Royal Institute of Technology
Department of Mathematics

Multi-class Supervised Classification Techniques for High-dimensional Data: Applications to Vehicle Maintenance at Scania

SF290X Degree Project in Mathematical Statistics, Spring 2017

Berlin, Daniel
dberl@kth.se

Supervisor:

Svensson, Jan
jan.svensson@scania.com

Pavlenko, Tatjana
pavlenko@math.kth.se

Examiner:

Pavlenko, Tatjana
pavlenko@math.kth.se

June 7, 2017

Abstract

In vehicle reparations, many times locating the cause of error could turn out more time consuming than the reparation itself. Hence a systematic way to accurately predict a fault causing part would constitute a valuable tool especially for errors difficult to diagnose. This thesis explores the predictive ability of Diagnostic Trouble Codes (DTC's), produced by the electronic system on Scania vehicles, as indicators for fault causing parts. The statistical analysis is based on about 18800 observations of vehicles where both DTC's and replaced parts could be identified during the period march 2016 - march 2017. Two different approaches of forming classes is evaluated. Many classes had only few observations and, to give the classifiers a fair chance, it is decided to omit observations of classes based on their frequency in data. After processing, the resulting data could comprise 1547 observations on 4168 features, demonstrating very high dimensionality and making it impossible to apply standard methods of large-sample statistical inference. Two procedures of supervised statistical learning, that are able to cope with high dimensionality and multiple classes, Support Vector Machines and Neural Networks are exploited and evaluated. The analysis showed that on data with 1547 observations of 4168 features (unique DTC's) and 7 classes SVM yielded an average prediction accuracy of 79.4% compared to 75.4% using NN. The conclusion of the analysis is that DTC's holds potential to be used as indicators for fault causing pars in a predictive model, but in order to increase prediction accuracy learning data needs improvements. Scope for future research to improve and expand the model, along with practical suggestions for exploiting supervised classifiers at Scania is provided.

keywords: Statistical learning, Machine learning, Neural networks, Deep learning, Supervised learning, High dimensionality

Sammanfattning

Övervakade Klassificerings Modeller för Högdimensionell Data och Multipla Klasser: Tillämplningar inom Fordonsunderhåll på Scania

Många gånger i samband med fordonsreparationer är felsökningen mer tidskrävande än själva reparationen. Således skulle en systematisk metod för att noggrant prediktera felkällan vara ett värdefullt verktyg för att diagnostisera reparationsåtgärder. I denna uppsats undersöks möjligheten att använda *Diagnostic Trouble Codes* (DTC:er), som genereras av de elektroniska systemen i Scanias fordon, som indikatorer för att peka ut felorsaken. Till grund för analysen användes ca 18800 observationer av fordon där både DTC:er samt utbytta delar kunnat identifieras under perioden mars 2016 - mars 2017. Två olika strategier för att generera klasser har utvärderats. Till många av klasserna fanns det endast ett fåtal observationer, och för att ge de prediktiva modellerna bra förutsättningar så användes endast klasser med *tillräckligt* många observationer i träningsdata. Efter bearbetning kunde data innehålla 1547 observationer 4168 attribut, vilket demonstrerar problemets höga dimensionalitet och gör det omöjligt att applicera standard metoder för statistisk analys på stora datamängder. Två metoder för övervakad statistisk inlärning, lämpliga för högdimensionell data med multipla klasser, Södvectormaskiner (SVM) samt Neurala Nätverk (NN) implementeras och deras resultat utvärderas. Analysen visade att på data med 1547 observationer av 4168 attribut (unika DTC:er) och 7 klasser kunde SVM prediktera observationer till klasserna med 79.4% noggrannhet jämfört med 75.4% för NN. De slutsatser som kunde dras av analysen var att DTC:er tycks ha potential att användas för att indikera felorsaker med en prediktiv modell, men att den data som ligger till grund för analysen bör förbättras för att öka noggrannheten i de prediktiva modellerna. Framtida forskningsmöjligheter för att ytterligare förbättra samt utveckla modellen, tillsammans med förslag för hur övervakade klassificerings modeller kan användas på Scnaia har identifierats.

Acknowledgements

I would like to thank my supervisor Tatjana Pavlenko, associate professor at the department of mathematics at KTH, Royal Institute of Technology for her academic expertise and encouragements during this thesis. I would also like to thank Scania CV AB for providing me with the opportunity and data which made the thesis project possible. Above all I would like to express my sincerest gratitude to my supervisor at Scania, Jan Svensson, for many rewarding discussions and encouragements throughout the extensive work that has lead to this thesis. Without your insights and technical skills the results would have suffered. I would also like to thank Erik Påledal at Scania for your expertise in Splunk which facilitated the generation of data. Special thanks to my peers at KTH which have engaged in discussions and motivated me, not only during this thesis but for five intense years of studies. Lastly I would like to send my gratitude to my partner Ida for all your support and everything you have done to help out which has provided me the opportunity to focus on my studies.

Stockholm, June 2017
Daniel Berlin

Abbreviations

DTC	Diagnostic Trouble Code
ECU	Electronic Control Unit
SDP3	Computer program used to troubleshoot a vehicles electronic systems
WHI	Workshop History - database containing information on workshop history
SVM	Support Vector Machine
NN	Neural Network
SQL	Structured Query Language
PCA	Principle Component Analysis
SSE	Sums of Square Error
CE	Cross Entropy

Contents

1 Introduction **1**
- 1.1 Problem description . . . 1
- 1.2 Objectives . . . 2
- 1.3 Outline . . . 2

2 Data preprocessing **3**
- 2.1 The ideal dataset . . . 3
- 2.2 Source of data . . . 3
  - 2.2.1 SDP3 log file . . . 4
  - 2.2.2 WHI database . . . 4
- 2.3 Finding matching events . . . 5
- 2.4 Generating classes . . . 5
- 2.5 Concerns . . . 7
  - 2.5.1 Connection . . . 8
  - 2.5.2 Maintenance . . . 8
  - 2.5.3 Noise . . . 9
- 2.6 Analyzed data . . . 9
  - 2.6.1 Initial strategy . . . 9
  - 2.6.2 Improved strategy . . . 10
  - 2.6.3 Reducing dimensionality . . . 12

3 Theoretical background **15**
- 3.1 Statistical learning . . . 15
  - 3.1.1 What constitutes a learning problem? . . . 16
  - 3.1.2 Supervised vs Unsupervised learning . . . 17
- 3.2 Linear separability . . . 17
- 3.3 Searating hyperplanes . . . 17
- 3.4 Perceptron algorithm . . . 18
- 3.5 Support Vector Machines . . . 19
  - 3.5.1 Maximal margin classifier . . . 19
  - 3.5.2 Support vector classifier . . . 21
  - 3.5.3 Support vector machine . . . 24
  - 3.5.4 Multiclass Support Vector Machines . . . 26
- 3.6 Neural Network and Deep learning . . . 26
  - 3.6.1 The McCulloch-Pitts neuron . . . 27
  - 3.6.2 Perceptron revised . . . 28
  - 3.6.3 Neural networks . . . 29
  - 3.6.4 Fitting Neural networks . . . 32

4 Method and Implementation **37**
4.1 Performance Evaluation . . . 37
4.1.1 Empirical error . . . 37
4.1.2 Confusion Matrix . . . 37
4.1.3 Random Guess . . . 38
4.1.4 Naive Classifier . . . 39
4.2 Implementation . . . 40
4.2.1 Neural Networks . . . 40
4.2.2 Support Vector Machines . . . 40

5 Results **41**
5.1 Parameter evaluation . . . 41
5.1.1 Neural Networks . . . 41
5.1.2 Grid Search . . . 42
5.2 First data set . . . 44
5.3 Second data set . . . 45

6 Discussion **50**

7 Conclusion **57**
7.1 Future work . . . 58
7.2 My recommendations . . . 58

Chapter 1

Introduction

In modern age people and goods are transported between various locations many times each day. The transport sector is a fragile system which can easily be disturbed resulting in delays, accidents and economic losses. Hence effective and reliable transport systems are highly requested. Such a system requires that vehicles are available when they are planned to operate, it is crucial to avoid unplanned halts and breakdowns. Inevitable vehicles break down or needs to exchange spare parts form time to time. In such an event, in order to foster sustainable development and minimize the time of reparation, identifying the reason of error is essential.

This thesis investigates the possibilities of applying machine learning algorithms in order to predict parts in need of replacement based on electronic errors exhibited by vehicles. A strong correlation between electronic error codes and fault causing parts could yield a powerful tool in diagnostics. The possibility to collect electronic error codes form operating vehicles could enable for a tool which could foresee problems in an early stage. Vehicles could then visit a workshop prior failure, thereby avoiding breakdowns or unscheduled halts.

1.1 Problem description

Every Scania vehicle offered today is modular, meaning that it is customized according to customer specifications by combining a large set of parts and components. This means that each truck and bus is different to the next, which in turn requires knowledge about how the individual components function in each vehicle and how they should be serviced and maintained in order to provide high quality products. Each vehicle has a number of *electronic control units* (ECU) that controls various parts of the vehicle. Typically a vehicle would have 20-80 different ECU's, depending on vehicle configuration and intended use. Ideally if a problem occurs, the affected ECU would generate an error code that describes the problem. Implementing such a system is obviously not an easy task and often the information of an error code is not descriptive enough to define the problem at hand. The error codes, known as *diagnostic trouble code* (DTC), is typically a result of a drop in voltage or similar. Hence it is not always obvious what caused the problem given an DTC.

Problem that arises can vary in severity, a problem believed to be very serious will most likely generate a DTC that immediately notifies the driver, while most problems generates DTC's that is stored and visible only later as the vehicle is maintained. By connecting the vehicle to a computer one can use a software program, developed by Scania, named SDP3 to troubleshoot the vehicles electronic system. During such a process all DTC's that have been indicated and stored will be observed. At this point some DTC's might indicate problems that are already solved, some are irrelevant and possibly only few of them are good indicators of problems where parts needs to be exchanged.

Some problems are easily understood by examining observed DTC's, others can be very hard to diagnose. If a problem cannot be diagnosed, the mechanic are sometimes forced to exchange fully functional parts in pursuit of a solution. Changing a functional part is costly and bad for the environment and a stationary vehicle at a workshop can carry large loss in income. Thus a predictive model which is able to diagnose parts which needs to be exchange comes with many benefits.

1.2 Objectives

The ambition of the master's thesis is to investigate if DTC's are good indicators for parts that needs to be exchanged (fault causing parts), by applying statistical learning techniques. The aim is to accurately be able to predict spare parts for a vehicle based observed DTC's. In mathematical terms this amounts to solve a multivariate supervised classification problem. The problem turned out to be a high dimensional problem.

A suitable data set did not exist, hence a prerequisite for the thesis was to generate an appropriate data set. Such a data set should contain events of observed error codes for a vehicle and those parts that had been exchanged on that vehicle based of those error codes.

Two methods Neural networks and Support Vector Machines will be implemented and compared.

1.3 Outline

This thesis starts with a chapter on data preprocessing, which not only describes the data used and how it was obtained but also involves a lot of background which facilitates the understanding of the thesis. This chapter starts by describing what would constitute an ideal data set, then explained the sources from which data was obtained and the process of coupling data together and then explains concerns if data which separates is from the ideal set. At the end of the chapter the data which is used in the thesis is described and analyzed and an algorithm to reduce the dimensionality of data is described.

Following this chapter is the theoretical background which explains the relevant concepts and methods that are used to create the classifiers. This chapter is quite extensive and the intention is that one not so familiar to these methods could get a grasp of how they work in order to understand why e.g. the methods are sensitive to data and so that the output of the classifiers could be properly understood. In the next chapter the evaluation measures are described and a naive classifier to which the prediction accuracy can be compared is suggested. The chapter also involves the implementation of the learning algorithms in `R` and describes how suitable parameters were obtained.

Finally the results are given followed by a discussion. The last chapter contains the conclusions drawn from the analysis and future research opportunities to further improve and expand the capabilities of a predictor are identified.

Chapter 2

Data preprocessing

The basis for any statistical analysis is *good* data. The meaning of "good" may vary between different problems and in this chapter it will be described what would constitute good data for the problem of this thesis. Prior to this thesis, to my knowledge, no data set connecting observed DTC's and exchanged spare parts on vehicles were in existence. Thus in order to conduct the thesis project such a data set had to be generated. This task came with various problems which will be discussed in this chapter. The data set was continuously updated and improved upon during the course of the thesis as meaningful discussions and insights were gathered.

We will start this chapter by explaining how an ideal data set would look like and then explain how the dataset used were created and in what ways this deviates from the ideal set. The reader unfamiliar with statistical learning might benefit from reading section 3.1 prior to this chapter.

2.1 The ideal dataset

The intended use for the data set is to implement a supervised classification algorithm. This means that we would like to structure data into *events*. One event would consist of one vehicle, its observed errors and the part(s) exchanged to solve the problem. The number of events should be arbitrary large so that one could choose a sample size, and if performance were poor one could always obtain more observations.

The implication here is that there is a perfect correlation between observed DTC's and exchanged parts, that is the DTC's are generated in such a way that they indicate problems with one or several parts that needs to be exchanged. We would like to structure data into a table or matrix such that each row corresponds to one event. We would have one column for each DTC and one for each class. Thus if we would have n events, p DTC's and k classes, the matrix would be off size $n \times pk$. It would be a binary matrix with a one indicating that a DTC have been observed or that an event belongs to a class. An illustration of such a matrix is given in table 2.1. The set of DTC's will represent the input variables in the learning algorithms and the classes will represent the output or response variables. One event should belong to only a single class. Thus one class need not to correspond to a single part but could be composed from several parts.

Next there will be given some background on how data is gathered to facilitate the understanding of why an ideal data set could not be achieved.

2.2 Source of data

One challenge in generating the data were that the relevant information were not compiled at a single source. The error codes were found in log-files to SDP3, the program used to troubleshoot

Table 2.1: *Illustration of the desired structure of data. A matrix where each row corresponds to one event, the first p columns corresponds to the p DTC's respectively and the last k columns correspond to the k different classes. For each event the observed DTC's and the corresponding class is indicated with ones.*

	DTC_1	DTC_2	...	DTC_p	class_1	class_2	...	class_k
Event_1	1	0	...	0	1	0	...	0
Event_2	1	1	...	0	0	0	...	1
⋮	⋮	⋮	⋱	⋮	⋮	⋮	⋱	⋮
Event_n	0	1	...	0	0	1	...	0

the electrical system of a vehicle. The actions taken were gathered in work orders stored in a database. This called for some systematic way of combining the information by the different sources to create the desired events. Key elements for this includes a *timestamp* and *chassis number*. With timestamp we referee to the date the vehicle visited the workshop and a chassis number is like an identification number, unique to each vehicle.

2.2.1 SDP3 log file

SDP3 is a software program developed by Scania intended for maintenance of vehicles. The software communicates with the vehicles electronic control units (ECU's) and it is possible to troubleshoot a vehicles electronic system. When the program is used a log-file is created which continuously registers actions the user takes, as well as information regarding the vehicle and e.g. observed DTC's. The original intention of the log-file was for development purposes not to gather statistics regarding how it is used or vehicle health status, hence the available information has its limits. Fortunately for the purpose of this thesis it is possible to extract information such as date, chassis number and DTC's which is necessary to create desired events.

Upon launch, SDP3 initializes a log-file. The log is closed, saved and sent to Scania as the program is closed. This means that a single log-file could contain information from several vehicles and dates. Typically a log-file contains thousands of rows, each corresponding to a logged event, sometimes even millions. Thus it is necessary for an efficient strategy to extract the relevant information for these log files. Fortunately Scania uses a software called Splunk to index every row in such log files. Over the course of a year there are almost $30 \cdot 10^9$ rows indexed with available information. Using Splunk it is possible to extract the relevant information.

2.2.2 WHI database

There are lots of information contained within the workshop history (WHI) database. In this thesis we are interested of the information in *work orders* stored in the data base. Usually when a vehicle visits a workshop a work order is created by the mechanic. A work order specifies measure taken to the vehicle e.g. manual labor, exchanged parts etc. One work order is specific to one vehicle and the next time that vehicle visits a work shop a new order is established. Each work order contains a timestamp, chassis number and information about measures taken. Since the information is kept in a data base one could use SQL-queries to obtain desired information. However at Scania different groups *manages* different data. There is user information in data, thus it needs to be managed confidentially. This meant that I could not get direct access to the WHI data base and had to ask for an excerpt.

Note that many times when a vehicle visits a workshop it is not necessary to exchange parts.

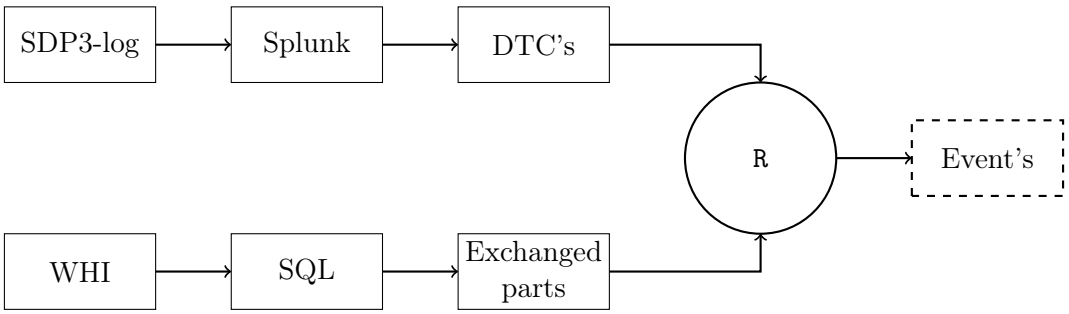

Figure 2.1: *Schematic overview of the process of creating an event. Using Splunk, SDP3s logfile are searched and information of chassis number, timestamp and observed DTC's are extracted and saved. Similarly using SQL-queries work orders are searched and information of chassis number, timestamp and exchanged parts are extracted and saved. This data are then imported to R where events are generated.*

Sometimes manual labor solves problems, sometimes vehicle visits for routine control or maintenance. Thus not all work orders are of interest. To keep things in line with the ideal data set, we wish only to use work orders were parts have been exchanged based upon observed DTC's.

2.3 Finding matching events

In the process of creating an event there are several steps to obtain data which then has to meet a number of criteria. As described above, data on observed DTC's are found in the log files of SDP3. Using Splunk one can search these files and extract relevant data. Once this is complete one are left with a data set containing chassis number, timestamp and observed DTC's. Similarly work orders are found in WHI data base and using SQL-queries one can search and extract relevant information. This again leaves us with a data set containing chassis number, timestamp and exchanged parts. Next one must combine the data sets and search for matching chassis number and timestamps. This is done in `R`, which is a free software environment for statistical computing and graphics. `R` can be downloaded from `http://cran.r-project.org/`. An schematic overview of the process is depict in Figure 2.1.

When events are to be matched one has to be cautious, we strive to only keep events that are true representatives of a reparation involving the exchange of a spare part based on observed DTC's. This is not a simple task for various reasons which will be discussed in detail in the sections to come. For now it is enough to know that it is only a small portion of the available data that are considered to be suitable as training samples for the learning algorithms. Figure 2.2 illustrates the different sources of data and what portion that are considered suitable.

2.4 Generating classes

Once data is imported to `R` there are a few things to consider prior to matching chassis number and timestamp to create events. Sometimes when a vehicle is repaired the workshop uses *local parts* when a part is replaced. In this report a local part refers to a part not made by Scania (could be a part that Scania provides but also other suppliers, or a part that Scania does not supply e.g. motor oil). When a work order contains local parts it is often impossible to know what that part is, thus some actions must be taken. One approach could be to simply omit those rows in a work order containing local parts and keep the rest of the event. This approach could

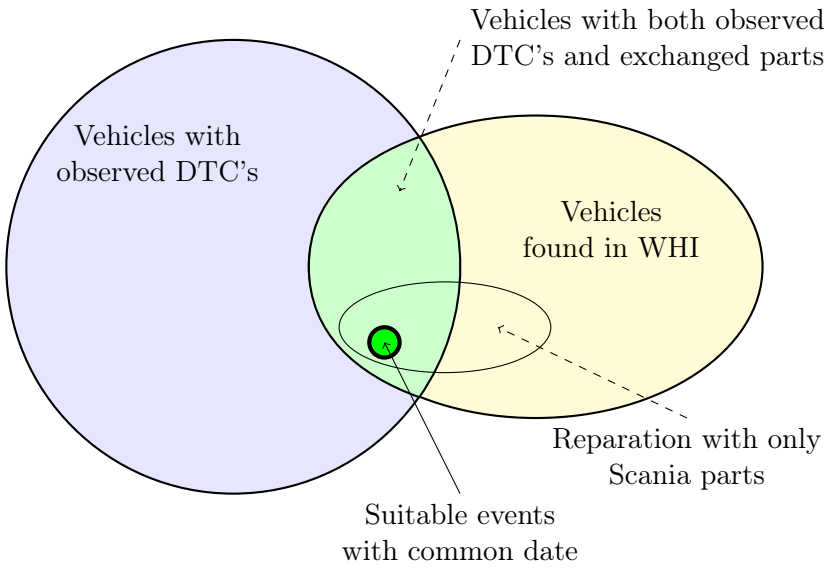

Figure 2.2: *A simple illustration of the portion of data considered suitable. The blue circle corresponds to events of observed error codes, the yellow ellips to events where parts have been exchanged. The light green overlapping area represents those vehicles for which there exists both observed errors (DTC's) and exchanged parts. The small ellipse partially within the overlapping light green area represents reparation where only Scania parts were used and finally the green circle inside the overlapping area corresponds to events considered suitable for classification.*

work in many cases e.g. if the local part is a screw or windscreen whippers or similar non-crucial parts. But if the local part happens to be the fault causing part indicated by DTC's then viable information is lost and bias will be introduces by such an event. Hence for the purpose of this thesis all work orders containing local parts has been discarded. This resulted in a dramatic loss of data and possible events but in the end the data sample were considered large enough using this strategy.

Once all unknown parts have been removed we can create *classes*. As mentioned in section 2.1 in a supervised classification framework an observation should belong to a single class. Hence in reparation of vehicles where several parts are replaced, it is the set of replaced parts that should constitute a class. The initial strategy were such that class labels were designed by directly observing exchanged parts for one chassis number and timestamp. Each unique set represented one class. This strategy has it flaws, which is discussed below. The initial strategy were later abandoned by one which involved a lot more data management.

The problem with the first approach was that there could be several classes virtually representing the exact same measure. For example, consider a vehicle with a defect turbocharger. During reparation it is decided that a gasket needs to be exchanged as well as the turbo. Now consider another vehicle with defect turbocharger, during reparation the turbocharger is replaced but not the gasket since it was considered to be in good shape. A during reparation of a third vehicle with defect turbo possibly the driver requests to have the windscreen wipers replaced whilst in the workshop. Examining the work orders of these events one would find three different sets of replaced parts. Using the initial strategy this yields three different class labels for virtually equal measures. Ideally in the example above we would like to indicate the turbocharger as fault causing part and have the three events belonging to a single class. Another concern is that different vehicles (production year, model, configuration) might use different turbochargers, i.e.

they would have different part numbers which again would generate several classes for virtually equal measures.

Learning algorithms were implemented with a data set defining classes using the initial approach. As performance were evaluated the flaws became apparent. Consider events where a turbocharger is exchanged, it is likely that the set of DTC's are quite similar for those events. We could even, for the purpose of illustrate the problem, assume that the set of DTC's are equal for those events. The aim of a learning algorithm is to predict to which class a set of DTC's belong. As described these observations would fit any of a number of classes but in the test data we say that it belongs to only one. If there are three classes which really should be represented by one there is at best $\frac{1}{3}$ chance of predicting the correct class. We cannot expect good performance of an algorithm operating under these conditions.

Improved strategy

By manually examine the set of exchanged parts for one chassis number and timestamp it is quite easy to determine suitable classes, however the number of events were way to many for this to be feasible. Hence it was called for a systematic way to determine suitable classes. The improved strategy used a software program called Multi, which is designed by Scania and contains information regarding spare parts. By searching for an article number, one obtains not only that parts, but one can also see to which other parts that specific part is included in. For example by searching an article number of a screw one would obtain a list of all parts in which that screw are used like a turbocharger or a water pump or similar. Hence if both a screw and a turbocharger are present on a work order, those article number are searched individually and if a single part can be identified to which both parts are included then that part will be used as a class for that observation. In the example above it is likely that the screw is included in several different parts, but the turbocharger are likely constitute a part on its own. If that is the case the observation would be labeled as a turbocharger. If the screw were the only part in the work order then it would have been omitted from the data.

For each chassis number and timestamp the corresponding spare parts were searched for in multi. If all parts could be identified to a single part then that would be used as class for that observation. This strategy has its pros and cons. Its advantage is that minor parts, like cable ties, screws and similar are filtered out if such parts belongs to more than one group (which they usually do), i.e. only events where a *vital* part has been exchanged is kept. Also events where vital parts are exchanged in combination with minor parts relevant to that reparation will be kept and labeled as one class instead of several. The disadvantage is that in reparations where several different vital parts has been exchanged will be discarded. Moreover reparations where a vital part has been exchanged, but also perhaps the windscreen wipers will be removed since they could not be identified as one single part.

Below we will discuss noise in data, in the improved strategy a few classes were removed manually since they were believed to be noise (events where DTC's are uncorrelated with replaced parts). Also a few DTC's which obviously could not be generated by a fault causing part were removed.

2.5 Concerns

The process of generating a data set for the learning algorithm came with a number of concerns. A few, like finding matching events, and generating classes have all ready been discussed above. In this section we will have a closer look at some aspects that needs some attention in order to

obtain good performance and have a data set as close as possible to an ideal one. In section 2.3 it is mentioned that only a small portion of available data are considered suitable and we start this section by having a closer look at the connection of data between the sources.

2.5.1 Connection

Our goal is to structure data into events. One event should represent a reparation of one vehicle. Ideally then there would exist a single source where information about observed DTC's and exchanged parts is compiled. This would eliminate any problems combining data from various sources might bring. The foremost problem of combining data is how to make sure that data belongs together. Fortunately identifiers like chassis number and timestamp are available in both sources of data which could be used to connect them. The question is why only a small portion of data can be used?

The primary reason is that many times a vehicle is connected to SDP3 and DTC's are registered no parts are exchanged, likewise it is not every time a work order is created that there are readouts about DTC's. But there are reparations with both observed DTC's and replaced parts that cannot be matched since either chassis number or timestamp is erroneous. In SDP3's log file, where observed DTC's are found, the timestamp is most often very accurate. The timestamp is determined by the local computers time, hence if the time on the computer running SDP3 is of, then the timestamp of those observations will be erroneous. In work orders, stored in WHI database, the date is determined by the date the vehicle arrived at the workshop. This date however is entered manually by the mechanic and thus holds no guarantee to be correct. What this means is that some data will be thrown away since the date between sources differ. This is unfortunate but not of a great concern as long as there is enough data.

The main issue with the connection is the uncertainty that arises. As data is gathered by different sources and the coupled together there is a possibility of erroneous events, e.g. false dates that happens to match, or even erroneous chassis numbers, meaning that observed DTC's may not be suitable indicators for replaced parts. More on this under section on noise below. As mentioned, an event aims to describe one reparation of one vehicle. In data an event is defined by a unique combination of chassis number and timestamp. Whilst there are deviations most events are believed to be true in the sense that they do describe one vehicle reparation.

2.5.2 Maintenance

Most vehicles are regularly maintained based on operating hours or mileage. Events where vehicles are maintained are not desired to have in the learning data. If a vehicle visits a workshop for a scheduled maintenance then the parts that will be exchanged are most likely determined prior to any observed DTC's. In fact the DTC's are registered in SDP3's log file as soon as a vehicle is connected, thus they could be found in data even if a mechanic has not examined them. This means that we should not expect there to be a connection between observed DTC's and any replaced parts for such events.

In the improved strategy, events where vehicles have been maintained was removed manually from learning data.

2.5.3 Noise

In this setting noise in data refers to events that meets all criteria to be classified as an event but still is not a true representation of an event by our definition. Hence in events considered as noise we believe that there is no correlation between observed DTC's and replaced parts. Above we have discussed maintenance of vehicles, such events are perfect examples of what we mean by noise in data. It is believed that all maintenance work have been removed from the final data set, however it would be a bald statement to make, and if any remains we would consider these events noise.

An other example of noise would be if the replaced part apparent in the work order is not the true fault causing part indicated by DTC's. Let us take a moment to recall that a DTC is most often a result by a drop in voltage or similar and implemented by a software engineer. Some errors, like a broken light or similar might be easy to diagnose, but far from all problem are so easy. And for sure there will arise problems that could not be foreseen. Hence when we say that DTC's indicate a fault causing part it is not always obvious what parts that are the reason for activating DTC's. For this reason it is plausible that fully functional parts are replaced trying to solve a problem. In such events the exchanged part in our data sample is not the true fault causing part. Such events are very hard to identify and if present would add noise to data.

So far we have discussed noise amongst the events, that is unwanted events in data which for some reason could not be filtered or removed. Another type of noise in data is noise within an event. This happens when there are some of the observed DTC's that are uncorrelated with the replaced parts. This type of noise is believed to be quite common. Examining common DTC's one finds e.g. that there are DTC's to indicate if the vehicle has moved without the key. Such a DTC could be activated if the vehicle is toed, or even by strong wind. Hence for a vehicle with engine problems that has to be toed to a workshop among observed DTC's there will not only be DTC's indicating the problem we would like to diagnose, but possible also a DTC indicating that the vehicle has been moved without a key. As there were thousands of unique DTC's all of them could not be evaluated in order to find such that adds noise and no extensive such search has been performed. However in the improved strategy a few could be identified and removed manually from data.

It is hard to estimate how much noise the data contains, and no such efforts has been made. Hence it is unknown how much noise there is within data. Since an event must have same chassis number, date and cannot contain local parts it's reasonable to believe that most events are at least observations of one vehicle reparation. Uncertain is how well DTC's can indicate what parts that is in need of repair. With a large enough data sample, noise should hopefully not be a great concern.

2.6 Analyzed data

In this section we will examine the resulting data sets used in the learning algorithms. The section will be divided into two sections, one for the data resulting using the initial strategy and one for the data generated using the improved strategy.

2.6.1 Initial strategy

The data used in the report were collected in march of 2017. SDP3's log files are stored about one year which limited the possible events for this thesis. Both data sources were searched for suitable data for the period 2016-03-01 to 2017-03-11. However for unknown reasons no events

Table 2.2: *The table shows the number of DTC's and classes, in data generated using the initial strategy, with observations greater than or equal to various thresholds.*

Threshold value	1	10	50	100	1 000
DTC	18 844	3621	1009	534	39
Class	5 565	148	28	11	-

matched prior to 2016-06-02, in fact it was first after 2016-07-12 that events matched as expected. Hence the used data set is based on reparations performed during 2016-07 to 2017-03.

Following the initial strategy described in section 2.4 the resulting data set contained 12 605 matching events. This is quite a disappointing figure considering that there were 1 260 165 possible matches of observed DTC and 75 924 possible matches of replaced parts. Among the common events there were 18 844 unique DTC's and 2 981 unique parts. Recall that a class were determined by a set of spare parts exchanged for one reparation. Thus among the 12 605 events there were 5 565 unique combinations of replaced parts. In terms of a learning problem this means that we would have 18 844 features (input variables), 5 565 classes and 12 605 observations. In Table 2.3 these numbers are compared with those when data was generated using the improved strategy.

Examining the data one finds that most DTC's have only a few observations. In Figure 2.3 there is a histogram of the DTC's. One can see that it is far from an even distribution, in fact the most common DTC occurs 7 210 times but there is only 39 DTC's with more than 1000 occurrences and 534 DTC with 100 or more occurrences. There is a similar distribution among the classes which is illustrated in Figure 2.4. The most common class has 238 observations but only 11 classes has more than 100 observations. There is 28 classes with more than 50 observations and 148 classes with 10 or more observations. More than 82% of the classes has just a single observation. In Table 2.2 the number of DTC's and classes with frequency greater than or equal to various thresholds is displayed.

The distribution of DTC's are not a major concern, few occurrences are not ideal but at least as the *true* indicators are many this should not have a dramatic effect on the performance. Few observations to each class however is a problem. Consider a learning problem with few observations to each class. Data is partitioned randomly into training and test sets, hence with only few observations in each class it is possible that all observations of one class ends up in the test set. Then it seems unreasonable to expect a learning algorithm to perform well on classifying observations to classes never seen before. Imagine that you are asked to classify tuna species, you might know that there exists many, 61 in fact [21], but if you never got the chance to study them chances are slim that you would get many correct. There are really only two alternatives here; either one has to obtain more data to increase the number of observations in each class or some classes has to be omitted. As explained the data set all ready contains those observations available to us, hence we must omit most classes.

2.6.2 Improved strategy

In section 2.4 we discussed a way to improve the generation of classes. As some classes were essentially copies when generated using the initial strategy and the improved strategy claimed to solve the issue we expect the situation to improve using the improved strategy. One drawback is that using the improved strategy we lose many events. We have already discussed some reasons

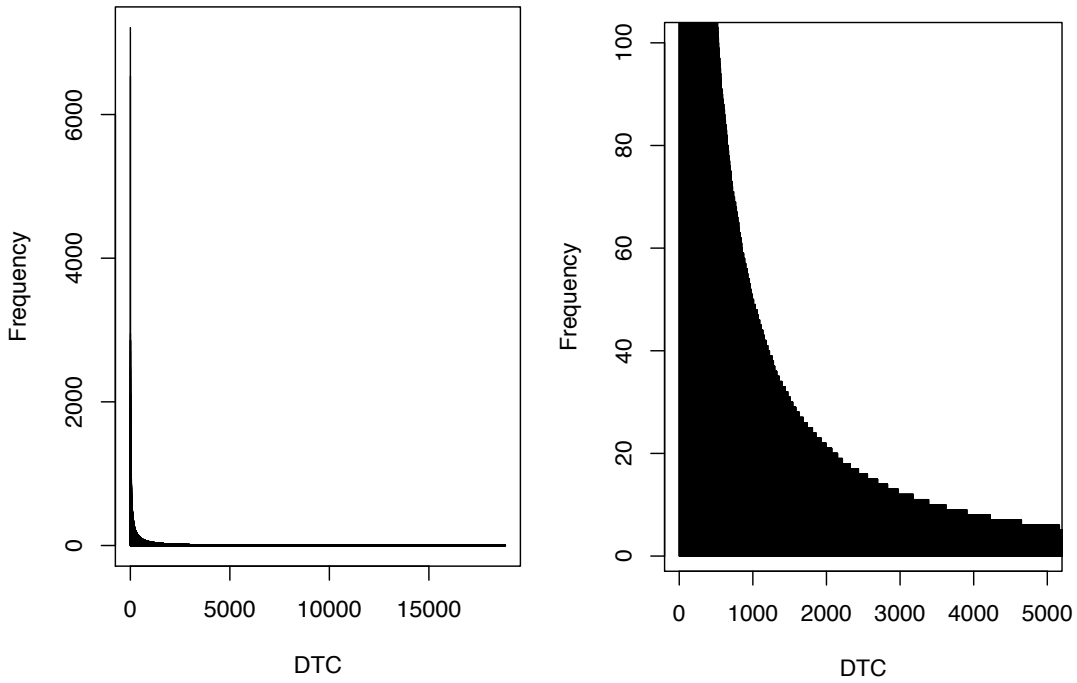

Figure 2.3: *Left panel shows a histograms of DTC's in data generated using the initial strategy and the right panel presents a zoomed version of the plot.*

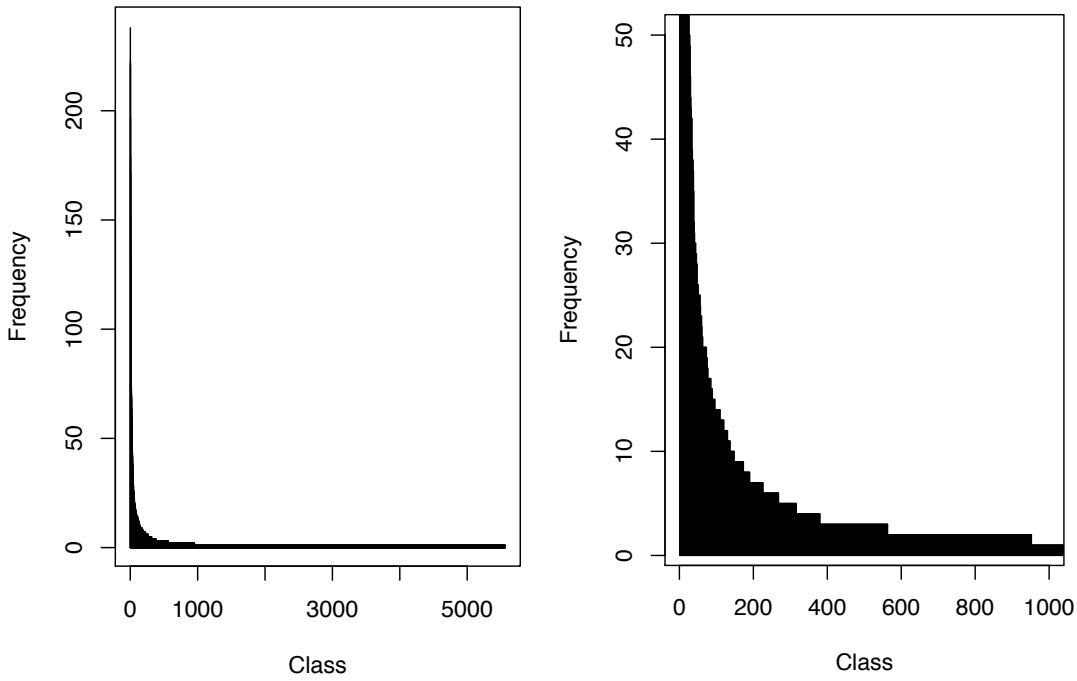

Figure 2.4: *Left panel shows a histograms of classes in data generated using the initial strategy and the right panel presents a zoomed version of the plot.*

Table 2.3: *Overview of the learning data set using the initial and improved strategy for generating classes*

Strategy	Matching events	Input variables	Classes
Initial	12 605	18 844	5 565
Improved	6 002	13 360	477

Table 2.4: *The table shows the number of DTC's and classes, in data generated using the improved strategy, with observations greater than or equal to various thresholds.*

Threshold value	1	10	50	100	1 000
DTC	13360	2019	498	257	9
Class	470	103	22	9	-

why events could be lost in section 2.4 and we recall that it is not necessarily a bad thing that events are removed. Some will have been removed since we believe that they are not suitable for classification, but still there will inevitably have been some undesired loss of events.

The improved strategy is really an extension of the original model and this becomes clear when it is implemented. We must first generate a data set basically using the initial strategy up to the point of generating classes. Hence the data set using the improved strategy could never end up being larger than the initial one. Applying the described algorithm 5 430 out of the 12 605 available events were removed, that is about 43%. This is quite a substantial loss, and even more events are omitted as a few classes and DTC's are removed manually according to the method described in section 2.4. Totally 6 603 events were removed which leaves 6 002 events. The number of DTC's and classes are displayed in Table 2.3 along the results using the initial strategy. Examining these numbers one can see that the number of matching events using the improved strategy is about 48% compared to the initial one. About 71% of the input variables (features) are kept but the most significant difference lies in the number of classes which is only about 8.5% compared to the number of classes generated using the initial strategy. Then, as the number of classes are reduced a lot more than the number of events and DTC's there will be relatively more observations to each class. In figures 2.5 and 2.6 there is histograms of the DTC's and classes in the data generated using the improved strategy.

Comparing the histograms in figures 2.3 and 2.4, generated using the initial strategy to those in figures 2.5 and 2.6, generated using the improved strategy one can see that they are quite similar. In Table 2.4 the number of DTC's and classes in the data generated using the improved strategy is displayed. Comparing the numbers to those of table 2.2 we can see that even though the number of events are halved using the improved strategy, the number of classes with 50 or 100 observations are basically the same. Even tough we believe that the improved strategy is superior to the initial one, we cannot be sure if it lives up to its name before testing the performance of data in the learning algorithms.

2.6.3 Reducing dimensionality

In general reducing dimensionality of input data could be crucial to get good performance using a learning algorithm. If we, in a model with many parameters (high dimensionality of the feature space), try to determine those parameters optimally by waiting for the algorithm to converge, we run the risk of overfitting [2] (p.343). For the sake of this thesis, and due the structure of

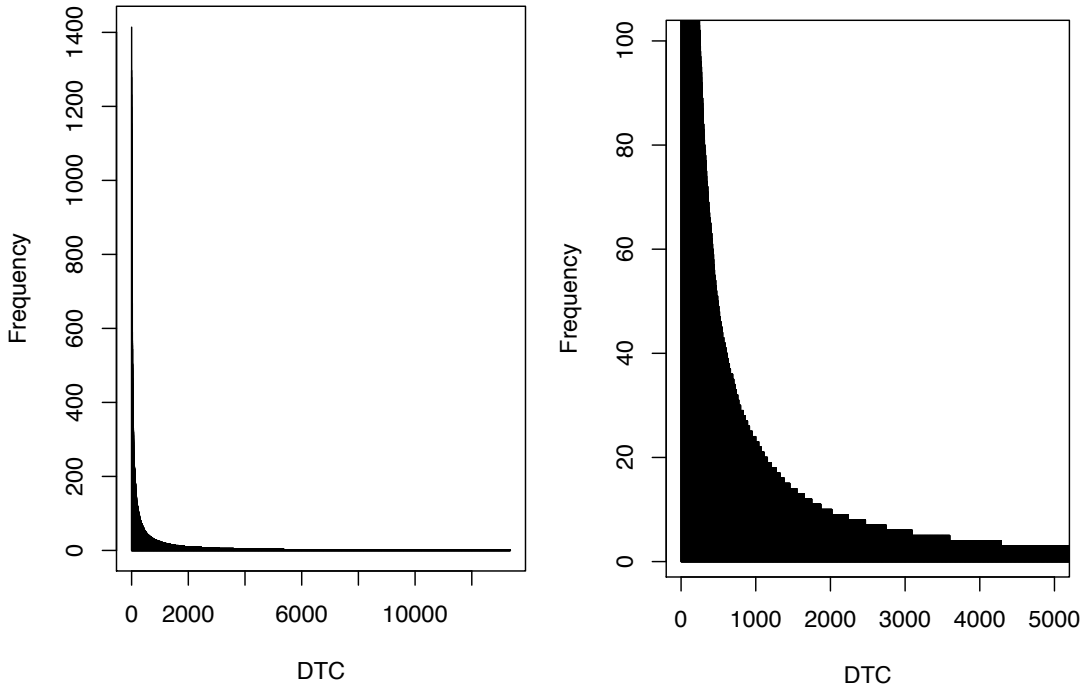

Figure 2.5: *Left panel shows a histograms of DTC's in data generated using the improved strategy and the right panel presents a zoomed version of the plot.*

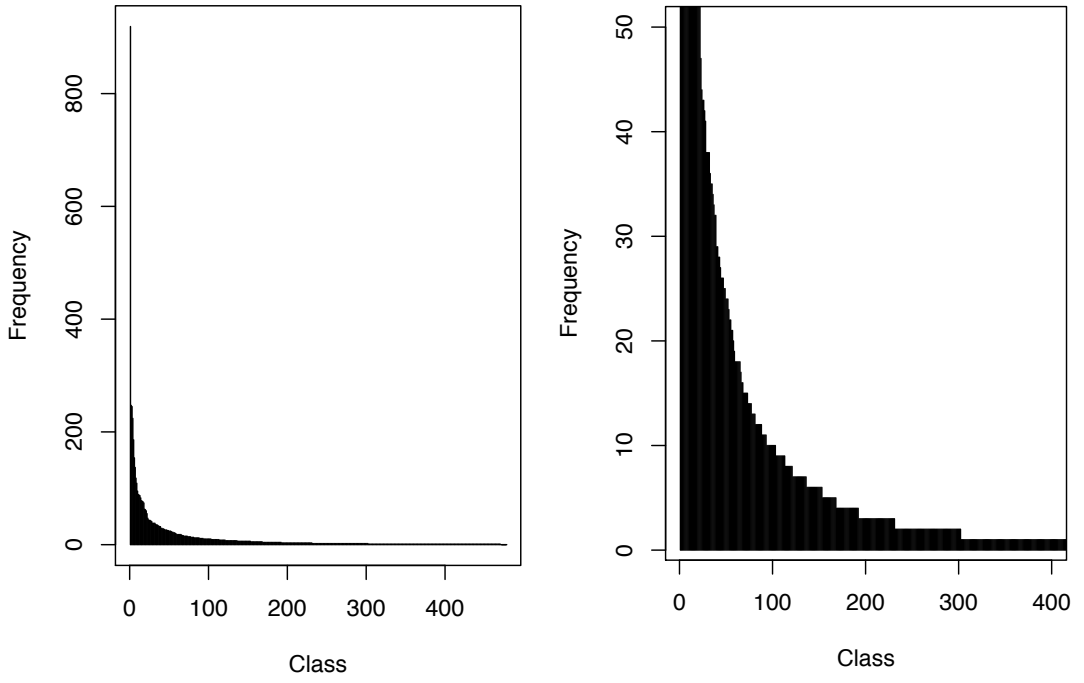

Figure 2.6: *Left panel shows a histograms of classes in data generated using the improved strategy and the right panel presents a zoomed version of the plot.*

the problem, reducing the dimensionality is crucial.

When we reduce the dimensionality of data we wish not only to reduce the number of input parameters, but also the number of classes. The later may not be typical for a classification problem, but for the purpose of this thesis it is not important to be able to classify any type of reparation but rather to investigate if such predictions is possible. In this pursuit the quality of data is much more important than the number of classes. Also there are a huge number of possible classes and to create a full scale network would demand an immense data sample, enormous computational power and is beyond the scope of this thesis. Further the primary problem at hand is the lack of observations, we cannot train a learning algorithm without a fair number of observations to each class and expect good performance.

As suggested in [2] (p.334-335) one could use principle component analysis (PCA) to reduce dimensionality of input data. Other suggestions include to set some connection weights to zero [12], but such an operation could have a significant effect on the remaining parameters in the model if the inputs are close to being collinear. Hence, in general it is not recommended to set more than one connection weight to zero [13], which defeats the purpose of reducing the network size. The approach in this thesis has been less technical but more efficient for the problem at hand, considering the structure of the problem.

In pursuit of reducing the dimensionality, the number one priority is to enhance the performance of the learning algorithm. Moreover a benefit of having a smaller data sample is the computational time needed to train the learning algorithms. Two criteria were used when the data samples were managed, events were removed using the following algorithm:

1. Sort all DTC's based on its total number of observations
2. Remove all events which contains a DTC observed less than or equal to ξ times.
3. Sort all classes based on the number of observations
4. Remove all classes with less than or equal to δ observations.

The set of parameters (ξ, δ) can be treated as tuning parameters and allows certain flexibility. ξ corresponds to a threshold for the number of times a DTC must have been indicated for observations containing that DTC to be included in learning data. Similarly δ corresponds to a threshold on the number of observations a class must have for it to be included in learning data. The motivation to use the frequency of DTC's as a criteria to remove events was the intuition that few observations of an input variable (a DTC), would be seen as noise by a classifier. Hence by removing DTC's with only few observations we hoped that the performance would improve. Note that we remove not only DTC's with few observations but whole events that contains such DTC's. If we were to eliminate DTC's without considering the entire event it is possible to introduce bias by removing a DTC indicating fault causing part. The second criteria, to remove events based on number of observations, seems logical and has already been discussed, it seems unreasonable to expect a classifier to perform well without a chance to train.

Note that the tuning parameters (ξ, δ) give a great deal of flexibility, e.g. if we choose $\xi = 0$ in the algorithm above no events would be removed based on the first criteria.

Chapter 3

Theoretical background

This chapter provides an overview of the methods and scientific basis relevant for this thesis. It is not my intention to give an in-depth discussion here, but rather to give an introduction to, and highlight a few key properties of these techniques. The first part of the chapter will focus on statistical learning in general after which a few more technical sections will follow covering concepts like linear separability and hyperplanes, and classification techniques such as support vector machines and Neural networks.

3.1 Statistical learning

At its core, *statistical learning* is about learning form data. But what there is to learn and how to learn from data can be very different and largely depend on data at hand. When I think of statistical learning I think of computers and algorithms that uses sets of data to *learn* a decision rule or similar in order to make predictions about some variable. My view is largely influenced by our modern age with enormous data sets and low computational cost of storing and processing data. Many techniques that today are considered modern and best practice were developed several decades ago and virtually ignored by the statistical community for a long time. It is because of the current focus on large data sets that these techniques are now regarded as serious alternatives to traditional statistical techniques [2] (p1-3).

Statistical learning naturally has a intimate relation to many areas of science, finance and industry. Example (found in [1] p.1) of situations where learning from data is useful includes:

- Predict whether a patient, hospitalized due to a heart attack, will have a second heart attack. The prediction is to be based on demographic, diet and clinical measurements for that patient.
- Predict the price of a stock in 6 months from now, on the basis of company performance measures and economic data.
- Identify the numbers in a handwritten ZIP code, from a digitized image.
- Estimate the amount of glucose in the blood of a diabetic person, from the infrared absorption spectrum of that person's blood.
- Identify the risk factors for prostate cancer, based on clinical and demographic variables.

The ability to learn plays a key role in the fields of statistics, data mining and artificial intelligence, intersecting with areas of engineering and other disciplines. Next we will describe a typical learning problem and explain the elements involved.

3.1.1 What constitutes a learning problem?

In a typical learning problem there are measurements, outcomes of some variable of interest, often refereed to as *response variable*. Such measurements can usually be described as *quantitative* or *qualitative*, and we would like to predict the outcomes of the response variable based on a number of *features* that describes it. The outcome and feature measurements of an *object* constitutes a *data set* which randomly is partitioned into *training-* and *test*-set (sometimes one also includes a validation set). The training set is used to build a model, intended to predict outcomes of unseen events. The aim is to create a prediction model that accurately can predict such outcomes.

Object and its features

The object is the central part of a learning problem, it is some aspect of an object that we are interested in. Objects has attributes or features that can be used to describe them, e.g. the radius is an attribute of a sphere. A house has dozens of attributes such as, size, number of rooms, location etc. In a learning problem, if we which to predict the price of a house we would use suitable features which we believe explain the response variable (the price) in order to make a prediction. Note that if we are interested in predicting the size of a house, then the price might serve as an attribute, a feature. Hence, the response variable of an object is one of its attributes, which one depends on the problem. An object can be a patient, house, company, or as in this thesis a vehicle. Features are measurements of an object that should describe the output variable. We consider again one of the given examples above, if a patient is our object and our aim is to predict if whether a patient hospitalized due to a heart attack, will have a second heart attack then demographic, diet and clinical measurements are features.

Quantitative/qualitative variable

An output variable is *quantitative* if it is numerical and represents a measurable quantity. For example, the population of a city is the number of people in the city - a measurable attribute of the city. Therefore, population is a quantitative variable. Other examples include a stocks price, the size of a house etc. A *qualitative* or *categorical* variable is one that takes names or labels such as the color of a ball or breed of a dog. Categorical variables have in common that their values have no natural order. If a patient has a heart attack or not is also an example of a categorical variable. There is a third type of variable, an *ordinal* variable [22] which has the property of both identity and magnitude. Characteristic for an ordinal variable is that it can be ranked but the margin between variables is subjective. For example, a grading system is ordinal, lets say that movies are graded on a scale from 1 to 5, 5 being the highest score. We say that a movie with a rating of 5 is better than one with a rating of 4 and that a movie with a rating of 4 is better than a movie with a rating of 3 however we cannot say that a 5 is twice as good as a 3.

Data-, training- and test-set

A data set consists of measurements of output and features of an object. Using a data set for a learning problem it is common practice to randomly divide the data into *training* and *test* set [2] (p.11), assuming that the data set is large enough. Sometimes a *validation* set is also used. Training data is usually 70 percent of the observations and is used to build the prediction model and remaining 30 percent of data is used to test the performance of the classifier. It is important that the test set is unseen by the prediction model. Sometimes a validation set is used in the fitting process [1] (p.219-223). Using a validation set, data could typically be partitioned e.g. 70:15:15 into three sets; training, testing and validation respectively. While the model is being fitted, the validation set can be used to test the performance. Typically we would see an initial

phase where validation error is reduced but as overfitting starts to occur we would expect the validation error to increase. Then the validation error could be used as a stopping criteria. In this approach it is important to separate the validation and test set as the test result would be biased otherwise.

3.1.2 Supervised vs Unsupervised learning

In statistical learning we talk about two different types of learning, *supervised* and *unsupervised* learning. Most common is supervised learning where the outcomes of each observations is known, and used to build the prediction model. In unsupervised learning problems there are measurements of features of an object but no measurements of an outcome variable. The task is rather to describe how the data is organized or clustered [1] (p.2). Above a typical learning problem was described which is an example of supervised learning problem. The focus of this thesis is on the statistical learning in supervised setting.

3.2 Linear separability

In this section we will define the concept of linear separability of sets of points in euclidean space.

Definition 3.1. Let X_0 and X_1 be two sets of points in p-dimensional euclidean space $\mathcal{X}$. Then X_0 and X_1 are said to be *linearly separable* if there exists a real vector $\boldsymbol{\beta} \in \mathbb{R}^p$ such that for every point $\mathbf{x} \in X_0$ satisfies $\boldsymbol{\beta}^T\mathbf{x} > k$ and every point $\mathbf{x} \in X_1$ satisfies $\boldsymbol{\beta}^T\mathbf{x} < k$ for some $k \in \mathbb{R}$.

In two dimensional space ($p = 2$) linear separability means that two sets of points can be separated by a straight line.

3.3 Searating hyperplanes

Inspired by the theory on separating hyperplanes given in [1] (p.129-135 and [2] (p.371-373) we defining a *hyperplane*. Consider a p-dimensional feature space $\mathcal{X}$, a hyperplane $\mathcal{D}$ on $\mathcal{X}$ is a flat affine subspace of dimension $p - 1$. It is characterized by the linear equation

$$\mathcal{D} = \{\mathbf{x} \in \mathcal{X} : f(\mathbf{x}) = 0\} \tag{3.1}$$

where

$$f(\mathbf{x}) = \beta_0 + \sum_{i=1}^{p} \beta_i x_i \tag{3.2}$$

for some parameters $\beta_0, \beta_i,\ i = 1, 2, \ldots, p$ and at least one $\beta_i \neq 0,\ i = 1, 2, \ldots, p$. We may think the hyperplane $\mathcal{D}$ as all points $\mathbf{x}$ such that $f(\mathbf{x}) = 0$ is satisfied. In two dimensional space ($p = 2$) a hyperplane is described by

$$\beta_0 + \beta_1 x_1 + \beta_2 x_2 = 0 \tag{3.3}$$

which is simply a straight line. Hence all points on the line constitutes the hyperplane. Figure 3.1 demonstrates hyperplane in two and three dimensional space.

In real spaces, a hyperplane separates the feature space $\mathcal{X}$ into two subspaces (half-spaces [23]):

$$\mathcal{R}_1 = \{\mathbf{x} \in \mathcal{X} : f(\mathbf{x}) > 0\}, \qquad \mathcal{R}_2 = \{\mathbf{x} \in \mathcal{X} : f(\mathbf{x}) < 0\}. \tag{3.4}$$

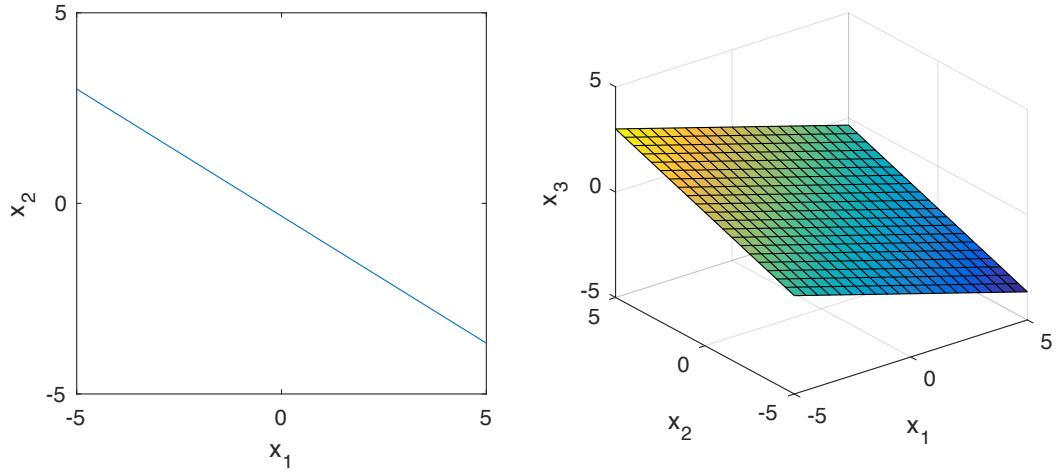

Figure 3.1: *Illustaration of hyperplanes in* $\mathbb{R}^2$ *(left) and* $\mathbb{R}^3$ *(right). To the left is the hyperplane* $1 + 2x_1 + 3x_2 = 0$ *and to the right is the hyperplane* $1 - x_1 + x_2 + 3x_3 = 0$.

Suppose that we are interested in binary classification and consider the output space $\mathcal{Y} \in \{-1, 1\}$. Assume that we have n observations from the feature space $\mathcal{X}$

$$\mathbf{x}_1 = \begin{pmatrix} x_{11} \\ x_{12} \\ \vdots \\ x_{1p} \end{pmatrix}, \mathbf{x}_2 = \begin{pmatrix} x_{21} \\ x_{22} \\ \vdots \\ x_{2p} \end{pmatrix}, \ldots, \mathbf{x}_n = \begin{pmatrix} x_{n1} \\ x_{n2} \\ \vdots \\ x_{np} \end{pmatrix}, \tag{3.5}$$

and that each of those observations belongs to one of the two classes in the output space, that is, $y_1, y_2, \ldots, y_n \in \mathcal{Y}$. That is our training set is

$$\mathcal{T} = \{(\mathbf{x}_i, y_i) \in (\mathcal{X}, \mathcal{Y}) : i = 1, 2, \ldots, n\}. \tag{3.6}$$

Further assume that the two classes are linearly separable, then by Definition 3.1 there exists a hyperplane characterized by the parameters $\beta_0, \boldsymbol{\beta}$ such that

$$y_i(\beta_0 + \boldsymbol{\beta}^T \mathbf{x}_i) > 0, \quad i = 1, 2, \ldots, n \tag{3.7}$$

where $\boldsymbol{\beta} = (\beta_1, \beta_2, \ldots, \beta_p)$ and ^{T} indicates transpose. Hence we can classify new observations $\mathbf{x}^* \in \mathcal{X}$ using the decision rule: assign $y^* = \text{sgn}(f(\mathbf{x}^*))$, where $f(\mathbf{x})$ is given by Equation (3.2).

3.4 Perceptron algorithm

It is hard to talk about machine learning, or any classification problem without mentioning the *perceptron*. The perceptron is a concept introduced by Rosenblatt in 1958 [5], in essence a perceptron is a classifier that computes linear combinations of input features and returns e.g. $\{-1, +1\}$ as class labels.

The perceptron algorithm tries to find a hyperplane by minimizing the distance of misclassified points in $\mathcal{T}$ to the decision boundary, i.e. the separating hyperplane. Following the theory suggested in [1] (p.130-132) we may express this in mathematical terms as the problem of minimizing

$$D(\beta_0, \boldsymbol{\beta}) = -\sum_{i \in \mathcal{M}} y_i \left(\beta_0 + \boldsymbol{\beta}^T \mathbf{x}_i\right) \tag{3.8}$$

where $\mathcal{M}$ is the set of indices of misclassified points. Assuming that $\mathcal{M}$ is fixed the gradient is given by

$$\frac{\partial}{\partial \beta_0} D(\beta_0, \boldsymbol{\beta}) = -\sum_{i \in \mathcal{M}} y_i, \tag{3.9}$$

$$\frac{\partial}{\partial \boldsymbol{\beta}} D(\beta_0, \boldsymbol{\beta}) = -\sum_{i \in \mathcal{M}} y_i \mathbf{x}_i. \tag{3.10}$$

In order to minimize the distance of misclassified points to the decision boundary the algorithm uses *stochastic gradient decent*. This means that the parameters $\beta_0, \boldsymbol{\beta}$ are updated based of each observation rather than computing the sum of the gradient contributions of each observation and taking a step in the negative gradient direction. For every misclassified observation $\mathbf{x}_i, i \in \mathcal{M}$ the parameters are updated following the scheme

$$\begin{pmatrix} \beta_0 \\ \boldsymbol{\beta} \end{pmatrix} \leftarrow \begin{pmatrix} \beta_0 \\ \boldsymbol{\beta} \end{pmatrix} + \eta \begin{pmatrix} y_i \\ y_i \mathbf{x}_i \end{pmatrix} \tag{3.11}$$

where $\eta > 0$ is the *learning-rate* parameter. Hence for every misclassified observation the hyperplanes direction is changed and the hyperplane is shifted parallel to itself. If the classes are linearly separable then the perceptron algorithm finds *one* separating hyperplane that is dependent on the initial values used in the stochastic gradient decent for $\mathcal{T}$. Moreover if the classes are linearly separable it can be shown that the perceptron algorithm converges after a finite number of steps [2] (p.326-328).

The perceptron algorithm does not come without flaws and a number of problems are summarized in [6]. First of all data must be linearly separable for the algorithm to converge. Moreover if data is linearly separable there exist infinite separating hyperplanes, which one found by the algorithm depends on the initial conditions. Also even though it can be shown that the algorithm converges in a finite number of steps, it can be very large.

3.5 Support Vector Machines

In this section our aim is to introduce the support vector machine (SVM). We will start by discussing the maximal margin classifier, then continue defining the support vector classifier and end the section by introducing the SVM. Sometimes people loosely refer to the maximal margin classifier, the support vector classifier and the support vector machine as *support vector machines*, but hopefully the following section will make the difference clear.

The theory on SVM described in this section is inspired by [1] (p.129-135,417-438), [2] (p.369-391) and [4] (p.337-356)

3.5.1 Maximal margin classifier

In section 3.3 we discussed what a separating hyperplane is and how it can be used for classification. Following this in section 3.4 we introduced the perceptron algorithm, which is a systematic way of finding a separating hyperplane. In general, as mentioned above, if the condition on linear separability holds, then there exists infinitely many separating hyperplanes, so how can we

decide which hyperplane to use?

A natural choice is the hyperplane which maximizes the distance to any observation. To find such a hyperplane one can compute the orthogonal distance between each training observation and a given hyperplane. For a given hyperplane the distance to a training observation $\mathbf{x}_i$ can be computed as

$$d_{\mathbf{x}_i} = \frac{f(\mathbf{x}_i)}{\|\boldsymbol{\beta}\|} = \frac{\beta_0 + \beta_1 x_{i,1} + \beta_2 x_{i,2} + \cdots + \beta_p x_{i,p}}{\sqrt{\beta_1^2 + \beta_2^2 + \cdots + \beta_p^2}}. \tag{3.12}$$

Note that if $\beta_0 + \boldsymbol{\beta}^T\mathbf{x} = 0$, defines a hyperplane, then so does $\tilde{\beta}_0 + \tilde{\boldsymbol{\beta}}^T\mathbf{x} = 0$, where

$$\tilde{\beta}_i = \frac{\beta_i}{\|\boldsymbol{\beta}\|}, \quad i = 1, 2, \ldots, p. \tag{3.13}$$

Hence, by normalizing so that $\|\boldsymbol{\beta}\| = 1$, the distance between a given hyperplane and a training observation is simply given by

$$d_{\mathbf{x}_i} = \tilde{f}(\mathbf{x}_i) = \tilde{\beta}_0 + \tilde{\boldsymbol{\beta}}^T\mathbf{x}_i \tag{3.14}$$

The smallest distance between a given hyperplane and any training observation defines the *margin* and is illustrated in Figure 3.2. Our aim is to find the hyperplane that maximizes the distance to any observation, that is the hyperplane with the largest *margin* and it is called the *maximal margin hyperplane* [4] (p.341) or the *optimal separating hyperplane* [1] (p.132). Classifying observation based on the optimal separating hyperplane is known as the *maximal margin classifier*[4] (p.341). The problem of finding the optimal separating hyperplane can mathematically be expressed as an optimization problem, namely:

$$\boxed{\begin{aligned} \max_{\beta_0, \boldsymbol{\beta}} \quad & M \\ \text{Subject to} \quad & \|\boldsymbol{\beta}\|^2 = 1 \\ & y_i(\beta_0 + \boldsymbol{\beta}^T\mathbf{x}_i) \geq M, \quad i = 1, 2, \ldots, p. \end{aligned}} \tag{3.15}$$

The set of constraints ensures that each observation is on the correct side of the hyperplane and at least a distance M from the hyperplane, we seek the largest M and associated parameters $\beta_0, \boldsymbol{\beta}$. The optimization problem can be simplified by replacing the constraints by

$$\frac{1}{\|\boldsymbol{\beta}\|} y_i \left(\beta_0 + \boldsymbol{\beta}^T\mathbf{x}_i\right) \geq M, \quad i = 1, 2, \ldots, p. \tag{3.16}$$

Using the property that the parameters can be rescaled, i.e. for any $\beta_0, \boldsymbol{\beta}$ satisfying the inequality (3.16), any positively scaled multiple satisfies it as well, we may arbitrarily set $\|\boldsymbol{\beta}\| = 1/M$. Thus the optimization problem (3.15) is equivalent to

$$\boxed{\begin{aligned} \min_{\beta_0, \boldsymbol{\beta}} \quad & \frac{1}{2}\|\boldsymbol{\beta}\|^2 \\ \text{Subject to} \quad & y_i(\beta_0 + \boldsymbol{\beta}^T\mathbf{x}_i) \geq 1, \quad i = 1, 2, \ldots, p. \end{aligned}} \tag{3.17}$$

Note that in this setting $M = 1/\|\boldsymbol{\beta}\|$ is the distance between the optimal separating hyperplane and the closest point(s) from either class. We choose the parameters $\beta_0, \boldsymbol{\beta}$ to maximize this distance. We have arrived at a convex optimization problem with linear inequality constraint. The Lagrange primal function to be minimized w.r.t. $\beta_0, \boldsymbol{\beta}$ is

$$L_P = \frac{1}{2}\|\boldsymbol{\beta}\|^2 - \sum_{i=1}^{p} \lambda_i \left(y_i \left(\beta_0 + \boldsymbol{\beta}^T\mathbf{x}_i\right) - 1\right). \tag{3.18}$$

Setting derivatives to zero yields

$$\boldsymbol{\beta} = \sum_{i=1}^{p} \lambda_i y_i \mathbf{x}_i, \tag{3.19}$$

$$0 = \sum_{i=1}^{p} \lambda_i y_i. \tag{3.20}$$

Substituting equations (3.19) and (3.20) in Equation (3.18) to obtain the corresponding Wolfe dual function

$$L_D = \sum_{i=1}^{n} \lambda_i - \frac{1}{2} \sum_{i=1}^{p} \sum_{k=1}^{p} \lambda_i \lambda_k y_i y_k \mathbf{x}_i^T \mathbf{x}_k. \tag{3.21}$$

Maximization of the Wolfe dual function is w.r.t. the Lagrange multipliers $\lambda_i, i = 1, 2, \ldots, p$ and subject to the constraints

$$\lambda_i \geq 0, \tag{3.22}$$

$$\sum_{i=1}^{p} \lambda_i y_i = 0. \tag{3.23}$$

The problem of maximizing L_D is a convex optimization problem to which standard mathematical software can be applied. Note that the solution must satisfy the KKT-conditions which for this particular problem is given by equations (3.19), (3.20), (3.22) and

$$\lambda_i \left(y_i \left(\beta_0 + \boldsymbol{\beta}^T \mathbf{x}_i \right) - 1 \right) = 0, \quad i = 1, 2, \ldots p. \tag{3.24}$$

The last condition implies that if $y_i \left(\beta_0 + \boldsymbol{\beta}^T \mathbf{x}_i \right) > 1$, then $\lambda_i = 0$. These are all observations in $\mathcal{T}$ that lies outside the margin. Conversely if $\lambda_i > 0$ then $y_i(\beta_0 + \boldsymbol{\beta}^T \mathbf{x}_i) = 1$ which is observations on the boundary to the margin. Recall condition (3.19)

$$\boldsymbol{\beta} = \sum_{i=1}^{p} \lambda_i y_i \mathbf{x}_i,$$

which implies that the solution vector $\boldsymbol{\beta}$ is determined only by the training observations that lies on the boundary of the margin, i.e. those observations closest to the separating hyperplane. These points are called *support vectors* and will be discussed in the following section.

3.5.2 Support vector classifier

The main problem with the Maximal margin classifier is the condition on data to be linearly separable. In many problem this is a condition that cannot be meet and the optimization problem (3.15) cannot be solved. Thus we now consider the case where the classes are not linearly separable, that is the classes overlap in the feature space $\mathcal{X}$. One approach to deal with such a problem is to still use a linear decision boundary but allow for some observations to be misclassified. That is, we still want to fit a hyperplane but allow for some points to be located on the wrong side of the margin. We can try to achieve this by adding a few constraints to the maximal margin classifier (3.15), consider the optimization problem

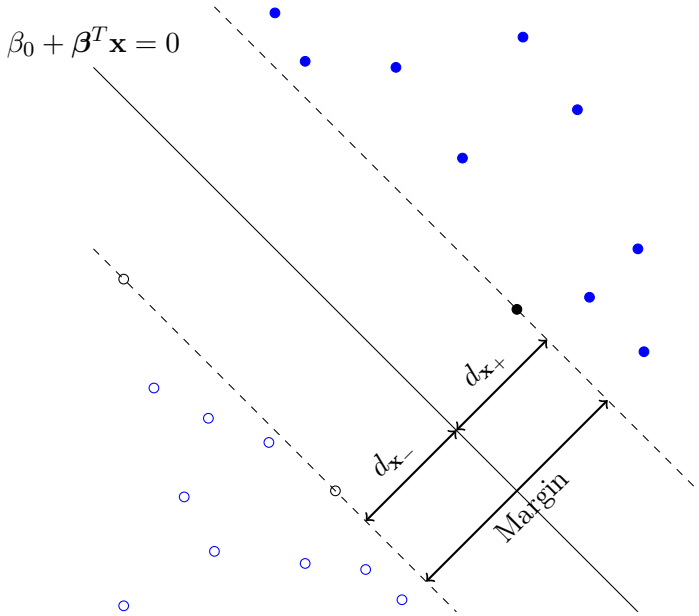

Figure 3.2: *The filled circles indicates observations of one class and the non-filled circles indicates observations of a different class. The solid black line is the optimal separating hyperplane* $\beta_0 + \boldsymbol{\beta}^T\mathbf{x} = 0$ *and the orthogonal distance between the dashed black lines indicates the margin.* $d_{\mathbf{x}_-}$ *and* $d_{\mathbf{x}_+}$ *indicates the distance from the optimal separating hyperplane to the closest training observations of respective class.*

$$\max_{\beta_0,\boldsymbol{\beta},\xi_1,\ldots,\xi_p} \quad M$$

$$\text{Subject to} \quad \begin{cases} \|\boldsymbol{\beta}\|^2 = 1 \\ y_i(\beta_0 + \boldsymbol{\beta}^T\mathbf{x}_i) \geq M(1-\xi_i), \quad i = 1,2,\ldots,p \\ \xi_i \geq 0, \quad i = 1,2,\ldots,p \\ \sum_{i=1}^{p} \xi_i < C \end{cases} \tag{3.25}$$

where $C > 0$ is a cost- or tuning parameter and ξ_i's are slack variables. It is the slack variables that allow for observations to be misclassified and the cost parameter regulates how much misclassification that is allowed. More precisely:

- ξ_i The proportional amount by which observation i is on the wrong side of the margin. $\xi = 0$ means that observation does not violate the margin, $0 < \xi < 1$ means that the observation is within the margin and $\xi > 1$ indicates that the observation is outside the margin on the wrong side, i.e. the observation is misclassified.
- C A bound for the sum $\sum_i \xi_i$, thus bounds the total proportional amount by which predictions falls on the wrong side of their margin.

Note that as misclassification occurs when $\xi_i > 1$, hence by bounding $\sum_i \xi_i$ at value C, bounds the total number of misclassified observation to at most $\lfloor C \rfloor$. Thus increasing C will result in a wider margin. The solution to the optimization problem (3.25) is known as the *Support vector classifier* [1] (p.419).

Similar to the maximal margin classifier we may rewrite the optimization problem slightly. Consider

$$\boxed{\begin{aligned} &\min_{\beta_0,\boldsymbol{\beta},\xi_1,\ldots,\xi_p} \quad \frac{1}{2}\|\boldsymbol{\beta}\|^2 + C\sum_{i=1}^{p}\xi_i \\ &\text{Subject to} \quad \begin{cases} y_i(\beta_0 + \boldsymbol{\beta}^T\mathbf{x}_i) \geq 1 - \xi_i, & i = 1, 2, \ldots, p \\ \xi_i \geq 0, & i = 1, 2, \ldots, p \end{cases} \end{aligned}} \tag{3.26}$$

which is equivalent to the optimization problem given by (3.25). Note that $C \to \infty$ yields the maximal margin classifier. The Lagrange primal function is

$$L_P = \frac{1}{2}\|\boldsymbol{\beta}\|^2 + C\sum_{i=1}^{p}\xi_i - \sum_{i=1}^{p}\lambda_i\big(y_i\left(\beta_0 + \boldsymbol{\beta}\mathbf{x}_i\right) - (1-\xi_i)\big) - \sum_{i=1}^{p}\mu_i\xi_i, \tag{3.27}$$

which is to be minimized w.r.t. parameters $\beta_0, \boldsymbol{\beta}$ and ξ_i's. Setting derivatives to zero yields

$$\boldsymbol{\beta} = \sum_{i=1}^{p}\lambda_i y_i \mathbf{x}_i, \tag{3.28}$$

$$0 = \sum_{i=1}^{p}\lambda_i y_i, \tag{3.29}$$

$$\lambda_i = C - \mu_i, \quad i = 1, 2, \ldots, p. \tag{3.30}$$

Moreover λ_i, μ_i and ξ_i are non-negative. By substituting equations (3.28)-(3.30) in Equation (3.27) the Lagrangian Wolfe dual objective function

$$L_D = \sum_{i=1}^{p}\lambda_i - \frac{1}{2}\sum_{i=1}^{p}\sum_{k=1}^{p}\lambda_i\lambda_k y_i y_k \mathbf{x}_i^T\mathbf{x}_k, \tag{3.31}$$

which is to be maximized subject to the constraints $0 \leq \lambda_i \leq C$ and $\sum_{i=1}^{p}\lambda_i y_i = 0$. In addition to equations (3.28)-(3.30) the KKT-conditions include

$$\lambda_i\big(y_i\left(\beta_0 + \boldsymbol{\beta}\mathbf{x}_i\right) - (1-\xi_i)\big) = 0, \tag{3.32}$$

$$\mu_i\xi_i = 0, \tag{3.33}$$

$$\big(y_i\left(\beta_0 + \boldsymbol{\beta}\mathbf{x}_i\right) - (1-\xi_i)\big) \geq 0, \tag{3.34}$$

for $i = 1, 2, \ldots, p$. The equations (3.28) - (3.34) uniquely characterize the solution to the primal and dual problem. Equation (3.28) entails that the solution for $\boldsymbol{\beta}$ has the form

$$\boldsymbol{\beta} = \sum_{i=1}^{p}\lambda_i y_i \mathbf{x}_i, \tag{3.35}$$

with non-zero coefficients λ_i for those observations where $\big(y_i\left(\beta_0 + \boldsymbol{\beta}\mathbf{x}_i\right) - (1-\xi_i)\big) = 0$, due to the constraint (3.32). The corresponding observations $\mathbf{x}_i$ for which $\big(y_i\left(\beta_0 + \boldsymbol{\beta}\mathbf{x}_i\right) - (1-\xi_i)\big) = 0$ is true are the support vectors. The Lagrangian multipliers are such that $0 < \lambda_i < C$ if observation $\mathbf{x}_i$ is a support vector on the margin and $\lambda_i = C$ if observation $\mathbf{x}_i$ lies inside the margin. This becomes clear by conditions (3.30) and (3.33) and the fact that $\xi_i = 0$ if $\mathbf{x}_i$ is on the edge of the margin and $\xi_i > 0$ if $\mathbf{x}_i$ is inside the margin.

Note that the resulting hyperplane still creates a linear decision boundary in feature space $\mathcal{X}$. There are of course problems where a linear decision boundary is not suitable even when allowing for some misclassification. Thus next we will look at ways to create non-linear decision boundaries.

3.5.3 Support vector machine

Building on the support vector classifier, possibly the easiest way of creating a non-linear decision boundary would be to enlarge the feature space. Consider for example a polynomial decision boundary of second degree, then we could let

$$\mathbf{x}' = (x_1, x_1^2, x_2, x_2^2, \ldots, x_p, x_p^2), \tag{3.36}$$

constitute the enlarged feature space (instead of $\mathbf{x} = (x_1, x_2, \ldots, x_p)$). With $\mathbf{x}'$ we can use the support vector classifier to compute a linear decision boundary in the enlarged feature space. In general linear boundaries in enlarged spaces achieve better training class separation and translates to non-linear decision boundaries in the original feature space $\mathcal{X}$.

There are however some issues with the above described approach, one of which is computational issues. Consider the general case where we have L basis functions $h_m(\cdot) : \mathbb{R}^p \to \mathbb{R}, m = 1, 2, \ldots, L$, and fit a support vector classifier using features

$$h(\mathbf{x}_i) = \big(h_1(\mathbf{x}_i), h_2(\mathbf{x}_i), \ldots, h_L(\mathbf{x}_i)\big), \quad i = 1, 2, \ldots, p. \tag{3.37}$$

The associated hyperplane is given by

$$f(\mathbf{x}) = \beta_0 + \boldsymbol{\beta}^T h(\mathbf{x}). \tag{3.38}$$

Using this approach computations are performed explicitly in the enlarged space, $\mathbb{R}^L$, and typically $L >> p$. For example the feature space spanned by $\mathbf{x}'$ yields $L = 2p$. There are countless ways to combine the features in $\mathcal{X}$ in order to enlarge the feature space and create more flexible decision boundaries. $\mathbf{x}'$ given by (3.36) is one simple example, one could choose to also include higher order polynomial terms, or use interaction terms, $x_i x_k, i \neq k$. Undoubtedly, unless we are careful the number of features could grow to a vast number making computations prohibitive. The *Support vector machine* is an extension of the support vector classifier that allows for the feature space to become very large (infinite even) whilst computations are kept efficient.

Consider the optimization problem associated with the support vector classifier (3.27), for some basis functions $h(\cdot) = \big(h_1(\cdot), h_2(\cdot), \ldots, h_L(\cdot)\big)$ the Lagrange Wolfe dual function takes the form

$$L_D = \sum_{i=1}^{p} \lambda_i - \frac{1}{2} \sum_{i=1}^{p} \sum_{k=1}^{p} \lambda_i \lambda_k y_i y_k \langle h(\mathbf{x}_i), h(\mathbf{x}_k) \rangle, \tag{3.39}$$

where $\langle \cdot, \cdot \rangle$ indicates the inner product. The solution function $f(\mathbf{x})$ takes the form

$$\begin{aligned} f(\mathbf{x}) &= \beta_0 + \boldsymbol{\beta}^T h(\mathbf{x}) \\ &= \left\{ \boldsymbol{\beta} = \sum_{i=1}^{p} \lambda_i y_i h(\mathbf{x}_i) \right\} \\ &= \beta_0 + \sum_{i=1}^{p} \lambda_i y_i \langle h(\mathbf{x}), h(\mathbf{x}_i) \rangle. \end{aligned} \tag{3.40}$$

Note that equations (3.39) and (3.40) only involves the basis functions through their inner product. In fact, we need not to specify the basis transforms $h(\mathbf{x})$ explicitly, but require only knowledge about the *kernel* function. Using kernels it is possible to produce non-linear decision boundaries by constructing a linear boundary in a large, transfromed version of the feature space. A kernel K is a function $K : \mathbb{R}^p \times \mathbb{R}^p \to \mathbb{R}$ such that for all $\mathbf{x}, \tilde{\mathbf{x}} \in \mathbb{R}^p$

$$K(\mathbf{x}, \tilde{\mathbf{x}}) = \langle h(\mathbf{x}), h(\tilde{\mathbf{x}}) \rangle. \tag{3.41}$$

The kernel should be a symmetric and positive (semi-) definite function. Some common choices of K are given in Table 3.1.

Table 3.1: *Example of kernel functions, where d is an integer, $\gamma > 0$ is a scale parameter and $\kappa_1, \kappa_2 \geq 0$*

Kernel	$K(\mathbf{x}, \tilde{\mathbf{x}})$
Polynomial of degree d	$(1 + \langle \mathbf{x}, \tilde{\mathbf{x}} \rangle)^d$
Radial basis	$\exp\left\{ -\gamma \left\| \mathbf{x} - \tilde{\mathbf{x}} \right\|^2 \right\}$
Hyperbolic Tangent	$\tanh(\kappa_1 \langle \mathbf{x}, \tilde{\mathbf{x}} \rangle + \kappa_2)$

We will not go into details of properties of different kernel functions, but for a specific choice the function characterizing the hyperplane associated with the support vector classifier takes the form

$$f(\mathbf{x}) = \beta_0 + \sum_{i=1}^{p} \lambda_i y_i K(\mathbf{x}, \mathbf{x}_i). \tag{3.42}$$

When the support vector classifier is combined with a non-linear kernel function the resulting classifier is known as a *Support vector machine* [4] (p.352). Thus a support vector machine is much like fitting an support vector classifier in an enlarged feature space. The difference lies in the fact that we need not to work explicitly in the enlarged space but can by computing $\binom{p}{2}$ values of $K(\mathbf{x}_i,\ \mathbf{x}_k), i \neq k$ determine the λ_i's.

Example: *Computation of basis functions for second degree polynomial kernel*
Consider a feature space with two inputs, $\mathbf{x} = (x_1, x_2)$ and a second degree polynomial kernel. Then

$$\begin{aligned} K(\mathbf{x}, \tilde{\mathbf{x}}) &= (1 + \langle \mathbf{x}, \tilde{\mathbf{x}} \rangle)^2 \\ &= (1 + x_1\tilde{x}_1 + x_2\tilde{x}_2)^2 \\ &= 1 + 2x_1\tilde{x}_1 + 2x_2\tilde{x}_2 + (x_1\tilde{x}_1)^2 + (x_2\tilde{x}_2)^2 + 2x_1\tilde{x}_1x_2\tilde{x}_2. \end{aligned} \tag{3.43}$$

This could be achieved by explicitly enlarging the feature space using $L = 6$ basis functions. If we choose

$$h(\mathbf{x}) = \left(1, \sqrt{2}x_1, \sqrt{2}x_2, x_1^2, x_2^2, \sqrt{2}x_1x_2\right), \tag{3.44}$$

and analogous for $h(\tilde{\mathbf{x}})$, then $K(\mathbf{x}, \tilde{\mathbf{x}}) = \langle h(\mathbf{x}), h(\tilde{\mathbf{x}}) \rangle)$ for all choices of $\mathbf{x}, \tilde{\mathbf{x}}$. Using the kernel function we include second degree terms of the features whilst still working in $\mathbb{R}^2$. This simple example highlights the power of kernels, by enlarging the feature space we would be working in $\mathbb{R}^{12}$.

As mentioned, a support vector machine is the result of combining the support vector classifier with a non-linear kernel. The role of the cost parameter C becomes clearer in the enlarged feature space. As before a large value of C will discourage large values of ξ_i's, i.e. an increase of C means that we are less tolerable of misclassified observations. Since perfect separation is often achievable in the enlarged space this lays the foundation for an overfit wiggly decision boundary in the original feature space. On the contrary a small value of C will encourage small value of $\|\boldsymbol{\beta}\|$, thus resulting in a more smooth boundary.

So far we have considered only separation of two classes, and it does not appear obvious how a hyperplane can be used for classification of more than two classes. Next follows a discussion of how to use the SVM in the setting of arbitrary many classes.

3.5.4 Multiclass Support Vector Machines

In a multiple class setting we have observations $\mathbf{x} \in \mathbb{R}^p$, of feature space $\mathcal{X}$ as before, but the output space $\mathcal{Y}$ now has K class labels, $\mathcal{Y} \in \{1, 2 \ldots, K\}$. There are numerous problems trying to use the concept of separation hyperplanes (upon which SMV are based) in multivariate classification. There has been several attempts to extend the support vector machine in order to find a strategy for managing arbitrary classes. We will briefly discuss two of the most popular approaches.

One-versus-one approach

The idea is to use the SVM classifier as we have all ready discussed, but in a clever way to allow for multivariate classification. The strategy is to compare two classes at a time fitting a SVM, in the end we assign a observation to the class it was most frequently classified as. This amounts to construct $\binom{K}{2}$ SVM's, one for each distinct pair of classes. Each test observation is classified $\binom{K}{2}$ times, class 1-vs-2, 1-vs-3, ..., 1-vs-K, 2-vs-1, 2-vs-3, etc. Then for each test observation $\mathbf{x}$ aggregate the *votes* for each class and assign $\mathbf{x}$ to the most popular class. In case of a tie between classes, choose one of the tied classes at random.

One-versus-rest approach

This approach utilizes the same basic principle as the one-versus-one approach, but now the strategy is to compare one of the K classes to all others, one at a time. Hence we fit K SVM's and for each of the K classes we compare it to the remaining $K - 1$. Corresponding to the kth class ($k \in [1, K]$) we obtain a function f_k associated with the decision boundary of the kth subproblem. Each time we encode the kth class $+1$ and the union of the remaining classes as -1. Then a test observation $\mathbf{x}$ is assigned to the class y according to

$$y = \underset{k \in \{1,2,\ldots,K\}}{\arg\max} f_k(\mathbf{x}). \tag{3.45}$$

The reasoning is that a large value of $f_k(\mathbf{x})$ indicates with some confidence that test observation $\mathbf{x}$ belongs to class k, rather that to any of the other $K - 1$ classes.

Which approach to choose is not obvious and which yields the best performance seems to be dependent on data [2] (p.391). One obvious benefit to the one-versus-rest approach over one-versus-one is the computational effort, for classification problems with even a moderate number of classes, say $K = 10$, one-versus-one approach requires 4.5 times as many fittings. With 100 classes one-versus-one approach requires almost 50 times (4950 in total) the number of fittings compared to one-versus-rest.

3.6 Neural Network and Deep learning

Neural Network (NN) is a branch of statistical learning originating form the fields of artificial intelligence and expert systems which sought after to answer questions such as: What makes the human brain such a formidable machine in processing cognitive thought? What is the nature of this thing called "intelligence"? And, how do humans solve problems? [2]

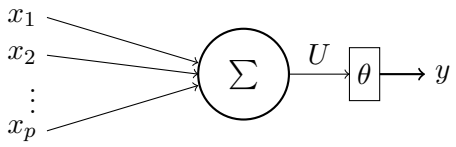

Figure 3.3: *McCulloch-Pitts neuron with input* $\mathbf{x} \in \mathbb{R}^p$*, output* y *and threshold* θ*.*

As an overly simplified model of the neuron activity in the brain, NN were originally designed to mimic brain activity. Now, NN's are treated more abstractly, as a network of highly interconnected nonlinear computing elements [2]. NN are well suited for problems of high-dimensionality and with large sample sizes. For example, problems of speech recognition, handwritten character recognition, face recognition, and robotics.

In order to introduce and understand the mechanics of a neural network some knowledge about how the human brain works is helpful as those ideas constitutes the foundation on which the theory of Neural Networks are built. For the purpose of this thesis however we limit ourself to the mathematical development that has lead to neural networks. The theory presented in this section is inspired by [1] (p.389-401), [2] (p.315-344). To get started we review the work of McCulloch and Pitts [7].

3.6.1 The McCulloch-Pitts neuron

The McCulloch-Pitts neuron takes an multidimensional (say p-dimensional) input vector $\mathbf{x} = (x_1, x_2, \ldots, x_p)^T$ and produces a binary output. Each component in the input vector takes a binary value, i.e. $x_i \in \{0, 1\},\ i = 1, 2, \ldots, p$. The sum of the input signal, $U = \sum_{i=1}^{p} x_i$, is compared to a threshold value θ to determine the output signal

$$y = \begin{cases} 1 & \text{if } U \geq \theta, \\ 0 & \text{if } U < \theta. \end{cases} \tag{3.46}$$

Note that as U can never be greater than p, $\theta > p$ yields a neuron that always returns $y = 0$, contrary $\theta = 0$ always yields $y = 1$. The McCulloch-Pitts neuron is given a graphical representation in Figure 3.3.

We may interpret the problem geometrically, the input space is an p-dimensional unit hypercube. Associated with each of the 2^p vertices of the hypercube is an output value (0 or 1). Given a value of θ the hyperplane $\sum_{i=1}^{p} x_i = \theta$ divides the hypercube into two half spaces where those vertices corresponding to $y = 1$ lies on one side of the hyperplane and those corresponding to $y = 0$ lies on the other side.

The design of the McCulloch-pitts neuron is well suited for computations of logistic functions taking p arguments and returns *TRUE* ($y = 1$) or *FALSE* ($y = 0$). For example the logical functions AND and OR are displayed in Figure 3.4. Note that if threshold $\theta = 0$ the output will always be $y = 1$ and similarly if $\theta > p$ the output will always be $y = 0$ independent of input $\mathbf{x}$. The AND- and OR functions are the basis for logical functions and all other logical functions can be computed by creating networks of several layers of McCulloch-Pitts neurons.

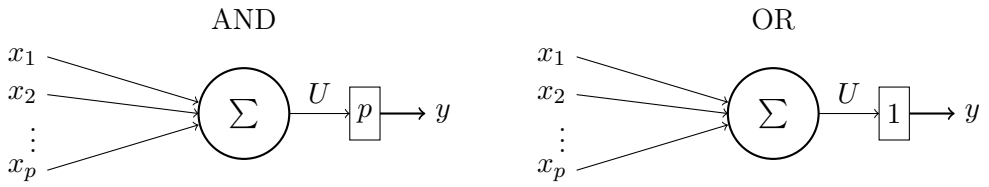

Figure 3.4: *McCulloch-Pitts neuron for the logical functions AND on the left, and OR on the right. The functions take p binary input arguments and returns output y. The AND-function uses threshold $\theta = p$, hence all inputs must equal one for the function to return TRUE. The OR-function uses threshold $\theta = 1$ thus if any input argument is one the function will return TRUE.*

While the McCulloch-Pitts neuron are easy to understand, and serves well as an introduction to Neural networks, there are other similar ideas which are more flexible and thus more suitable as to create decision boundaries.

3.6.2 Perceptron revised

In section 3.4 we introduced the perceptron algorithm, now we will see that it can be given an interpretation very similar to the McCully-Pitts neuron. A perceptron is essentially a McCulloch-Pitts neuron but each input is multiplied with a real-valued *weight* β_i, $i = 1, 2, \ldots, p$. The components of the input vector is no longer restricted to binary values but can now take any real value. The weights adds lots of flexibility to the model, a small magnitude of a connection weight β_i makes the model less sensitive to input x_i and likewise a large magnitude yields a model very sensitive to the corresponding input. The model is sometimes refereed to as the single layer perceptron and is illustrated in Figure 3.5. A weighted sum $U = \sum_{i=1}^{p} \beta_i x_i$ is computed and compared to threshold θ. The output y is determined precisely as for the McCulloch-Pitts neuron given by Equation 3.46. Note by the construction of the McCulloch-Pitts neuron U was bounded by the interval $[0, p]$, but now there is no bounds to U since both input components and the corresponding connection weights can be arbitrary large and take positive or negative values.

It is common to convert the threshold θ to 0 by introducing a *bias element*, $\beta_0 = -\theta$ and comparing $\beta_0 + U$ to 0. Note that

$$\begin{aligned} 0 &= \beta_0 + U \\ &= \beta_0 + \sum_{i=1}^{p} \beta_i x_i \\ &= \beta_0 + \boldsymbol{\beta}^T \mathbf{x}, \end{aligned} \tag{3.47}$$

where $\boldsymbol{\beta} = (\beta_1, \beta_2, \ldots, \beta_p)^T$, constitutes a hyperplane. The output signal y is determined based on which side of the hyperplane an observation $\mathbf{x}$ lies. If we slightly change the notations so that

$$U = \sum_{i=0}^{p} \beta_i x_i, \tag{3.48}$$

i.e. we include β_0 in U and set $x_0 = 1$, then the output signal is determined as

$$y = \begin{cases} 1 & \text{if } U \geq 0, \\ 0 & \text{if } U < 0. \end{cases} \tag{3.49}$$

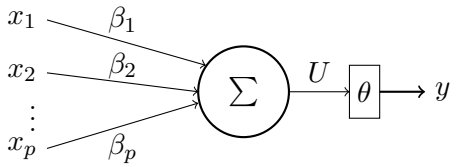

Figure 3.5: *Single layer perceptron with p input arguments and corresponding connection weights β_i, $i = 1, 2, \ldots, p$. The weighted sum is compared to threshold θ to give binary output y.*

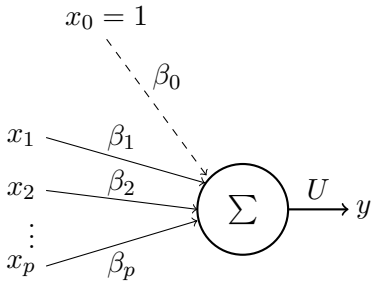

Figure 3.6: *An equivalent perceptron to the single layer perceptron in Figure 3.5, here $\beta_0 = -\theta$ and $x_0 = 1$.*

Note that we still require the classes to be linearly separable as discussed in section 3.4. The similarities of the Perceptron and McCulloch-pitts neuron are many, the finally discussed version of the single layer perceptron is displayed in Figure 3.6.

3.6.3 Neural networks

The idea of *layering* perceptrons was suggested in 1969 by Minsky and Papert [8], however it took almost 2 decades, until mid 1980's when computational power became widely available, before these ideas were adopted by the research community. Their suggestion was that the limitations of the perceptron could be overcome by layering the perceptrons and applying nonlinear transformations prior to combining the transformed weighted inputs. These ideas could be regarded as the core of what has lead to neural networks as we consider them today.

Neural networks are multi-stage regression or classification models that takes p input variables and the central idea is to extract linear combinations of the input variables as *derived features*, and then model the target (output) variables as non-linear combinations of these features. A typical representation of a single hidden layer neural network is given by Figure 3.7. In the typical regression setting there is one continuous output variable y ($K = 1$) while in a classification problem of K classes there is K output variables $y_1, y_2, \ldots, y_K$.

The derived features Z_m, $m = 1, 2, \ldots, M$ are created from linear combinations of the inputs x_i, $i = 1, 2, \ldots, p$, and the output variables y_k, $k = 1, 2, \ldots, K$ are modeled as functions of linear combinations of Z_m's. Let $\beta_{i,m}$ be the connection weight between input x_i and feature Z_m and $\alpha_{m,k}$ be the connection weight between derived feature Z_m and output y_k then

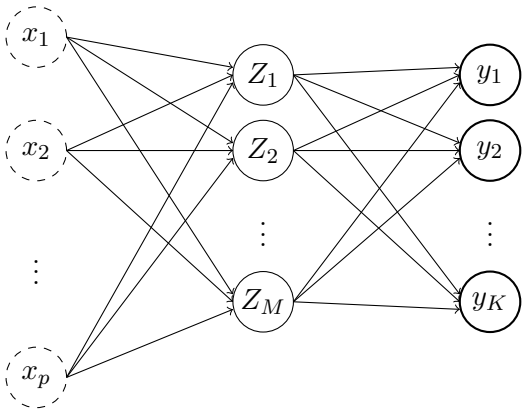

Figure 3.7: *Graphical representation of a fully connected, neural network with p input nodes, one hidden layer with M nodes and an output layer with K neurons.*

$$Z_m = \sigma_m\left(\beta_{0,m} + \boldsymbol{\beta}_m^T \mathbf{x}\right), \quad m = 1, 2, \ldots, M, \tag{3.50}$$

$$T_k = \alpha_{0,k} + \alpha_k^T Z, \quad k = 1, 2, \ldots, K, \tag{3.51}$$

$$f_k(\mathbf{x}) = g_k(T), \quad k = 1, 2, \ldots, K, \tag{3.52}$$

where $\beta_{0,m}$ and $\alpha_{0,k}$ are bias terms, $Z = (Z_1, Z_2, \ldots, Z_M)$, $T = (T_1, T_2, \ldots, T_K)$. Combining these equations yields an expression for the kth output node

$$y_k = f_k(\mathbf{x}) + \epsilon_k, \tag{3.53}$$

where ϵ_k, $k = 1, 2, \ldots, K$ are error terms and

$$f_k(\mathbf{x}) = g_k\left(\alpha_{0,k} + \sum_{m=1}^{M} \alpha_{m,k}\sigma_m\left(\beta_{0,m} + \boldsymbol{\beta}_m^T \mathbf{x}\right)\right), \quad k = 1, 2, \ldots, K. \tag{3.54}$$

The functions $g(\cdot)$ and $\sigma(\cdot)$ are known as *activation functions*. In Table 3.2 some examples of activation functions are given. For $\sigma(\cdot)$ the Logistic and Hyperbolic tangent activation functions are frequently used. The function $g(\cdot)$ allows for a final transformation, for regression it is usually taken to be the identity function, and for classification the softmax function

$$g_k(T) = \frac{e^{T_k}}{\sum_{l=1}^{K} e^{T_l}}, \quad k = 1, 2, \ldots, K, \tag{3.55}$$

is commonly used. In the above representations the activation functions are given indices to indicate that different activation functions can be used at different nodes. It is however common to use the same activation function for all nodes. If both $\sigma(\cdot)$ and $g(\cdot)$ are linear, then Equation (3.54) are just a linear combination of the inputs.

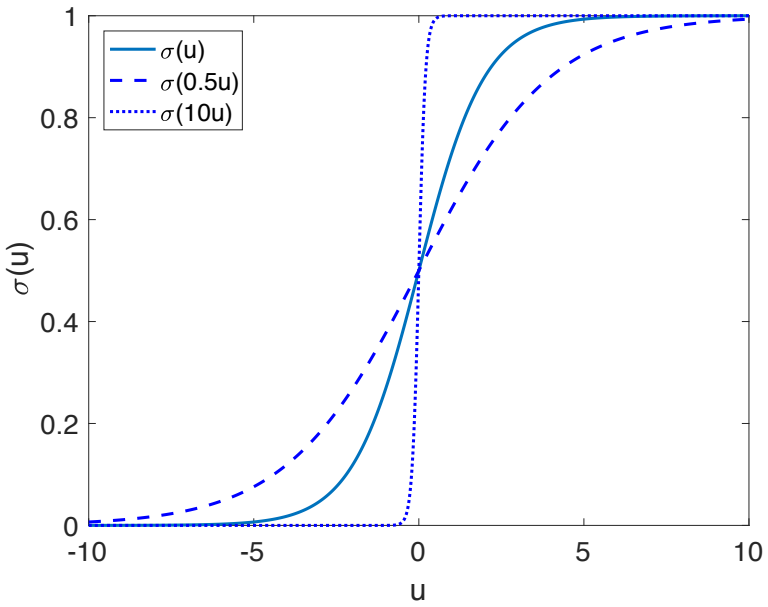

Figure 3.8: *Plot of logistic activation function (sigmoid)* $\sigma(u) = 1/(1 + e^{-u})$*, the solid blue line. Also included are* $\sigma(tu)$*, for* $t = \frac{1}{2}$*, dashed line and* $t = 10$*, dotted line. All lines shows the characteristic S-shape of a sigmoidal function.*

Table 3.2: *Examples of activation functions*

Activation function	$\sigma(u)$	Range of values
Identity, linear	u	$\mathbb{R}$
Step (threshold)	$\mathbb{1}_{u\geq 0}$	$\{0, 1\}$
Gaussian radial basis	$\frac{1}{\sqrt{2\pi}} e^{-u^2/2}$	$\mathbb{R}$
Logistic	$(1 + e^{-u})^{-1}$	$(0, 1)$
Hyperbolic tangent	$(e^u - e^{-u})/(e^u + e^{-u})$	$(-1, 1)$

The Logistic, Hyperbolic tangent and softmax functions are called *Sigmoidal* functions and have a characteristic S-shape. By using a non-linear transform we greatly enlarge the class of linear models, thus sigmoidal functions play an important role in network design. See Figure 3.8 for example of a sigmoidal function, one can see the flexibility of a sigmoid, e.g. for u close to zero, $\sigma(u)$ is close to linear.

The derived features Z_m computed in the middle of the network are parts or the *hidden layer* and in general we can have any number of hidden layers in our model. They are called hidden since the values of Z_m are not directly observed. Due to the fact that one can choose the number of hidden layers as well as the number of neurons in each hidden layer there are endless possibilities when designing a neural network. This makes it hard to tune a network in order to yield the best performance. For many applications one hidden layer is enough and as a rule of thumb one can start of with 2/3 of the input size as number of neurons. Deep learning can be described as layers of nonlinear processing units for feature extraction and transformation where each successive layer uses the output from the previous layer as input [9]. Hence, neural networks with more than one hidden layer are sometimes refered to as deep learning [3].

3.6.4 Fitting Neural networks

In this section we will have a look at how to fit or train a neural network. This includes the *back-propagation* algorithm.

Training a neural network amounts to determine the unknown connection weights. The aim is to determine the parameters in such a way that the model fits training data well. Consider a fully connected, single hidden layer network, like the one in Figure 3.7, and let Θ denote the complete set of weights;

$$\Theta = \{\alpha_{0,k}, \boldsymbol{\alpha}_k, \beta_{0,m}, \boldsymbol{\beta}_m;\ k = 1, 2, \ldots, K,\ m = 1, 2, \ldots, M\}, \tag{3.56}$$

where $\boldsymbol{\alpha}_k = (\alpha_{1,k}, \alpha_{2,k}, \ldots, \alpha_{M,k})^T$ and $\boldsymbol{\beta}_m = (\beta_{1,m}, \beta_{2,m}, \ldots, \beta_{p,m})^T$. In total there are $M(p+1) + K(M+1)$ parameters. Moreover we consider N p-dimensional input vectors

$$\mathbf{x}_n = (x_{1,n}, x_{2,n}, \ldots, x_{p,n})^T, \quad n = 1, 2, \ldots, N. \tag{3.57}$$

For regression, sum-of-squared errors are used as error function:

$$R(\Theta) = \sum_{k=1}^{K} \sum_{n=1}^{N} \big(y_{n,k} - f_k(\mathbf{x}_n)\big)^2, \tag{3.58}$$

and for classification one can use either squared error or cross-entropy

$$R(\Theta) = -\sum_{n=1}^{N} \sum_{k=1}^{K} y_{n,k} \log\big(f_k(\mathbf{x}_n)\big) \tag{3.59}$$

as measure of fit. For a K-class classification problem, each output node models the probability that observation $\mathbf{x}$ belongs to class k, $k = 1, 2, \ldots, K$. The corresponding classifier assigns new observations to the most probable class;

$$G(\mathbf{x}) = \arg\max_k\ f_k(\mathbf{x}), \quad k = 1, 2, \ldots, K \tag{3.60}$$

where $f_k(\mathbf{x})$ is given by equation (3.54). The calculations in the hidden units are equivalent to a linear logistic regression model if the softmax fuction is used for activation function and cross-entropy is chosen as error function, then all parameters are estimated by maximum likelihood.

Usually we do not search for the global minimizer of $R(\Theta)$, as this would likely yield an overfit solution. Hence there is need for some sort of stopping criteria and most common is to either introduce a penalty term or by early stopping. The generic approach to minimizing $R(\Theta)$ is by gradient descent, usually referred to as *back-propagation* [10] in this setting. The algorithm computes the first derivatives of the error function with respect to the network weights $\{\beta_{i,m}\}$ (connecting input node i and hidden node m) and $\{\alpha_{m,k}\}$ (connecting the mth hidden node to output node k). These derivatives are then used to estimate the weights by minimizing the error function through an iterative gradient descent method. In Figure 3.9 a simplified diagram indicates the weights in a single hidden layer network.

Now, for the nth input vector let $\mathbf{Z}_n = (Z_{1,n}, Z_{2,n}, \ldots, Z_{M,n})$ and

$$U_{m,n} = \beta_{0,m} + \boldsymbol{\beta}_m^T \mathbf{x}_n, \quad m = 1, 2, \ldots, M \tag{3.61}$$

$$Z_{m,n} = \sigma_m\left(U_{m,n}\right), \quad m = 1, 2, \ldots, M, \tag{3.62}$$

$$T_{k,n} = \alpha_{0,k} + \boldsymbol{\alpha}_k^T \mathbf{Z}_n, \quad k = 1, 2, \ldots, K, \tag{3.63}$$

$$f_k(\mathbf{x}_n) = g_k(T_{k,n}), \quad k = 1, 2, \ldots, K. \tag{3.64}$$

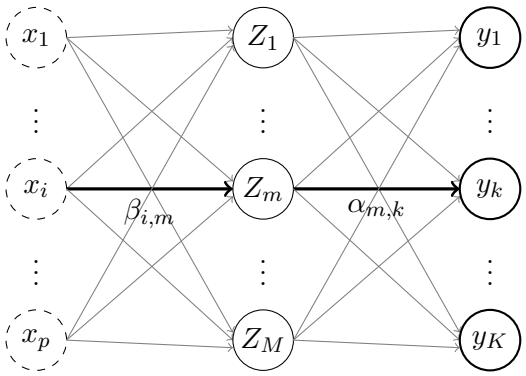

Figure 3.9: *Graphical representation of a fully connected, neural network with p input nodes, one hidden layer with M nodes and an output layer with K neurons. In the diagram the weights $\beta_{i,m}$ connecting input node i to hidden node m and $\alpha_{m,k}$ connecting hidden node*

Moreover consider

$$\begin{aligned} R(\Theta) &= \sum_{n=1}^{N}\sum_{k=1}^{K}(y_{k,n} - f_k(\mathbf{x}_n))^2 \\ &= \sum_{n=1}^{N}\sum_{k=1}^{K}(e_{k,n})^2 \\ &= \sum_{n=1}^{N} R_n, \end{aligned} \tag{3.65}$$

These notations will simplify computations of the derivatives. The back-propagation algorithm is an iterative process. Let

$$\begin{aligned} \boldsymbol{\alpha}^{(r)} &= (\boldsymbol{\alpha}_1^{(r)T}, \boldsymbol{\alpha}_2^{(r)T}, \ldots, \boldsymbol{\alpha}_K^{(r)T})^T \\ &= (\alpha_{1,1}^{(r)}, \alpha_{2,1}^{(r)}, \ldots, \alpha_{M,1}^{(r)}, \alpha_{1,2}^{(r)}, \ldots, \alpha_{M,K}^{(r)})^T, \end{aligned} \tag{3.66}$$

be the $M \cdot K$ vector of weights connecting the hidden layer to the output layer at the rth iteration. Similarly let

$$\begin{aligned} \boldsymbol{\beta}^{(r)} &= (\boldsymbol{\beta}_1^{(r)T}, \boldsymbol{\beta}_2^{(r)T}, \ldots, \boldsymbol{\beta}_M^{(r)T})^T \\ &= (\beta_{1,1}^{(r)}, \beta_{2,1}^{(r)}, \ldots, \beta_{p,1}^{(r)}, \beta_{1,2}^{(r)}, \ldots, \alpha_{p,M}^{(r)})^T, \end{aligned} \tag{3.67}$$

be the $p \cdot M$ vector of weights connecting the input layer to the hidden layer at the rth iteration. Under the assumption that the activation functions $g_k(\cdot)$ and $\sigma_m(\cdot)$ are differentiable the gradient can be computed using the chain rule for differentiation:

$$\begin{aligned}\frac{\partial R_n}{\partial \alpha_{m,k}^{(r)}} &= \frac{\partial R_n}{\partial e_{k,n}} \frac{\partial e_{k,n}}{\partial f_k(\mathbf{x}_n)} \frac{\partial f_k(\mathbf{x}_n)}{\partial T_{k,n}} \frac{\partial T_{k,n}}{\partial \alpha_{m,k}^{(r)}} \\ &= -2e_{k,n} g_k'(T_{k,n}) Z_{m,n} \\ &= -2\big(y_{k,n} - f_k(\mathbf{x}_n)\big) g_k'\big(\alpha_{0,k}^{(r)} + \boldsymbol{\alpha}_k^{(r)T} \mathbf{Z}_n\big) Z_{m,n}, \end{aligned} \tag{3.68}$$

$$\begin{aligned}\frac{\partial R_n}{\partial \beta_{i,m}^{(r)}} &= \frac{\partial R_n}{\partial Z_{m,n}} \frac{\partial Z_{m,n}}{\partial U_{m,n}} \frac{\partial U_{m,n}}{\partial \beta_{i,m}^{(r)}} \\ &= -2 \sum_{k=1}^{K} e_{k,n} g_k'\big(T_{k,n}\big) \alpha_{m,k}^{(r)} \sigma_m'\big(U_{m,n}\big) x_{i,n} \\ &= -2 \sum_{k=1}^{K} \big(y_{k,n} - f_k(\mathbf{x}_n)\big) g_k'\big(\alpha_{0,k}^{(r)} + \boldsymbol{\alpha}_k^{(r)T} \mathbf{Z}_n\big) \alpha_{m,k}^{(r)} \sigma_m'\big(\beta_{0,m}^{(r)} + \boldsymbol{\beta}_m^{(r)T} \mathbf{x}_n\big) x_{i,n}, \end{aligned} \tag{3.69}$$

where we have used that

$$\begin{aligned}\frac{\partial R_n}{\partial Z_{m,n}} &= 2 \sum_{k=1}^{K} e_{k,n} \frac{\partial e_{k,n}}{\partial Z_{m,n}} \\ &= 2 \sum_{k=1}^{K} e_{k,n} \frac{\partial e_{k,n}}{\partial T_{k,n}} \frac{\partial T_{k,n}}{\partial Z_{m,n}} \\ &= -2 \sum_{k=1}^{K} e_{k,n} g_k'\big(T_{k,n}\big) \alpha_{m,k}^{(r)}. \end{aligned} \tag{3.70}$$

The gradient descent update at the $(r+1)$st iteration takes the form

$$\alpha_{m,k}^{(r+1)} = \alpha_{m,k}^{(r)} - \eta_r \sum_{n=1}^{N} \frac{\partial R_n}{\partial \alpha_{m,k}^{(r)}}, \tag{3.71}$$

$$\beta_{i,m}^{(r+1)} = \beta_{i,m}^{(r)} - \eta_r \sum_{n=1}^{N} \frac{\partial R_n}{\partial \beta_{i,m}^{(r)}}, \tag{3.72}$$

where η_r is the learning rate parameter that specifies how large each step should be in the iterative process. Define $\delta_{k,n}$ and $s_{m,n}$ such that equations (3.68) and (3.69) can be expressed

$$\frac{\partial R_n}{\partial \alpha_{m,k}^{(r)}} = \delta_{n,k} Z_{m,n}, \tag{3.73}$$

$$\frac{\partial R_n}{\partial \beta_{i,m}^{(r)}} = s_{m,n} x_{i,n}. \tag{3.74}$$

That is let

$$\delta_{k,n} = -2\big(y_{k,n} - f_k(\mathbf{x}_n)\big) g_k'\big(\alpha_{0,k}^{(r)} + \boldsymbol{\alpha}_k^{(r)T} \mathbf{Z}_n\big), \tag{3.75}$$

$$s_{m,n} = \sigma_m'\big(\beta_{0,m}^{(r)} + \boldsymbol{\beta}_m^{(r)T} \mathbf{x}_n\big) \sum_{k=1}^{K} \alpha_{m,k} \delta_{k,n}. \tag{3.76}$$

We may think of the quantities $\delta_{k,n}$ and $s_{m,n}$ as *errors* from the current model at the hidden and output layer units respectively. Combining equations (3.71) and (3.72) with (3.73) and (3.74) one obtains

$$\alpha_{m,k}^{(r+1)} = \alpha_{m,k}^{(r)} - \eta_r \sum_{n=1}^{N} \delta_{k,n} Z_{m,n}, \tag{3.77}$$

$$\beta_{i,m}^{(r+1)} = \beta_{i,m}^{(r)} - \eta_r \sum_{n=1}^{N} s_{m,n} x_{i,n}, \tag{3.78}$$

which defines the back propagation algorithm. It is a two stage algorithm:

Forward pass is the first stage in which the current weights are fixed and the predicted values $\hat{f}_k(\mathbf{x}_n)$ are computed from equation (3.76).

Backward pass is the second stage. Here the errors $\delta_{k,n}$ are computed and then back propagated via (3.76) to gives the errors $s_{m,n}$.

This yields estimates of the gradient necessary to update the parameters in the model using (3.77) and (3.78). Upon start all weights have to be assigned initial values. The back propagation algorithm has also been known as the *delta rule* [11]. Above we have derived the algorithm for the sum of squares error, using cross entropy yields similar update rules.

In the updates (3.77) and (3.78) the parameters are summed over all the training observations like a *batch*-learning. One could also choose *online*-learning, and update the gradient based on each training observation, cycling through the training cases many times. In the case of online-learning the sums of the update equation are replaced with a single summand. Online training allows networks to manage very large training sets, and also to update weights as new observations are introduced.

In general the model is overparametrized and the problem is a nonconvex optimization problem which usually is hard to solve. The problem is likely to become unstable unless one is careful while implementing. Next some important considerations are briefly discussed.

Stopping and convergence

As mentioned, usually we would like to stop the iterative process prior to the global minimum of $R(\Theta)$ to avoid overfitting. Moreover there is no proof that the back propagation algorithm always converges and experience has shown that the algorithm is a slow learner, there may exist many local minimums and that convergence can be hard to achieve in practice. Common approaches to decide on a stopping criteria include;

Early stopping - In early development of neural networks an early stopping rule were used to avoid overfitting (whether this was by design or accident is still debated). By this approach the training of the network is stopped well before approaching the global minimum. Since the network initially is a highly regularized linear model (asuming that the initial weights are small), this has the effect of shrinking the final model towards a linear model. More resent development suggest to use a *validation* set, discussed in section 3.1, to determine an appropriate time of stopping, since we expect the validation error to increase as the global minimum is approached.

Weight decay - This is a more explicit approach of regularization in which a penalty term is added to the error function $R(\Theta) + \lambda p(\Theta)$ where $\lambda \geq 0$ is a tuning parameter and

$$p(\Theta) = \sum_{m,k} \alpha_{m,k}^2 + \sum_{i,m} \beta_{i,m}^2. \tag{3.79}$$

Large value of λ has the effect of shrinking the parameters towards zero. In the back propagation algorithm the effect of $p(\Theta)$ is the additional terms $2\alpha_{m,k}$ and $2\beta_{i,m}$ to the gradient, equation (3.71) and (3.72) respectively. An alternative form of penalty is given by

$$p(\Theta) = \sum_{m,k} \frac{\alpha_{m,k}^2}{1+\alpha_{m,k}^2} + \sum_{i,m} \frac{\beta_{i,m}^2}{1+\beta_{i,m}^2}, \tag{3.80}$$

and is known as *weight elimination* penalty. This has the effect of shrinking smaller weights more than (3.79).

Starting values

In a setting where the error function $R(\Theta)$ is a nonconvex with multiple minima, the solution is likely to depend on the staring values of the weights, hence it is usually a good idea to try different starting values. Starting with weights randomly generated near zero is usually a good idea. Small values of the parameters means that we are operating in the linear part of the sigmoid see Figure 3.8, hence resulting in an approximately linear model. As the weights are increased the model becomes non-linear, ideally this method comes with great flexibility as nonlinearities can be introduce locally at nodes in the model where needed. Using start values exactly equal to zero will result in an algorithm that never moves and too large starting values usually yields poor solutions.

Network architecture

One of the main problems in network design is to decide upon the number of hidden layers and corresponding nodes. The design of the network might have a drastic effect on its performance and it is far from obvious how to design a network in an optimal way, or in a good way for that matter. For many applications one hidden layer is sufficient and as a rule of thumb a good starting value for the number of neurons is $2/3$ of the input size. Choosing too few parameters it is likely that the model will not have enough flexibility to capture non-linearities in the data. With too many parameters in the model there is always a risk of overfitting, and computations can become very expensive. In theory, performance could still be good using too many parameters since the extra weights could be shrunken towards zero if appropriate regularization is used. There are suggestion to use cross-validation to choose the number of nodes in a hidden layer [2] (p.343). However, depending on $R(\Theta)$ this might not yield conclusive results. One could instead apply cross-validation to choose the parameters for regularization [1] (p.400). In many application of NN the network architecture is either decided from the context of the problem or by trial and error.

Scaling of the input variables

Input variables often comes with a variety of different units and different scales which may affect the relative contribution of each input to the resulting analysis. Hence it is usually a good idea to scale the input variables. There are different approaches to this but the most common one is to standardize each input to have zero meen and unit standard deviation [1] (p.399-400). This will also assure that all parameters are treated equally in the regularization process.

Chapter 4

Method and Implementation

This chapter explains the performance measures used in data evaluation. The implementation of two methods, Neural Networks and Support Vector Machines, are described.

4.1 Performance Evaluation

When evaluating the performance of the learning algorithm we need a benchmark to which it can be compared. We must also have a measure of the error of the learning algorithm. In this section we will have a look at just that.

4.1.1 Empirical error

The error rate of a classifier will be evaluated as the empirical error rate. This is simply the number of missclassified observations divided by the total number of observations in the test set. Let g be a classifier and consider n observations x_i with corresponding class labels y_i, $i = 1, 2, \ldots, n$. Then the empirical error or test error of the classifier g is

$$err(g) = \frac{1}{n} \sum_{i=1}^{n} \mathbb{1}_{g(x_i) \neq y_i}. \tag{4.1}$$

The overall performance or prediction accuracy of a multi-label classifier is estimated by $1 - err(g)$.

4.1.2 Confusion Matrix

In a classification problem with a moderate number of classes the performance of a classifier can easily be evaluated using a confusion matrix, also known as an error matrix [19]. A confusion matrix is a table layout that allows for visualization of the performance of a classifier. Each row in the matrix represents a predicted class and the columns represents the true class of an observation. In Table 4.1 there is an sample of an confusion matrix. A element ω_{ij} of the matrix indicates the number of observations with true class j that is predicted to class i. If $i = j$ the prediction is correct and the element is located at the diagonal of the matrix. If $i \neq j$ then ω_{ij} indicates the number of misclassified observations with true class j that have been predicted as class i. Hence all ω_{ij} indicating misclassification are located outside the diagonal. Ideally then a classifier would produce a diagonal matrix which would indicate that all observations are correctly classified.

A confusion matrix can be used to quickly get an idea of how a classifier performs. Not only can one by examine the diagonal get a good idea of the overall performance, but by examine

Table 4.1: *An illustration of a confusion matrix. In the matrix column k corresponds to observations which belongs to class k. The rows of the matrix corresponds to the predicted classes of the observations. Each element $\omega_{k,k}$ that lies on the diagonal corresponds to the number of correct classifications to class k. Each element $\omega_{i,j}$, $i \neq j$ lies outside the diagonal and corresponds to the number of misclassified observations predicted to class i with true class j.*

		True class			
		1	2	...	K
Predicted class	1	ω_{11}	ω_{12}	...	ω_{1K}
	2	ω_{21}	ω_{22}		ω_{2K}
	$\vdots$	$\vdots$		$\ddots$	$\vdots$
	K	ω_{K1}	ω_{K2}	...	ω_{KK}

elements outside the diagonal one can see if observations are more often misclassified to some classes than others. Similarly one can see if observations from a certain class are more often erroneously predicted than others. However as the number of classes becomes large a confusion matrix becomes somewhat problematic to view.

Based on a confusion matrix one can perform a more in depth evaluation of how the classifier performs on each class. Define *within class prediction accuracy*, p_k of class k, as the portion of correctly classified observations from class k:

$$p_k = \frac{\omega_{k,k}}{\sum_{i=1}^{K} \omega_{i,k}}, \quad k = 1, 2, \ldots, K. \tag{4.2}$$

The within class prediction accuracy is a useful measure to see how well a classifier can predict observations from a certain class. Another interesting aspect is if misclassified observations are predicted more often to certain classes. Such a measure could also be obtained from a confusion matrix, define *class prediction error* ε_k as the portion of observations incorrectly classified as class k:

$$\varepsilon_k = \frac{\sum_{j \neq k} \omega_{k,j}}{\sum_{j=1}^{K} \omega_{k,j}}, \quad k = 1, 2, \ldots, K. \tag{4.3}$$

The purpose of a confusion matrix in this thesis is rather to illustrate how the results of a classifier could be evaluated, than to perform in depth analysis of every classifier fitted.

4.1.3 Random Guess

Often in binary classification setting with two classes random guess is used as a good measure of performance. Then a random guess would have an prediction accuracy of $\frac{1}{2}$ on average. Extending this to a multi class setting with K classes a random guess would on average have an prediction accuracy of $\frac{1}{K}$. If however the number of observations to each class are unbalanced, i.e. some classes have more observations than others the theoretical prediction accuracy might change. Using basic probability theory we can compute the theoretical prediction accuracy.

Consider K classes with corresponding *class probability* π_k, $k = 1, 2, \ldots, K$. The class probability is simply the number of observations to class k divided by the total number of observations. The theoretical prediction accuracy of a random guess would be

$$\text{prediction accuracy} = \sum_{k=1}^{K} \pi_k P(\text{guess class } k). \tag{4.4}$$

Assuming that we will guess using the same ratio as the class probabilities we would obtain

$$\text{prediction accuracy} = \sum_{k=1}^{K} \pi_k^2. \tag{4.5}$$

Consider an example with $K = 3$ classes and $\pi_1 = 2/3$, $\pi_2 = \pi_3 = 1/6$. Then the theoretical prediction accuracy using (4.5) is $(2/3)^2+(1/6)^2+(1/6)^2 = 0.5$. Note that if all class probabilities are equal then the theoretical prediction accuracy is $1/K$ which is in line which our intuition.

4.1.4 Naive Classifier

If there is imbalance in the number of observations among classes and the class probabilities are accessible there is an easy strategy by which there is a good chance to beat a random guess. Consider the example above, if we would guess at random without any information on the class probabilities there is no reason to assume that we would guess using the same ratio as the class probabilities but rather 1/3 to each class. As the class probabilities sums to one the theoretical prediction accuracy would be 1/3. As we saw above if we assume that we guess with the same ratio as the class probabilities the theoretical prediction accuracy were 1/2. But if we assume that we guess using the radio of the class probabilities, they must be known and in such a setting we could beat a random guess by always classify to the class with largest class probability. In the example above this would mean that the theoretical prediction accuracy would be 2/3, which is more accurate than any of the others.

Note that there exist a classifier called Naive Bayesian classifier, or just Bayesian classifier, which assigns each observation to the most likely class, given its predictor values [4] (p.37-39). In the context of this thesis however the naive classifier does not refer to the Bayesian classifier but rather the naive approach of always guessing to the class with largest estimated class probability.

Using the available data one can compute the class probability that a observation belongs to one class. The estimated class probability $\hat{\pi}_k$ of class k is given by

$$\hat{\pi}_k = \frac{1}{n} \sum_{i=1}^{n} \mathbb{1}_{y_i=k}, \quad k = 1, 2, \ldots, K, \tag{4.6}$$

where n is the number of available observations. The corresponding naive classifier would classify all observations $\mathbf{x}$ to the class with highest class probability

$$g(\mathbf{x}) = \arg\max_k \hat{\pi}_k, \quad k = 1, 2, \ldots, K. \tag{4.7}$$

Note that the classifier $g(\mathbf{x})$ is independent of the observations. Also note that the number of observations that belongs to each class should be obtained from training data as the classes in test data must remain unseen by a classifier. Since the data set is divided into training and test set at random, using training data to compute class probabilities should give a good idea of which class that contains most observations. As seen in section 2 the problem of this thesis comes with unbalanced classes. Hence to compare the result of the classifiers we use the naive classifier (4.7) as benchmark.

4.2 Implementation

Data analysis and modeling was performed in R, an interpreted programming language and a free open source software environment for statistical computing and graphics [17]. Packages created by users extend its capabilities. In the implementation of Neural Networks the package `neuralnet` [14] were used and in implementation of SVM the packages `e1071` [15], and `LIBSVM` [16] were used.

The following sections describes the process of finding suitable parameters for the learning algorithms. When parameters were chosen the final models were evaluated 10 times for each data set and the accuracy of each classifier were evaluated and compared to the naive classifier of always choosing the class with highest class probability. Each time the data were randomly partitioned into training and test set, with 70 percent of data in the training set and 30 percent in the test set. The minimum, median, average and maximum value of the prediction accuracy are presented.

4.2.1 Neural Networks

Training a Neural Network involve the choice of activation function, error function and the structure of the hidden layers and neurons. In the pursuit of finding a well performing network different activation- and error functions were tested and combinations of these. Also a number of networks configurations were tested and test error were evaluated to find a suitable network configuration. In this pursuit we wish to use the same training and test data since different sets could effect the performance. However as the initial weights in the model are randomly initialized, training a identical networks often yield different test errors.

Initially single hidden layer networks were tested for the error functions Sums of Square Error (SSE) (3.58) and Cross Entropy (CE) (3.59) and the Logistic and Hyperbolc Tangent (tanh) activation functions. See table 3.2 for description of the activation functions. Then some arbitrary multiple hidden layer configurations were evaluated.

4.2.2 Support Vector Machines

Using the `smv()` function in the `e1071` package one can choose different kernel functions, and tuning parameters C and γ. C is the cost parameter described in section 3.5 which affects the amount of misclassification allowed and thereby the with of the margin. γ is a tuning parameter in the kernel functions. Two kernel functions were tested, namely Radial

$$K(\mathbf{x}, \tilde{\mathbf{x}}) = \exp\big\{ -\gamma \left\| \mathbf{x} - \tilde{\mathbf{x}} \right\|^2 \big\}, \tag{4.8}$$

and Sigmoid

$$K(\mathbf{x}, \tilde{\mathbf{x}}) = \tanh(\gamma \langle \mathbf{x}, \tilde{\mathbf{x}} \rangle + \kappa). \tag{4.9}$$

Note that these are the same equations as given by table 3.1 but the parameters of the hyperbolic tangent has been renamed to fit notations of `svm()` function. For each choice of kernel function a grid search were conducted to choose suitable values of C and γ. In a grid search a number of values for each parameter are chosen and SVM are fitted using all combinations of the parameter values. The test error are recorded for each SVM creating a grid. In the grid suitable values for parameters can be identified by locating a minimum of the test error.

Chapter 5

Results

In this chapter the results will be presented. As discussed in section 2, two main strategies refereed to as initial- and improved strategy were used to generate sets of data used for evaluation. First the results of the analysis to choose suitable parameters for the learning algorithms are presented. Thereafter the results are divided into two sections, one for each set of data generated by the different strategies.

The result of Neural Networks and Support Vector Machines are given for various combinations of the parameters (ξ, δ), used in algorithm from section 2.6.3 to reduce the dimensionality of the problem. For those combinations both learning algorithms are implemented their results are compared.

5.1 Parameter evaluation

The result of the analysis to choose suitable parameters for the learning algorithms are presented in this section. First the results for the Neural Networks are presented followed by the grid search for the Support Vector Machines.

5.1.1 Neural Networks

Networks were trained and performance were evaluated for combinations of the SSE and CE error functions and the logistic and tanh activation functions. Due to trouble with the tanh activation function, which will be discussed in the discussion, the logistic activation function were used. The analysis were quite time consuming, hence only performed once on the data generated by the improved strategy from section 2.4, using $(\xi, \delta) = (10, 50)$ in the algorithm for reducing the dimensionality, described in section 2.6.3. In Figure 5.1 the test error using different number of neurons in a single hidden layer network, for both SSE and CE error functions are displayed along elapsed time for training the networks. The following number of neurons in the hidden layer were tested;

$$\text{Number of neurons} = 1, 10, 20, 30, 50, 100, 150, 200, 250.$$

The test error using more than one neuron are quite similar for the networks independent of the error function. Lowest test error is 0.227 and are obtained using CE error function and 150 neurons. Examining the right plot of Figure 5.1 one can see the exponential growth of the training time, i.e. the computational time for training a network, as the number of neuron increases.

Based on the results of the single layer networks, a few networks with two hidden layers were trained. In Figure 5.2 the results of such are displayed. Networks with 10, 50 and 150 neurons in a first hidden layer were combined with a second layer of 5,10,20,50 and 100 neurons. No

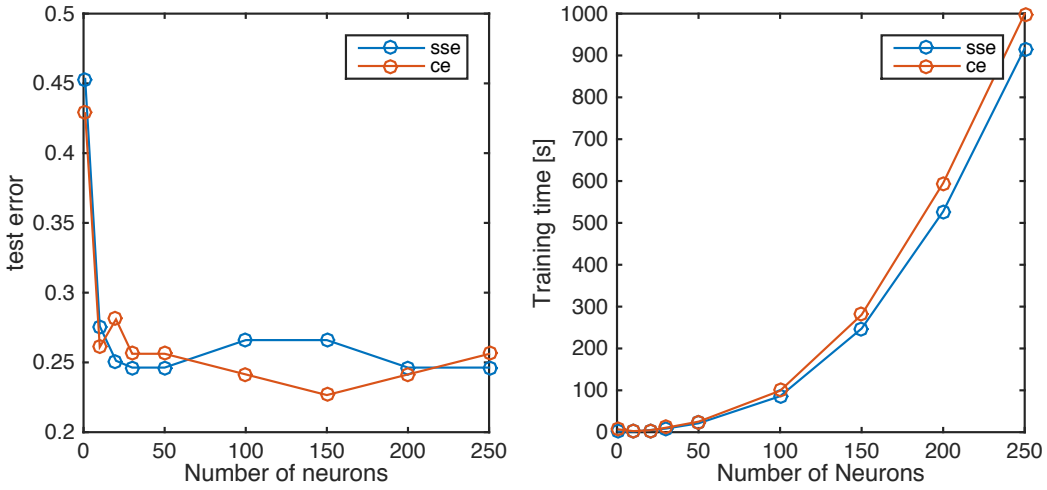

Figure 5.1: *In the left panel test error for networks with a single hidden layer of 1,10,20,30,50,100,150,200 and 250 neurons is displayed. The logistic activation function are used and both SSE and CE error function are tested. The right panel presents the corresponding training times.*

large deviations in performance are obtained and time of training were similar to that of just a single hidden layer with corresponding number of neurons in the first layer. For the analysis of the two hidden layer networks the CE error function were used. Lowest test error of 0.236 are obtained using a network with 50:10 neurons in the first:second hidden layer (test error of 0.237 are obtained using 50:20 and 150:20).

Networks with tree hidden layers were also tested using 10,50 and 150 neurons in the first hidden layer, 20 neurons in the second hidden layer and 5,10,20,50 and 100 neurons in the third hidden layer. The test error and training time for these evaluations are displayed in Figure 5.3. Again CE are used as error function and the performance is quite similar even though using 50 neurons in the first layer seems to perform slightly better most times. The lowest test error of 0.237 are obtained by a network with 50:20:20 neurons in the first:second:third hidden layer respectively.

The overall analysis shows that out of the tested configurations a single hidden layer neural network with 150 neurons, logistic activation function and cross entropy error function had the lowest test error, however the performance are quite similar for many configurations tested. The final model was chosen as a network with two hidden layers using 50:20 neurons in the first:second hidden layer. The choice are based on the fact that the difference in test error are only about 4 percent compared to the best performing model, but the computational time are more than 10 times faster. The logistic activation function and cross entropy error function are used in the final model.

5.1.2 Grid Search

As described in section 4.2.2 to determine suitable parameter values for the SVM classifier a grid search are applied. It is performed once for each set of data obtained by different choices of parameters (ξ, δ) used in the algorithm to reduce dimensionality, and for both Radial and Sigmoid kernels. In Figure 5.4 the result from a grid search using $(\xi, \delta) = (10, 50)$ and Sigmoid kernel is displayed. The logarithm of C and γ are used to give an smother surface. In the grid search the following values, and all combinations of them, are used:

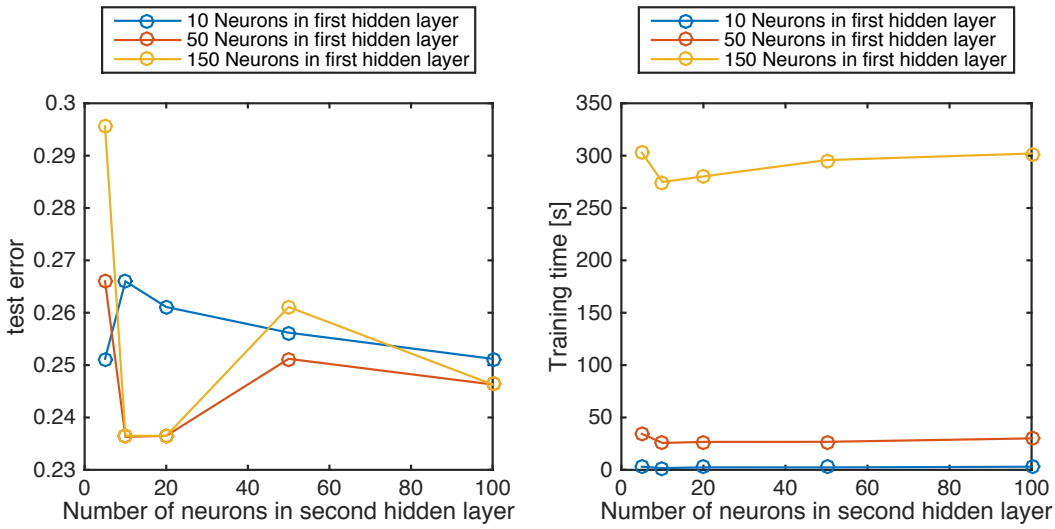

Figure 5.2: *In the left panel the test error for networks with 10,50 and 150 neurons in the first hidden layer, using 5,10,20,50 and 100 neurons in a second hidden layer are displayed. Logistic activation function and Cross Entropy as error function are used. The plot in the right panel displays the corresponding training times.*

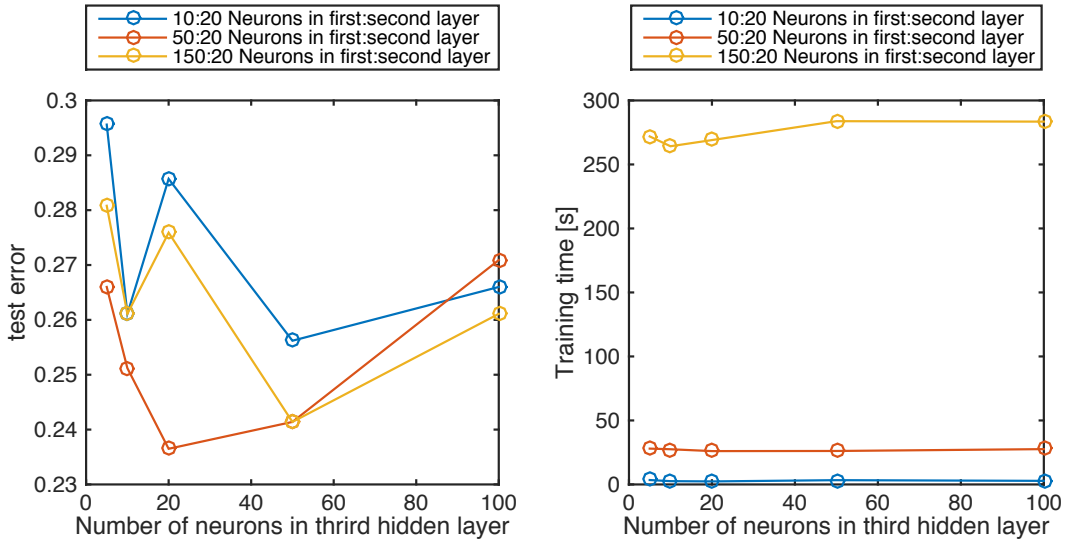

Figure 5.3: *In the left panel the test error for networks with 10,50 and 150 neurons in the first hidden layer, 20 neurons in the second hidden layer and using 5,10,20,50 and 100 neurons in a third hidden layer are displayed. Logistic activation function and Cross Entropy as error function are used. The plot in the right panel displays the corresponding training time.*

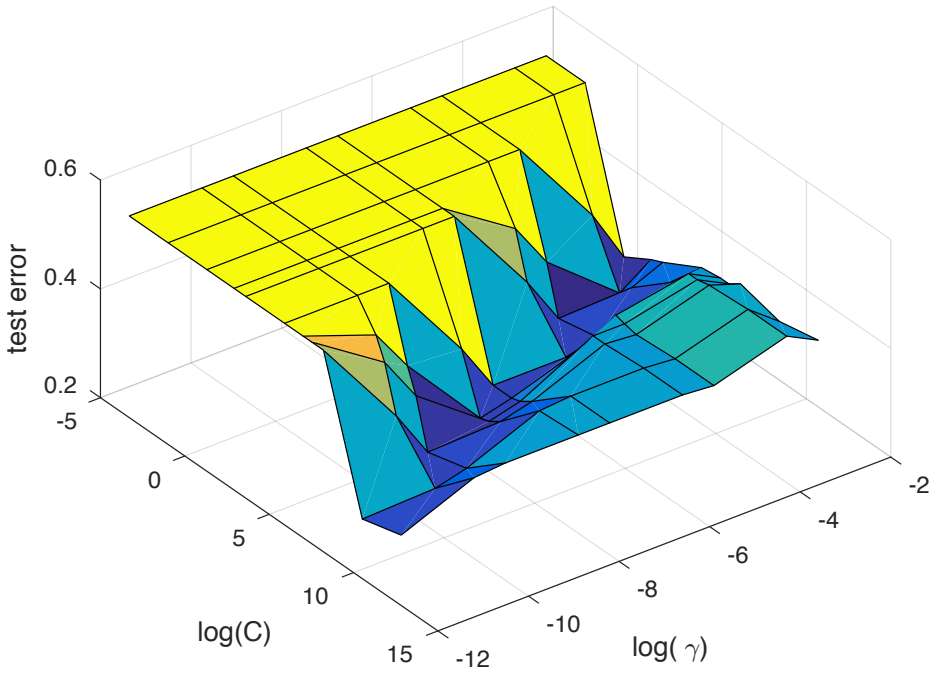

Figure 5.4: *The surface illustrates how the test error of SVM classifier varies with the parameters C and γ. The x-axis shows $log(C)$ and the y-axis $log(\gamma)$. The logarithm is used to yield a smother surface. This particular surface were obtained using the Sigmoid kernel function on data generated using the improved strategy and parameters an the algorithm to reduce dimensionality set to $(\xi, \delta) = (10, 50)$.*

$$C = 0.01, 0.1, 1, 5, 10, 100, 300, 500, 1000, 10000,$$
$$\gamma = 0.1, 0.05, 0.01, 0.005, 0.001, 0.0005, 0.0001, 0.00005, 0.00001.$$

Examining Figure 5.4 one can see similar results of the test error for various combinations of the parameters. In fact, in most cases there were several combinations of the parameters for which the test error were at its minimum (among the tested values). Similar results were obtained for the other grid searches performed. These results are not included in the report since the grid search is merely a systematic way of determining suitable parameters. Approximately 15 grid searches were performed using different kernels on different data. Among the tests $C = 100$ and $\gamma = 0.001$ in combination with Sigmoid kernel were identified as suitable since this combination yielded a minimum, or close to a minimum test error for all grid searches performed.

5.2 First data set

The result in this section were obtained using a data set generated by the initial strategy described in section 2.6.3. Before applying the algorithm to reduce the dimensionality the dataset contained 12605 observations of 18844 features (unique DTC's) and 5565 classes. Using (10,50) and (1,100) for the parameters (ξ, δ) respectively in the algorithm to reduce dimension two sets of data are obtained and used to evaluate the performance of the learning algorithms.

The results are summarized in Table 5.1. For each learning method and choice of (ξ, δ) the number of observations, features and classes are reported together with the accuracy of the clas-

Table 5.1: *The table shows the minimum, median, mean and maximum prediction accuracy of the classifiers when tested 10 times. The parameters (ξ, δ) are from the algorithm in section 2.4 used to reduce the dimensionality of input data. The corresponding number of observations, features and classes of the learning data are also given.*

Method	(ξ, δ)	Observations	Features	Classes	Min.	Median	Mean	Max.
SVM	(10,50)	604	1243	8	0.624	0.657	0.670	0.718
Naive	(10,50)	604	1243	8	0.182	0.213	0.213	0.249
NN	(10,50)	604	1243	8	0.536	0.572	0.580	0.630
Naive	(10,50)	604	1243	8	0.193	0.207	0.210	0.243
SVM	(1,100)	1286	4084	9	0.661	0.698	0.696	0.728
Naive	(1,100)	1286	4084	9	0.140	0.163	0.161	0.184
NN	(1,100)	1286	4084	9	0.570	0.628	0.629	0.707
Naive	(1,100)	1286	4084	9	0.140	0.163	0.164	0.184

sifier. Learning data are randomly partitioned (70:30) into training and test set 10 times and the minimum, median, mean and maximum value for the accuracy of the learning algorithm are given and compared to the naive classifier described in section 4.1.4.

Analysis shows that for the tested parameters of the learning algorithms both Neural Networks and Support Vector Machines outperforms the naive classifier with a great margin. The Support Vector Machines has a superior performance compared to Neural Network for both tests. Using (10,50) for the parameters (ξ, δ) respectively data contained 604 observations of 1243 features and 8 classes. The SVM had an average prediction accuracy of 67 percent compared to 58 percent using NN or about 21 percent using the naive approach. The best accuracy were 71.8 and 63 percent for SVM and NN respectively. Using (1,100) for (ξ, δ) data contained 1286 observations of 4084 features and 9 classes. The average prediction accuracy using SVM is 69.8 percent compared to 62.9 using NN.

In Table 5.2 the result from one prediction using SVM on data with parameters $(\xi, \delta) = (10, 50)$ is displayed as a confusion matrix. Classes are labeled using number 1-8. Based on this confusion matrix the within class prediction accuracy (4.2) and class prediction error (4.3) are evaluated, the results are summarized in table 5.3. The average within class prediction accuracy for this test were 63.8%. Best within class prediction accuracy was 88.2% recorded for class 7 while class 1 had worst within class prediction accuracy of just 32%. No class are erroneously predicted to class 7 while 50% of the observations predicted as class 3 were misclassified.

5.3 Second data set

The result in this section were obtained using a data set generated by the improved strategy described in section 2.6.3. Before applying the algorithm to reduce the dimensionality the dataset contained 6002 observations of 13360 input variables (unique DTC's) and 477 classes. Using (10,50) and (1,100) for the parameters (ξ, δ) respectively in the algorithm to reduce dimension two sets of data are obtained and used to evaluate the performance of the learning algorithms. The results are summarized in Table 5.4. For each learning method and choice of (ξ, δ) the number of observations, features and classes are reported together with the accuracy of the classifier. Learning data are randomly partitioned (70:30) into training and test set 10 times and the minimum, median, mean and maximum value for the accuracy of the learning algorithm are given and compared to the naive classifier described in section 4.1.4.

Table 5.2: *Confusion matrix from SVM on data generated by the initial strategy and using* $(\xi, \delta) = (10, 50)$*,* $C = 100$*,* $\gamma = 0.001$ *and sigmoid kernel. Prediction accuracy 64.1 percent.*

		True class							
		1	2	3	4	5	6	7	8
Predicted class	1	8	1	0	0	0	4	0	1
	2	0	13	2	0	0	0	0	0
	3	2	7	32	3	11	4	0	5
	4	0	0	3	7	1	0	0	0
	5	0	1	1	0	8	0	0	1
	6	15	0	0	0	0	24	0	0
	7	0	0	0	0	0	0	15	0
	8	0	0	1	0	0	0	2	9

Table 5.3: *The resulting within class prediction accuracy* p_k *and class prediction error* ε_k *calculated from the confusion matrix given in Table 5.2*

k	1	2	3	4	5	6	7	8	Mean
p_k	0.320	0.591	0.821	0.700	0.400	0.750	0.882	0.563	0.628
ε_k	0.429	0.133	0.500	0.364	0.273	0.385	0	0.250	0.292

Using $(\xi, \delta) = (10, 50)$ the learning data contained 677 observations of 1252 features belonging to 6 classes. The SVM yielded the best prediction accuracy of 76.7 percent on average and 81.3 percent at best. NN yielded 73.7 percent prediction accuracy on average, with best performance of 79.8 percent. Both SVM and NN beat the naive classifier with great margin. Using $(\xi, \delta) = (1, 100)$ data contained 1547 observations of 4168 features and 7 classes. Again both SVM and NN outperformed the naive classifier and the SVM had a superior performance to NN, 79.4 percent average prediction accuracy compared to 75.4 percent. The best prediction accuracy of 84.1 percent were obtained using SVM compared to 80.2 percent using NN. The naive classifiers best prediction accuracy were 48.5 percent.

On the second data set generated by the improved strategy some additional evaluations were performed choosing parameters (ξ, δ) to include more classes. This means that the feature space will grow, making computations more demanding. Due to limitations in computer hardware NN could not be evaluated on the larger data samples. In Table 5.5 the minimum, median, mean and maximum prediction accuracy of SVM are given when tested 10 times for $(\xi, \delta) = (0, 100),\ (1, 50),\ (1, 20)$ respectively.

For $(\xi, \delta) = (0, 100)$ we obtain 2328 observations of 7833 features and 9 classes. The average prediction accuracy was 76.7 percent and 79.1 percent at best. Using the naive classifier the corresponding numbers were 39.5 and 43.6 percent respectively. For $(\xi, \delta) = (1, 50)$ data contained 2289 observations of 4895 features and 19 classes. The average prediction accuracy of SVM classifier is 60.3 percent, minimum and maximum prediction accuracy is 58.4 and 62.7 percent respectively. The SVM classifier outperformed the naive classifier for which the best prediction accuracy is 32.2 percent and 30.7 on average. Using $(\xi, \delta) = (1, 20)$ the learning data contained 3036 observations of 5534 features and 46 classes. The SVM classifier yielded a prediction accuracy of 45.6 percent on average and 48.3 percent at best. The corresponding naive classifier has an average prediction accuracy of 22 percent an 24.1 percent at best.

Table 5.4: *The table shows the minimum, median, mean and maximum prediction accuracy of the classifiers when tested 10 times. The parameters (ξ, δ) are from the algorithm in section 2.4 used to reduce the dimensionality of input data. The corresponding number of observations, features and classes of the learning data are also given.*

Method	(ξ, δ)	Observations	Features	Classes	Min.	Median	Mean	Max.
SVM	(10,50)	677	1252	6	0.729	0.761	0.767	0.813
Naive	(10,50)	677	1252	6	0.448	0.505	0.496	0.517
NN	(10,50)	677	1252	6	0.704	0.734	0.737	0.798
Naive	(10,50)	677	1252	6	0.424	0.470	0.474	0.522
SVM	(1,100)	1547	4168	7	0.763	0.790	0.794	0.841
Naive	(1,100)	1547	4168	7	0.425	0.444	0.447	0.485
NN	(1,100)	1547	4168	7	0.724	0.754	0.754	0.802
Naive	(1,100)	1547	4168	7	0.405	0.445	0.443	0.485

In Table 5.6 the result from one prediction using SVM on data with parameters $(\xi, \delta) = (1, 100)$ is displayed as a confusion matrix. Classes are labeled using number 1-7. Based on this confusion matrix the within class prediction accuracy (4.2) and class prediction error (4.3) are evaluated, the results are given in table 5.7. The average within class prediction accuracy were 73.6%. Two classes yielded a prediction accuracy above 95% but there are also two for which the within class prediction accuracy is just about 54%. Very few observations are erroneously predicted to class 1 and 6 but 40% of the observations predicted as class 7 is misclassified.

In Figure 5.5 the average prediction accuracy of SVM classifier are plotted against the number of classes used. The average is taken over 10 evaluations. For this analysis data generated by the improved strategy are used and the parameters in SVM is set to $C = 100$, $\gamma = 0.001$ and the sigmoid kernel function are used. In the algorithm to reduce dimensionality $\xi = 1$ and δ was varied to obtain different number of classes. δ corresponds to the minimum number of observations each class should have to be included in the learning data. The following values are used:

$$\delta = 189, 179, 140, 130, 124, 102, 71, 56, 48, 31, 27, 21, 18, 12, 9.$$

The corresponding number of classes is:

$$\text{Number of Classes} = 2, 3, 4, 5, 6, 7, 10, 15, 20, 25, 31, 40, 52, 65, 80.$$

The SVM performs better than the naive classifier every time and one can se that the prediction accuracy tends to decrease with the number of classes.

Table 5.5: *The table shows the minimum, median, mean and maximum prediction accuracy of the SVM classifier when tested 10 times. Corresponding accuracy of the naive classifier are also given. The parameters (ξ, δ) are from the algorithm in section 2.4 used to reduce the dimensionality of input data. The corresponding number of observations, features and classes of the learning data are also given.*

Method	(ξ, δ)	Observations	Features	Classes	Min.	Median	Mean	Max.
SVM	(0,100)	2328	7833	9	0.746	0.764	0.767	0.791
Naive	(0,100)	2328	7833	9	0.370	0.393	0.395	0.436
SVM	(1,50)	2289	4895	19	0.584	0.606	0.603	0.627
Naive	(1,50)	2289	4895	19	0.284	0.309	0.307	0.322
SVM	(1,20)	3036	5534	46	0.443	0.451	0.456	0.483
Naive	(1,20)	3036	5534	46	0.195	0.218	0.220	0.241

Table 5.6: *Confusion matrix from SVM on data generated by the improved strategy, using $(\xi, \delta) = (1, 100)$, $C = 100$, $\gamma = 0.001$ and sigmoid kernel. Prediction accuracy 81 percent.*

		True class						
		1	2	3	4	5	6	7
Predicted class	1	43	0	0	0	2	0	0
	2	0	198	5	11	12	2	20
	3	0	0	31	1	1	2	0
	4	0	1	3	19	1	0	0
	5	2	2	0	0	30	2	2
	6	0	0	0	0	0	28	1
	7	0	7	3	4	4	0	27

Table 5.7: *The resulting within class prediction accuracy p_k and class prediction error ε_k calculated from the confusion matrix given in Table 5.6*

k	1	2	3	4	5	6	7	Mean
p_k	0.956	0.952	0.738	0.543	0.600	0.824	0.540	0.736
ε_k	0.044	0.202	0.114	0.208	0.211	0.035	0.400	0.173

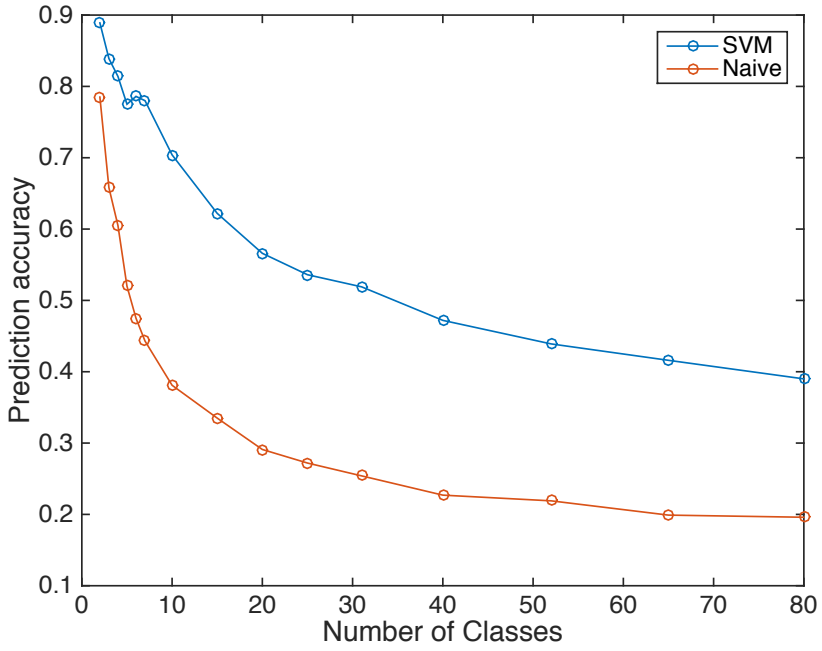

Figure 5.5: *Average prediction accuracy (1-error) of SVM evaluated over 10 times using* $C = 100$, $\gamma = 0.001$, *sigmoid kernel for* $\xi = 1$ *and* δ *is varied to obtain different number of classes. The blue line shows the prediction accuracy of SVM and the red line the prediction accuracy of the naive classifier.*

Chapter 6

Discussion

In chapter 2 it is explained how data are gathered and processed in order to generate a suitable data set for the learning algorithms. At the start of this project there were no data compiled to constitute a suitable sample and the process of generating one turned out to be more troublesome and time consuming than anticipated. Due to that data containing information on exchanged parts were not directly accessible to me, I had to rely on other people to generate such data. Thus in the early stages of this thesis project there were some uncertainties about structure on data and the amount of available data. The first data set generated by the strategy refereed to as *initial* strategy, described in section 2.4, were evaluated to some extent before realizing its flaws. The analysis of data generated by this strategy is included in the report partly since the *improved* strategy, also described in section2.4, were established at a late stage but primarily do illustrate the importance of *good* data in analysis. Comparing the average prediction accuracy for the classifiers using data generated by the initial and improved strategy, found in Table 5.1 and 5.4, we see improvements of about 10 percent.

Using initial strategy of generating classes, a class were defined by a unique combination of exchanged parts. The main trouble with this approach is that if a part had been exchanged by its own on one vehicle and in combination with other part(s) on another, more than one class would have been generated. Hence sets of DTC's indicating the same problem could belong to several classes which obviously is not ideal for a classifier. This problem is illustrated by a in depth analysis of the confusion matrix in Table 5.2 generated by a SVM classifier on data generated by the initial strategy. The classes were labeled 1-8, but in Table 6.1 the names of the spare part(s) corresponding to each class are given. (Note that the name of the spare part were checked only after classifiers on the initial strategy had been evaluated, hence at the time classes were merely article numbers without interpretation. This is why classes like washer fluid is present among the classes.) Class 1 and 6 both contains NOx-sensor, but class 1 also contains cable tie. It is reasonable to believe that the DTC's indicates a problem with the NOx-sensor and that the set of DTC's belonging to class 1 and 6 are quite similar. In Table 5.3 we see that the within class prediction accuracy of class 1 is 32 percent, and the confusion matrix shoes that 15 of 25 observations belonging to class 1 are incorrectly classified to class 6.

The aim of the improved strategy were to address this issue. It is by no means flawless, and one drawback of the improved strategy is that many observations are lost due to that they could not be assigned to a class. The improved strategy involves taking the set of exchanged parts of one vehicle and try to indicate a single fault causing part. How this is done is described in section 2.4. Using the improved strategy, observations from class 1 and 6 in the initial strategy, are labeled as a single class *Differentialtryckgivare, Partikelfilter* labeled 2 in Table 6.2. In table 5.7 we can see the within class prediction accuracy of 95.2 percent for this class compared to 32 and 75 percent of class 1 and 6 respectively from the initial strategy. It seems as is the improved

Table 6.1: *Class labels corresponding to confusion matrix in table 5.2*

1	NOx-givare, Bandklämma
2	Filtersats
3	Vattenventil
4	Spolarvätska
5	Differentialtryckgivare
6	NOx-givare
7	Reduktionsmedelsdoserare
8	Säkring, Relä

Table 6.2: *Class labels corresponding to confusion matrix in table 5.6*

1	COO, Electronisk styrenhet
2	Differentialtryckgivare, Partikelfilter
3	Rör och slangar, Reduktionsmedeldoserare, SCR
4	Reduktionsmedelpump, SCR
5	Säkring, Relä, Diod
6	SCR, EEC, Sensor
7	Vattenventil, Rör- och slangdragning

strategy for generating classes lives up to its name. In table 5.3 and 5.7 we see that the average within class prediction accuracy is 73.6 percent for the SVM trained on data generated by the improved strategy compared to 62.8 percent when trained on data generated by the initial strategy.

In Chapter 5 it is established that the SVM classifier had better prediction accuracy than the NN classier for all evaluations performed. Note however that all comparisons were based on Neural networks with two hidden layers of 50:20 neurons respectively and with logistic activation- and cross entropy error function. In the parameter evaluation several other network architectures were tested, one of which actually had better prediction accuracy. The goal of the parameter evaluation is to find a suitable network configuration. Among the infinite possible networks only few were tested and there is no guarantee that other configurations could perform even better than those tested. Also since the weights are randomly initialized training on same data often yields different results which makes comparing various configurations a bit problematic. Initially one would expect the test error to decrease as the model complexity increases, but a too complex model is likely for overfitting to occur and we would expect test error to increase. In Figure 5.1 fitting a single hidden layer network with SSE error, the test error initially decreases with the number of neurons. Then test error increases a little to decrease again as the number of neurons increase. It is possible that the if the test were run again it would look somewhat different due to the random initial weights. A better approach would be to record the test error 10 times and use the average. One could also include larger networks with more neurons and/or hidden layers in the parameter evaluation analysis. This was not done due to the computational time of training large networks.

Apart from the computational time of the neural networks I experienced a few problems with implementation and training Neural Networks. As mentioned in section 5.1 I experienced some problems using hyperbolic tangent (tanh) activation function. Using tanh in combination with SSE the algorithm did not converge (`stepmax` was set to 10000) which was never a problem using logistic error function. Possible reasons why the algorithm experienced problems has not been investigated. Trying to use tanh activation function in combination with CE error function

generated an error message:

```
Warning messages:
1:  In log(x) :  NaNs produced
2:  'err.fct' does not fit 'data' or 'act.fct'
```

At the time this was not researched further and as explained the logistic activation function were chosen instead. However the problem was probably that the output was not normalized, e.g. using softmax function, hence the output from tanh would be in the range $[-1, 1]$. Recall CE error function (3.59), which involves the logarithm, hence we cannot have negative values as inputs. Another problem experienced when fitting NN on large data were that the computer did not have enough memory to handle the computations. There would be an error message like:

```
Error:  cannot allocate vector of size 7.6 Gb.
```

On R documentation [20] these types of errors are explained:

"Error messages beginning cannot allocate vector of size indicate a failure to obtain memory, either because the size exceeded the address-space limit for a process or, more likely, because the system was unable to provide the memory"

Hence it seems as it is the RAM memory on the computer that was the limit and that larger could be evaluated if desired using appropriate hardware.

In figure 5.5 the average prediction accuracy of SVM classifier is displayed as a function of the number of classes used in the learning data set. The average is taken over 10 predictions. The overall trend is that the prediction accuracy decreases with the number of classes. This may not come as a surprise, but there are a few things that should be said. First of all it is not obvious how to rate the performance of classifiers tested on different data. Could one say that a prediction accuracy of 80 percent in a problem with 6 classes is a *better* performance than a prediction accuracy of 50 percent in a problem with 50 classes? But still, if Scania which to deploy a classifier to predict fault causing parts based on sets of DTC's it would have to be able to accurate predict many classes otherwise it would not be of much assistance. There could be many possible reasons why prediction accuracy drops as more classes are included. In general this is expected since the model complexity increases with the number of classes, but the observed decrease seems quite substantial, even for small number of classes, so could this be explained?

In mathematical terms we would expect poor performance if the observations overlap in feature space. In the setting of this problem it would mean that observations of different classes have common DTC's. This is present in learning data, and has been observed by inspection, but no in depth analysis has been conducted on the subject. In data generated by the improved strategy and using $\xi = 1$, $\delta = 50$ in the algorithm to reduce dimensionality there is 2289 observations of 4895 features and 19 classes. In this data there are 2285 unique sets of DTC's, i.e. virtually every set of DTC's are unique. Note that this does not mean that observations form a single class does not have one or several DTC's in common, but rather that there almost always are observed DTC's of different cinds that probably is uncorrelated to the problem that lead to the exchange of some part. It would be quite interesting to examine the similarities between the observations, within one class and between different classes, but as mentioned no such analysis has been performed. Another possible explanation is that the number of observations play an important role. Note that to include more classes in learning data we change the parameter δ in

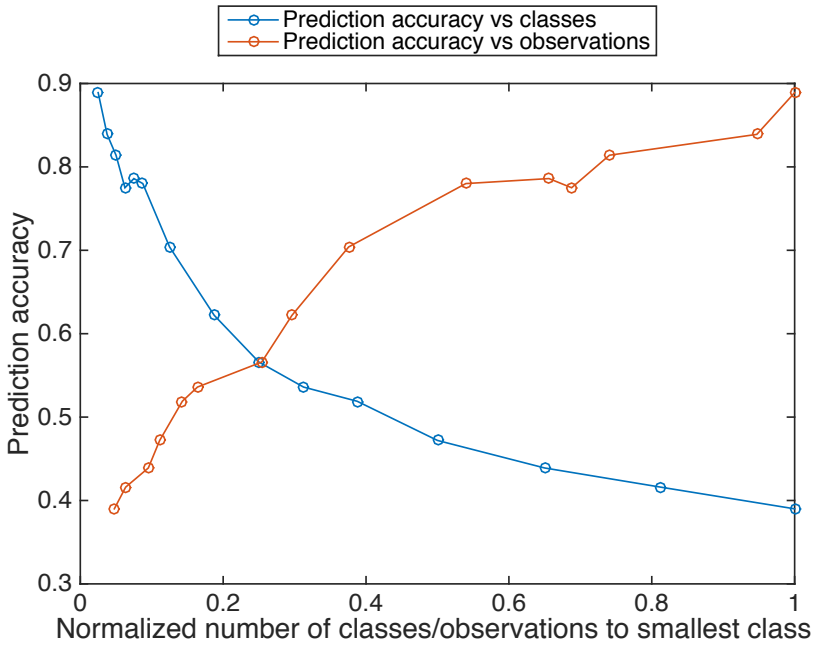

Figure 6.1: *Average prediction accuracy of SVM evaluated 10 times using $C = 100$, $\gamma = 0.001$, sigmoid kernel for $\xi = 1$ and δ is varied to obtain different number of classes. The blue line shows the prediction accuracy as a function the number of classes and the red line shows the prediction accuracy as a function of the smallest number of observations in learning data (δ). The x-axis has been normalized to fit both scales. Delta is varied in the interval [189,9] which yields classes in the range [2,80].*

the algorithm to reduce dimensionality. Recall that δ corresponds to the minimum number of observations to a class for it to be included in learning data. Hence, as we evaluate the prediction accuracy in Figure 5.5 we must consider that as more classes are included in the problem the number of observations to the additional classes are decreasing. For example using $\delta = 18$ we have a number of classes with less than 20 observations, then by randomly partitioning data into training (70%) and test set (30%) we can expect to have only 14 observations to those classes in the training set. Considering the high dimensionality of about 5600 features, this seems like a hard task. For comparison if $\delta = 100$ we have at least 100 observations to each class and we could expect to have about 70 observations in training data to those classes with fewest observations. Also the dimensionality is reduced to about 4200 features. It comes as no surprise that the later learning data yields better prediction accuracy. Figure 6.1 shows the average prediction accuracy of SVM, evaluated 10 times, fitted on data generated by the improved strategy using $C = 100$, $\gamma = 0.001$, sigmoid kernel. In the algorithm to reduce dimensionality $\xi = 1$ is held constant and δ is varied in the interval [189,9] to obtain different number of classes for learning data. The prediction accuracy is displayed as a function of the number of classes in learning data as well as a function of the number of observations of the smallest class (δ). The x-axis is normalized to fit both scales. The blue line in Figure 6.1 is the same as in Figure 5.5 but plotted on a normalized scale. The red line shows the prediction accuracy as a function of the number of observatons to the smallest class and we can see that the prediction accuracy seems to drop quicker towards the lower half of the scale indicating that a small number of observations could influence the prediction accuracy.

We could also look at the within class prediction accuracy as a function of the number of observations to each class. Note that there could be many reasons why the prediction accuracy varies

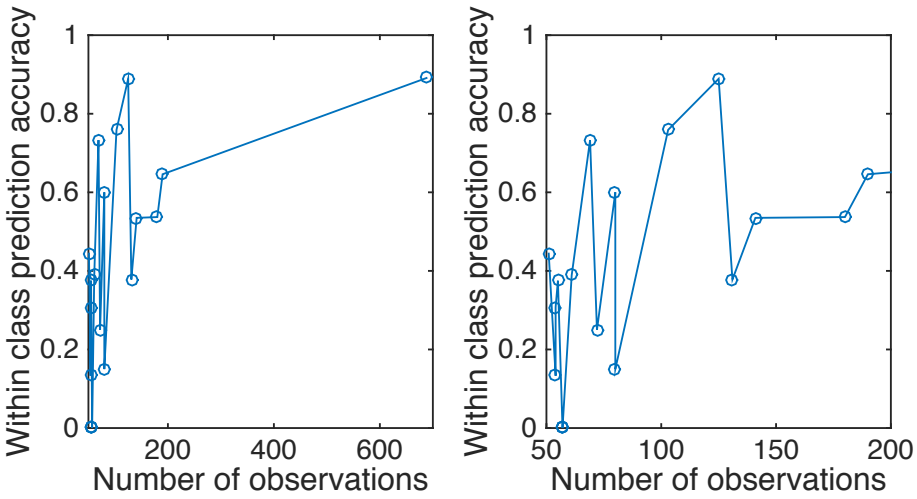

Figure 6.2: *Within class prediction accuracy for 19 classes plotted against the number of observations. The plot on the right is a zoomed version of the plot on the left.*

between classes, e.g. the correlation between DTC's and exchange parts might be stronger for certain classes, and as described, some classes were considered not suitable and removed manually from the learning data. This was only done for about the top 10 classes which were included in the smaller sets generated. It was only in a late stage of this thesis that performance of larger data sets, including more classes were decided upon and it was forgotten to remove unsuitable classes with few observations. Hence analysis of data containing more than 9 classes includes ones like *windscreen wipers* which is not suitable for classification. Still it would be interesting to see how the within class prediction accuracy varies with the number of observations to respective class.

Using data generated by the improved strategy and reducing dimensionality with parameters $\xi = 1$, $\delta = 50$ we obtain a data set with 2289 observations of 4895 features and 19 classes. There are 12 classes with the number of observations in the range 50-100 and 7 classes with more than 100 observations (6 in the range 100-200 and one with 688 observations). Training a SVM classifier with parameters $C = 100$, $\gamma = 0.001$, sigmoid kernel function on the data to obtain a confusion matrix we can compute the within class prediction accuracy for each class. In table 6.3 the resulting confusion matrix is given and in table 6.4 the corresponding within class prediction accuracy and class prediction error are given. In Figure 6.2 the withing class prediction accuracy for the different classes are plotted against the number of observations to each class. The average within class prediction accuracy for those classes with more than 100 observations are 66.2 percent compared to 28.1 percent for those classes with observations in the range [50,100]. The overall prediction accuracy of the classification were 59.5 percent. This also indicates that the number of observations to each class seems to have a large effect on the prediction accuracy of a classifier. Note that the number of observations to each class is based on the learning data prior it is partitioned into training and test set.

In Figure 5.5 we see that having only two classes the naive classifier yields a prediction accuracy of nearly 80 percent. This shows the large deviation on the number of observations to each class. The largest class has 688 observations compared to 190 observations in the second largest class. Ideally there should be an equal number of observations to each class. but since most classes had few observations it was chosen to use all available observations.

Table 6.3: *Confusion matrix from SVM on data generated by the improved strategy, using* $(\xi, \delta) = (1, 50)$, $C = 100$, $\gamma = 0.001$ *and sigmoid kernel. Prediction accuracy 59.5 percent. Each column correspond to the true class and each row to the predicted class of an observation*

		True class																		
		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
P	1	6	0	0	0	0	0	1	0	1	1	0	0	1	0	0	0	0	1	1
	2	0	32	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
	3	1	0	196	8	2	4	13	7	6	9	9	8	11	1	5	6	5	8	3
	4	0	0	3	7	0	0	0	0	0	0	0	0	1	0	0	0	0	1	0
	5	0	0	0	1	11	1	1	0	3	0	0	1	1	0	0	0	2	1	0
	6	0	0	0	0	0	15	0	0	0	0	0	0	1	0	3	0	0	0	0
	7	0	0	0	0	0	0	0	2	0	0	0	0	0	0	0	0	0	0	0
	8	1	0	3	0	0	0	0	6	1	0	0	0	0	0	0	0	0	0	0
	9	0	0	1	1	0	0	0	0	0	1	0	1	0	0	0	0	0	1	0
	10	0	0	1	0	0	0	0	0	1	23	0	0	0	1	0	0	0	1	0
	11	0	0	1	0	0	0	0	1	0	0	3	2	0	0	0	0	0	1	0
	12	0	0	4	0	0	0	0	1	1	1	6	14	0	0	2	0	0	1	0
	13	1	2	3	1	0	0	0	0	0	5	0	1	29	5	3	0	0	0	2
	14	3	1	0	0	0	3	0	0	0	2	0	0	3	22	1	0	0	2	0
	15	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	16	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	2	0	0	0
	17	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	4	0	0
	18	3	1	7	0	2	2	3	7	7	1	2	9	6	0	3	7	2	31	4
	19	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	8

Table 6.4: *The resulting within class prediction accuracy* p_k *and class prediction error* ε_k *calculated from the confusion matrix given in Table 6.3*

k	1	2	3	4	5	6	7	8	9	10
p_k	0.375	0.889	0.891	0.389	0.733	0.600	0	0.250	0	0.535
ε_k	0.500	0.030	0.351	0.417	0.500	0.211	1.00	0.455	1.00	0.148
k	11	12	13	14	15	16	17	18	19	Mean
p_k	0.150	0.378	0.537	0.759	0	0.133	0.308	0.646	0.444	0.422
ε_k	0.625	0.533	0.442	0.405	1.00	0.333	0.200	0.680	0.1111	0.470

For a long time in this thesis project the primary focus was to improve the generation of learning data. Good data is the key to finding a model which can produce accurate predictions. Initially it seemed as that there were almost unlimited data, but as events were coupled together to create observations of DTC's and replaced parts for vehicles it turned out that this was definitely not the case. The issue was not with the number of observations but rather the distribution to various classes. Table 2.2 illustrates this problem, on about a years worth of data there were only 18 844 matching events. These events were distributed on 5565 different sets of replaced parts. Among the matching events only 11(!) unique sets of replaced parts were observed more than 100 times. In our problem classes are based on the exchanged parts, directly using the initial strategy and indirectly using the improved, and as we have discusses above the number of observations to one class seems to be vital for good performance. This really call for a better approach to generate data.

Through out this project replaced parts has been used as indicator of a fault causing part. We have assumed that the observed DTC's of a vehicle is the result of those parts that is exchanged. The credibility of this assumption has already been discussed several times and considering available data there really were not any options. A better approach would be to log reparations as events storing information in a log by a single software program. This does not necessarily call for enormous measures, using e.g. SDP3 as base, the DTC's will be observed here and they are all ready accessible in the log file. One could include a function where the user (mechanic) could report the fault causing part once the reparation is done. Done properly this would eliminate confusions regarding if the exchanged part truly were the fault causing part. Moreover by this approach it is not necessary for any part to be exchanged in order to classify the cause of the observed error codes. Furthermore the problem of changing one or several parts discussed would also be eliminated, it would not matter if the windscreen wipers were exchanged while at the workshop, the fault causing part would be indicated by the mechanic.

The analysis shows that SVM classifier on average yields the overall most accurate prediction for all evaluations performed. When evaluated 10 times the smallest prediction accuracy of SVM were better than that of NN. In fact, the results in table 5.1 and 5.4 shows that in three out of four learning data samples the minimum prediction error of SVM were better than the average prediction accuracy of NN. The best recorded prediction accuracy were obtained for SVM in every test. Hence based on the results the SVM seems to yield better performance than NN.

As mentioned in section 4.1, the purpose of a confusion matrix in this thesis is rather to illustrate how the results of a classifier could be evaluated, than to perform in depth analysis of every classifier fitted. Hence only few confusion matrices has been included in the report to illustrate just that.

Chapter 7

Conclusion

In this thesis project the aim was to investigate if DTC's are good indicators for parts that needs to be exchanged and to generate a model that could accurately predict fault causing parts for a vehicle based observed DTC's. Succeeding with the model would imply that DTC's indeed hold useful information that could be used to indicate false causing parts.

For every test conducted the learning algorithms outperformed both a random guess and a naive classifier based on the class probabilities. Many times the prediction accuracy of the learning algorithms were twice as good compared to the naive classifier. This implies that there is correlation between observed DTC's and classes in learning data. The best prediction accuracy of 84.1 percent were obtained by a Support Vector Machine classifier trained on data with 1547 observations of 4167 features (unique DTC's) belonging to 7 classes. On this data each class had at least 100 observations. The average prediction accuracy, evaluated over 10 predictions, on the same data set were 79.4 percent. One test on this data showed that the within class prediction accuracy on average were 73.6 percent, but two classes yielded a within class prediction accuracy above 95 percent while the lowest were 54 percent. That a classifier can predict observations to some classes with an accuracy of 95 percent suggests that DTC's could be good indicators of fault causing parts, but the average within class prediction accuracy of 73.6 percent shows that there are large variations between classes.

In order to include more classes in the evaluation, the threshold on the number of observations to each class had to be reduced. Figure 5.5 shows the decreasing prediction accuracy of SVM as the number of classes are increased. In the discussion the effect of few observations to each class was analyzed, which indicated that the performance of the classifier was largely dependent on the number of classes. Hence it is hard to draw conclusions about the potential of a classifier operating on data with many classes. What is safe to say is that in order to give a classier a fair chance the learning data needs to be improved and that those prediction accuracies recorded in data with many classes are nothing to get to excited about.

A classifier that could accurately predict fault causing parts on vehicles based on nothing but observed DTC's could bring great benefits for Scania. Such a classifier would ideally handle tens of thousands possible DTC's and probably thousands various classes to indicate fault causing parts. By regularly updating the learning data with new observations and retraining the classifier it could be able learn indicate new parts solely based on the actions in the workshop. The classifiers generated in this thesis falls a bit short of this vision, but still are a step in the right direction. By improving generation of data and collecting more samples, such a classifier might just be around the corner.

7.1 Future work

In this thesis two classifiers were evaluated, one could extend to include several other classifiers in the analysis e.g. Bayesian networks. An interesting idea would be to extend the analysis and predict the time of a reparation. The time for labor can be obtained by the work orders where replaced part were found.

One possible way to improve upon the existing data could be to evaluate the set of DTC's present in data more carefully. In this thesis only a few were removed manually since they obviously could not be correlated with the replaced parts. Given time, such an analysis could be done to remove possibly many DTC's which would reduce noise within data. Moreover additional ways of generating classes could be tested, possibly identifying a better way to indicate a fault causing part.

Changing the way data is logged opens possibility connect DTC's and fault causing part more easily and accurately. Note that a fault causing part need not to be a replaced part on a vehicle but should be a part indicated by the DTC's. By involving feedback from the mechanic as briefly suggested in the discussion this would be possible. Overall the important thing is to have a lot of observations to each class. Hence one may have to expand the search window in order to obtain more observations and parts which rarely causes faults will most likely remain hard to classify. Since high prediction accuracy would be very important such a classifier one could examine the output of the predictor more closely. If it predicts to a class with a probability less than say 90% one could choose to indicate this so that the end user would know how probable the indicated fault causing part is.

On a larger scale one could see interesting opportunities of a classifier that could indicate fault causing parts based on DTC's. As most of the vehicles Scania sells today can connect to internet, one could evaluate DTC's on the road as soon as they occur. To some extent this already done today, most vehicles have a *check engine* light. But by sending information of observed DTC's as they occur they could be evaluated by a classifier similar to those of this thesis and the driver could most likely obtain much more detailed information about errors that might occur during operation. Consider a case where a vehicle is driving and there emerge a problem with e.g. the turbo charger and that it must be exchanged. If a classifier could predict this based on the DTC's the driver could be notified, the spare part could be ordered to a workshop on the vehicles rout and the time of the reparation could be minimized.

7.2 My recommendations

Based on the overall analysis in this thesis I would recommend the Support Vector Machine as a classifier, partly due to the better prediction accuracy compared to Neural Networks but also since it is easier to implement. Moreover SVM could cope with higher dimensionality for a given sample size and the computational time to fit the classifier were substantially less than that of a Neural Network.

I would strongly recommend to consider changing the way data is logged in order to facilitate the generation of training datasets which are most suitable for supervised learning. By doing so many of the uncertainties within data could be eliminated.

Bibliography

[1] Hasti T, Tibshirani R and Friedman J. *The Elements of Statistical Learning*. Springer-Verlag, 2009

[2] Izenman A.J. *Modern Multivariate Statistical Techniques*. Springer-Verlag, 2008

[3] Nielsen M.A. *Neural Networks and Deep Learning*. Determination Press, 2015

[4] James G, Witten D, Hasti T and Tibshirani R. *An Introduction to Statistical Learning*. Springer-Verlag, 2015

[5] Rosenblatt, F. (1958). The perceptron: a probabilistic model for information storage and organization in the brain, Psychological Review 65

[6] Ripley, B. D. (1996). Pattern Recognition and Neural Networks, Cambridge University Press.

[7] McCulloch, W.S. and Pitts, W. (1943). A logical calculus of the ideas immanent in the nervous activity, Bulletin of Mathematical Biophysics, 5, 115–133.

[8] Minsky, M.L. and Papert, S.A. (1969). Perceptrons, Cambridge, MA: MIT Press.

[9] Deng, L.; Yu, D. (2014). "Deep Learning: Methods and Applications". Foundations and Trends in Signal Processing. 7 (3–4): 199–200.

[10] Werbos, P.J. (1974). Beyond regression: new tools for prediction and analysis in the behavioral sciences, Ph.D. dissertation, Harvard University.

[11] Widrow, B. and Hoff, M. (1960). Adaptive switching circuits, IRE WESCON Convention record, Vol. 4. pp 96-104; Reprinted in An- dersen and Rosenfeld (1988).

[12] Hassibi, B., Stork, D.G., Wolff, G., and Wanatabe, T. (1994). Optimal brain surgery: expensions and performance comparisons, Advances in Neural Information Processing Systems, 6, 263–270

[13] Ripley, B.D. (1996). Pattern Recognition and Neural Networks, Cambridge, U.K.: Cambridge University Press. p.169

[14] Fritsch S, Guenther F, Suling M and Mueller M. (2016), neuralnet: Training of Neural Networks (R package), `https://cran.r-project.org/web/packages/neuralnet/neuralnet.pdf`

[15] Meyer D, Dimitriadou E, Hornik K, Weingessel A, Leisch F, Chang C.C and Lin C.J. (2017). e1071: Misc Functions of the Department of Statistics, Probability Theory Group (Formerly: E1071), TU Wien (R package), available at `https://cran.r-project.org/web/packages/e1071/e1071.pdf`

[16] Chang C.C and Lin C.J. (2016) LIBSVM: a library for Support Vector Machines, available at http://www.csie.ntu.edu.tw/~cjlin/libsvm

[17] R Development Core Team. (2008). R: A language and environment for statistical computing, http://www.R-project.org. R Foundation for Statistical Computing.

[18] Powers, David M W (2011). "Evaluation: From Precision, Recall and F-Measure to ROC, Informedness, Markedness & Correlation". Journal of Machine Learning Technologies. 2 (1): 37–63.

[19] Stehman, Stephen V. (1997). "Selecting and interpreting measures of thematic classification accuracy". Remote Sensing of Environment. 62 (1): 77–89.

[20] R documentation webpage, viewed 2017-05-29 at https://www.rdocumentation.org/packages/base/versions/3.4.0/topics/Memory-limits

[21] SPC Oceanic Fisheries Programme (2010), Tuna Species caught in the WCPO, webpage viewed 2017-04-12 at www.spc.int/oceanfish/en/tuna-fisheries/tuna-species

[22] StatTreck.com (2017), Scales of Measurements in Statistics, webpage, viewed 2017-03-01 at http://stattrek.com/statistics/measurement-scales.aspx

[23] Weisstein, Eric W. "Half-Space." From MathWorld - A Wolfram Web Resource. Viewed 2017-05-31 at http://mathworld.wolfram.com/Half-Space.html