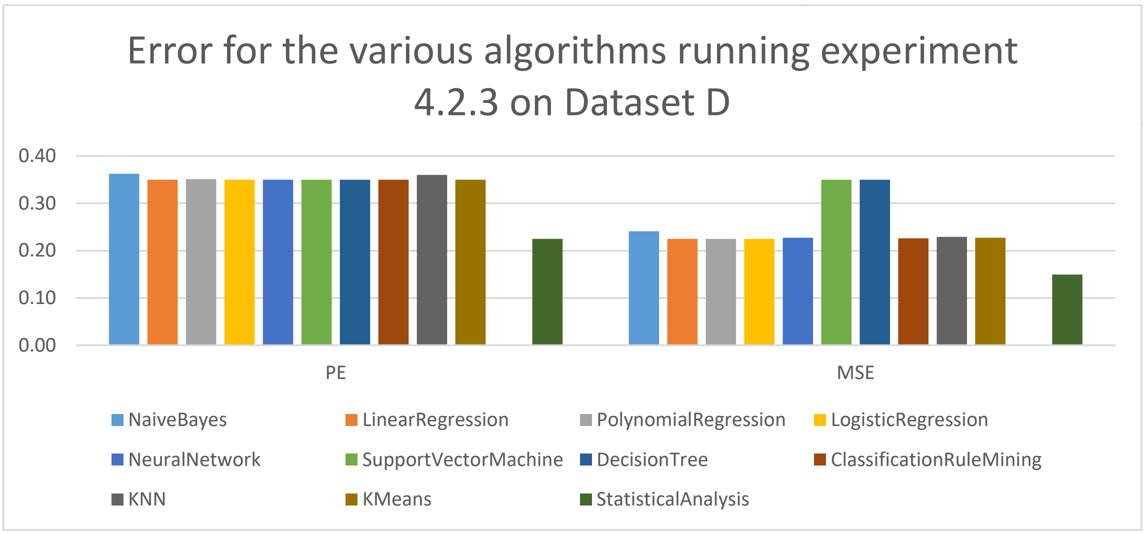


Figure B.12: The results of Experiment 4.2.3 for dataset D.

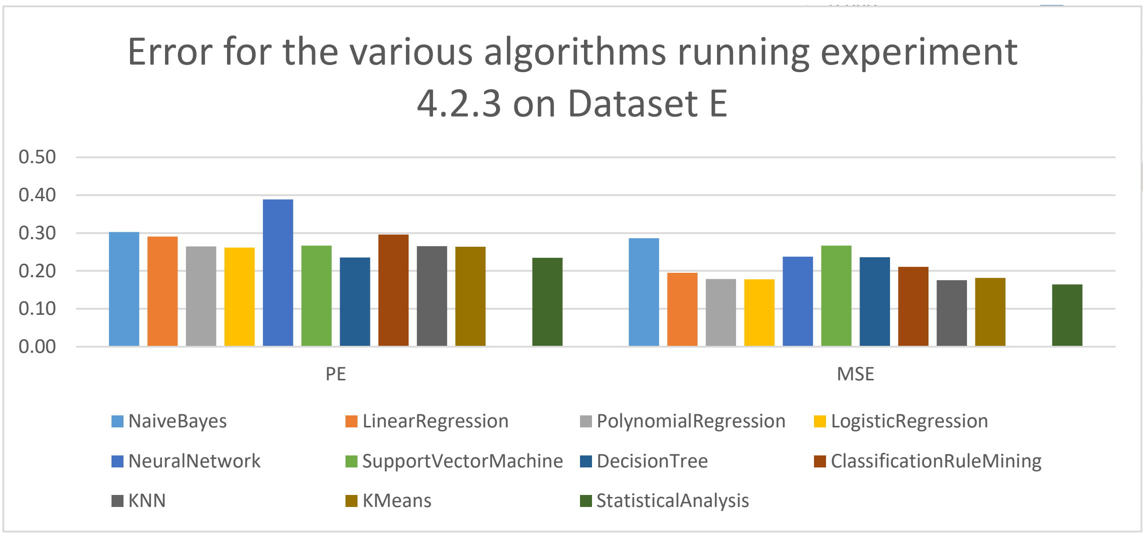


Figure B.13: The results of Experiment 4.2.3 for dataset E.

B.2 How fast can machine learning algorithms predict insurance product interest for different datasets?

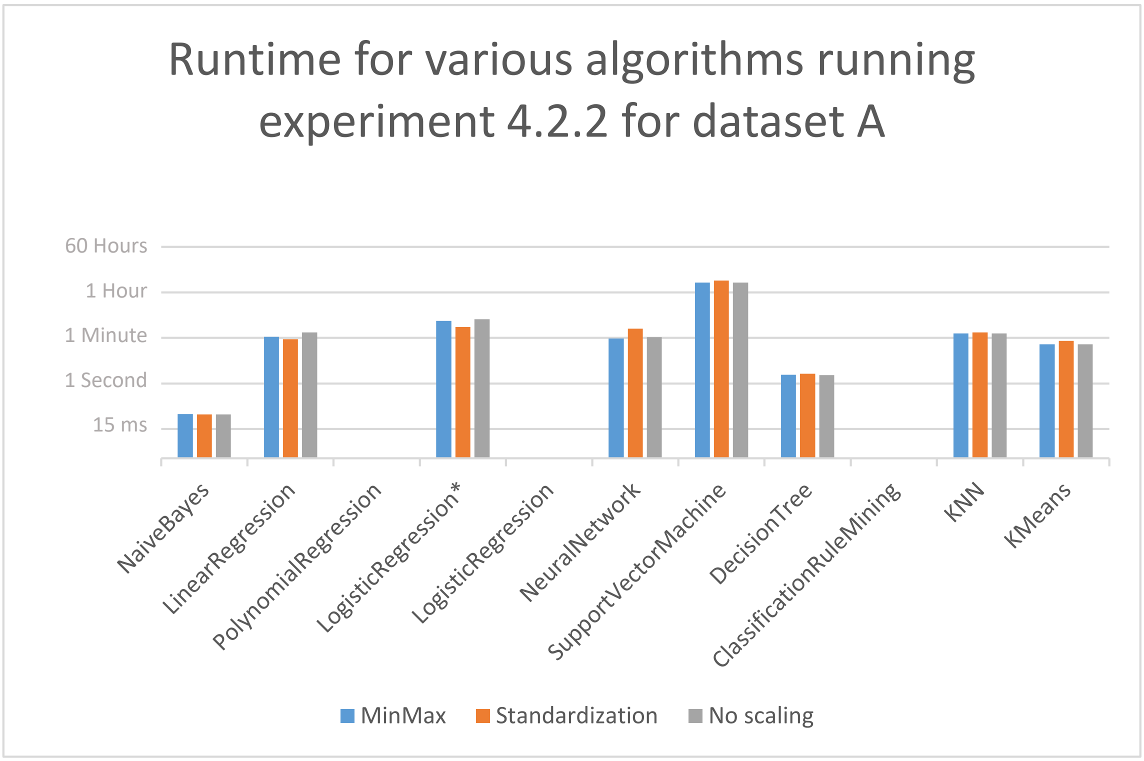


Figure B.14: The results of Experiment 4.2.2 for dataset A.

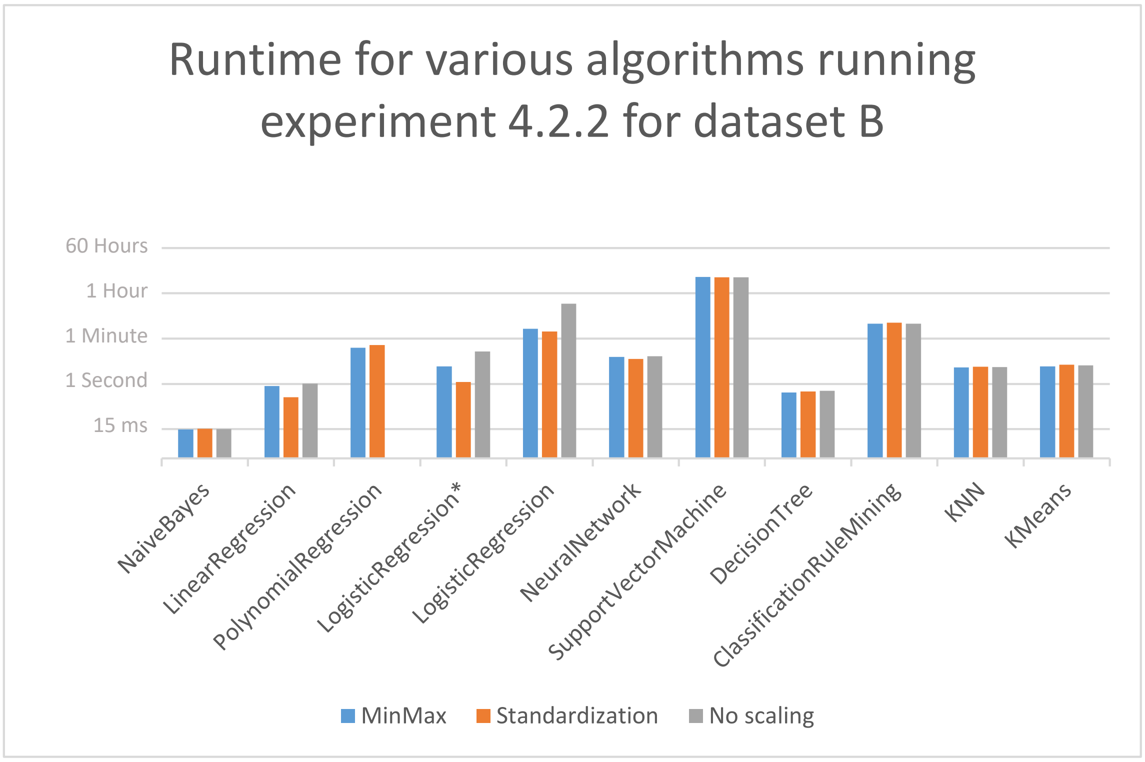


Figure B.15: The results of Experiment 4.2.2 for dataset B.

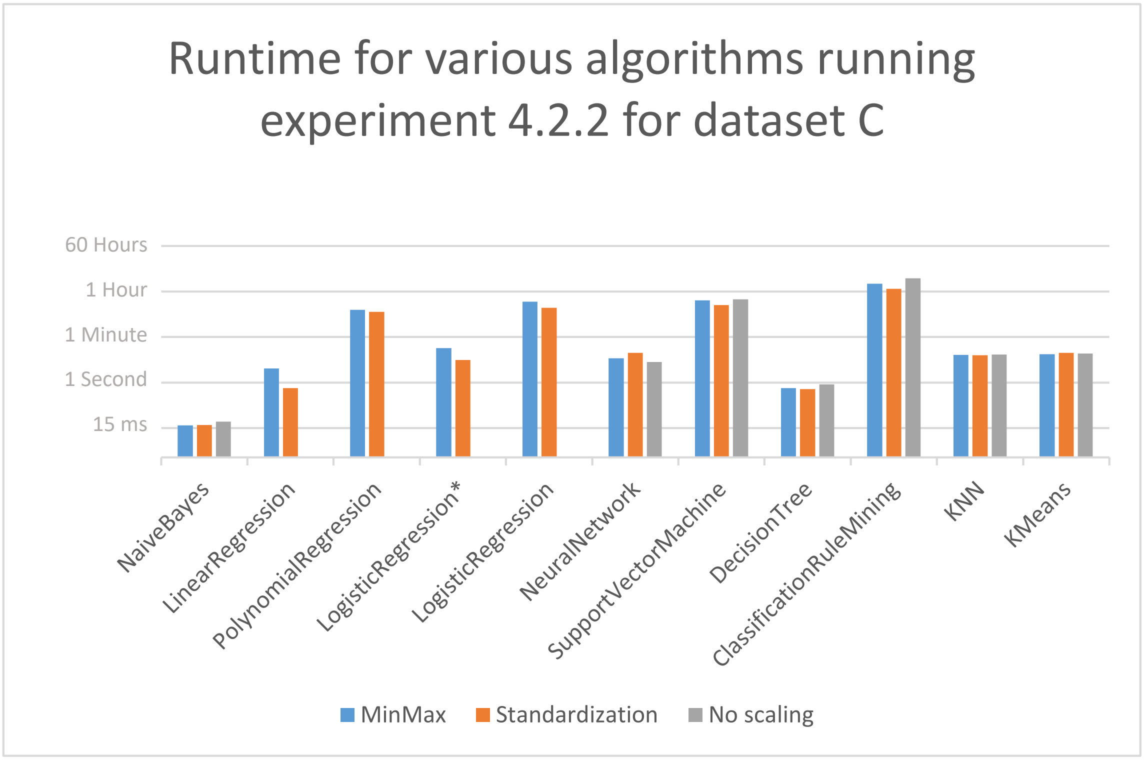


Figure B.16: The results of Experiment 4.2.2 for dataset C.

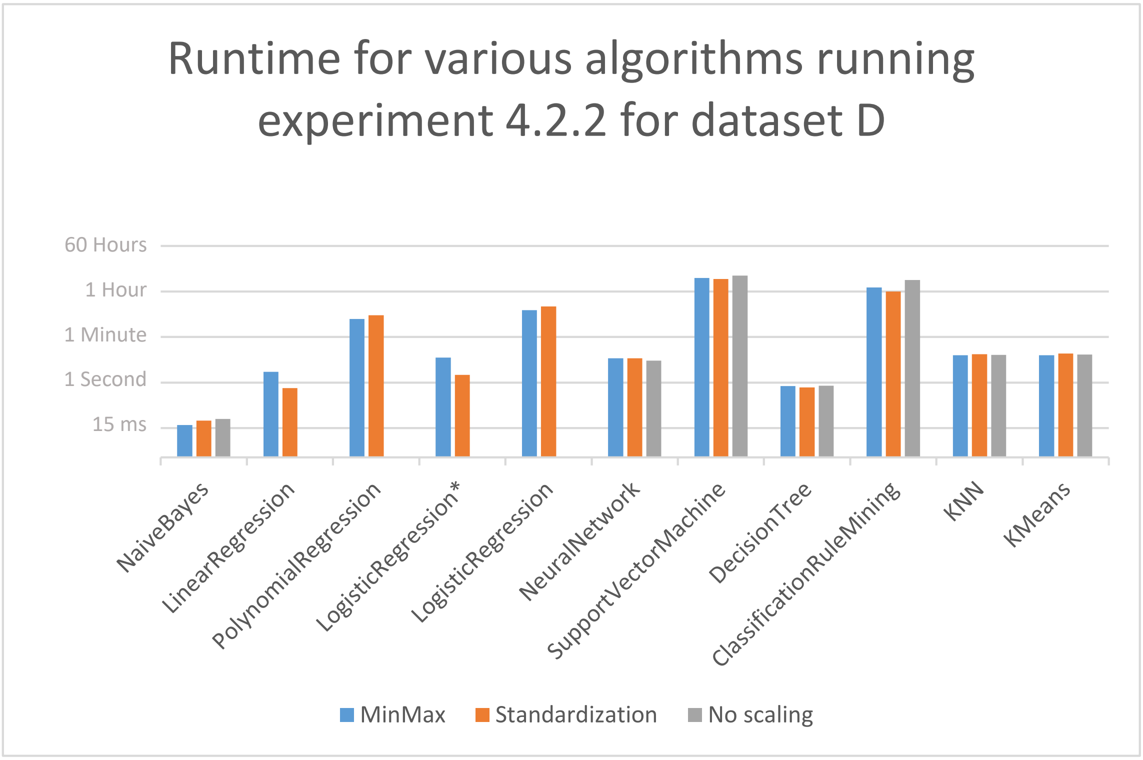


Figure B.17: The results of Experiment 4.2.2 for dataset D. Note that logistic regression is ran without $\circ = 2$

B.2.1 How much do various parameters influence the prediction accuracy of the algorithms? Can we eliminate the tuning phase?

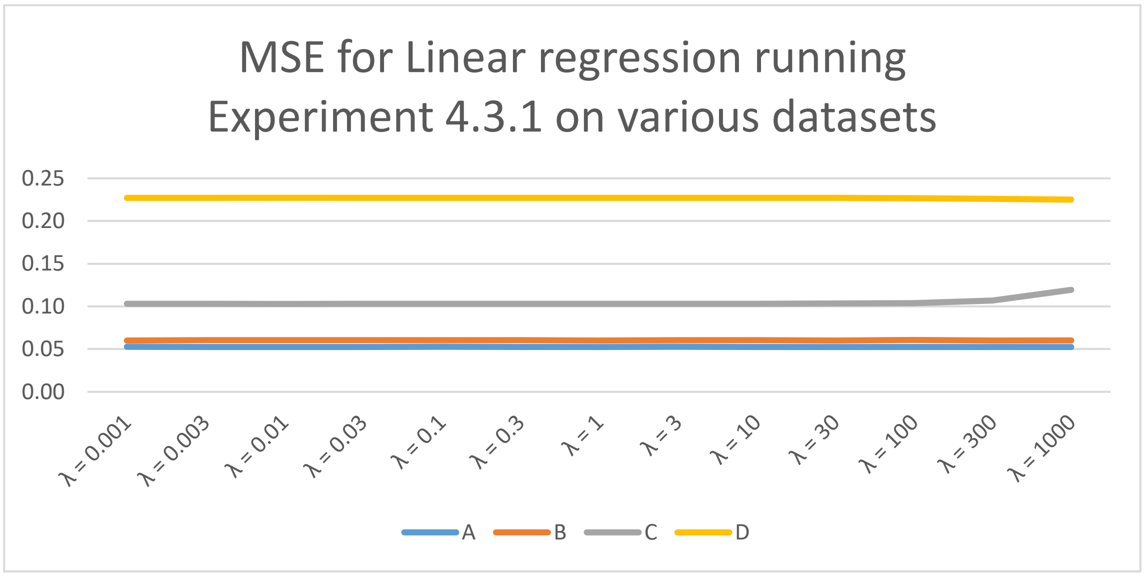


Figure B.18: The results of Experiment 4.3.1 for Linear regression.

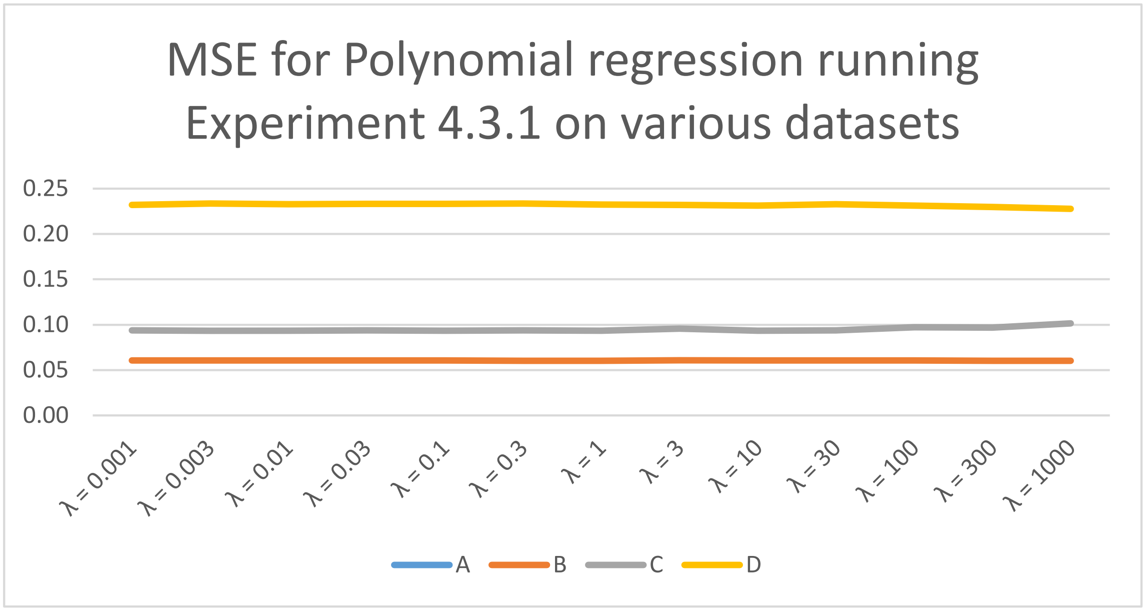


Figure B.19: The results of Experiment 4.3.1 for Polynomial regression.

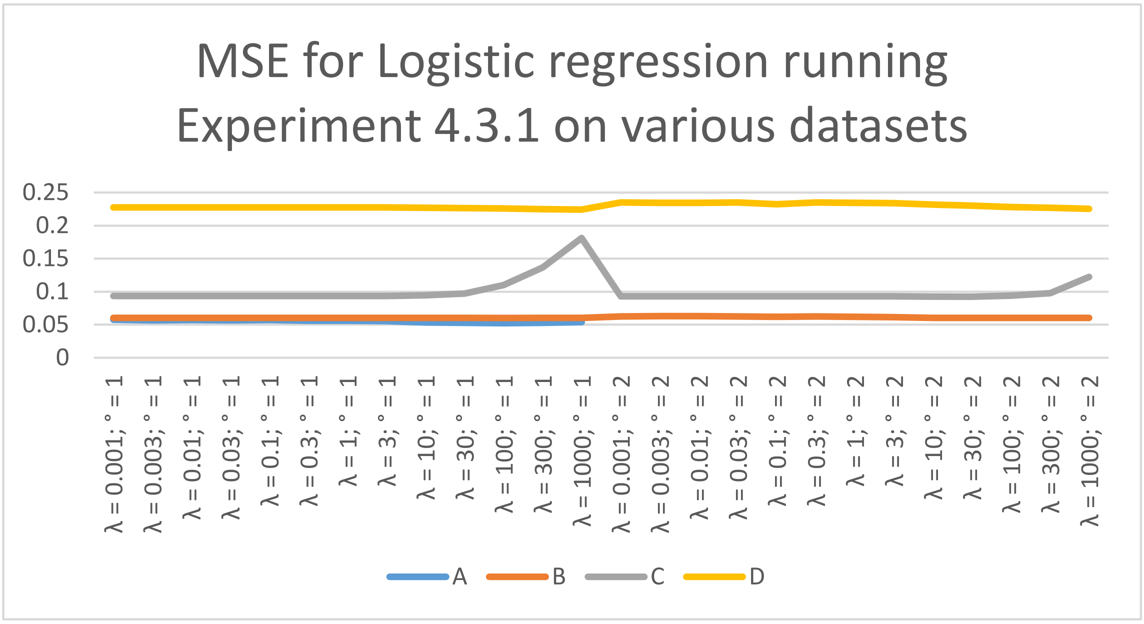


Figure B.20: The results of Experiment 4.3.1 for Logistic regression.

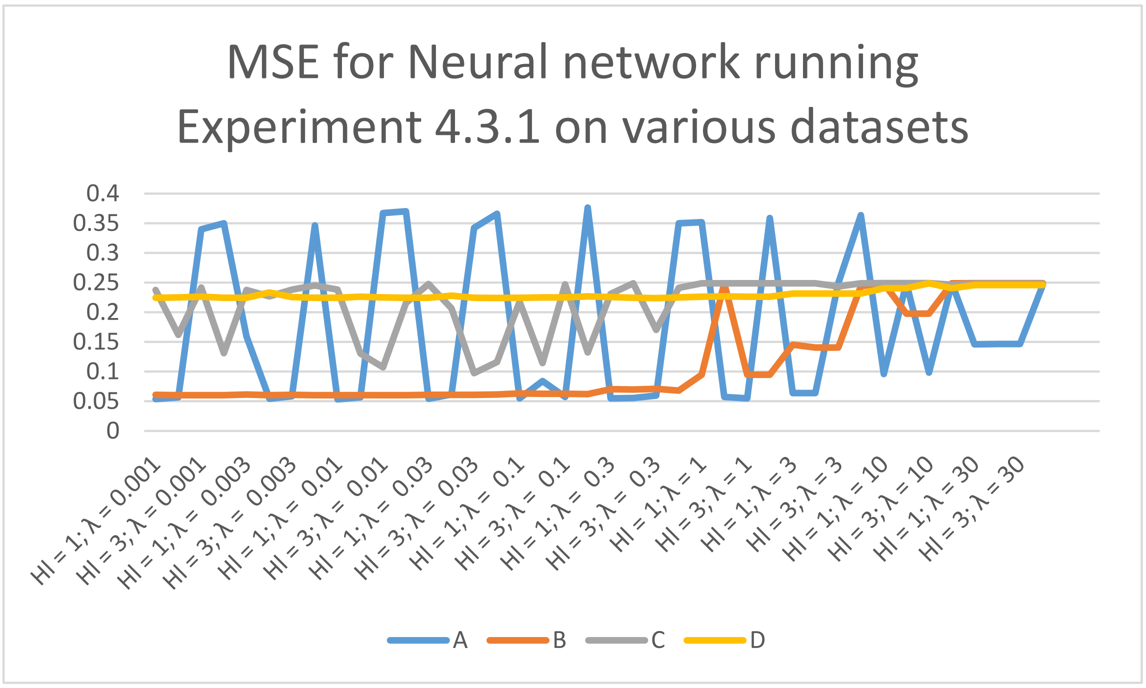


Figure B.21: The results of Experiment 4.3.1 for Neural network.

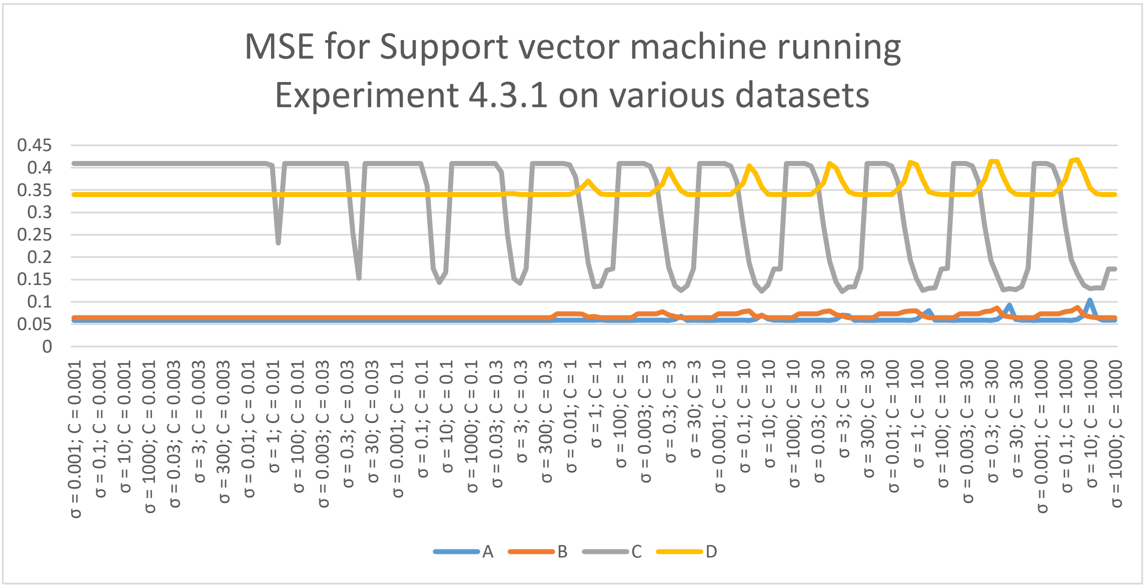


Figure B.22: The results of Experiment 4.3.1 for Support vector machine.

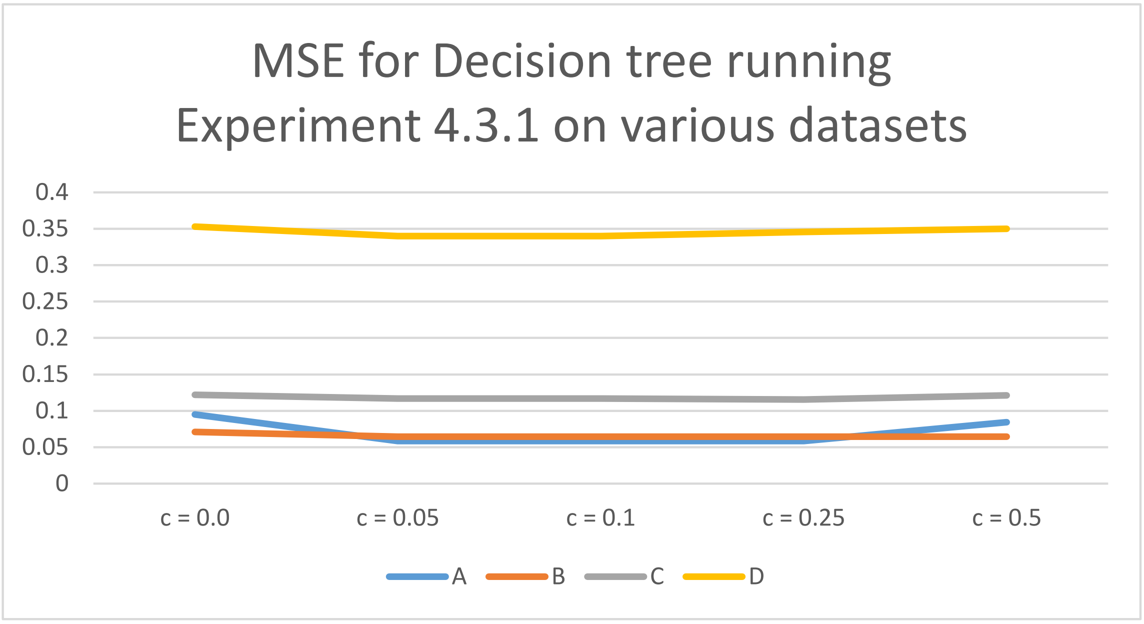


Figure B.23: The results of Experiment 4.3.1 for Decision tree.

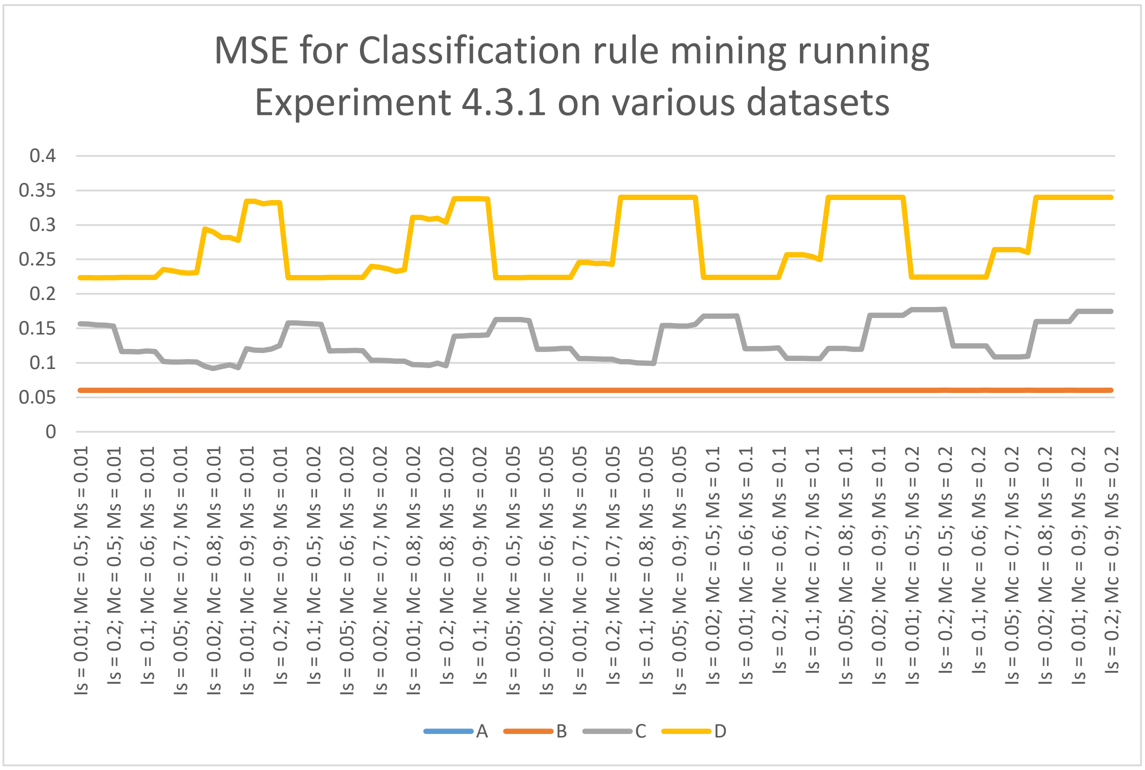


Figure B.24: The results of Experiment 4.3.1 for Association rule mining.

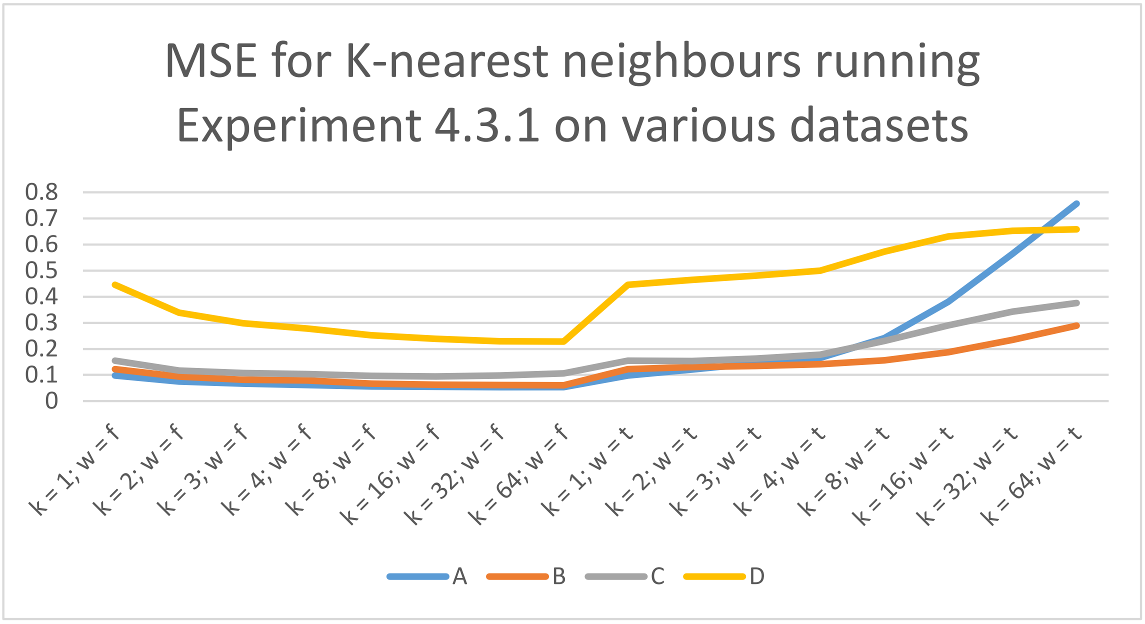


Figure B.25: The results of Experiment 4.3.1 for K-nearest neighbours.

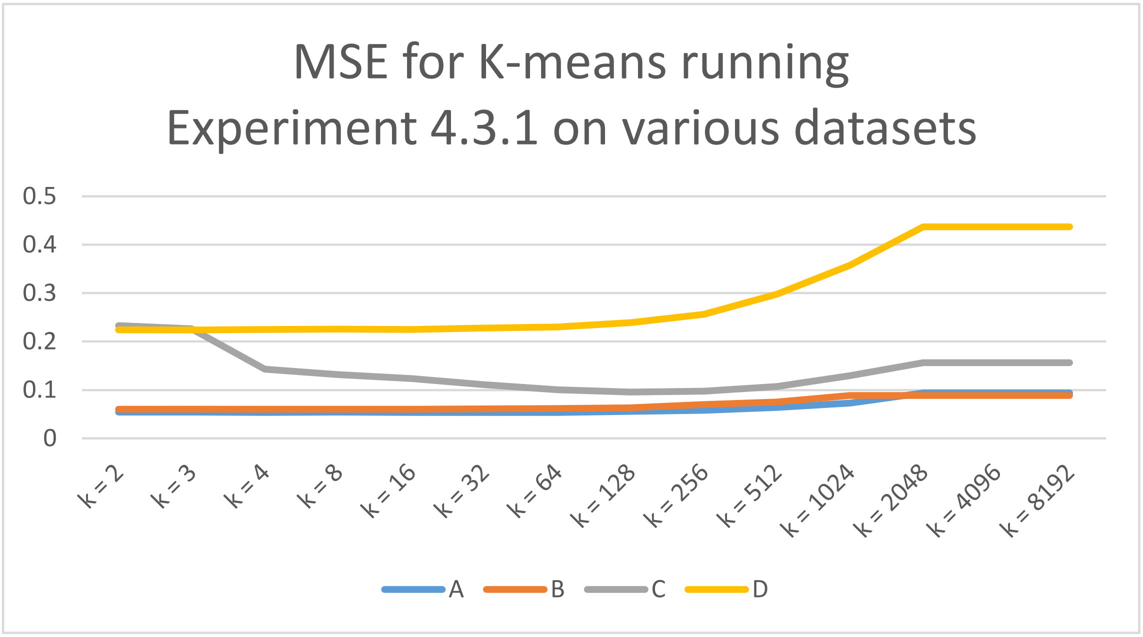


Figure B.26: The results of Experiment 4.3.1 for K-means.

B.2.2 How well do algorithms scale to large data sets?

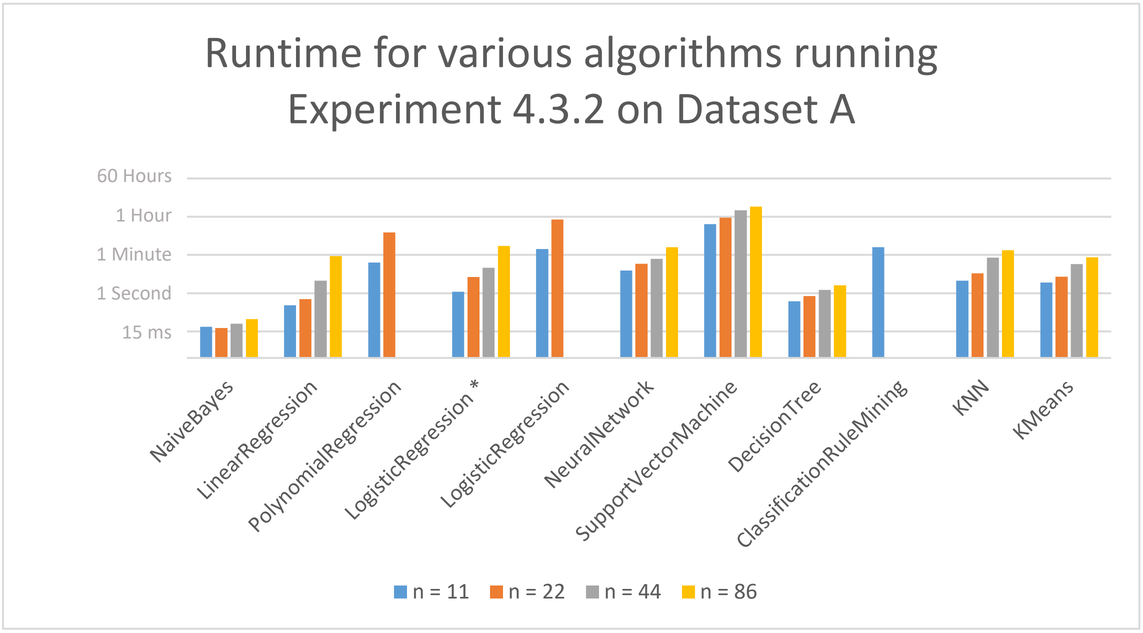


Figure B.27: The results of Experiment 4.3.2 for variations of dataset A.

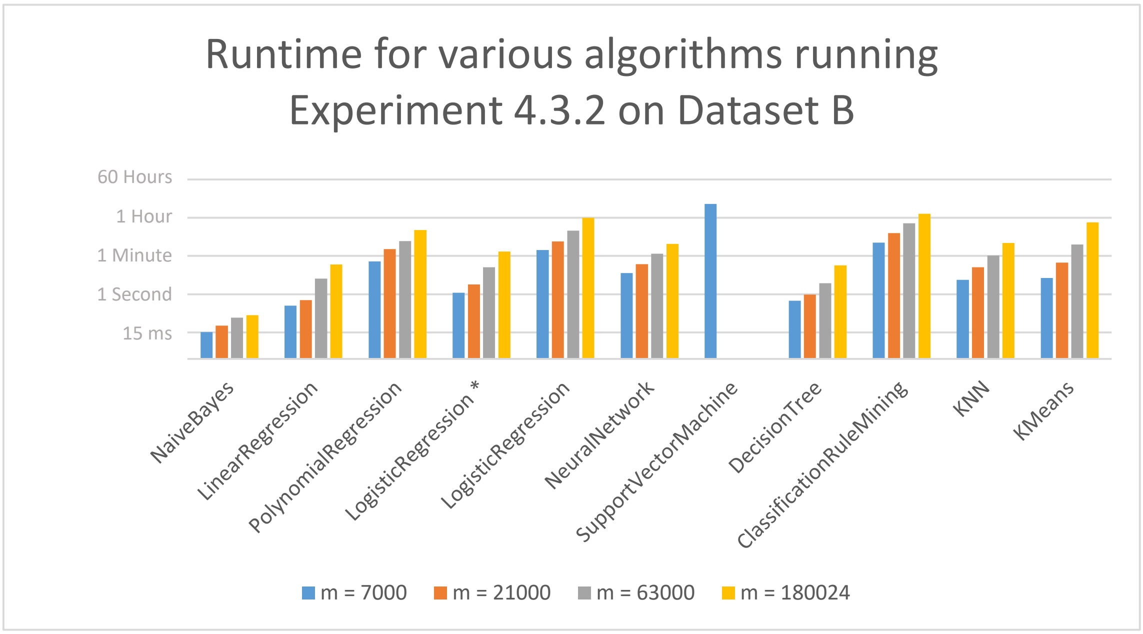


Figure B.28: The results of Experiment 4.3.2 for variations of dataset B.

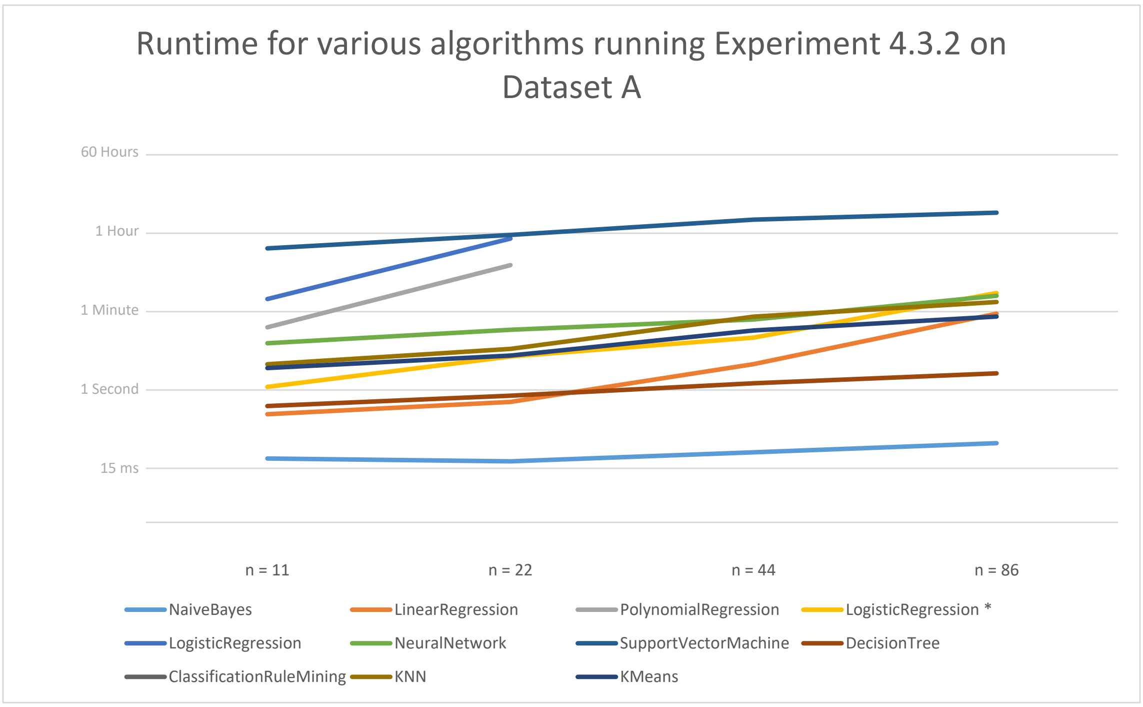


Figure B.29: The results of Experiment 4.3.2 for variations of dataset A.

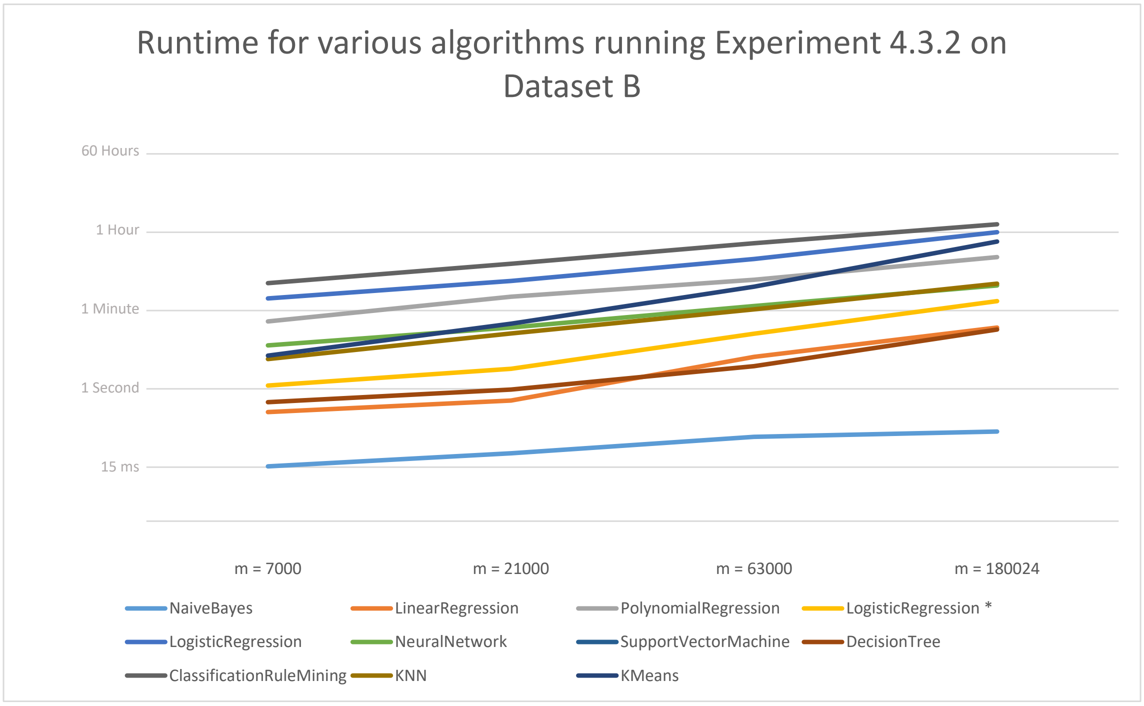


Figure B.30: The results of Experiment 4.3.2 for variations of dataset B.

B.3 How well can machine learning algorithms deal with non information?

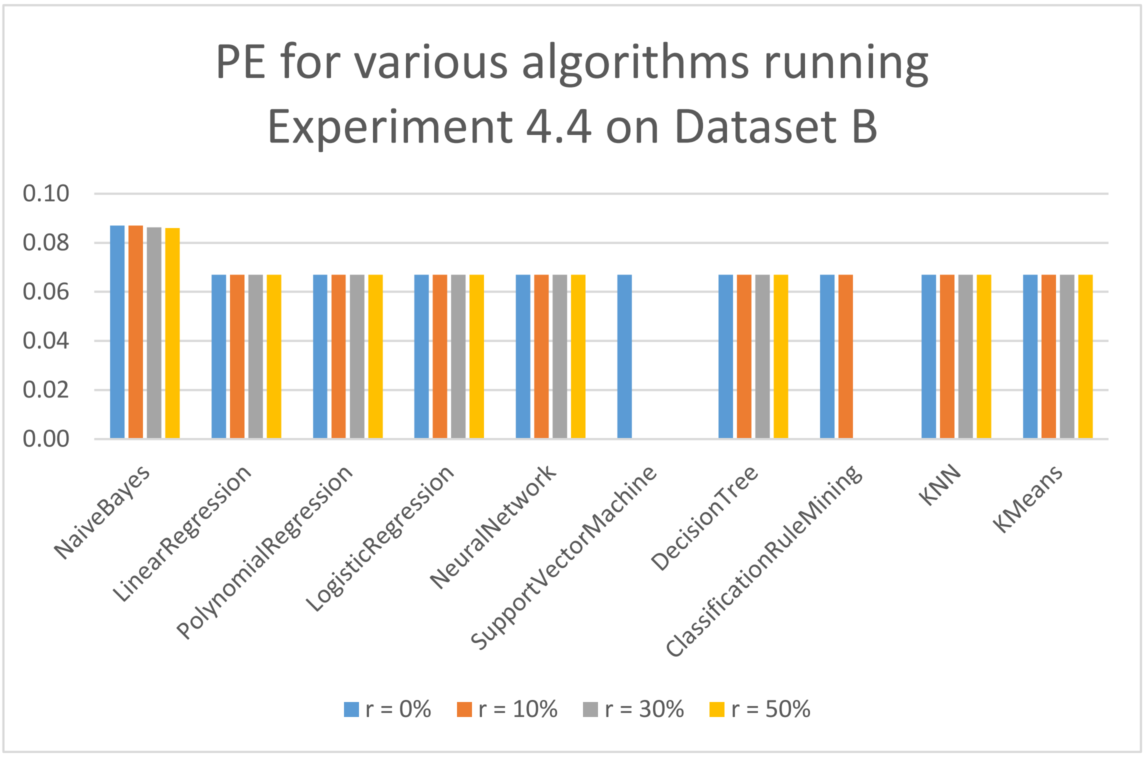


Figure B.31: The results of Experiment 4.4 for dataset A and error measurement PE.

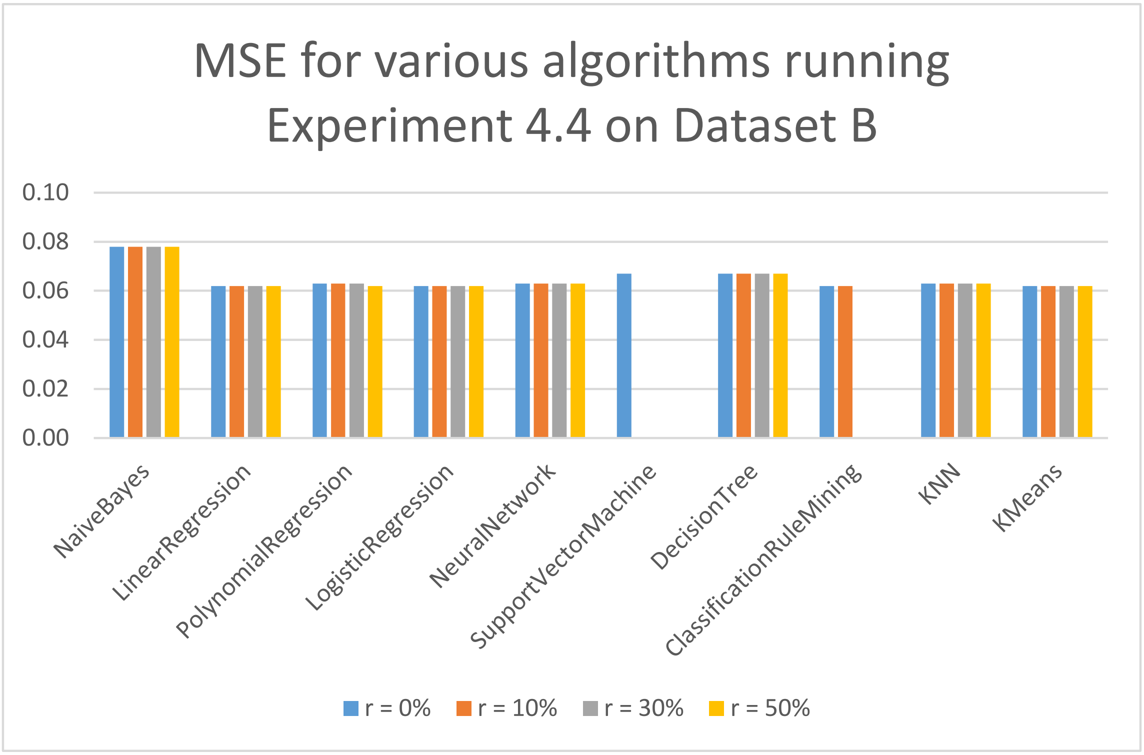


Figure B.32: The results of Experiment 4.4 for dataset A and error measurement MSE.