Efficient piecewise linear classifiers and applications

Thesis by

Dean Webb

Submitted to the degree of Doctor of Philosophy

A DISSERTATION

Presented to the Graduate School of Information Technology and Mathematical Sciences

University of Ballarat
PO Box 663
University Drive, Mount Helen
Ballarat, Victoria 3353
Australia

Abstract

Supervised learning has become an essential part of data mining for industry, military, science and academia. Classification, a type of supervised learning allows a machine to learn from data to then predict certain behaviours, variables or outcomes. Classification can be used to solve many problems including the detection of malignant cancers, potentially bad creditors and even enabling autonomy in robots.

The ability to collect and store large amounts of data has increased significantly over the past few decades. However, the ability of classification techniques to deal with large scale data has not been matched. Many data transformation and reduction schemes have been tried with mixed success. This problem is further exacerbated when dealing with real time classification in embedded systems. The real time classifier must classify using only limited processing, memory and power resources.

Piecewise linear boundaries are known to provide efficient real time classifiers. They have low memory requirements, require little processing effort, are parameterless and classify in real time. Piecewise linear functions are used to approximate non-linear decision boundaries between pattern classes. Finding these piecewise linear boundaries is a difficult optimization problem that can require a long training time. Multiple optimization approaches have been used for real time classification, but can lead to suboptimal piecewise linear boundaries.

This thesis develops three real time piecewise linear classifiers that deal with

large scale data. Each classifier uses a single optimization algorithm in conjunction with an incremental approach that reduces the number of points as the decision boundaries are built. Two of the classifiers further reduce complexity by augmenting the incremental approach with additional schemes. One scheme uses hyperboxes to identify points inside the so-called "indeterminate" regions. The other uses a polyhedral conic set to identify data points lying on or close to the boundary. All other points are excluded from the process of building the decision boundaries.

The three classifiers are applied to real time data classification problems and the results of numerical experiments on real world data sets are reported. These results demonstrate that the new classifiers require a reasonable training time and their test set accuracy is consistently good on most data sets compared with current state of the art classifiers.

Statement of Authorship

This thesis contains no material which has been accepted for the awards of any other degree or diploma in any university and is less than 100,000 words in length excluding tables, maps, footnotes, bibliographies and appendices. To the best of my knowledge and belief this thesis contains no material previously published by any other person except where due acknowledgement has been made.

Dean Webb

December 2010

Acknowledgements

Firstly I would like to thank my supervisor Assoc. Prof. Adil Bagirov and my associate supervisor Dr. Mammadov. Their guidance, availability and persistance along the way has been invaluable.

I would also like to thank Dr. Ugon and Dr. Soukhoroukova for all their help and encouragement. They are truly good friends and colleagues. All the work that they put into reading and highlighting the mistakes in my thesis, made the writing that much easier.

To all my other colleagues at the University of Ballarat, I would like to thank you for all the encouragement and support that you gave me. When the research seemed to be going no where, it was always good to have people to discuss with. Also I would like to thank all the anonymous referees who read and gave me useful feedback on my papers.

I would like to thank the University of Ballarat for allowing me the oppurtunity to undertake this research. Special thanks to Dianne Clingin for the support that she gave me towards the end of the writing phase.

Lastly but certainly not least, I would like to thank my family for enduring the roller coaster ride that is a Ph.D. Especially to my wife, it is not easy having two young children and a husband studying fulltime. For your patience I am eternally grateful. Thank you Kathryn, Bailey and Sienna.

List of Publications

[1] Adil Bagirov, Moumita Ghosh, and Dean Webb. A derivative-free method for linearly constrained nonsmooth optimization. *Journal of Industrial and Management Optimization*, 2(3):319–338, 2006.

[2] Adil Bagirov, Julien Ugon, and Dean Webb. An efficient algorithm for the incremental construction of a piecewise linear classifier. *Information Systems*, 36(4):782–790, 2011. Selected Papers from the 2nd International Workshop on Similarity Search and Applications SISAP 2009.

[3] Adil Bagirov, Julien Ugon, Dean Webb, and Bülent Karasözen. Classification through incremental max-min separability. *Pattern Analysis & Applications*, 14:165–174, 2011. 10.1007/s10044-010-0191-9.

[4] Adil Bagirov, Julien Ugon, Dean Webb, Gurkan Ozturk, and Refail Kasimbeyli. A novel piecewise linear classifier based on polyhedral conic and max-min separabilties. 2010. Submitted.

[5] Adil Bagirov, Julien Ugon, and Dean Webb. Fast modified global k-means algorithm for incremental cluster construction. *Pattern Recognition*, 44(4):866–876, 2011.

[6] Dean Webb, John Yearwood, Liping Ma, Peter Vamplew, Bahadorreza Ofoghi, and Andrei Kelarev. Applying clustering and ensemble clustering approaches to phishing profiling. *In Proceedings of the Australasian Data Mining Conference: AusDM*, Melbourne, Australia, December 2009.

[7] John Yearwood, Musa Mammadov, and Dean Webb. Profiling phishing activity based on hyperlinks extracted from phishing emails. *Social Network Analysis and Mining*, 1–12, 2011. 10.1007/s13278-011-0031-y.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In today's information age, knowledge discovery is becoming more important than ever before. Industry, military, academia and science all endeavour to learn and then predict future outcomes within their respective fields.

Intelligence agencies want to predict any impending man made catastrophes; companies would like to increase their product buying power or predict stock market trends; banks would like to identify potential loan defaulters; medical science is concerned with predicting accurate diagnoses and the robotics industry is interested in developing further robot autonomy.

Solutions to such problems can be achieved through the data mining task of supervised learning. More specifically through the sub task known as classification. Classification is a type of machine learning where the computing device employs a classification algorithm to learn from data observations, in order to make future predictions.

Over the past few decades the collecting and storing of data have increased at such a rate that the current classification techniques are unable to deal with the large volumes of data. Attempts have been made to reduce the data by applying various transformation and reduction techniques.

These techniques have been met with varying degrees of success, as most of

them depend heavily on the underlying distributions of the data provided. Furthermore, certain classification domains place additional demands on classifiers. One such application domain, is that of real time classification.

Real time classification problems usually need classifiers to have low memory requirements, small training times and to classify data in real time. Generally, such applications demand the use of large scale data, this is especially true when device sensors are sampling data at very high rates. Some examples of real time applications include small reconnaissance robots, automated visual surveillance systems, embedded systems and portable electronic devices. Applications involving embedded systems and portable electronic devices also add power and space constraints, further emphasizing computational and memory efficiency of the classifier.

In most cases current mainstream classifiers achieve good classification accuracy. However, many such classifiers fail to deal with large scale data, have long training times, require human input through the tuning of parameters and can not provide real time classification. As many real time applications deal with large scale data and learn without human intervention, this makes these types of classifiers inappropriate for such applications. On the other hand, it has been shown that piecewise linear (PWL) classifiers are ideal for real time classification [87].

PWL classifiers can provide real time classification with low memory, processing and power requirements. They are also very simple to implement and do not contain parameters that depend on the data set. However, the determination of PWL boundaries used for the classification process can be a complex global optimization problem [141].

The objective function in this problem is nonconvex and nonsmooth where Newton-like methods cannot be applied. It may have many local minimizers, where only global minimizers provide PWL boundaries with the least number of

hyperplanes. Furthermore, the number of hyperplanes needed to separate sets is not known a priori. As a result of these difficulties, PWL classifiers can require a long training time which creates challenges for their practical application.

In order to overcome these problems and reduce training time, most PWL techniques try to avoid solving optimization problems when computing piecewise linear boundaries. Many multiple optimization approaches apply different forms of heuristics to determine and compute the number of hyperplanes [18, 32, 65, 87, 118, 142, 141, 149]).

Often these algorithms are quite fast, however, they do not always find the global minimizers of the classification error function. Working with large scale data only intensifies the problem as the number of minimizers of the classification error function increases.

This thesis develops three real time piecewise linear classifiers that deal with large scale data. Each classifier uses a non smooth single optimization algorithm to calculate the number and placement of hyperplanes for the decision boundary. Learning is based on an incremental approach that employs heuristics to decrease the number of data points,while building a decision boundary for the separation of classes.

Incremental approaches have been shown to be efficient problem solving tools [127]. Such approaches can be applied effectively for the construction of PWL boundaries, as information from previous iterations can be built upon. Building on information can reduce complexity and aid in the determination of an optimal number of hyperplanes.

Furthermore, two of the classifiers further reduce complexity by augmenting the incremental approach with additional schemes. One classifier uses hyperboxes to identify points inside the so-called "indeterminate" regions. The other algorithm uses a polyhedral conic set to identify data points lying on or close to the defined boundary. All other points from both schemes are excluded from the

process of building the decision boundaries.

Results of numerical experiments using a number of real-world data sets are reported. These results demonstrate that the new algorithms consistently produce good test set accuracy and are trained in a reasonable time. These results are compared to a number of other classifiers, where all experiments were run on real world data sets.

This thesis is organized as follows: Chapter 2 presents a comprehensive review of the Knowledge discovery in data bases process. This process highlights the tasks needed for data mining, including data collection, data storage, data processing and data mining itself. A comprehensive review is given on the classification process. A data mining software package is presented where the classification process is shown in practice using a number of classification techniques.

This chapter also presents a review on real time classification, where a number of classification constraints are highlighted. The multi-class problem is presented and a review of solutions is given. This leads onto a review of a number of main stream classification techniques, where problems associated with large scale data are highlighted.

Chapter 3 gives a brief explanation on optimization and then moves into piecewise linear classification, where a description is given. The next sections provide a detailed overview of multiple optimization and single optimization piece wise linear classification techniques. This critique highlights the advantages of these classifiers, but also points out their weaknesses in terms of large scale data and long training times.

An incremental approach is given, highlighting the benefits of algorithms that deal with large scale problems sequentially. This then leads onto schemes that focus on data, where a review on data pre-classification is given. A number of schemes are presented where the benefits of pre-classifying data is made relevant to dealing with large scale data.

The next three chapters present the three real time piece wise linear classifiers developed in the thesis. The first one, given in chapter 4 introduces the idea of incrementally building a piecewise linear boundary from a least complexity approach.

Chapter 5 extends this idea by introducing polyhedral conic functions. The polyhedral conic functions are used in conjunction with the incremental algorithm, where points are removed before the incremental algorithm is applied.

Chapter 6 uses an alternative idea to that of polyhedral conic functions. Instead, hyperboxes are used to eliminate points between each pair of classes. As with the use of polyhedral conic functions, hyperboxes eliminate points before the incremental approach is applied, thus reducing complexity.

Results of numerical experiments are presented in each of the these three chapters and this is followed by a discussion that draws a conclusion from the presented results.

Finally chapter 7 concludes the thesis by drawing on conclusions from the results of the three classifiers. Further research ideas are presented, and based on the three classifiers future work and research directions are highlighted.

Chapter 2

Data mining and classification

2.1 Introduction

It is natural for Human Beings to abstract objects into groups based on certain characteristics. When a human comes in contact with an unfamiliar object, identification would be made on a closest fit to one of the previous defined groups. Early nomadic tribes people serve as a good example of this as they learned the differences between edible and poisonous plants. Learning the characteristics of both groups of plants allowed them to quickly identify unknown plants likely to be poisonous. "The ability to distinguish between objects is the fundamental to learning intelligent behaviour in general."[25].

Today we are able to identify a plant or animal by its physical attributes and functionality using a dichotomous classification tree, Systema Naturae [97]. This work dates back to Aristotle and then Karl von Linne in the 18th century. Although refinements have been made, this hierarchical classification model is still being used today. Its primary function is to identify different plants or animals. If part of the taxonomy is missing or a contradiction is found then additions or alterations to the model can be made. This highlights the importance of the classification process. We can store what we have learnt, quickly identify the

new unseen or update the unknown or contradicting information. However, work done by von Linne (Linnaeus) and other scientists since the 18th century has been laborious and tedious. Painstakingly all found plants and animals were manually recorded, collated and then added to the classification model.

With the advent of the computer these manual processes have now become automated. Computers have allowed us to minimize the work load in terms of collecting and storing data. We are also able to analyse, make predictions and find useful patterns from the data. The classification task is performed by the computer where it learns from the data in order to later classify unseen objects. This is known as machine learning and can be used in all types of domains. In business, classification can be used in customer segmentation to identify customers who would be a credit risk, or others who would be likely to pay their entire monthly credit card balance. In medicine, researchers may want to predict a patient's response to a drug, or determine the likelihood of a patient having breast cancer.

Data Mining is the field of work which deals with collecting, analysing, predicting and learning from data. Machine learning has become a very important field in data mining where many different classification techniques can be found. These include techniques that are based on trees, statistics, mathematical optimization, heuristics and many others. With as many classification techniques, come many more classification problems which need to be solved. One such problem domain can be found in real time classification. Many applications for real time classification can be seen in science, industry and academia. These include robotics, embedded systems, facial recognition, prosthetic automation and many more.

Although technology has allowed for many advancements in the field of data mining, it has also inadvertently created problems. Computer processors and memory have become progressively cheaper and quicker. This has allowed for

increased amounts of data to be stored, but also quickly retrieved. Consequentially the desire for data collection is now matched by the capability of technology, where vast amounts of data is being collected. However, the amount of data being collected is disproportional to our ability to utilize it. Current machine learning techniques are becoming less capable when dealing with large volumes of data.

This is especially the case in real time applications. Many real time systems contain sensors that can sample data at very high rates, thus generating masses of data over a very short period of time. In terms of sensor collected data, it has been shown that even as early as the late 1980s, *automated* collection, storage and better retrieval methods led to large amounts of data being collected [136]. This trend has only intensified, where memory storage capacity has increased by a factor of a million and its price has decreased accordingly.

Many mainstream machine learning techniques currently used today have problems when dealing with large scale data. Often, only subsets of the data can be used for training, and in terms of classification, most classifiers are unable to classify data in real time. However, it has been shown that piecewise linear classifiers are ideal for real time classification [87]. Piecewise linear classifiers can be broken up into two main groups, multiple and single optimization piecewise linear classifiers. The multiple optimization approach [142], [141] can suffer from accuracy problems and long training times. On the other hand, the single optimization approach theoretically is shown to be ideal for such an application [9] and [5].

This thesis investigates the ability of a single optimization piecewise linear classifier to solve real time large scale classification problems. It builds on previous work grounded in mathematical optimization, using the max-min optimization technique [7] to minimize a classification error function that incrementally builds a piecewise linear decision boundary between classes. It highlights the challenges presented in solving a single optimization problem that deals with non smooth-

ness, non convexity and finding global solutions amongst a large number of points. It also highlights the benefits, as max-min functions only take up little memory and allow for classification to happen in a fraction of a second.

Therefore, can a single optimization piece wise linear classifier be used to solve real time classification problems using large scale data?

In order to understand and justify this question, some context must be given to the process of classification, real time applications and the growing presence of large scale data. Data mining is the link between all three and is the encompassing field that deals with all tasks and applications that attempt to convert useful knowledge from masses of collected data. It is, therefore, a good starting point for this literature review.

Finally, it is worth noting that this thesis considers large scale data as hundreds of thousands of observations containing thousands of features, with no more than a hundred classes.

2.2 Data Mining

2.2.1 History

Data mining can be traced along three lines, these include statistics, artificial intelligence and machine learning.

Statistics which is its longest and strongest contributor is at the core of its foundations. Such things as regression, standard distribution, discriminate and cluster analysis were some of the original techniques used to analyse data.

Artificial intelligence, or AI was next to follow. At this time computing power and memory had increased significantly. Also scientists were becoming more interested in techniques other than just statistical based ones to analyse and use data.

This led to a shift in wanting computers to analyse and behave in terms of

the way humans think. Philosophy, psychology and computer science drove the whole AI concept. Such techniques as artificial neural networks and intelligent agents were born from this paradigm.

AI, however, was a discipline strongly built on heuristics where many of the underlying structures used were rule and tree based techniques. Trees are generally exponential in storage capabilities, and rule based systems suffered similar problems when used in the decision making process. This meant that learning and decision making could only be used on small problem sets, unless more computing power and memory was found. The computing power and memory needed for such applications during the early 80's was very expensive and hard to gain access to.

Machine learning was the next to follow and was born out of the problems associated with AI techniques. Machine learning encompasses statistics, heuristics and mathematical techniques in its quest to learn patterns from the data. Generally machine learning is an inductive learning process where statistics, heuristics or mathematical techniques, or any combination of three learn based on examples from a given data.

To date, data mining has become an interdisciplinary field. Increased technology and the demands for more information from other disciplines have now meant that data mining embraces more areas than ever before. These include data base systems, statistics, machine learning, visualization and information science. Depending on the type of data to be mined, or the data mining application this could include such things as spatial data analysis, information retrieval, pattern recognition, image analysis, signal processing, web technology and many other applications.

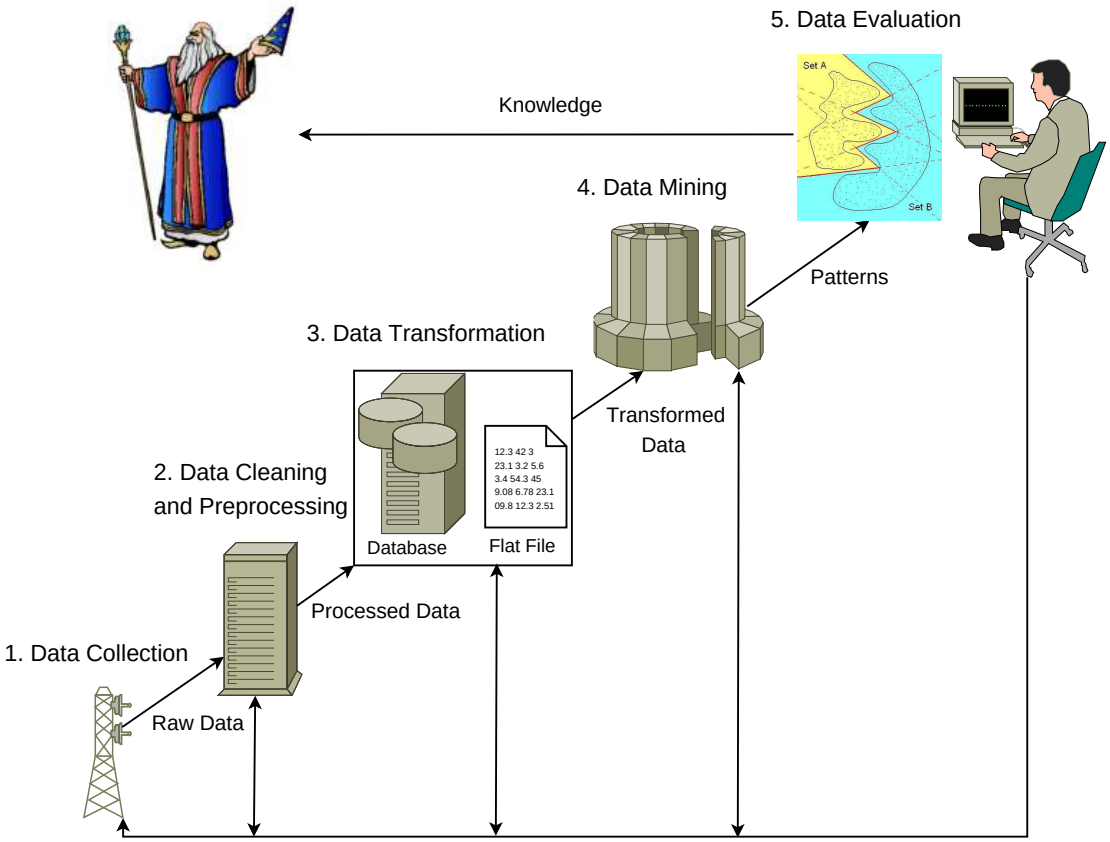


Figure 2.1: **Knowledge discovery Process**

2.2.2 Knowledge Discovery in Databases

Data mining can be considered as the extraction of useful information or knowledge from large amounts of collected data. This useful information could be a model, summary or just derived values pertaining to a problem definition [82].

Recently, the definition of data mining has become synonymous with the process of Knowledge Discovery in Databases (KDD) [61]. Although the KDD process actually includes data mining as one of its steps, in this thesis we use the term data mining to mean the KDD process itself. I include the KDD process in this thesis to provide background to the work that must be undertaken before any classification algorithm can begin training or be used as a classifier.

Data mining can be seen as an iterative process, where many steps are performed to extract and refine knowledge from the data provided. This process is

synonymous with the algorithms developed in this thesis, as they also iteratively refine data in order to reduce training complexity and improve the accuracy of the classifier.

As shown in figure 2.1 the Knowledge Discovery in Databases is an in depth and iterative process. The main flow of this process is from bottom left to top right where each previous step is a prerequisite to the following step. However, once the data mining and evaluation steps have been performed any of the previous steps can be revisited any number of times.

Knowledge discovery is an iterative process and rarely is a problem defined precisely enough that all steps are traversed only once. It is also rare that only one data mining technique alone is used. Often the results from a number of data mining techniques are evaluated and expert opinion maybe used to determine the success of the results. Based on these results, data may need to be modified using a number of statistical and analytical techniques. The KDD process may then be reapplied using the updated data, using new or the same data mining techniques and again the results are re-evaluated. This process can happen many times, to the point where even the problem definition may change or new data may need to be collected.

The KDD process has been described as a voyage of discovery where at each new iteration of this process more relevant knowledge maybe unlocked. The following subsections represent the steps in the KDD process where a detailed description is given.

2.2.3 Data Collection

Data can be collected in many different ways. There is a large amount of pre-existing data that can be sourced from repositories all over the world. Often companies or academic institutions have collected data relevant to solving other institution or company's problems. However, questions maybe raised to the in-

tegrity and the means used to source the data. Even being part of the data collecting process may include a number of repeat collections, before useful patterns can be found.

The methods used to collect data will depend on the application and problem definition. Social scientists generally collect data via observational techniques. This is a systematic process of watching and recording behaviour and characteristics. Another method is interview based, where questioning of respondents is done either individually or as a group.

When humans are involved in the collection process, possible errors can be made. Respondents or observers could be tired or distracted leading to incorrect values being recorded. Survey questions can be ambiguous or misleading which can lead to incorrect, redundant or no answers at all.

Generated data is another type of data, usually sourced via sensors measuring certain activities, or data generators designed to create certain distributions for testing or simulating problems. Generally this is an automated process, where large amounts of data can be generated. High sampling rates and fast data generation programs lead to large amounts of data being collected or created.

The collected data must be stored in such a way that relevant data remains related to each other and can be called up to be used quickly and easily. With large amounts of data sourced, it is possible that the data may be stored on multiple machines and over multiple sites. Many storage solutions such as databases, data warehouses or other information repositories are currently being used.

Data warehouses offer the ability to have multiple data bases linked to one another. Retrieval of relevant data is efficient and in a format suitable for the clients. Some warehouses even offer the ability to have the data mined. Data can be stored in many ways, these include spread sheets, a flat file containing related measurements, many types of databases or even just raw data. For more information on data warehouses refer to [84] and [47].

2.2.4 Data cleaning and preprocessing

The previous step shows that erroneous values can make their way into data. During manual collection, humans due to tiredness, boredom or inadequate surveys may make errors. Automated collection may incur problems such as damaged sensors, incorrectly calibrated measuring equipment or measuring devices exposed to electromagnetic interference. These errors manifest themselves as missing, noisy or outlier data, and may cause problems when data mining is performed.

Outliers are values that are not consistent with the values contained in other observations. They occur infrequently, generally only representing about two percent of all measurements pertaining to the observations [82]. Noisy data may sometimes be included as outliers, though only in the case of intermittent occurrences.

Noisy data is found more often in automated collection. It is more problematic with sensor collected data, as random noise can occur both as bursts or just intermittently. Noisy data can contradict values that describe classes, breaking down the natural pattern correlations and decreasing the ability to distinguish between the certain patterns. It is worth noting however, that irrelevant attributes contribute no noise as their values are not indicative of any inherent patterns.

There has been a lot of work undertaken in data analysis, preprocessing and preparation. As a result, a host of data analysis and preprocessing tools are available. [13] proposed a new method for dealing with missing values by using an imputation method based on the k nearest neighbour algorithm [116]. [13] also reviewed a number of existing methods, these include ignoring missing values; disregarding instances containing missing values and maximum likelihood procedures [42].

[66] presented a comprehensive survey of contemporary outlier detection techniques. These include a large array of statistical, neural and machine learning techniques. [66] highlight the advantages and disadvantages of each technique

and point out that an algorithm's suitability is based on the data's distribution model, correct attribute types, scalability, speed and modelling accuracy. [66] also show the benefits of removing observations in terms of noise and data size complexity. This is further shown in [98] which employed optimization to minimize the data size while performing data mining. The authors point out that this strategy maintains the mining quality and reduces complexity.

Data reduction can be crucial in improving the performance of a data mining technique, both in speed and its effectiveness [165]. Though there are many techniques for reducing the number of observations [126], there are also numerous techniques for selecting and removing attributes. Reasons such as complexity reduction, poorly chosen attributes containing noise, little information or too many missing values are good candidates for such removal techniques.

The authors in [134] reviewed a number of attribute selection techniques to deal with large dimensional data sets in bioinformatics. The review incorporates many techniques, including filter, wrapper and embedded based. This review highlights the benefits and disadvantages of such techniques, focussing on over fitting, prediction performance, computational complexity and attribute dependencies. Highlighted in the review, are the problems associated with classification techniques attempting to learn patterns from large dimensional data. Furthermore, the review shows that any supervised learning model using such data needs to incorporate attribute selection as a step in the process.

Identifying and removing redundant or irrelevant attributes is a complex problem where finding an optimal attribute subset is intractable [81]. [165] use both attribute relevance and redundancy to build a frame work to approximate an optimal subset. Alternatively, there are many other strategies that construct attributes from a basic attribute set [101]. Such strategies attempt to avoid attribute dependencies during construction as this can influence the accuracy of the classifier.

Furthermore, to hight light the importance of this step, [168] shows that approximately 80% of the total data engineering work is spent on data cleaning and preparation. They show that most data mining techniques expect data to be nicely distributed, contain no missing or incorrect values and only informative features remain. If data preprocessing has not been applied, then it is likely that useful hidden patterns will be disguised leading to the decreased performance of the data mining techniques. The overall benefits of preprocessing means a smaller, cleaner and higher quality data set which yields more concentrative patterns.

A detailed description of attribute selection strategies can be found in [61] and a number of data preprocessing techniques can be found in [159].

2.2.5 Data Transformation

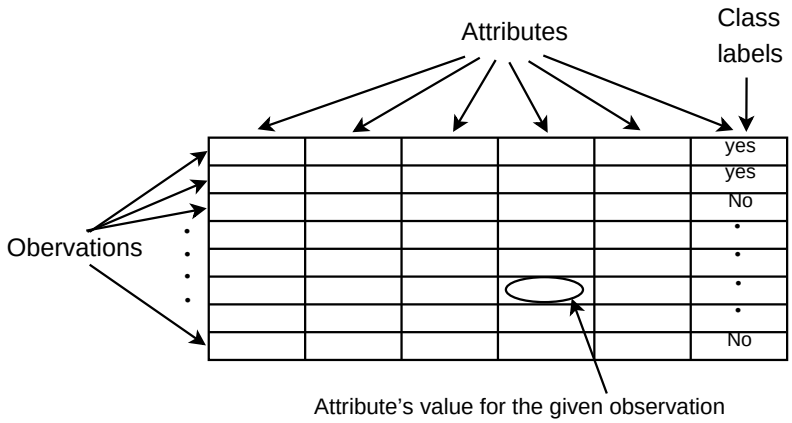


Figure 2.2: **Data in a tabular representation for classification**

Before data mining can be performed, the preprocessed data needs to be formatted in a way that best suits the data mining technique. As mentioned in the Data Collection step, data may have been stored in one or many different types of data bases. However, this may not be suitable for the data mining technique to read in the data. Certain programs or data mining applications expect data in a certain format. As an example, some Support Vector Machine (SVM) classifiers [26] expect rows containing feature values split by colons. The

attribute number is on one side and the attribute value is on the other, showing which value belongs to which attribute.

Further transformations may include the conversion of category labelled data into certain memory allocated sized integer or continuous values. Such data transformations are implemented so as to improve the performance of the classification algorithm. This can allow for quicker parsing of the data; a better memory structure to contain the currently needed values from the data set; and to allow for the use of more efficient data structures by the classification algorithm. Though such tasks seem trivial, much time can be spent formatting data sets for the currently nominated classification technique.

Generally machine learning techniques are trained using a two dimensional flat data file containing rows of measurements. This format is shown in figure 2.2. Each row contains a group of features or attributes, the row is known as an observation or an instance. The observation or instance is the recorded collection of measurements that together describe an instance of an entity or event. The data set contains n rows and m columns, used as collective evidence for the machine learning algorithm. In supervised machine learning each observation will have a class label attached to it. Often this is the last feature column. Class labels are generally assigned manually by an expert in the field.

As an entity or an event is described by a collection of attributes, it is often the case that the scale of magnitude between the attribute values may vary greatly. As an example, one attribute may have a value range of $[0, 1]$ and another may have a value range of $[-1000, 10000]$. In this case, the magnitude of difference is significantly greater between the two attribute measurements and depending on the data mining technique, a greater weighting maybe given to the second attribute.

Normalization is a process often used to scale attribute values so that they fall within a small specified range. This helps to reduce any bias between the

attributes. It is also particularly useful for classification techniques that rely on distance measures, often improving their performance [139]. [139] review a number of scaling techniques, the following are just a few of the commonly used techniques.

Using data set A containing i observations and j attributes, normalization is performed on attribute A_j:

min-max is a linear transformation on A_j.

$$A'_{ij} = (A_{ij} - min(A_j))/(max(A_j) - min(A_j))$$

z-score is based on the mean and standard deviation of A_j.

$$A'_{ij} = (A_{ij} - mean(A_j))/stdev(A_j)$$

decimal scaling moves the decimal point of value A_{ij}.

$$A'_{ij} = (A_{ij}/10^p)$$

where p is the smallest integer such that $\max(|A_j|) < 1$

The choice of the normalization technique can depend on the classification algorithm, the type of data collected and the effect of any other preprocessing techniques used on the data. As an example [151] present a comparative study on different normalization techniques for the use in discriminating odours in black tea. They evaluate these normalization techniques based on the Principal Component Analysis (PCA) [140] preprocessing method for a back- propagation multilayer perceptron classifier [28]. Their results showed that normalizing the data improved accuracy over using raw data. Furthermore, their results showed that the min-max normalization technique allowed for the best classification accuracy.

The classification algorithms developed in this thesis use the min-max normalization technique. Our previous experiments showed that the best classification accuracy was gained using this technique. Furthermore, a number of accounts can be found in current literature to support these findings. As an example, [139] show that a number of classification algorithms, including k-nearest neighbour [49] and tree based classifiers, performed consistently better using the min-max normalization technique over other normalization techniques.

2.2.6 The data mining task

Data mining is the next stage in the KDD process and in essence is an analysis of the data. Given the prepared data, what can be learnt from this data. In practice, there are two primary goals of data mining. These include prediction and description.

Prediction uses a model or rule derived from the data to predict unknown or future variables of interest. There are a number of data mining tasks that can fulfil these two goals. For example, predictive tasks may include classification, which is described in this thesis. Another task is regression, which finds a predictive learning function that maps to real value prediction variables.

Description on the other hand, focuses on finding the inherent patterns in the data. Examples of such tasks may include clustering, which seeks to identify a finite set of categories or clusters to describe the data. Also, summarization, which is a descriptive task that uses methods to find a compact description for a set or subsets of the data.

There are many data mining tasks that can be performed. However, this thesis focusses on the process of classification and current classification techniques relevant to this research. There is some intersection between tasks, as a few classification techniques also include regression or clustering as part of their classification technique. However, this is addressed later in the thesis. For detailed

explanations on all other data mining tasks refer to [46].

As classification is a type of supervised machine learning, the following subsections give context to machine learning and supervised learning tasks. These subsections serve to provide context to the processes and models used for classification under the confines of data mining and the KDD process. Also, they highlight the role of computers as automated predictive tools and how they can learn from data.

Machine learning

The field of machine learning (ML) develops computer programs to induce inherent patterns found in the data. These patterns can be derived from rules or models to either find certain behaviours or relationships, or be used in predicting unseen objects or events. Machine learning algorithms learn by searching through an n dimensional data set space attempting to find an acceptable generalization.

One of the most fundamental machine learning tasks is inductive learning. It is defined as the process of estimating an unknown input/output dependency between a finite number of measurement based observations. Hence, the machine learning algorithm uses a priori knowledge (training data), to build a selected class of approximating functions or model that describes the generalized mapping between the inputs and outputs for future prediction.

Machine learning can be broken up into three main categories. These include supervised learning, unsupervised learning and reinforcement learning.

Supervised learning The data has class labels assigned to its observed measurements. The ML algorithm learns the mapping between the input vector and output class label of each observed measurement. Then it uses these mappings to predict the class label of unseen observations. This conforms to the two classical inference learning types:

1. Induction - progressing from particular observations (training data) to

a model or rule.

2. Deduction - progressing from the model or rule, given input values (test data) to particular cases of output values (classes).

Unsupervised learning The data has no class labels assigned, instead the ML algorithm searches for relationships within the data. The ML technique attempts to learn the inherent structures within the data by grouping together examples given a similarity measure.

Reinforcement learning This type of learning pertains more to artificial intelligence. ML agents learn and then act within a simulated environment created using the data. Given delayed rewards for appropriate behaviour and penalties for inappropriate behaviour the ML agents learn ideal behaviour to maximize the ML technique's performance. It is roughly based around the carrot and stick methodology.

Classification (Supervised learning)

The classification problem is one of the most widely studied within the field of machine learning. Also known as supervised learning, classification is the process of finding a model or function that learns a mapping between the set of observations and their respective classes.

As mentioned previously, the purpose of a classification model or function is to predict the class label of any unseen observations. The step of data mining allows for the classification problem to be solved with many tools. The WEKA [159] data mining application has many machine learning algorithms, including a large number of classification algorithms. Often, the question arises to which classification algorithm is the best to use, given the data set. Furthermore, which set of parameters, if any, are more appropriate given the classification algorithm.

Presented next is a classification process which is a structured iterative pro-

cess used to solve classification problems. One of its main aims is to help address the problem of choosing an appropriate classification algorithm and an optimal parameters iteratively. Furthermore, this classification process can be modelled directly using data mining software. This includes the WEKA data mining application presented in subsection 2.2.8.

Classification process

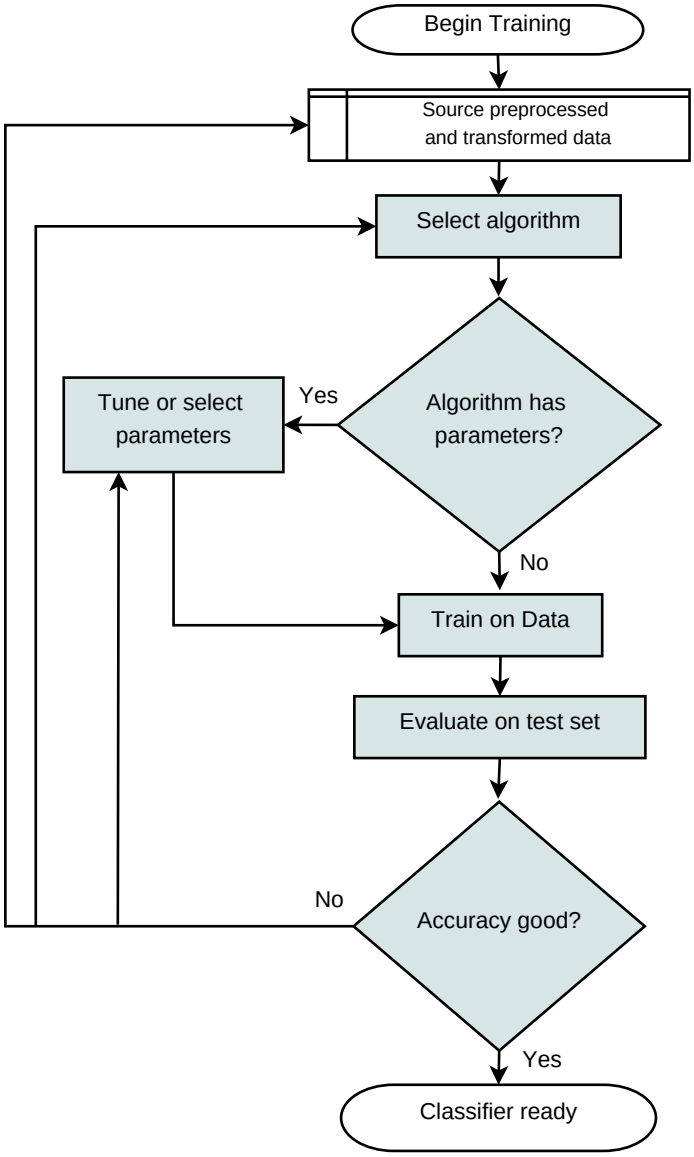


Figure 2.3: **Process of classification**

The process of classification of real world problems is shown in figure 2.3. The

data being used in this process is part of the KDD process and assumes iterative changes both in preprocessing and transformation. One can view this classification process as the data mining step. Failure to build an accurate and reliable classifier from the given data may mean that the preprocessing and transformation steps are performed again.

Furthermore, the classification model itself must be validated and verified to ensure that the training process is robust and non-biased. A number of classification measure accuracies can be used for this comparison. These measures may be dependent on the type of classifiers and problem domain. A number of statistical methods for comparing multiple algorithms on multiple datasets can be found in [43]. To prevent an infinite number of iterations with little to no progress, a comparison of these measures after each training phase or a maximum iteration threshold may need to be set.

All the shaded steps in figure 2.3 relate to the following subsections.

Algorithm selection

Often a review of classification algorithms is done before any supervised machine learning is undertaken. It is likely that several classification algorithms will be needed to determine the best prediction model or rule. The most appropriate classifier may be dependent on the selection of classifiers on offer, data type, problem domain and problem definition.

Brain-computer interface systems [160] is one example of a problem domain. These systems look at captured brain wave data that is both time dependant and contains continuous values. A comparison between classification algorithms for suitability in this domain is performed [99]. The authors point out that when choosing an appropriate classifier, the properties of the classifiers must be known. They create four definition groups for the classifiers based on their properties and assess them given the problem domain and data type.

Once a selection of classifiers has been decided upon and preliminary testing has been done, an evaluation can be made through training and testing. This evaluation is generally based on a prediction accuracy or training time measures of the classifier. This preliminary testing can serve as a good guide to which family of classifiers works best on the current data. By sampling classifiers from a number of families, it is possible to gauge if certain individual classifiers will respond poorly to the data, hence allowing for their omission from selection.

Algorithm training

Training on the data can be done using a number of techniques, where the data is divided up into training and test sets. Generally with large data sets, it is considered acceptable to divide the data set into two thirds training and one third test. However, in smaller data sets, other strategies are employed. This can include the use of cross-validations, where the data is randomly broken up into n number of subsets and trained iteratively. Presented next are three re-sampling techniques used for training on data.

The first technique that can be used is the holdout method which divides the data set into two thirds training and one third testing. This technique is computationally preferable for large datasets. If the classes are divided proportionally then the classification algorithm will have a balanced sample of training and testing observations to gain an accurate rate of prediction. The observations selected from the initial data set are normally chosen from an uniform distribution and allocated proportionally to the training and testing subsets.

The next technique often used is cross-validation. Cross-validation works by dividing the data set up into n equally sized mutually exclusive subsets. A training set is the union of $n-1$ subsets, in which the remaining subset is the test set. Training and testing is done n times, where all subsets have been included in each union. To estimate the classification error, the average is taken over n

predictions of the test sets. This technique works well for medium sized data sets, as there is a balance between the number of computations and the classifier learning from re-sampling the data.

The last popular training technique worth considering, is the leave-one-out validation. This technique is very similar to the cross-validation technique. In fact it is considered as a special case, as the n subsets in this technique are actually single observations. This technique is computationally intensive and is best suited for very small data sets. It has been shown to be the most accurate estimate of the classifier's error [88]. However, due to its computational complexity, it can not be considered when dealing with large scale data.

[85] presents a comprehensive survey of cross-validation and boot strap methods. The author highlights problems such as incorrect sampling and the use of incorrect training methods. These problems can lead to over fitting or poor estimates of the training data. As the evaluation of a classification algorithm is generally based on prediction accuracy, it is important that the classification algorithm learns from a well proportioned cross section of the data. Furthermore, [85] states that choosing the correct technique is based on the size and type of the data and the type of classification algorithm.

The three classifiers developed in this thesis use the hold out method of dividing the data set into two thirds training and one third testing.

Other problems that can be encountered while training may include an imbalance in the number of classes, i.e. a larger proportion of instances for one class compared to the others. It has been found that certain datasets derived from nature contain considerable noise and have class imbalances, leading to a lower accuracy compared to other data sets [60].

Often a balance must be found between the type of classification algorithm, parameters chosen and data sampling method used. As mentioned earlier, some algorithms perform better on certain types of data. Badly suited classifiers or

poorly chosen parameters may cause problems for the classification algorithm to converge, in turn leading to a poor accuracy or long training times.

Selecting and tuning parameters Certain classification algorithms need parameters passed to them before they can begin training. As an example, support Vector machine classifiers need parameters to find the best approximated kernel to divide the classes. This can be an iterative process of trial and error where the updating of the parameters will lead to better accuracy and faster learning. As an example, a parameter grid search tool for the Radial Basis kernel support vector machine can be found in [69].

It is worth noting however, that given the combinations of optimal parameters needed for a classification algorithm, finding the optimal parameters can be a tedious process. Parameters can not be altered until after the algorithm has trained and tested for accuracy. Given the size of the data, the training could take a long period of time. Furthermore, if the initial parameters chosen are far from optimal, then the time taken for the algorithm to converge may also be extended.

Generally, choosing the correct parameters is an iterative process based on the previous accuracy where a better set of parameters can be chosen using positive feedback. The stopping criterion is generally based on a consecutive run of decreasing accuracy measures or when a predefined accuracy value is reached. There are a number of tools to aid with the choice of appropriate parameters, but these can be limited and can also be time consuming [163].

2.2.7 Pattern evaluation

Pattern evaluation is a crucial step in the whole KDD process. Potentially the data mining process can generate thousands or even millions of patterns. It is therefore reasonable to conclude that only a fraction of the generated patterns

will be useful. It is imperative that the data miner understands the problem domain well and that a problem definition has been well constructed. When the data miner is reviewing the results, the problem definition should be in the back of his or her mind. Calling in an expert at this stage to help understand and verify the results will be beneficial.

The results from the data mining model should ultimately help in the decision making process. These results need to be interpretable, as "black box" models can not be justifiable. The ultimate outcome would be an interesting pattern that is easily understood, valid on test data with a degree of certainty, useful and in the case of descriptive tasks, is novel.

In terms of classification, learning the class patterns more precisely is important. Different classification techniques work better on different problem domains with different data types. It is then up to us, the humans to try other classification techniques or data transformations.

It is also possible that experts who have classified data or verified patterns may have overlooked certain relationships or variables. It may be that the experts have to reclassify, re-evaluate or reissue data. It is not unusual for experts to learn more about their respective field by performing analysis on data from their field.

This step is also about defining metrics that can be used to determine the validity of the results. Objective and quantifiable measures based on the structure of the pattern and statistics underpinning them can be used to determine the success of the results. An overall criteria for comparing algorithms could be based on the following:

- Predictive accuracy - How well the classifier can predict the class label of new unseen data;
- Speed and memory - Computer resources used by the classifier during training and classifying;

- Robustness - How well the classification algorithm can deal with all types of data, including noisy data.
- Scalability - The ability for the classification algorithm to deal with increasing amounts of data.
- Interpretability - Being able to understand the model or rule provided by the classifier.

This thesis develops three real time classification algorithms for use in large scale data problems. We are therefore, interested in the time taken by the classifiers to train and classify, the amount of memory needed to store the classifier and finally the classifier's predictive accuracy. Other than measuring time, the predictive accuracy is a relatively easy performance measure to quantify. Some performance measures include the ROC curve [21] and the F-score [31]. For a detailed survey of accuracy measures refer to [30].

Once the results are acceptable or the experts are happy with their evaluations then the knowledge can be presented. This may be in the form of reports or statistical representations. In the case of classification, it means that the computer is ready to classify unseen examples for the purpose intended. This also becomes an iterative process as the data evolves or changes over time and the KDD process will need to be run again to adapt to the change in the data.

2.2.8 WEKA

The name WEKA stands for Waikato Environment for Knowledge analysis (WEKA). It is a data mining software suite developed by the Department of Computer Science, University of Waikato, New Zealand [159]. Weka is an open source [20] application written in the Java programming language [17] and has been funded by the New Zealand government since 1993.

It was designed primarily as a work bench for machine learning tools to determine the necessary factors for the success of agriculture industries. Its secondary aim was to develop new methods and then assess their effectiveness. This has led to a data mining application that boasts many learning algorithms, preprocessing and post-processing tools and an intuitive graphical user interface.

It is widely used in research and education but also now being adopted by industry for data mining applications. It is an out of the box application that can be used immediately after installation.

There are various ways of interfacing with the WEKA software in order to tackle the data mining problem presented. As mentioned previously data can be a voyage, possibly one of exploration. Hence, one needs to explore the problem given the presented evidence. In our case, it is in the form of data. It is then prudent that the user begins the data mining process by using the application's 'Explorer' graphical user interface. The user interface presents a well laid out menu offering the following tasks. They include:

- Preprocess
- Classify
- Cluster
- Associate
- Select Attributes
- Visualize

These tasks roughly mimic the steps shown in the KDD process 2.2.2. In terms of the collection step, WEKA can connect to databases, however it does not allow for a storage solution itself. Data must already be collected and stored else where. It must also be converted into a flat file format so it can be read in by WEKA.

The preprocessing step includes an inbuilt program that transforms data into their so called "ARFF" file format, where a description of the data is at the beginning of the data set. Other preprocessing tasks include a group of filters that can discretize, normalize, resample, transform, combine attributes and many others. All of these filters ensure that a method is on hand that allows for an ideal conversion of the data that would be optimal for the machine learning algorithm. It also allows the user to find better insight into the data, as each transformation gives another perspective of the data.

Further preprocessing tasks pertain to the training of classification algorithms. The data can be randomly split into subsets and trained on each subset individually. This is explained in more detail in the algorithm training subsection of the classification model, subsection 2.2.6.

As far as the data mining step is concerned, all of the machine learning algorithms are contained in the classification, clustering and rule association menu sections. There are many algorithms contained in WEKA that fall under these three banners. In this thesis we are concerned primarily with classification algorithms. Most of the mainstream multi-purpose classification algorithms found in WEKA are presented in section 2.5. They are broken up into statistical, heuristic and mathematical optimization based algorithms.

In WEKA, once the user has chosen a classification algorithm, often configurable options are needed for the algorithm to perform well. These are parameters that may include, seed value, number of epochs, learning rate, time and many more. The type of parameters needed generally depend on the classification algorithm. However, WEKA provides default parameters so that all algorithms can run.

Also, there are options to determine types of output and accuracy measures. Charts and other outputs including a scatter plot, the actual model, i.e. an artificial neural network showing the input, output and hidden nodes with all weights

can be viewed. Furthermore, the WEKA package allows for the performance measures to be combined with the plots and the current classification model. This is to aid the user in determining the best model or rule for the classification problem.

WEKA has a comprehensive suite of preprocessing tools, for example it contains a set of attribute selection methods. These methods are broken into two groups. The first group include search methods that search through the attributes in an attempt to find the best attributes based on a number of criteria. The second group contain evaluation methods, which evaluate attributes based on such metrics as information gain and correlation. Both sets of methods aim to keep only the attributes that contribute to the inherent patterns while removing noise.

In terms of visualization WEKA also offers visualization of the data. All attributes and observations can be mapped onto a 2 or 3 dimensional space. The user can visualize how points and classes relate to each another. This gives the data miner insight into how the points are distributed or related. This also aids the data miner in determining what if any more preprocessing of the data is necessary. It may also give insight into what data mining method would be more appropriate given the data.

As data mining is an iterative process of refining the pattern search, WEKA offers the ability to iterate through the menu tasks. Furthermore, it allows for all previous preprocessing and machine learning algorithm's output to be stored. Hence, allowing for the ability to incrementally improve on the classification problem, by improving on the data; using more appropriate preprocessing tools and using more appropriate machine learning algorithms.

WEKA also offers an experiment environment where the classification algorithms can be automated and batched using different parameters and different preprocessed data. Analysis can be done after all tasks have completed, where visualization of the tasks is possible. This experiment environment saves the data

miner time, as all the tasks can be run off line without human interaction.

Finally, WEKA offers a knowledge flow interface that can mimic the KDD process graphically. Data sources, classifiers, evaluators, visualizers and summarizes can all be connected together via a flow diagram. A comprehensive data flow can be seen in diagram 2.4.

The main flow shows the data set being divided into subsets, for cross flow validation. It then has attributes selected for it, before it is sent to a support Vector machine classification algorithm for learning. It then has its output sent to a performance evaluator and then has that information output to a text viewer for evaluation.

Furthermore, it can be seen that the data is being sent to a visualizer, summarizer and plotted into a scatter plot matrix. These layouts can be saved and reused or changed at a later date. The knowledge flow interface completely automates an iteration of the KDD process. All entities can be changed, replaced or moved after each iteration.

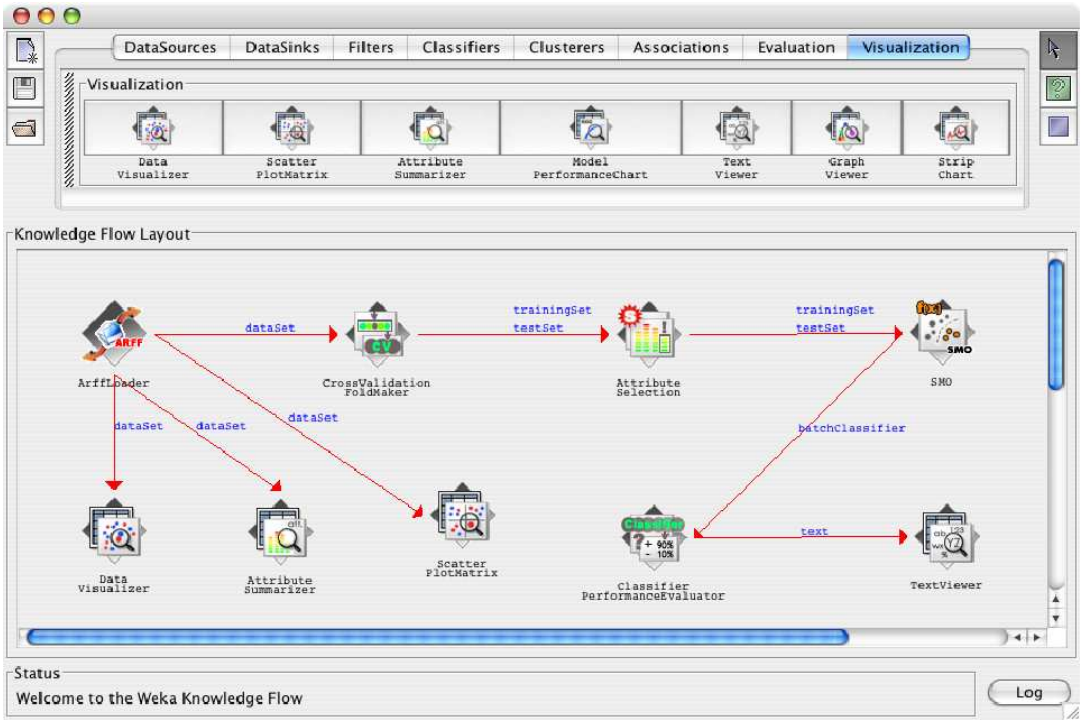


Figure 2.4: **Knowledge flow Graphical user interface**

2.3 Real time systems

"A real time system is a system where the time at which output is produced is significant. This is usually because the input corresponds to some movement in the physical world, and the output has to relate to the same movement. The lag from input time to output time must be significantly small for acceptable timeliness" [27].

Therefore, a real time system is one that relies heavily on the notion of response time. This response time can be broken up into two categories, hard real-time and soft real-time systems. Hard real-time systems are ones where the response time is critical to an application, i.e. a missile response system. Whereas, soft real time systems have response times that are important, but occasionally missed deadlines will not stop the system from functioning correctly. An example is a data sensor acquisition server, where computations can be rolled back to a previously established checkpoint in the event of a timing error.

As important as processing time is in real time systems, the output accuracy can not be overlooked. Correctness of a real time computation depends mainly on the time that the outputs are generated, but also the accuracy of the results. Presented in [44] is a real time system based on a 3D echo-cardiograph reconstruction system.

The system is used to evaluate important cardiac performance parameters such as stroke volume and ventricular mass. The authors point out that for the system to function correctly that both accuracy and timing requirements must be met. Failure to derive the correct information even within the allocated time frame could lead to incorrect estimates and thus endanger lives.

An in depth review on real-time systems can be found in [86], and a tutorial can be found in [56].

2.3.1 Embedded real time systems

Embedded systems are becoming one of the most popular applications for real time environments. An embedded system is a microprocessor based system that is built to control a function or a range of functions that are generally dedicated to one application [80]. An example is a washing machine. There are many functions for a wash cycle, but the main application is to wash fabrics.

As embedded systems are dedicated to specific tasks, they need only to be designed to meet such requirements. As a general rule, micro processors designed for embedded systems have reduced capabilities compared to the multi purpose processors found in personal computers.

[122] presents a review on design constraints for embedded real time control systems. The author highlights a number of embedded system limitations which include size, weight, power, cooling, performance and cost. As embedded systems are designed to be a component of a larger system or built into smaller mobile devices, they have their own operating requirements and manufacturing constraints.

Most devices, whether they are a mobile phone or one of the hundred dedicated micro processor units found in the latest cars, have strict size and weight limitations. Therefore, the internal electronics, including the processor and memory must be reduced in physical size and capacity. Furthermore, there is little room for cooling devices and battery storage. As a result of these constraints, it is essential that the power consumption and calculations of the processor is reduced.

[102] introduces further constraints, specifying that the program code size must be reduced as room for both static and dynamic memory is limited. He points out that many embedded packages have a one chip solution, "systems on a chip (SoCs)", where all information processing circuits are included on this single chip. This further highlights the need for run time efficacy, showing that

energy consumption, clock frequencies and supply voltages should be kept as low as possible.

For an in depth review on embedded systems, refer to [102] and [109].

2.3.2 Real time classification for embedded systems

[109] shows that embedded systems can be broken up into two different sub classes. These include embedded data-processing systems and embedded controllers. Embedded data-processing systems are dedicated to data communication and are data flow dominated. Processing the data must be done within a predefined time window.

Embedded controllers on the other hand, are dedicated to control functions and are flow dominated where they react to external events. The events are handled in real time, and accomplished within the range of milliseconds. These systems interface directly to the physical equipment that they are embedded into, and through sensors collect information about that environment. These systems are generally dedicated to monitoring or controlling specific operations within equipment.

They are known as reactive systems as they typically respond to incoming stimuli from the external environment [80]. Any reactive system is one that is in continual interaction with its environment and executes at a pace determined by that environment [16]. They can be thought of as being in a certain state, waiting for an input. For each input, they change their internal state to generate an output and/or a new state.

Numerous examples of such reactive systems can be found. These include most automotive embedded systems [94]. For example systems that control brakes, door windows, steering and air bags. Many robotic applications that include quality control or industrial machine fabrication. Also, applications for the use in condition monitoring and fault diagnosis (CMFD) of computers or machinery

[95].

Given the description and examples of reactive systems, it can be seen, that a classification algorithm with low memory, processor and power requirements can be ideal for such applications. A classifier can be trained off line and then the classifier's rule or module can be loaded into an embedded system. It would then wait for a set of inputs, compare the inputs against the model or rule and then provide the outputs based on the classification of the inputs.

A number of examples of classifiers being used for real time applications can be found in the literature. [93] presents work on monitoring power transformer faults using both support Vector machines [152] and artificial neural network classifiers [132]. [95] present case studies that look at monitoring different aspects of an internal combustion engine and a large class of static diagnosis problems. In this study both a multi layered perception and radial basis function neural network classifier were used to determine the faults in the system.

In another example, [68] develops a methodology for intelligent remote monitoring and diagnosis for manufacturing processes. In this work, a back propagation neural network classifier is used to monitor and identify faults in a real industrial case. The results showed that the back propagation neural network combined with the rough set approach [121] were promising. Furthermore, other work can seen in myoelectric control systems. Here an embedded device is implanted into a prosthetic limb. The classifier reads the myoelectric nerve data and converts this information into a set of movements [113].

2.4 Polychotomous classification (the Multi-class problem)

Before moving into a review on current classifiers, it is helpful to include some context about multi-class classification.

Most classifiers are designed to distinguish between only two different classes. In addition, classifiers that use a discriminant function not based on estimated densities, but instead fit a discriminant between two classes need some form of generalization to solve the multi-class problem [147]. Classifiers such as support vector machines and piecewise linear classifiers fall into this category. Constructing an approach that extends binary classification to a multi-class generalization for such classifiers is not always straightforward [2].

A number of approaches dealing with this problem are offered in the literature. The three most popular approaches include the "One- Against-One", "One-Against-All" and "Error correcting coding" strategies. For a detailed review of multi-class approaches refer to [129].

"One-Against-One" (OAO) is also known as the "All-against-All" or the "All pairs" strategy. Typically a "One-Against-One" strategy constructs $k(k-1)/2$ classifiers, where training is done between all pairs of classes [63]. This strategy reduces the problem to multiple binary classification problems that are solved separately. A framework that uses a scoring or voting system finds the most appropriate two class boundary, given a criteria, and classifies the new observation based on that boundary.

Many "One-Against-One" implementations for classification problems exist in the literature. Examples include [63] which builds on Friedman's procedure, the "max-wins" rule [52] and estimate pairwise class probabilities within a coupling model [70]. The authors solve the problem of multi-class classification for support vector machines. The authors also present various other methods, some include combining several binary classifiers and considering all classes at once.

"One-Against-All" (OAA) strategy constructs k classifiers where each single class is trained against all other classes. [3] presents a k class modular ap-

proach to neural networks whereby building a k single output feed forward networks. Training one class pattern against the other combined class patterns, for all classes. A high output is given for the single class pattern and low for the pattern of the remaining classes.

"Error-correcting coding" (ECC) strategy implements an encoding scheme for each class, using a matrix M of $-1, 1$ values of a $K \times C$ matrix, where C is the number of binary classifiers. A distance measure, often the Hamming Distance, is used between each row and the output.

$$f(x) = \arg\min \sum_{i=1}^{C} \left(\frac{M_{ri} f_i(x)}{2} \right) \tag{2.1}$$

A number of error encoding schemes can be found in [55]. The authors build a framework that uses an error-correcting code approach to partition classes into opposing subsets. Furthermore, [2] builds on the work of [55], where they focus their attention on a *margin-based* approach frame work.

There is merit in using any of the three strategies for solving the multi-class problem. From the literature the OAO approach appears to be the most popular. Though a lot of this work was undertaken using SVM classifiers, this work can still be generalized to other PWL classifiers. Results presented in ([2], [54], [70]) show that the OAO approach offers better performance than the OAA approach. Furthermore, results found in [2] show an increase in performance when using the ECC approach compared again, with the OAA approach.

A comprehensive review and comparison of the three strategies is presented in [129]. The authors contradict the previous results found by the previous authors, showing that when correctly applied on real world data sets, the OAA yields results as good as the other two approaches. It was assumed that the training times of the OAA approach were slower, see [54], their experiments did not show this.

The three classifiers developed in this thesis use the OAA approach to solve the multi-class classification problem.

2.5 Current classification techniques

There are many different classification techniques being used today and most are found in the WEKA data mining package. The following are popular mainstream classifiers, most of which were used in this thesis as comparison techniques for the three developed classifiers.

2.5.1 Logic based classifiers

Decision trees

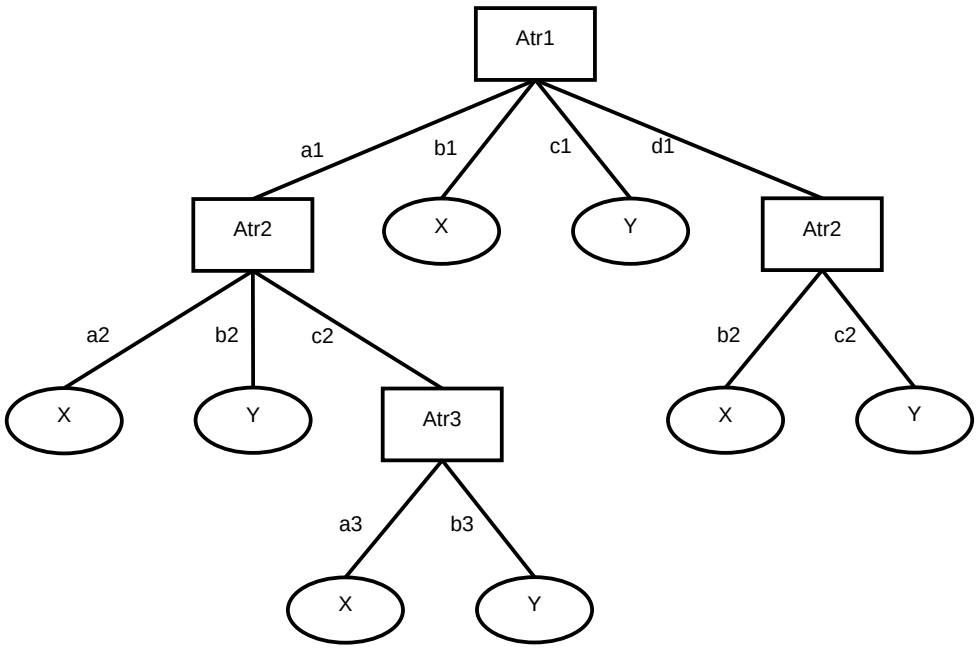


Figure 2.5: **A decision tree**

Decision trees are the most popular representation of logic based classifiers and are well presented in the literature ([74], [22] and [108]). Two well known implementations of decision trees include Classification and Regression Trees (CART)

Atr1	Atr2	Atr3	Class
a1	a2	a3	X
a1	b2	a3	Y
c1	a2	a3	Y
d1	b2	b3	X
a1	c2	a3	X
b1	a2	b3	X
b1	b2	b3	X
d1	c2	b3	Y
c1	a2	a3	Y
a1	c2	b3	Y

Table 2.1: Training set of decision tree

[22], and Quinlan's univariate tree growing algorithm, known as the Iterative Dichotomiser 3 algorithm (ID3) [123]. The C4.5 algorithm [124] extends the ID3 algorithm by allowing the classification algorithm to deal with numbers and not just categorical values as is the case for ID3.

The above implementation uses a top down approach as the learning model for building a decision tree. The algorithm searches the data set's attributes and incrementally creates the tree's nodes by splitting attributes, given the class labels. Most algorithms split attributes based on an information theory criteria. Each node then represents an attribute, and their outgoing branches correspond to all the possible outcomes that lead to further attribute nodes. The terminal or leaf nodes correspond to the output classes.

Classification of new observations starts at the root node and then traverse down the respective tree nodes until a class node is reached. At a nominated attribute node, a comparison is made between the observation's attribute value and the node rule. The outcome of this rule specifies the branch in which to continue from that node. This process continues until a class leaf node is reached.

Information theory is often used to minimize the number of branches created. This in turn reduces the number of tests that the new observation will need to do to reach its class node. This attribute selection process works on the assumption

that the complexity of the decision tree is strongly related to the amount of information conveyed by the value of that given attribute. It does this by using a heuristic based on selecting a feature that provides the most information gain [133]. Hence this measure favours the features that will partition the data into subsets that have the lowest class entropy.

Figure 2.5 is an example of one possible decision tree derived from the observations shown in table 2.1. The tree was incrementally constructed using the most informative attributes first [133]. Branches were constructed by splitting on each of these attributes, where any redundant branches were removed [123]. The tree comprises of univariate splits, allowing for a simple representation and making it easy to understand the inferred model. However, this also reduces its functional form and therefore reduces the approximation power of the model. For a detailed review on multi-variate decision trees refer to [24].

Information gain heuristic: Given a data set T with n outcomes for a given feature that is partitioned into subsets of training observations $T_1, T_2, \ldots, T_n$ then if S is any set of samples, let

$$freq(C_i, S) \in S \subseteq C_i \quad 1, \ldots, k$$

$$I(S) = -\sum_{i=1}^{k}((freq(C_i, S)/|S|) \cdot log(freq(C_i, S)/|S|))$$

after T has been partitioned according to n outcomes of feature X_i the expected entropy is the weighted sum over subsets

$$I(T) = -\sum_{i=1}^{n}((|T_i|/|T|) \cdot I(T_i))$$

with the quantity: $Gain(X) = I(T) - I_x(T)$

However finding the smallest decision tree is NP complete due to the combinatorial nature of the attribute selection process. An example given in [82] shows

that a data set of five attributes and twenty observations can lead to more than 10^6 decision trees, depending on the number of different values for each attribute. For this reason most decision tree construction methods use non backtracking and greedy algorithms, as they are efficient and easy to implement. However, this can lead to suboptimal solutions.

Rule learners

Rules can be derived from decision trees by recording the traversed path through the tree from the root node, via the respective attribute nodes to the class leaf [124]. However, for the most part, rules are induced from training data using dedicated rule based algorithms. A detailed review of current rule learners can be found in [53]. Some well known implementations of rule learners include RIPPER [38] and PART [51].

Most algorithms are based on the divide and conquer paradigm [104]. These algorithms search through the training set finding a rule that explains a number of instances for a given class and then separates these instances from the others. This separation is done recursively by conquering the remaining instances finding new rules and updating older rules until all instances have been separated. Therefore, the algorithm must run $m - 1$ times, where m is the number of classes and k is the number of rules. This means that the algorithm needs to make $n \cdot m \cdot k$ passes of the data set, where n is the number of instances.

During the learning phase problems can occur as the available data dwindles. Such problems include the fragmentation problem [117] and disjunct problem where fewer training observations can lead to a higher error rate [67]. However, the aim is to construct the smallest set of generalized rules that encapsulate all the necessary assumptions for each class. It is necessary to avoid a large number of rules as they tend to remember clauses, leading to overfitting and increased complexity.

The Disjunctive normal form (DNF) can represent each class as rules in the following form: $(X_1 \wedge X_2 \wedge \cdots \wedge X_n) \vee (X_{n+1} \wedge X_{n+2} \wedge \ldots X_{2n}) \vee \ldots \vee (X_{(k-1)n+1} \wedge X_{(k-1)n+2} \wedge \ldots \wedge X_{kn})$. k is the number of disjunctions and n is the number of conjunctions within each disjunction.

Typical rule based approaches solved the multi-class problem by running the algorithms separately for each individual class. This can lead to contradictions, i.e. the same observation is assigned to a different class and incomplete rules, i.e. observations are assigned to no classes. A number of solutions have been proposed to overcome these problems. One such solution includes a round robin approach [54]. This approach builds rules for each pair of classes and then uses a voting scheme to determine which two class solution the observation more closely belongs to. A more detailed description of the multi-class problem can be found in section 2.4.

2.5.2 Statistical based algorithms

Bayesian classifiers

Bayesian classifiers are statistical based methods that use probability and probability distributions to infer the class outputs of new observations. The Bayesian classifier learns the probability distribution of the data set by using a sampling technique to randomly select observations from the data set using a density estimator, usually a joint or naive density estimator. This distribution, known as prior distribution, is used as the initial information.

The data set is then arranged into subsets, usually dividing the inputs into their respective class outputs or creating functions to divide inputs over the class outputs. A new distribution is found within the new subsets, and is used to update the prior distribution. This new distribution becomes the posterior distribution. A maximum likelihood function based on this new distribution is used to predict the class outputs. This process is called induction of inference.

In [107], the author claims that it is a reasonable assumption to know how much training data is needed to attain a reliable estimation. When using conditional probability to estimate the probability distribution the amount of data needed is actually unknown. One hundred independently drawn training examples have been shown to be enough to obtain the maximum likelihood estimate of $P(Y)$, however, accurately estimating $P(X|Y)$ will need many more examples.

The author provides an example showing that if output Y was a boolean and vector X contains inputs of n boolean features then for this case the estimation of parameters $\theta_{ij} \equiv P(X = x_i|Y = y_j)$, where the index i takes on 2^n possible values. Therefore, this means that the estimation will be approximately 2^{n+1} parameters. This shows that the estimation is of an exponential nature and to obtain reliable estimates would require multiple parses of vector X.

The Naive Bayes classifier improves the complexity down to a linear form of $2n$. This is achieved by making the assumption that there is a conditional independence between all the features in the vector space. This assumption allows for the reduction in the number of parameter estimations needed for the conditional probability $P(X|Y)$.

However, with this complexity reduction there is a trade off with the accuracy of the Naive classifier. [128] highlights this problem within the text categorization domain. [128] shows that when one class has more training examples than another, the Naive Bayes classifier generally selects poor weights for the decision boundary. The assumed feature conditional independence will cause the magnitude of the weights for classes with strong word dependencies to become much larger than the classes with weak dependencies. Hence any domain that has feature independence integral to its distribution will fail due to this assumption.

The Naive Bayes classifier is a special case of a Bayesian Network. The learning structure and parameters of an unrestricted Bayesian network would be a logical improvement to the short comings of a naive Bayesian Network. A Bayesian

Network $B = \langle N, A, \Theta \rangle$ is a directed acyclic graph (DAG), where each node $n \in N$ represents a feature in the data set and each node n_i edge represents a probabilistic dependency, quantified using a conditional probability distribution $\theta_i \in \Theta$.

Although Bayesian Networks are used for analysis and predictive regression, they can also be used as a classifier. See [36]. Like naive Bayes, they also need to establish a prior and posterior distribution. However, in this case, the posterior distribution is taking into account relationships between all features, as the assumption is now a conditional dependency. This allows for feature inter-relationships and fixes the feature independence problem.

However, [36] state that given these new improvements the problem is NP hard. For this reason modifications, such as the ones shown in [58, 36, 71] attempt to reduce complexity. Many examples of successful modifications report high accuracy in training, but the learning time is often high. This can be seen in [36] which shows that for a $32000 \cdot 13$ dimensional dataset trained in ~ 20 minutes, that the time is scaled up by a factor of $O(n^2)$ in the number of features.

k-Nearest Neighbour

The k-Nearest Neighbour algorithm [49] and [40] is an instance or case based algorithm that does most of its work during classification time. It does not necessarily need to create a function rule or model, but instead compares each new instance to nearby existing instances using a distance metric. The closest k existing instances will determine which class the new instance will be assigned to. This is done via a majority vote of the neighbouring objects and the class containing the most number of objects among the k number of objects becomes the class to which that object is assigned to.

An example of this can be seen in figure 2.6. In this example $k = 5$, meaning the distance or radius is extended until five other neighbouring instances are

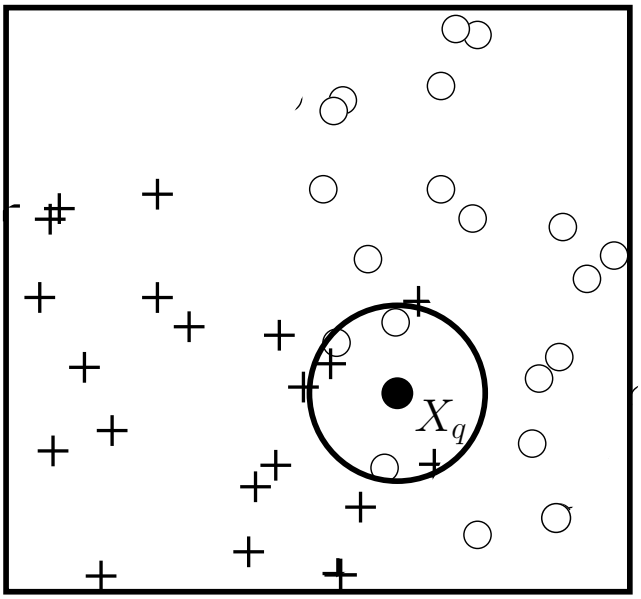


Figure 2.6: k-nearest neighbour, k=5

found. The figure also shows that the new instance X_q, has three instances of the same class and two that are not, as its closest neighbours. As mentioned earlier, a vote is taken and the majority of instances belonging to the same class within the radius of $k = 5$ is chosen as the class that the new instance is assigned. As can be seen in this case, the instance is correctly classified.

Furthermore, it is a good idea to have k set as an odd number. This will prevent an equal number of objects belonging to both sets of classes within the k number of points. Generally it is useful to look at more than one neighbour around the new instance. However, having one neighbour is widely used as a classifier. There is no set rule to defining the number of neighbours. Generally a number of neighbours are tried incrementally and the one that provides the best accuracy is used.

Given the training data $D = \{x_1, \ldots, x_n\}$ in R^n space. We have $(x_i)_{i \in [1,|D|]}$ training examples which are labelled by a class label $y_j \in Y$. The objective is to classify the unknown example q. $\forall x_i \in D$ where a distance measure can be used to find the distance $d(q, x_i)$.

$$Vote(y_j) = \sum_{c=1}^{k} \frac{1}{d(\mathbf{q}, \mathbf{x_c})^n} 1(y_j, y_c)$$

Though the $(k-NN)$ algorithm has been shown to work well on small datasets, it does not scale up well on large ones. [116] shows that the basic $k-NN$ classifier using a simple Minkowski distance will have a time behaviour of $O(|D||F|)$, where D is the training set and F is the number of features. In this simple case, the distance metric is linear in the number of features and the comparison process increases linearly with the amount of data. However, more complex distance measures will increase this complexity further, where the measure is directly related to the number of features.

Conversely a large training set may cause memory problems due to needing to store all of instances at classifying time. Work has been conducted in finding more efficient data tree storage structures and search methods. However, due to the number of nodes increasing exponentially in the number of instances, there has been limited success.

Data sets having a large disproportional number of instances belonging to one class can also be problematic. These instances can dominate the class prediction, thus affecting the prediction accuracy adversely. The most popular approach to over come this is to weight the nearest instances by their given distance. The further the distance, the less weighting the nominated neighbour should be given.

A final consideration is that k should be known a priori. Without knowledge of the data set, multiple runs of the algorithm may be needed to find the most appropriate k nearest neighbours. For these reasons the authors state that the $k - NN$ classifier can have a poor run time performance if the training set is large.

2.5.3 Artificial Neural Networks

Perceptron

An artificial neural network (ANN) is an inductive learning technique that models the brain by using a collection of nodes and weighted edges. ANNs have a long history with origins dating back to the early 1940s [162]. A detailed review on the history of ANNs can be found in [4]. In 1956, building on the work of [64] and [162], [131] created a unit called the perceptron. Most ANNs are based on a perceptron.

A perceptron has an input layer and an output layer which contains only one neuron. It calculates a linear combination of its inputs, passes a weighted sum of the inputs to an activation function that outputs one of two possible values. Figure 2.8 shows the basic model for an ANN, in the case of a perceptron, the receptor is the input layer and the output element is a single output.

Multilayered perceptrons

A single perceptron can construct a decision function that represents a hyperplane. However, [106] found that a perceptron was unable to learn the XOR function. This was mainly due to it having only two layers and thus capable of only linear separation. More layers, hidden layers, were introduced to improve separability from linear to bilinear. This then extended the network into a multilayered perceptron [132].

Multilayered perceptrons are fully connected networks containing at least three layers of neurons (nodes), input, hidden and output layers. The layers are connected by adjustable weighted links. The neurons in the input layer correspond to the attributes, and the output layer neurons correspond to the classes. The neurons in the hidden layer are connected to both the input and output layer nodes. One of the most popular multilayer network is the feed forward network.

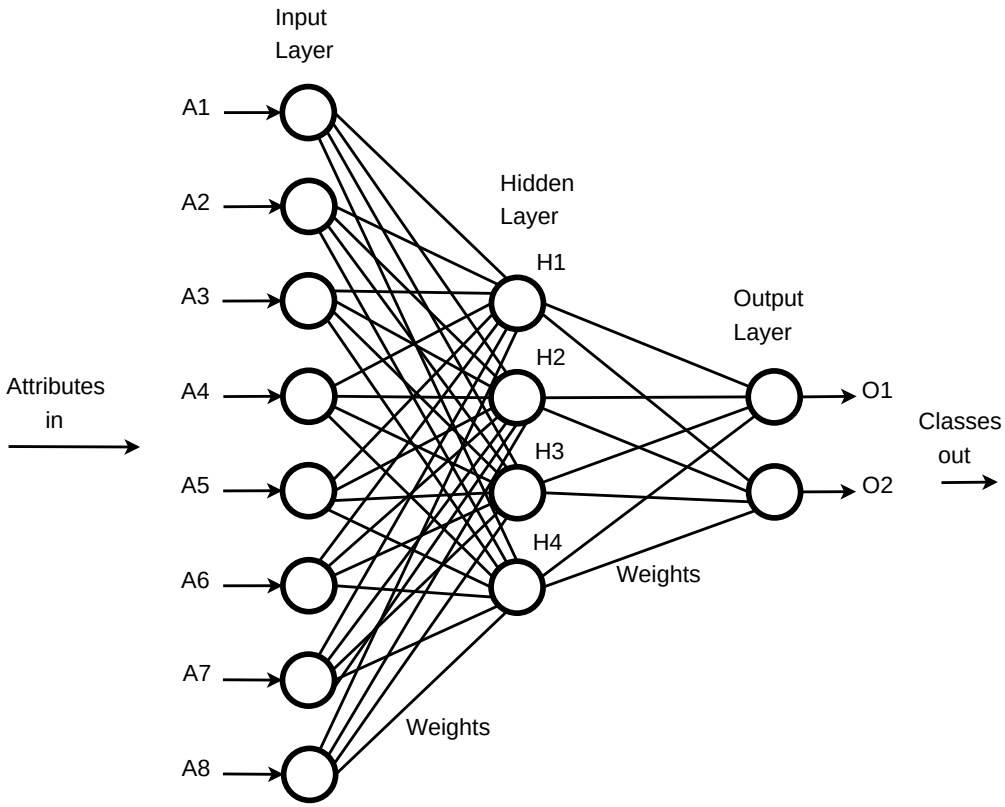


Figure 2.7: **Simple multilayer neural network**

Feed Forward networks have signals only travelling one way, from the inputs to the outputs. See figure 2.7.

The training and classification processes can be seen through the basic model shown in figure 2.8. The model is composed of four functional elements:

Receptor block is where the inputs x_i arrive, the inputs are either from the data or from a previous layer of neurons. These values are modified by the weighting factor of the links.

Adder gives a weighted sum of the inputs, or a linear combination of that neuron's incoming connections.

$$V_k = \sum_i w_i x_i$$

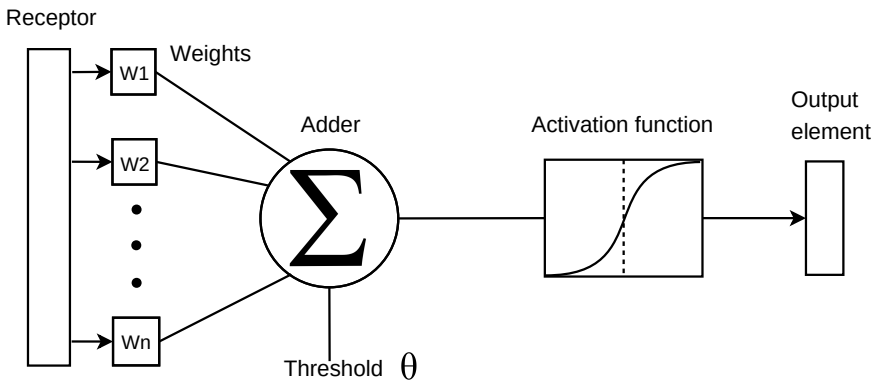


Figure 2.8: **Basic model of a neural network**

Activation function applies a threshold step function to the adder output to determine if that neuron should fire.

Output element produces and distributes the outputs.

To train the network, observations are repeatedly presented as training pattern vectors to the input layer. During each iteration the output vector is compared with the current class vector and the difference is used to adjust the interconnected weights. If the prediction value is too low then the weights are adjusted up and vice versa. Training is completed when the training error is less than some tolerance, or no more than an allocated number of epochs has been reached.

A number of search methods can be used for the training of multilayered networks. However, the back propagation method [132] is the most widely applied. It back feeds through the network, adjusting weights by using the derivative of the activation function in conjunction with other parameters. It often achieves high accuracy [48], but can get stuck in local minima and have a slow convergence rate [169]. Optimization algorithms based on gradient descent algorithms are often used to overcome these problems. Such algorithms include conjugate gradient methods, quasi-Newton BFGS and Leven-Marquadt methods, refer to [167]. A number of special classed problems involving bilinear separation ([91], [119] and [15]), use greedy algorithms trained by Linear programming [41] in feed forward

networks.

During classification the unknown pattern is propagated through the network to the output layer where the activation values determine the class values. The input layer passes the signals to their respective hidden layer nodes, which in turn pass signals onto the output layer nodes. Each neuron has an activation function, so only the signals that are activated will pass between each layer onto the next neuron. Since all link weights were determined during training, the firing of the activation function depends on the contribution of the incoming values to the weighted sum.

A number of activation functions can be found in the literature, these include threshold or binary, piecewise linear and non linear type functions. The most popular are non linear activation functions that are bounded, monotonically non-decreasing and differentiable. This allows for the desired output of unit interval $[0, 1]$ as a threshold to fire the neurons. Furthermore, differentiability makes the function suitable for the use in gradient search algorithms. The most popular among them are the sigmoidal function [103]. Gaussian nonlinear functions are popular in Radial basis function Networks (RBF) [35] and [79]. They are a generalization of nearest neighbour algorithms, but instead use an exponential distance function.

Determining the number of hidden nodes and layers is still an open problem. Much work is being done on finding bounds for the number of hidden neurons [72]. [29] presents a criterion based on an approximation of Chebyshev polynomials, [23] and [83] present work on establishing the number of hidden layers. Furthermore, [39] shows that adding extra hyperplanes, i.e. extra nodes in the first hidden layer can save the need for a second hidden layer.

Although the inclusion of another layer improves separability, [15] shows that two hyperplanes may not always solve the XOR problem. [23] and [83] attempt to find a minimal architecture using a hidden layer and present limitations of poly-

hedral dichotomies when separating certain XOR configurations. They present a conjecture showing four XOR configurations not computable by one hidden layer.

A further consideration, is to do with dimensionality problems when dealing with large scale data. As the number of features increase, both the input nodes and the number of connecting weights will also increase proportionally. This becomes a problem for gradient search methods, as the gradient calculation requires a nested summation over the weights and number of patterns [115]. The author shows empirically that halving the number of records actually reduced the training time by a factor of twelve.

Furthermore, as increased dimensionality can lead to an increased number of minima, finding a global solution becomes more difficult and time consuming. [150] shows that simple first order gradient descent methods, i.e. steepest descent method used in BP, have poor convergence properties. Although very small steps down the gradient guarantees convergence to a local a minimum [48], the time taken to converge can be long with no guarantees of finding a global minimum. Conjugate gradient methods greatly improve the performance of feed-forward networks, however, finding optimal parameters can be just as time consuming [150].

Artificial neural networks often deal with the multi-class classification problem as a single neural network system using O outputs. An example is shown in figure 2.7, depicting a multilayered feed forward network. However, in terms of improving on the backpropagation (BP) algorithm a multi-class solution is preferrable [114]. There are a number of possible multiple neural network architectures, most use the approaches presented in section 2.4. These include the one-against-one (OAO) and the one-against-all (OAA) strategies.

[3] presents work that builds a multiple neural network architecture and compares the efficiency and accuracy to the single neural network system. The authors divided a multilayer neural network into a number of single output feed forward

network modules. Their aim was to improve on the standard back propagation approaches. Their results showed that using the OAA approach gave a reduction in the number of iterations for training, a better rate of convergence, and scaled up well for a larger number of classes.

For an indepth review on multi-class pattern classification using neural networks refer to [114].

2.6 Conclusions

In this chapter we presented an in depth view on the data mining process, supervised machine learning and more specifically how the classification process works. It was shown that classification can be a long involved process, where a number of classification techniques, parameters and data preprocessing may be needed before a successful classifier is constructed.

Further, a review was given on a popular data mining package, WEKA. This review showed that such data mining tasks can be implemented using a dedicated data mining package. Also, it gave insight into the number of tools that can be found for data preprocessing and classification. Thus, allowing the data miner many options for the best presentation of the data using the most appropriate classification technique.

A comprehensive review was given on real time and embedded systems. This review described both systems and highlighted the operating constraints placed on a classification technique being used in such a domain. Furthermore, a brief review was given on current real time classification problems. A number of examples presented, showed the type of real time classification problems being solved and by what type of classifier.

The multi-class classification problem was highlighted and a number of solutions were presented. Also, a brief review was given on how some of the current

classification techniques deal with this problem. This led onto a detailed review of a number of current and popular mainstream classification techniques. Problems in these classification techniques were highlighted in terms of dealing with large scale data and real time classification constraints.

The next chapter presents a review on a number of optimization and piecewise linear based classifiers. This review presents the benefits of these classification techniques in being able to deal with real time classification. However, it also highlights problems with finding well placed decision boundaries efficiently. It further highlights the problems of these classifiers in terms of dealing with large scale data.

Chapter 3

Optimization and piece wise linear based classifiers

3.1 Introduction

It can be seen that the classification problem is also an optimization problem. The ultimate aim is to construct some form of boundary that best divides the classes from one another. When an unknown point is presented, it can then be assigned to a class region designated by the respective boundary. Therefore, the aim of classification is to find a function or rule that can minimize the number of misclassified points or maximize the number of points that contribute well to the construction of a decision boundary.

This chapter presents a number of optimization based classifiers. All of these classifiers fall under either a multiple or single optimization approach. As this thesis develops classification techniques based on a single optimization approach using max-min separation, a description of max-min classification is presented. As part of this description, an objective function based on the average error of separation between two sets is also presented.

Furthermore, as two of the three developed classifiers use data pre- classifi-

cation schemes and an incremental algorithm to reduce the computational complexity, a review of both are included in this chapter. These are sections 3.4 and 3.5.

3.1.1 Optimization based classifiers

The general form for an optimization problem is

$$\text{minimize} \quad f(x) \quad \text{subject to} \quad x \in X \tag{3.1}$$

where $x \in R^n$ is a vector of decision variables, $f(x)$ is the objective function and $X \subset R^n$ is the feasible region.

The Mathematical formulation for the classification problem that minimizes or maximizes the objective function, is dependant on the misclassified points and a number of parameters, used as variables. Most optimization problems associated with classification are based on the connected problem of separating two sets. The following must be determined when solving such a problem.

1. The type of objective function needed.
2. The parameters needed to be passed to the function.
3. A criteria on determining the quality of the objective function.
4. Ensuring the problem being solved is tractable.

The optimization based classifiers presented in sections 3.2 and 3.3 address these criteria. Furthermore, most of these classifiers extend the classification problem of two sets to solving the multi-class problem.

3.1.2 Piecewise linear classifiers

Piecewise linear classifiers have been a subject of study for more than three decades. They can be used to approximate non-linear decision boundaries be-

tween pattern classes. One hyperplane provides perfect separation when the convex hull of these pattern classes do not intersect. However, in many real-world applications this is not the case.

In many data sets the classes are disjoint, but their convex hulls intersect. In this situation, the decision boundary between the classes is non-linear. However, the decision boundary can be approximated using piecewise linear functions. Over the last three decades different algorithms to construct the piecewise linear decision boundary between classes have been designed these include the following classifier examples ([5, 9, 11, 18, 32, 65, 87, 118, 142, 141, 149]).

Most of these piecewise linear classifiers are very simple to implement; their memory requirements are very low and they provide real-time classification. Furthermore they have a small training time and do not contain parameters which depend on the data set. Most of them also make no assumptions of the underlying statistical distributions of the samples, and also most of the classifiers are generalised to only a mathematical function, i.e. max-min function.

Therefore they are ideally suited for applications such as small reconnaissance robots, autonomous mobile robots, intelligent cameras, embedded and real-time systems, portable devices, industrial vision systems, automated visual surveillance systems, monitoring systems etc. [87]. As previously mentioned, all of these applications require fast real-time classification and low memory usage with limited processing power. Whereas, most of the main stream classifiers were not designed with these requirements in mind.

There are however problems associated with piecewise linear classifiers. In general, the determination of piecewise linear boundaries is a complex global optimization problem [141]. In most cases the problem of finding such boundaries is reduced to the minimization of the classification error function which is non-differentiable and nonconvex. Nonconvexity of the error function means that, in general, it has many local minimizers.

As the global minimizers provide the piecewise linear boundary with the least number of hyperplanes, an optimization technique best suited to nonsmooth global optimization problems is needed. Smoothing techniques do not always provide a good approximation of the decision boundary. Also Newton-like or gradient based methods cannot be applied to solve such problems. Furthermore, the problem of building a decision boundary is more complicated, as the number of hyperplanes in the boundary is not known a priori.

These complications make training of the algorithm very complex and can lead to long training times, thus making the application of piecewise linear classifiers difficult. In order to reduce the training time most techniques try to avoid solving optimization problems when computing piecewise linear boundaries by using certain heuristics. For example, they apply fast clustering algorithms (such as k-means) to find clusters in each class. They then compute hyperplanes separating pairs of clusters from different classes. The final piecewise linear boundary is obtained as a synthesis of those hyperplanes (see [18, 87, 118, 32, 141, 142, 149, 65]).

Existing piecewise linear classifiers can be divided into two classes. The first class contains classifiers in which each segment of the piecewise linear boundary is constructed independently (*multiple optimization approach*). The second class contains classifiers in which the problem of finding a piecewise linear boundary is formulated as an optimization problem (*single optimization approach*).

3.2 Classifiers based on the multiple optimization approach

3.2.1 Early piecewise linear classifiers

Nilsson

Over the past few decades various approaches to design piecewise linear classifiers have been proposed. In the mid 1960s "Committee Machines" [110] were used for pattern recognition to build a piecewise linear system. However, the discrimination step was complex and computationally intensive [146]. Initial improvements included the idea of hyperplane placement by minimizing probability density functions [157]. However, a large sample size was needed to construct a well placed hyperplane boundary.

Sklansky and Michelotti

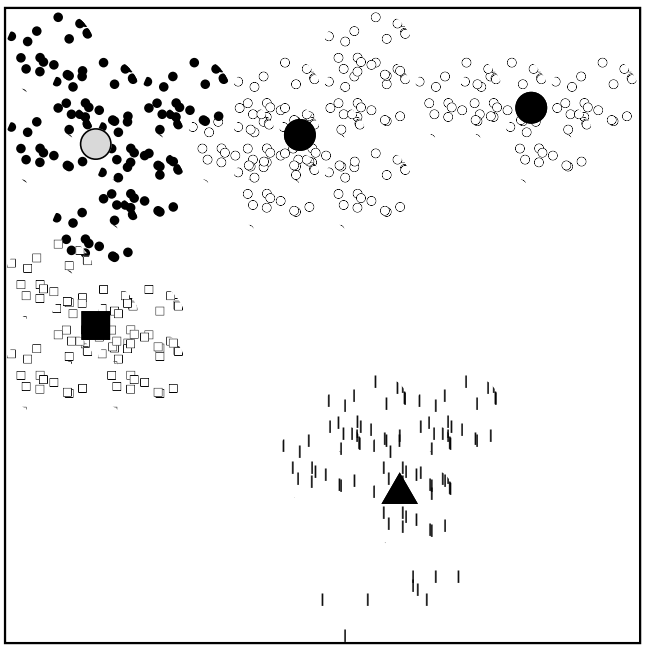


Figure 3.1: Prototypes found by clustering algorithm

More significant improvements came in, what is arguably, the first of the mul-

tiple optimization approaches. Though still a two-class problem, ([142], [141]) introduced a procedure to locally train piecewise linear decision boundaries. The authors recognized the time and memory complexities associated with committee machines when training on the whole data set. To improve on this, they introduce a methodology that reduced complexity by concentrating only on the points that contribute to separating the decision boundary. This in turn helps to find simplifications in the decision surface.

Firstly they identify prototypes using Forgy's algorithm [50], also known as the k-means clustering algorithm [100]. These prototypes are the cluster centers of each class, see figure 3.1. They then use the prototypes to locate the points of the two opposing classes that are either close or intersect. These points make up the areas known as the encounter zones, refer to figure 3.2. Each pair of encounter zones is known as a "close-opposed pair" or "link".

Training is done by sequentially adding hyperplanes to only these areas. A window training procedure [77] is used, where training is repeated until two successive iterations gives the same separation. This gives a stopping criteria and avoids the use of parameters. Finally, creating a decision boundary with a near-minimal set of hyperplanes that separate only the subsets of the k-closed pairs from the k prototypes.

Though this work addressed the complexity issues associated with committee machines, a number of problems still existed. Determining the number and the size of prototypes was left up to the user as a parameter. There was also a need to automate the merging and splitting of the linear segments. Furthermore, the technique needed to be generalized in order to solve the multi-class problem.

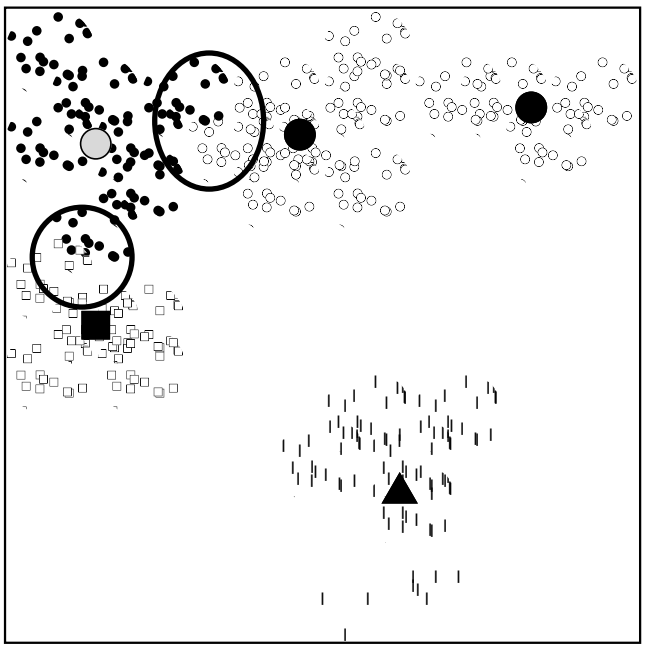


Figure 3.2: Encounter zone between classes in feature space

3.2.2 Prototype based piecewise linear classifiers

Park and Sklansky

Building on the previous work of prototypes and closed opposed pairs, the authors propose an improved method based on the cutting of straight line segments [120]. Tomek's method [75], a Condensed Nearest Neighbour (CNN) method [62] is used to find opposed pairs belonging to different classes. These pairs form the straight line segment (Tomek link), refer to diagram 3.3. Finding a set of Tomek links is $O(N^3)$ in the worst case, where N is the number of training set points. To reduce complexity, the prototypes are used instead of data points for the cutting of Tomek links.

Their procedure aimed to find a minimal subset of Tomek links and to maximize the number of Tomek links cut. This guarantees training set consistency and provides a good initial starting point for the hyperplanes. Thus allowing for efficient local region training. The authors show that incorporating the use of Tomek links and examining error rates, allow for the automation of a nearly opti-

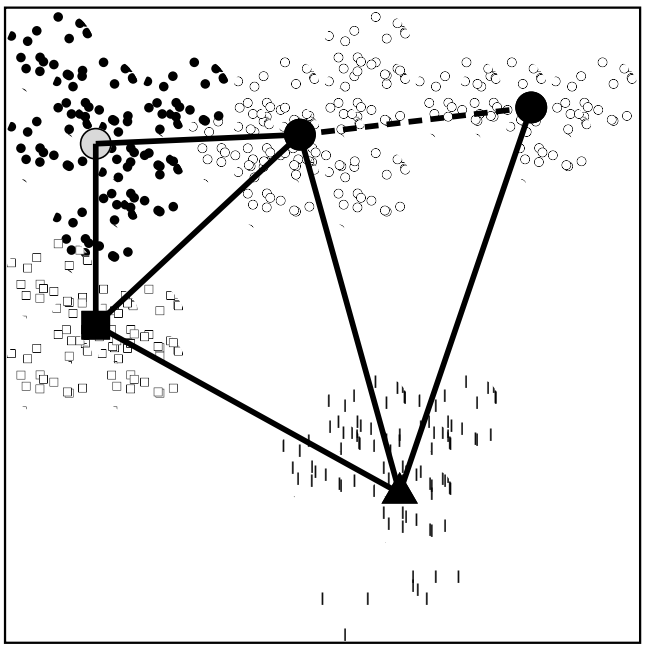


Figure 3.3: Tomek links

mal set of hyperplanes without the need of user specified parameters. To address the multi-class problem, data is temporarily relabeled into two groups. Using a two phase approach to train hyperplanes both locally and globally, an error rate then determines which phase hyperplane to use.

The authors demonstrate that this piecewise linear classifier provides a much faster decision than the k-nearest neighbors classifier for a similar accuracy. In [166] this classifier was compared with a neural network classifier. The neural network classifier performed slightly better than the piecewise linear classifier. On the other hand, the neural network classifier required a much longer training time.

Tenmoto, Kudo and Shimbo

The proposed technique presented in [149] and [148] builds directly on the use of prototypes and Tomek links found in the aforementioned classification work of Slansky and Park [120], refer again to diagram 3.3. They recognize the need for using a small number of prototypes instead of all the training points. This is mainly

due to the computational costs associated with constructing locally trained hyperplanes and discovering the spatial structure of the underlying distribution of the samples [149].

The authors identify problems relating to the prototypes found in the previous work of Slansky and Park. These include determining the number of prototypes, prototypes accurately representing the training samples and an appropriate set of prototypes in a high-dimensional space.

They point out that the previously mentioned classifier can produce undesirable results. This is due to prototypes being separated by a small number of hyperplanes and as a consequence ignoring local information. Furthermore, if an inappropriate set of prototypes is chosen then poorly constructed hyperplanes are possible.

To address these problems, a modification of the previous method [120] is proposed. This new method uses an incremental approach to construct hyperplanes where even samples of both prototypes and training examples are used. An error rate for this local training is kept under a threshold. The threshold is determined automatically by a probabilistic model criteria, the Minimum Description Length criterion [130], so as to avoid over fitting the training data. Furthermore, more training samples are chosen as two links are used instead of one link, as is the case in the previous method.

The results of the new method shows that it generally outperforms the previous method of Park and Slansky. The new method also keeps a lower error rate than that of Park and Sklansky, however more hyperplanes are needed as they search for training samples around four ends of two hyperplanes instead of one.

3.2.3 Tree Based Methods

Kostin

Further building on the concept of prototype patterns, an algorithm that creates a binary class partition tree is presented in [87]. The author proposes an improvement on the work of Park and Sklansky [120], and Tenmoto et al. ([149] and [148]). He points out that almost all the step procedures presented in Park and Sklansky's classification method are time consuming. Since the modified algorithm of Tenmoto et al. performs mostly the same steps found in Park and Sklansky's method it also is time consuming.

Furthermore, Kostin highlights the emphasis that is placed on the class boundary data points by Park and Sklansky's algorithm. As mentioned earlier, the algorithm is dependant on the prototypes properly representing these data points. He states that this is difficult to achieve as these marginal class training patterns are often represented by noisy data points. Therefore, the delicate optimization procedures performed within these class boundaries are of little value.

As mentioned before, Tenmoto et al. improve on Park and Sklansky's algorithm by maintaining a correct recognition rate over the training patterns, in order to avoid overfitting the training data. However, Kostin shows that the efficiency of their algorithm is still dependent on the results of the clustering and therefore is as complex as the algorithm of Park and Sklansky.

To avoid complex computational procedures in minimizing the number of hyperplanes in the decision boundaries and to handle the multi-class problem, Kostin implements a binary tree structure. This proposed classifier takes the centroids of the prototype patterns representing each class, and creates a rooted binary class partition tree. The tree contains a collection of segments of hyperplanes, that are broken up as perpendicular bisectors of the line segments linking the class centroids.

The proposed algorithm sequentially partitions the multi dimensional space containing the training set patterns. It computes for each non leaf node a set of hyperplanes that separates the training sets into two groups of classes that correspond to two outcome branches. These outcome branches represent the divided subregions and are partitioned according to the binary tree. These subregion partitions are continually created until one or more of the stopping criteria are met. The stopping criteria could include a separation error or an upper limit on the number of constructed hyperplanes. Each of the non leaf nodes within the tree represent the equation of the respective hyperplane.

To deal with multi-class pattern recognition, a hierarchical partition scheme is used. For further examples of hierarchical partition schemes for multi-classification, refer to [89] and [161]. In order to implement such a scheme, Kostin generalizes the original method so that the binary tree contains class centroids, or a partition tree of classes. The binary tree is configured so that the root and inner nodes correspond to groups of classes and the leaf nodes correspond to single classes.

As before, non leaf nodes are constructed sequentially. However, this time the algorithm divides the data set into two groups. From there, each group of classes in turn is divided into two smaller groups using a piecewise linear boundary. This is implemented by building a binary partition tree of all the classes. Each of these nodes decrease until the nodes consist of only two centroids of two simple classes.

For classification of unknown observations, an ordered traversal of the tree is handled by a set of codes created during the training phase. These codes control the choice of hyperplane sets chosen for the recognition process in determining which sub region the unknown observation belongs to.

In [87], Kostin claims that the computational complexity of the classifier is linear in the number of classes, but is quadratic in the number of attributes. As is the case with most tree based classifiers, storage is often a problem, where the memory complexity is quadratic in the number of nodes. A further consideration

is with accuracy. The accuracy may be compromised if there is an incorrect separation error tolerance or too few hyperplanes are selected during the separation of the initial regions.

Another tree based classifier is proposed in [32]. In this case, a linear binary decision tree classifier is presented, where the decision at each non-terminal node is made using a genetic algorithm. The authors apply this binary tree genetic algorithm approach for the use in pap smear cell classification. They aimed to build a classifier with a minimum misclassification error rate and low sensitivity-false alarm rate. Their results were favourable, however improvements could be made by adding more discriminatory features and increasing the statistical sufficiency of data samples.

3.2.4 Linear Regression based methods

The use of linear Regression as a technique in classification is common. Such work includes [73] which extend Support Vector machine classification to regression estimation to solve learning control for space robots. Also, [73] uses linear regression for classifying gene expression by using partial least squares to build a linear model.

In [138], a multiple optimization algorithm using linear regression is presented. The learning algorithm constructs a piecewise linear classifier for multi-class problems. In the first step of the algorithm linear regression is used to determine the initial positions of the discriminating hyperplanes for each pair of classes. Next an error function is minimized by a gradient method for each hyperplane separately. In the case of non-convex classes, the next step uses a clustering procedure to decompose the classes into appropriate subclasses, in order to make them linearly separable.

In terms of multi class classification, a pattern corresponding to a class is uniquely classified by the trained discriminating hyperplanes if it belongs to all

the half spaces for that class hyperplane set. It will not, however, belong to all the half spaces of all other sets. This is true for all class patterns in terms of discriminating against all other classes. If a class pattern is not uniquely classified, then that pattern is classified on the basis of the minimum distance to the class regions.

This classification technique has been implemented as a functional application and is named DIPOL92 (**DI**scrimination and **PO**st Learning). It was included in the STATLOG [105] project where it achieved good classification results on many data sets. The aim of the project was to review and compare different classification techniques on a wide range of known data sets.

3.2.5 Neural Network based and other methods

The paper [18] proposes an approach to construct a piecewise linear classifier using neural networks, refer to subsection 2.5.3. In this paper the training set is split into several linearly separable training subsets, and the separability is preserved in subsequent iterations. Furthermore, the authors completely reclassify these learning sub sets by a two layer neural network. This procedure uses a basis exchange technique in order to find a decision rule on the base of the learning sets and to find a solution within a finite number of iterations.

The authors highlight the benefits of this technique, including the fact that it shares the same properties of the nearest neighbour 1-NN rule, refer to subsection 2.5.2, as computational complexity is $O(M)$, where M is the number of data points. Furthermore, if the technique is implemented using a parallel network, the classification time would decrease significantly compared to the nearest neighbour 1-NN rule. They also showed that the computational complexity of the neural network is again linear and only increases in the number of neurons in the first layer.

The paper [118] proposes a piecewise linear classifier which starts with a linear

classifier. If it fails to separate the classes, then the sample space of one of the classes is divided into two subsample spaces. This sequence of splitting, redesigning, and evaluating continues until the overall performance is no longer improved. The results presented in the paper showed that the classifier worked well, as only a small number of discriminant functions were needed to obtain a reliable classifier.

However, the main strength of the classification technique, is in its ability to deal with the multi-class problem. The algorithm uses an iterative error correction procedure at classifying time. This procedure takes random samples of classes and adjusts a weight vector for each misclassification by saving the lowest error rate each time. Obviously the class with the lowest error is the one that is nominated.

3.3 Classifiers based on the single optimization approach

These classifiers learn the piecewise linear boundary globally and simultaneously, via solving a single optimization problem. Although they achieve better results than multiple optimization methods, their training time can still be long. One way to find such functions is to compute boundaries between pattern classes.

In general, the classes in real world data sets are distinct, however their convex hulls may intersect. In this case classes need to be separated by nonlinear functions. Such functions can be approximated by using piecewise linear functions.

Piecewise linear classifiers based on a single optimization approach were proposed in papers [5] and [9].

3.3.1 Linear Separability

Linear Separability uses one affine function to separate two sets.

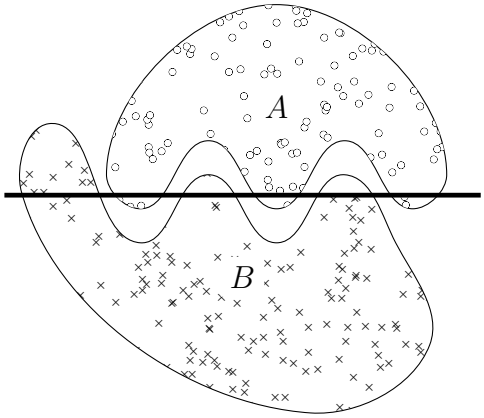


Figure 3.4: Linear separability

The sets $A = \{a^1, \ldots, a^m\}$ and $B = \{b^1, \ldots, b^p\}$ are linearly separable if there exists a hyperplane $\{x, y\}$, with $x \in \mathbb{R}^n,\ y \in \mathbb{R}^1$ such that

1) for any $j = 1, \ldots, m$

$$\langle x, a^j \rangle - y < 0,$$

2) for any $k = 1, \ldots, p$

$$\langle x, b^k \rangle - y > 0.$$

The sets A and B are linearly separable if and only if $\operatorname{co} A \bigcap \operatorname{co} B = \emptyset$.

In practice, it is unlikely for the two sets to be linearly separable. An example is that of Rosenblatt's perceptron or linear threshold unit (LTU)[131]. More precisely the Minsky and Pappert famous exclusive 'OR' problem [106]. Therefore it is important to find a hyperplane which minimizes some misclassification cost. In [14] the problem of finding this hyperplane is formulated as the following optimization problem:

$$\text{minimize} \quad f(x, y) \quad \text{subject to} \quad (x, y) \in \mathbb{R}^{n+1} \tag{3.2}$$

where

$$f(x,y) = \frac{1}{m}\sum_{i=1}^{m} \max\left(0, \langle x, a^i\rangle - y + 1\right) + \frac{1}{p}\sum_{j=1}^{p} \max\left(0, -\langle x, b^j\rangle + y + 1\right)$$

is an error function. Here $\langle\cdot,\cdot\rangle$ stands for the scalar product in $\mathbb{R}^n$. An algorithm for solving problem (3.2) was described in [14] . It was shown that the problem (3.2) is equivalent to the following linear program:

$$\text{minimize} \quad \frac{1}{m}\sum_{i=1}^{m} t_i + \frac{1}{p}\sum_{j=1}^{p} z_j$$

subject to

$$t_i \geq \langle x, a^i\rangle - y + 1, \quad i = 1, \ldots, m,$$

$$z_j \geq -\langle x, b^j\rangle + y + 1, \quad j = 1, \ldots, p,$$

$$t \geq 0, \quad z \geq 0,$$

where t_i is nonnegative and represents the error for the point $a^i \in A$ and z_j is nonnegative and represents the error for the point $b^j \in B$.

The sets A and B are linearly separable if and only if $f^* = f(x^*, y_*) = 0$ where (x^*, y_*) is the solution to the problem (3.2). It is proved that the trivial solution $x = 0$ cannot occur.

3.3.2 Support Vector Machines

Support Vector Machines (SVM) are similar to piecewise linear separators as they belong to the family of generalised linear classifiers. SVMs create decision planes that separate the data into their respective classes. They were first proposed in [152] and [153], due to their solid mathematical foundations are currently one of the most popular classification techniques. Their popularity extends to real time classification problems, including myoelectric control [113] and smart phone

malware detection [19]. A comprehensive review of SVMs can be found in [154] and [155].

Linear support vector machines

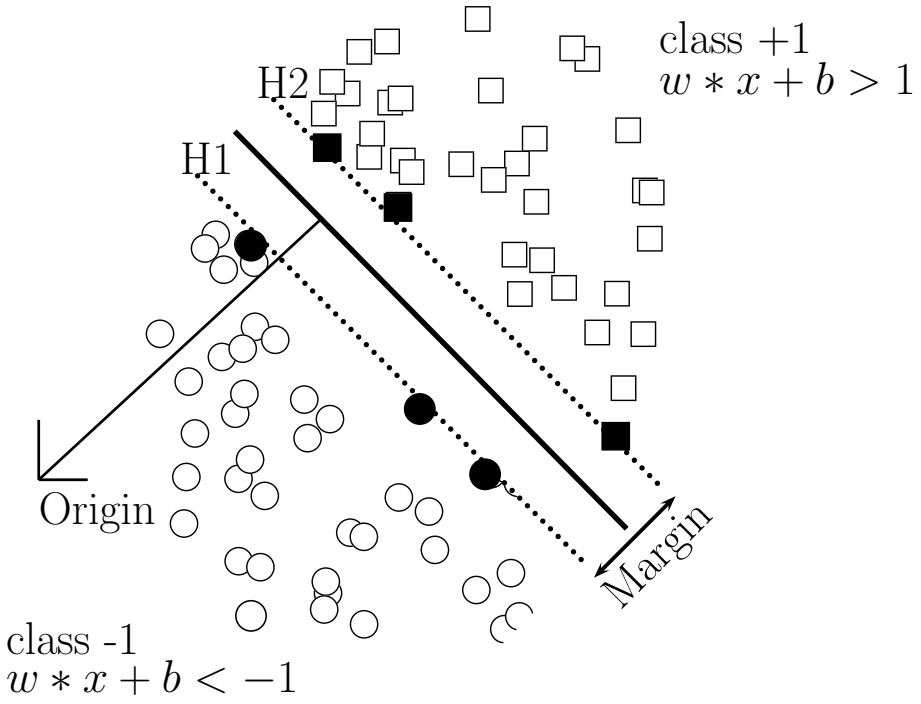


Figure 3.5: Linear separating hyperplanes as a separable case. The support vectors are filled in.

The simplest case is that linear machines are trained on separable data. If the training data is linearly separable then variables w weight vector and b the bias, exist such that

$$\begin{aligned} w^T X_i + b \geq 1, \text{for all } X_i \in M \\ w^T X_i + b \leq -1, \text{for all } X_i \in N \end{aligned} \tag{3.3}$$

The decision rule is given as

$$f_{w,b}(X) = sgn(w^T X + b), \tag{3.4}$$

where *sgn* is the sign function.

The overall aim is to create the largest possible distance between a separating hyperplane and the data points on either side. An optimal separating hyperplane can then be found by minimizing the squared norm of the hyperplane, such that a convex and quadratic programming (QP) problem is constructed. The problem is then defined as simultaneously minimizing the empirical classification error and at the same time maximizing the geometric margin between the two sets. Therefore to maximize the margin $\|w\|$ means the goal is to minimize $\frac{1}{2}\|w\|^2$ subject to 3.3.

The points lying on the margin of the newly found hyperplane are known as the support vectors. These points are significant as their linear combination *alone* represent the solution, refer to figure 3.5. As a result of the optimization problem, the number of support vectors are generally small. It can also be seen that the number of features have little effect on the complexity of an SVM.

However, SVMs are generally not favoured for large scale data mining. This is due to the training complexity being highly dependent on the size of the data set [164]. It has been shown that the training complexity is at least quadratic in the number of data points. Refer to [145] and [164] for further training complexity discussions.

In practice it is possible that a separating hyperplane may not exist. This could be attributed to noise in the data set that causes an overlap of the classes. To allow for observations that violate 3.3 slack variables can be introduced ([155] and [156]).

$$\Xi_i \geq 0, \quad i = 1, \ldots, l \tag{3.5}$$

A soft margin is created to deal with the misclassified points by relaxing the constraints to

$$y_i((w^T X_i) + b) \geq 1 - \Xi_i, \quad i = 1, \ldots, l \tag{3.6}$$

A generalized classifier is now constructed to control both its capacity through $\|w\|$ and an upper bound on the number of training errors through the sum of slack variables 3.5. Furthermore, the optimization problem remains convex.

One manifestation of this soft margin classifier is the C-SVM.

$$\gamma(w, \Xi) = \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{l} \Xi_i \tag{3.7}$$

subject to the constraints 3.5 and 3.6, for a constant $C > 0$.

Another possible manifestation of a soft margin classifier is the $\nu - \text{SVM}$

$$\min\|w\|^2 + C\sum_{i} \xi_i \text{ such that } c_i(w \cdot X_i - b) \geq 1 - \xi_i, \quad 1 \leq i \leq n \tag{3.8}$$

In this case the C parameter is replaced by the $\nu \in [0, 1]$ parameter which is the lower and upper bound of the (noisy) points that lie on the wrong side of the hyperplane respectively. For further information and improvements on these classifiers refer to [33] and [34].

Nonlinear support vector machines

Most real world classification problems deal with complex data containing noise and non linearity, such that no hyperplane can successfully separate the two sets. In this case, the data points are projected or mapped into a higher dimensional space. This *transformed feature space* is then used to define a separating hyperplane.

An appropriately chosen transformed feature space will allow for points to be better separated. Even to the point of linear separation as this transformed space corresponds to the original nonlinear input space [1]. The original input space D can be mapped by a function $\Phi : D \to H$. With this transformation it is now

necessary to solve only function K, the dot products in H such that:

$$K(x_i, x_j) = \Phi(x_i) \cdot \Phi(x_j) \tag{3.9}$$

It not necessary to determine Φ, only function K. Function K is a special class of functions known as a *Kernel function*. They allow for inner products to be calculated within the feature space without needing to perform any mapping to the higher feature space.

The following are three popular kernel functions.

- $K(x, y) = (x \times y + 1)^{\mathrm{p}}$
- $K(x, y) = \exp^{-\|x-y\|^2/2\sigma^2}$
- $K(x, y) = \tanh(\kappa x \times y - \delta)$

When the input data is projected onto a higher dimensional space one of these previously chosen kernels is used to separate the classes. The combination of a correctly chosen kernel and the transformation into the higher dimensional space allows for a greater margin of difference between the two sets and a better separation. For a more detailed explanation, refer to [26].

Currently the biggest limitation of the SVM approach lies in the choice of the appropriate kernel and then the subsequent choice of parameters [26]. The user must choose the kernel given the data set. This may be a time consuming process given the combination of parameters and kernels. Much work has been undertaken to remedy this situation. Examples include methods that use meta-learning and meta features for choosing parameters [143].

Furthermore, there exist statistical grid search methods that find parameters for tuning the Radial Basis kernels [12]. Guides and tutorials aimed to educate the novice user on tools for parameter tuning and intuitive kernel selections can be found in [69].

3.3.3 Polyhedral separability

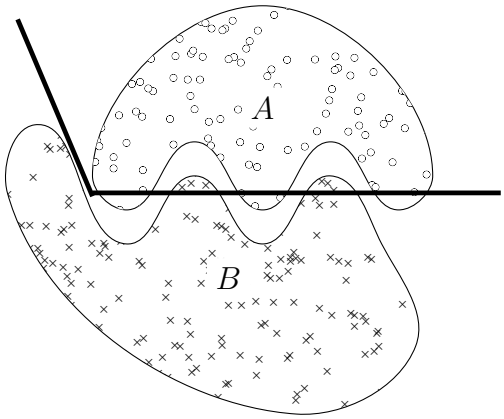


Figure 3.6: Polyhedral separability

The paper [5] introduces the concept of polyhedral separability. In this case a class is approximated by a polyhedral set and the rest of the space is used to approximate the second class. The classification error function is represented as the sum of convex and nonconvex nonsmooth functions. An algorithm for minimizing this function is developed in [5]. In the paper [112] the polyhedral separability algorithm was modified to derive classification rules by accurately learning from few data. It is achieved by solving a mixed integer programming model that extends the notion of discrete support vector machines.

The concept of h-polyhedral separability was developed in [5]. The sets A and B are h-polyhedrally separable if there exists a set of h hyperplanes $\{x^i, y_i\}$, with

$$x^i \in \mathbb{R}^n,\ y_i \in \mathbb{R}^1,\ i = 1, \ldots, h$$

such that

1. for any $j = 1, \ldots, m$ and $i = 1, \ldots, h$

$$\langle x^i, a^j \rangle - y_i < 0,$$

2. for any $k = 1, \ldots, p$ there exists at least one $i \in \{1, \ldots, h\}$ such that

$$\langle x^i, b^k \rangle - y_i > 0.$$

It is proved in [5] that the sets A and B are h-polyhedrally separable, for some $h \leq p$ if and only if

$$\operatorname{co} A \bigcap B = \emptyset.$$

The problem of polyhedral separability of the sets A and B is reduced to the following problem:

$$\text{minimize } f(x, y) \text{ subject to } (x, y) \in \mathbb{R}^{(n+1)\times h} \tag{3.10}$$

where

$$f(x, y) = \frac{1}{m} \sum_{j=1}^{m} \max \left[0, \max_{1 \leq i \leq h} \left\{ \langle x^i, a^j \rangle - y_i + 1 \right\} \right] + \frac{1}{p} \sum_{k=1}^{p} \max \left[0, \min_{1 \leq i \leq h} \left\{ -\langle x^i, b^k \rangle + y_i + 1 \right\} \right] \tag{3.11}$$

is an error function. Note that this function is a nonconvex piecewise linear function. It is proved that $x^i = 0,\ i = 1, \ldots, h$ cannot be the optimal solution. Let $\{\bar{x}^i, \bar{y}_i\}, i = 1, \ldots, h$ be a global solution to the problem (3.10). The sets A and B are h-polyhedrally separable if and only if $f(\bar{x}, \bar{y}) = 0$. If there exists a nonempty set $\bar{I} \subset \{1, \ldots, h\}$ such that $x^i = 0,\ i \in \bar{I}$, then the sets A and B are $(h - |\bar{I}|)$-polyhedrally separable. In [5] an algorithm for solving problem (3.10) is developed. One of the sets is approximated by a polyhedral set and the rest of the space is used to approximate the second set. The error function is represented as a sum of nonsmooth convex and nonsmooth nonconvex functions. The calculation of the descent direction at each iteration of this algorithm is reduced to a certain linear programming problem.

3.3.4 Max-min separability

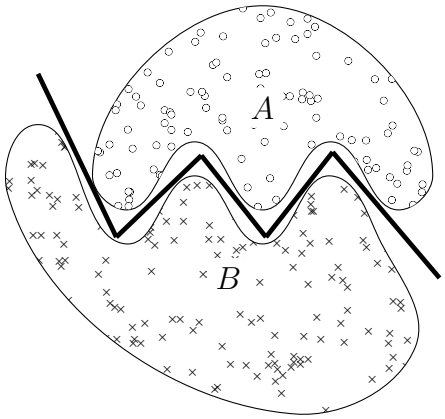


Figure 3.7: Max-min separability

The concept of max-min separability was introduced in [9]. In this approach two sets are separated using a continuous piecewise linear function. Max-min separability is the generalization of linear, bilinear and polyhedral separabilities [11]. It has been proven that any two disjoint finite point sets can be separated by a piecewise linear function. The error function in this case is nonconvex nonsmooth. The discrete gradient method [7, 8] is applied to minimize it.

Max-min separability is a generalization for the polyhedral separability introduced in 3.3.3

Definition and properties Let $H = \{h_1, \ldots, h_l\}$, where $h_j = \{x^j, y_j\},\ j = 1, \ldots, l$ with $x^j \in \mathbb{R}^n,\ y_j \in \mathbb{R}^1$, be a finite set of hyperplanes. Let $J = \{1, \ldots, l\}$. For a given $1 \leq r \leq l$ consider any partition of this set $J^r = \{J_1, \ldots, J_r\}$ such that

$$J_k \neq \emptyset,\ k = 1, \ldots, r, \quad J_k \bigcap J_j = \emptyset, \quad \bigcup_{k=1}^{r} J_k = J.$$

From now on we use the following notation for hyperplanes. If the set of hyperplanes is given without partition we use only one index for x and y. If the set of hyperplanes is given with partition we use two indices for x and y.

Let $I = \{1, \ldots, r\},\ 1 \leq r \leq l$. A particular partition $J^r = \{J_1, \ldots, J_r\}$ of the

set J defines the following max-min-type function:

$$\varphi(z) = \max_{i \in I} \min_{j \in J_i} \left\{ \langle x^{ij}, z \rangle - y_{ij} \right\}, \ z \in \mathbb{R}^n. \tag{3.12}$$

Let $A, B \subset \mathbb{R}^n$ be given disjoint sets, that is $A \bigcap B = \emptyset$.

Definition 1. *The sets A and B are max-min separable if there exist a finite number of hyperplanes $\{x^j, y_j\}$ with $x^j \in \mathbb{R}^n,\ y_j \in \mathbb{R}^1,\ j \in J = \{1, \ldots, l\}$ and a partition $J^r = \{J_1, \ldots, J_r\},\ I = \{1, \ldots, r\},\ 1 \leq r \leq l$ of the set J such that*

1) for all $i \in I$ and $a \in A$

$$\min_{j \in J_i} \left\{ \langle x^{ij}, a \rangle - y_{ij} \right\} < 0;$$

2) for any $b \in B$ there exists at least one $i \in I$ such that

$$\min_{j \in J_i} \left\{ \langle x^{ij}, b \rangle - y_{ij} \right\} > 0.$$

Remark 1. It follows from Definition 1 that if the sets A and B are max-min separable then $\varphi(a) < 0$ for any $a \in A$ and $\varphi(b) > 0$ for any $b \in B$, where the function φ is defined by (3.12). Thus the sets A and B can be separated by a function represented as a max-min of linear functions. Therefore this kind of separability is called a max-min separability.

Remark 2. Linear and polyhedral separability, introduced in [14] and [5], respectively, can be considered as particular cases of the max-min separability. If $I = \{1\}$ and $J_1 = \{1\}$ then we have the linear separability and if $I = \{1, \ldots, h\}$ and $J_i = \{i\},\ i \in I$ we obtain the h-polyhedral separability. Moreover, max-min separability is a generalization of the bilinear separation (see [11]).

Proposition 1. *(see [9]). The sets A and B are max-min separable if and only*

if there exists a set of hyperplanes $\{x^j, y_j\}$ *with* $x^j \in \mathbb{R}^n,\ y_j \in \mathbb{R}^1,\ j \in J$ *and a partition* $J^r = \{J_1, \ldots, J_r\},\ I = \{1, \ldots, r\},\ 1 \le r \le l$ *of the set* J *such that*

1) for any $i \in I$ *and* $a \in A$

$$\min_{j \in J_i} \left\{ \langle x^{ij}, a \rangle - y_{ij} \right\} \le -1;$$

2) for any $b \in B$ *there exists at least one* $i \in I$ *such that*

$$\min_{j \in J_i} \left\{ \langle x^{ij}, b \rangle - y_{ij} \right\} \ge 1.$$

Proposition 2. *(see [9]). The sets* A *and* B *are max-min separable if and only if there exists a continuous piecewise linear function separating them.*

Remark 3. It follows from Proposition 2 that the notions of max-min and piecewise linear separability are equivalent.

Proposition 3. *(see [9]). The sets* A *and* B *are max-min separable if and only if they are disjoint:* $A \bigcap B = \emptyset$.

Remark 4. Proposition 3 means any two disjoint finite point sets are max-min separable.

The next proposition shows that in most cases the number of hyperplanes necessary for the max-min separation of the sets A and B is not too large.

Proposition 4. *(see [9]). Assume that the set* A *can be represented as a union of sets* $A_i,\ i = 1, \ldots, q$ *and the set* B *as a union of sets* $B_j,\ j = 1, \ldots, d$ *such that*

$$A = \bigcup_{i=1}^{q} A_i, \quad B = \bigcup_{j=1}^{d} B_j$$

and

$$co\, A_i \bigcap co\, B_j = \emptyset \ \text{ for all } i = 1, \ldots, q,\ j = 1, \ldots, d. \tag{3.13}$$

Then the number of hyperplanes necessary for the separation of the sets A and B is at most $q \cdot d$.

Remark 5. Proposition 4 demonstrate that in most cases the cardinality of all sets of indices $J_i,\ i \in I$ is the same. If the assumptions of Proposition 4 are satisfied then the cardinality of all these sets is either d or q. We will use this fact for the design of an incremental algorithm.

3.3.5 Error function

[11] presents an algorithm for the max-min separability problem. They develop an algorithm to find that piecewise linear function by minimizing the following error function. This error function determines the quality of the of the max-min piecewise linear separation.

Given any set of hyperplanes $\{x^j, y_j\},\ j \in J = \{1, \ldots, l\}$ with $x^j \in \mathbb{R}^n,\ y_j \in \mathbb{R}^1$ and a partition $J^r = \{J_1, \ldots, J_r\},\ \ I = \{1, \ldots, r\},\ 1 \leq r \leq l$ of the set J, we say that a point $a \in A$ is well separated from the set B if the following condition is satisfied:

$$\max_{i \in I} \min_{j \in J_i} \left\{ \langle x^{ij}, a \rangle - y_{ij} \right\} + 1 \leq 0.$$

Then we can define the separation error for a point $a \in A$ as follows:

$$\max \left[0, \max_{i \in I} \min_{j \in J_i} \left\{ \langle x^{ij}, a \rangle - y_{ij} + 1 \right\} \right]. \tag{3.14}$$

Analogously, a point $b \in B$ is said to be well separated from the set A if the following condition is satisfied:

$$\min_{i \in I} \max_{j \in J_i} \left\{ -\langle x^{ij}, b \rangle + y_{ij} \right\} + 1 \leq 0.$$

Then the separation error for a point $b \in B$ can be written as

$$\max\left[0, \min_{i\in I}\max_{j\in J_i}\left\{-\langle x^{ij}, b\rangle + y_{ij} + 1\right\}\right]. \tag{3.15}$$

An averaged error function is defined as (see [9, 11])

$$f(X,Y) = f_1(X,Y) + f_2(X,Y) \tag{3.16}$$

$$f_1(X,Y) = (1/m)\sum_{k=1}^{m}\max\left[0, \max_{i\in I}\min_{j\in J_i}\left\{\langle x^{ij}, a^k\rangle - y_{ij} + 1\right\}\right],$$
$$f_2(X,Y) = (1/p)\sum_{t=1}^{p}\max\left[0, \min_{i\in I}\max_{j\in J_i}\left\{-\langle x^{ij}, b^t\rangle + y_{ij} + 1\right\}\right],$$

where $X = (x^{11}, \ldots, x^{lq_l}) \in \mathbb{R}^{nL},\ Y = (y_{11}, \ldots, y_{lq_l}) \in \mathbb{R}^{L},\ L = \sum_{i\in I} q_i,\ q_i = |J_i|,\ i \in I = \{1, \ldots, l\}$. $|J_i|$ denotes the cardinality of the set J_i. It is clear that $f(X,Y) \geq 0$ for all $X \in \mathbb{R}^{nL}$ and $Y \in \mathbb{R}^{L}$.

Proposition 5. *(see [9]). The sets A and B are max-min separable if and only if there exists a set of hyperplanes $\{x^j, y_j\}, j \in J = \{1, \ldots, l\}$ and a partition $J^r = \{J_1, \ldots, J_r\},\ \ I = \{1, \ldots, r\},\ 1 \leq r \leq l$ of the set J such that $f(x,y) = 0$.*

Remark 6. The error function (3.16) is nonconvex and if the sets A and B are max-min separable with the given number of hyperplanes, then the global minimum of this function $f(X^*, Y_*) = 0$ and the global minimizer is not always unique. Moreover, $X = 0 \in \mathbb{R}^{nL}$ cannot be an optimal solution [9].

The problem of max-min separability is reduced to the following mathematical programming problem:

$$\text{minimize } f(X,Y) \text{ subject to } (X,Y) \in \mathbb{R}^{(n+1)^L} \tag{3.17}$$

where the objective function f is described by equation (3.16).

Proposition 6. *(see [9]). Assume that the sets A and B are max-min separable with a set of hyperplanes $\{x^j, y_j\}, j \in J = \{1, \ldots, l\}$ and a partition $J^r = \{J_1, \ldots, J_r\}$, $I = \{1, \ldots, r\}$, $1 \leq r \leq l$ of the set J. Then*

1) $x^{ij} = 0$, $i \in I$, $j \in J$ cannot be an optimal solution;

2) if

(a) for any $t \in I$ there exists at least one $b \in B$ such that

$$\max_{j \in J_t} \left\{ -\langle x^{tj}, b \rangle + y_{tj} + 1 \right\} = \min_{i \in I} \max_{j \in J_i} \left\{ -\langle x^{ij}, b \rangle + y_{ij} + 1 \right\}, \quad (3.18)$$

(b) there exists $\tilde{J} = \{\tilde{J}_1, \ldots, \tilde{J}_r\}$ such that $\tilde{J}_t \subset J_t$, $\forall t \in I$, $\tilde{J}_t$ is nonempty at least for one $t \in I$ and $x^{tj} = 0$ for any $j \in \tilde{J}_t$, $t \in I$.

Then the sets A and B are max-min separable with a set of hyperplanes $\{x^j, y_j\}, j \in J^0$ and a partition $\bar{J} = \{\bar{J}_1, \ldots, \bar{J}_r\}$ of the set J^0 where

$$\bar{J}_t = J_t \setminus \tilde{J}_t, \ t \in I \quad \text{and} \quad J^0 = \bigcup_{i=1}^{r} \bar{J}_i.$$

In paper [11], an algorithm for solving problem (3.17) is presented. This algorithm exploits special structures of the error function such as piecewise partial separability.

In this algorithm it is assumed that the number of hyperplanes is known a priori. However this information is not always available. The classification accuracy is highly dependent on this number. A large number of hyperplanes may lead to overfitting of the training set. It is therefore imperative to calculate as few hyperplanes as needed to separate classes with respect to a given tolerance. The algorithms developed in this thesis use an incremental approach to find an appropriate number of piecewise linear functions to separate classes. A review of incremental learning algorithms is presented in section 3.5.

The complexity of the error function computation (3.16) depends on the number of data points. For data sets containing tens of thousands of points the error function becomes expensive to compute, and the algorithms proposed in [9, 11] become very time consuming. In the next section, section 3.4 a scheme to reduce the number of points is presented. The algorithms developed in chapters 5 and 6 build on this scheme.

Furthermore complexity is reduced at each iteration by eliminating points easily classified using simpler piecewise linear separators calculated at previous iterations. This allows for a significant reduction of the computational effort for large data sets. Also, this scheme reduces the risk of overfitting by only considering the data points that are relevant.

3.4 Data pre-classification

Pre-classification can be seen as the process of identifying data points for a given purpose. Examples of applications for such purposes include detection of outliers [111]. Different methodologies that identify points as either being "outliers" or "normal" are surveyed in [66]. In [66] an application for safety critical environments is presented, where the identification of outlier points can indicate abnormal running conditions. Examples include detecting defects in plane engines or flows in pipelines.

Another example includes filters and pattern discovery in images. In [76], the authors identify and then filter out pixels determined as "noise" in sonar images. They are then able to improve the quality of the images by removing these noisy points. Other image related work includes using clustering methods to determine so called "positive" and "negative" samples. These identified points help to build reliable models associated with a semantic concept to discover patterns in web images [144].

The use of clustering for the process of pre-classification is mainly based on prototype learning methods [45]. For examples of clustering techniques refer to ([100], [96], [124], and [10]). Prototype learning methods are widely used as data reduction schemes to train classification algorithms. They work by creating data sample prototypes, see figure 3.1, that aid in identifying candidate points to be removed from further training processes. This maybe one step in a multi-tiered approach to remove points.

Examples of such schemes include [142] in which the notion of encounter zones are introduced, see figure 3.2. Firstly creating prototypes from class centers, the encounter zones are defined as areas that contain points from the different classes or prototypes that overlap or are very close to one another. Also, [120] and [37] propose a scheme that uses the Condensed Nearest Neighbor (CNN) rule [62]. This scheme looks at finding reduced subsets of overlapping points, refer to section 3.2.2.

These schemes are ideal for the use in optimization based classifiers as they reduce the complexities associated with large data sets. A learning task can be simplified by eliminating the points that can be more easily classified. [142] shows that a reduction scheme used on a multi-optimization classifier reduces computations/time when training hyperplanes on subsets of the data. They also show that this helps to avoid overfitting of the data. Furthermore, [37] shows that when applying a generalized version of CNN as a pre-process step for SVM classifiers, the classifier produces better accuracy than without pre-classification.

[59] proposes a pre-classification scheme that takes points belonging to two data types and assigns them to three data types. Their approach is based on the separation law for convex sets by sets, where they attempt to separate the two sets of classes with another set. The three sets are labelled as 'malignant' (A), 'benign' (B), the class sets, and 'indeterminate' (S), the separating set. The data points belonging to set A and B contain only easily discriminated points for each

respective class. Whereas, set S contains indeterminate points from both sets A and B, and is empty in the case of linear separability.

The authors work with the underlying assumption that the data is "convex separable". Thus, the three regions A, B, and S containing data points from their respective classes and the indeterminate region, form convex sets. Therefore, there is no convex combination leading from set A to set B that does not enter through set S.

Given that a separating set, set S is the natural generalization from a hyperplane, the authors are particularly interested in constructing a separating polytope from set S, with a minimal volume. They use nonsmooth optimization to minimize the number of indeterminate points within set S.

If set S separates the sets A and B then the three types can be specified as:

Type I :malignant(the set $A \setminus S$)

Type II :benign(the set $B \setminus A$)

Type III :indeterminate (the set $S \bigcap (A \bigcup B)$).

The classification algorithms presented in the next chapters, chapter 4, 5 and 6 of this thesis, build on the idea of separating a data set into specific regions. These regions are useful for determining well classified data compared to data that is much harder to separate in terms of convex sets. As with the work presented in [59], nonsmooth optimization is applied to the indeterminate region, but in our case to build a piecewise linear decision boundary.

However, unlike the work in [59], our aim is not to find accurate separation regions, but instead to find a fast approximation of the three sets to minimize the number of points involved in building the decision boundary.

3.5 Incremental learning algorithms

Incremental learning algorithms are becoming increasingly popular in supervised and unsupervised data classification (see, for example,[135, 90, 78, 125]). Generally, they build on accumulative knowledge only adding complexity when needed. Intuitively this process would use less time and memory as it focuses its resources only on the relevant information.

The concept of incrementally learning has been a field of study since the early 1970s [158, 104]. Much of this work was in observing the way humans learn. From this research incremental machine learning models were derived.

Incremental learning can be seen through human learning. "People learn concept descriptions from facts and incrementally refine those descriptions when new facts or observations become available." [127]. The authors give two main reasons to why humans need to learn incrementally.

1. Sequential flow of information. The authors state that a human typically receives information in steps and that they must learn to deal with a situation before all information becomes available. As new information comes in, there is no time to reformulate all currently known information.

2. Limited memory and processing power. People are unable to store and access all information that they are exposed to. People tend to store the most prominent facts and generalizations and progressively modify these generalizations as more information is made available.

The authors can see the parallel between human learning and machine learning. Both receive information sequentially and both have to deal with it in such a way that the most important factors are stored, and the essential concepts are modified incrementally as new information arrives.

Furthermore, due to the scale of the data, limited memory and processing power is also an important factor. Therefore, any algorithm must be efficient

in being able to extract and build a model that can encapsulate the important information. Intuitively, an incremental learner would be ideal as it only adds or modifies to a model or rule as new, but different information arrives.

Building on this premise, the authors successfully build an automated incremental learning model which modifies concept descriptions [127], using only examples. They show that the incremental learning model can accommodate new information without forgetting previously seen facts. Their results show that the incrementally learning model significantly outperformed the single step learners, both in time and quality of the results.

Around the same time, work was done in defining a framework for incremental learning models. [137] outlines certain "learning from experience" components essential for an incremental learning model. With other authors including [92], a broad outline of the basic components needed to build an incremental machine learning algorithm was outlined. These are as follows,

1. Clustering the instances into known classes or groups. i.e. define a mapping between the observation and a class.

2. Initializing the groups, using descriptors or functional assignments to distinguish between groups in the model.

3. Project the matching description of new observation against subsequent collective changes in the model. Change the state of the model to reflect the update.

4. Evaluate the effectiveness of the function or model with the addition of the new observation.

5. Refine the model to improve the effectiveness as measured by the evaluation.

6. Aggregate the changes broadly across the model.

7. Store if the update successfully improves the model.

The above components allow for an incremental learning algorithm to take predefined groups of observations and convert them into a form recognizable to the data structures used by the learning algorithm. From there a sequential process that utilises the knowledge found in the previously given observations are added to a model or rule that allows for the algorithm to move closer to a global solution.

Therefore, the search space and starting point in step n is determined by the solution found in the previous step. Furthermore, at each iteration the algorithm moves closer to the global solution terminating when a criteria is met. This is usually a preset tolerance given an error margin or a predefined number of steps.

The evaluation, refinement and aggregation steps are all used to ensure that the algorithm sequentially builds an appropriate model or rule. This is based on a criteria or form of optimization that leads to a solution that solves the objective. The benefits, therefore, include less use of resources and complex tasks are handled in parts by sequentially building on more simple tasks.

For large scale problems, data points that do not contribute to the solution will be discounted and need not be seen again. Whereas only the points that are interesting to the learning algorithm and contribute to the solution are added to the model.

Optimization based classifiers are generally more time consuming and such a strategy is ideal. A single optimization piecewise linear classifier can build on its complexity as it constructs a decision boundary between classes. It would only increase its complexity if a separation criteria was not met.

In terms of refinement and aggregation, the classification algorithm would start with a linear separation then move to polyhedral separation and finally, if needed, using max-min separation. At each step, evaluating the rule to determine if the separation for the decision boundary meets the criteria. If so, then no need to increase the complexity any further.

3.6 Conclusions

In this chapter we reviewed a number of piecewise linear classification algorithms and found that most of the algorithms are suitable for real time classification. However, a number of the multiple optimization classification algorithms suffered from long training times. Some of the schemes used to improve on these training times led to a poor approximation of the decision boundaries. Thus making these classification techniques unsuitable for use on large scale data.

The SVM family of classifiers though very successful and well used, were shown to be problematic when needing to choose an appropriate function kernel and set of parameters for a given classification problem. Training times would be affected adversely, if time was needed in finding the appropriate parameters. Also, these classifiers maybe unsuitable for real time applications that can not use parameters.

A review of max-min separation and use as a classification method was presented in this chapter. As shown in this review, in theory, the separation is ideal for such applications involving nonlinear set separation for use in real time classification. However, choosing the least number of hyperplanes for set separation and avoiding overfitting of the data can be a problem.

Furthermore, this chapter presented a comprehensive review on incremental algorithms, highlighting the theoretical and practical advantages for use in large scale applications that can be solved sequentially. Also, a review on pre-classification schemes were presented. In this review, a number of examples showed the benefits of pre-classifying data into certain groups before other techniques are applied.

The developed classification algorithms presented in chapters 4, 5 and 6 builds on max- min separation. These classifiers address the problems of finding the necessary number of hyperplanes, overfitting and dealing with large scale data. This is done by using an incremental approach and pre- classifying points before

the training. The next three chapters develop three real time classifiers for the use in large scale data applications.

Chapter 4

Classification through incremental max-min separability

4.1 Introduction

The first piecewise linear classification technique proposed uses a new incremental algorithm to find piecewise linear boundaries between pattern classes. In this algorithm piecewise linear boundaries are built by gradually adding new hyperplanes until separation is obtained, with respect to some predefined tolerance.

Incremental learning algorithms are becoming increasingly popular in supervised and unsupervised data classification (see, for example,[78, 90, 125, 135]). Generally, they build on accumulative knowledge only adding complexity when needed, refer to section 3.5.

This type of approach breaks up the data set into observations that can be classified using simple separators, and observations that require more elaborate ones. This allows one to simplify the learning task by eliminating the points that can be more easily classified. Furthermore, at each iteration, information gathered during prior iterations can be exploited.

In the case of piecewise linear classifiers, this approach allows for the compu-

tation of as few hyperplanes as necessary to separate the sets, without any prior information. Additionally, this approach allows for the algorithm to reach a near global solution of the classification error function. At the same time, by using the piecewise linear function obtained at a given iteration as a starting point for the next iteration. Thus it reduces computational effort and avoids possible overfitting.

In this chapter we present the incremental algorithm and explain the classification rule. We discuss the implementation of the algorithm and how training is performed. We apply the proposed algorithm to solve supervised data classification problems using 15 publicly available data sets. We report the results of numerical experiments and compare the proposed classifier with 9 other mainstream classifiers found within the WEKA data mining suite. Finally we conclude on our results.

4.2 Incremental algorithm

In this section we describe an incremental algorithm for finding piecewise linear boundaries between finite sets. We assume that we are given a data set A with q classes: $A_1, \ldots, A_q$.

At each iteration of the algorithm we solve problem (3.17) with a preset number of hyperplanes to find a piecewise linear boundary between a given class and the rest of the data set. This is done for all classes using the one vs all approach. After computing piecewise linear boundaries for all classes we define data points which can be easily classified using the piecewise linear boundaries from this iteration. Then all these points are removed before the next iteration.

The algorithm stops when all remaining points, if any, belong to only one set. i.e. there are no sets left to separate. For each set, the previous iteration's classification accuracy and the objective function value are compared to the current

iteration's values. The difference in these values are used as the stopping criterion for the final piecewise linear boundary between this set and the rest of the data set.

For the sake of simplicity we split the incremental algorithm into two parts: Algorithm 1 (outer) and Algorithm 2 (inner). Algorithm 1 contains the main steps of the method. These steps are the initialization of starting points, the number of hyperplanes and the update of the set of undetermined points. Algorithm 2 is called at each iteration of Algorithm 1. It computes the piecewise linear boundaries for a given set; refines the set of undetermined points; updates starting points and the number of hyperplanes for the next iteration of Algorithm 1.

4.2.1 Algorithm

First, we describe the outer algorithm. Let $\varepsilon_0 > 0$ be a tolerance.

Algorithm 1. *An incremental algorithm*

1: *(*Initialization*) Set* $A_u^1 = A_u,\ Q_u^1 = \emptyset,\ u = 1, \ldots, q$. *Select any starting point* (x, y) *such that* $x \in \mathbb{R}^n, y \in \mathbb{R}^1$ *and set*

$$X^{1u} = x, Y_{1u} = y, \forall\ u = 1, \ldots, q.$$

Set

$$C^1 = \{1, \ldots, q\},\ I_{1u} = \{1\}, J_1^{1u} = \{1\}, r_{1u} = 1, s_{1u}^1 = 1,\ u = 1, \ldots, q,$$

the number of hyperplanes for class u: $l_{1u} = 1$ *and iteration counter* $k = 1$.

2: *(*Stopping criterion*) If* $|C^k| \leq 1$ *then stop. Otherwise go to Step 3.*

3: *(*Computation of piecewise linear functions*) For each* $u \in C^k$ *apply Algorithm 2. This algorithm generates a piecewise linear boundary* (X^{ku*}, Y_{ku*}), *the set of indices* $I_{k+1,u},\ J_i^{k+1,u},\ i \in I_{k+1,u}$, *a number of hyperplanes* $l_{k+1,u}$,

a starting point $(X^{k+1,u}, Y_{k+1,u}) \in \mathbb{R}^{(n+1)l_{k+1,u}}$ *for class* u*, the set* A_u^{k+1} *containing "undetermined" points and the set* Q_u^k *of easily separated points from class* u*.*

*4: (*Refinement of set C^k*) Refine the set* C^k *as follows:*

$$C^{k+1} = \{u \in C^k : \ |A_u^{k+1}| > \varepsilon_0 |A_u|\}.$$

Set $k = k + 1$ *and go to Step 2.*

We will now present the inner algorithm for separating class $A_u,\ u \in \{1, \ldots, q\}$ from the rest of the data set. At each iteration k of Algorithm 1 we get the subset $A_u^k \subseteq A_u$ of the set $u \in C^k$ which contains points from this class which are not easily separated using piecewise linear functions from previous iterations. Let

$$\overline{Q}_u^k = \bigcup_{j=1,\ldots,k} Q_u^j$$

be a set of all points removed from the set A_u during the first $k > 0$ iterations. We denote

$$D_k = \bigcup_{t=1,\ldots,q} \left(A_t \setminus \overline{Q}_t^k\right), \ \ \underline{A_u^k} = \bigcup_{t=1,\ldots,q, t\neq u} \left(A_t \setminus \overline{Q}_t^k\right).$$

Algorithm 2 finds a piecewise linear function separating the sets A_u^k and $\underline{A_u^k}$. Let $\varepsilon_1 > 0, \varepsilon_2 > 0, \varepsilon_3 > 0$ be given tolerances and $\sigma \geq 1$ be a given number.

Algorithm 2. *Computation of piecewise linear functions*

Input *Starting points* $(X^{ku}, Y_{ku}) \in \mathbb{R}^{(n+1)l_{ku}}$*, the set of indices* I_{ku}*,* $J_i^{ku},\ i \in I_{ku}$ *and the number of hyperplanes* l_{ku} *at iteration* k *of Algorithm 1.*

Output *A piecewise linear boundary* $(X^{ku*}, Y_{ku*}) \in \mathbb{R}^{(n+1)l_{ku}}$*, the set of indices* $I_{k+1,u}$*,* $J_i^{k+1,u},\ i \in I_{k+1,u}$*, a number of hyperplanes* $l_{k+1,u}$*, a starting point* $(X^{k+1,u}, Y_{k+1,u}) \in \mathbb{R}^{(n+1)l_{k+1,u}}$ *for class* u*, a set of undetermined points*

A_u^{k+1} and a set Q_u^{k+1} of easily separated points from class u.

1: (Finding a piecewise linear function) *Solve problem (3.17) over the set D_k starting from the point $(X^{ku}, Y_{ku}) \in \mathbb{R}^{(n+1)l_{ku}}$. Let (X^{ku*}, Y_{ku*}) be the solution to this problem, f_{ku}^* be the corresponding objective function value, and $f_{1,ku}^*$ and $f_{2,ku}^*$ be values of functions f_1 and f_2, respectively. Let E_{ku} be the error rate for separating the sets A_u^k and $\underline{A_u}^k$ at iteration k over the set A, that is*

$$E_{ku} = \frac{|\{a \in A_u^k : \varphi_u^k(a) > 0\} \cup \{b \in \underline{A_u^k} : \varphi_u^k(b) < 0\}|}{|A|},$$

where

$$\varphi_u^k(a) = \max_{i \in I_{ku}} \min_{j \in J_i^{ku}} \left(\langle x^{ij*}, a \rangle - y_{ij*} \right).$$

2: (The first stopping criterion) *If $\max\{f_{1,ku}^*, f_{2,ku}^*\} \leq \varepsilon_1$ then set $A_u^{k+1} = \emptyset$, $Q_u^{k+1} = A_u \setminus \overline{Q}_u^k$ and stop. (X^{ku*}, Y_{ku*}) is the piecewise linear boundary for set A_u.*

3: (The second stopping criterion) *If $k \geq 2$ and $f_{k-1,u}^* - f_{ku}^* \leq \varepsilon_2$ then set $A_u^{k+1} = \emptyset$, $Q_u^{k+1} = \emptyset$ and stop. (X^{ku*}, Y_{ku*}) where $X^{ku*} = X^{k-1,u*}, Y_{ku*} = Y_{k-1,u*}$ is the piecewise linear boundary for set A_u.*

4: (The third stopping criterion) *If $E_{ku} \leq \varepsilon_3$ then set $A_u^{k+1} = \emptyset$, $Q_u^{k+1} = A_u \setminus \overline{Q}_u^k$ and stop. (X^{ku*}, Y_{ku*}) is the piecewise linear boundary for set A_u.*

5: (Refinement of sets of undetermined points) *Compute*

$$f_{ku,min} = \min_{a \in \underline{A_u^k}} \varphi_u^k(a)$$

and the following set of easily classified points by function φ_u^k:

$$Q_u^{k+1} = \left\{ a \in A_u^k : \ \varphi_u^k(a) < \sigma f_{ku,min} \right\}.$$

Refine the set of undetermined points from the set A_u as follows:

$$A_u^{k+1} = A_u^k \setminus Q_u^{k+1},$$

*6: (*Adding new hyperplanes*)*

1. *If $f_{1,ku}^* > \varepsilon_1$ then set*

$$s_{k+1,u}^i = s_{ku}^i + 1, J_i^{k+1,u} = J_i^{ku} \cup \{s_{k+1,u}^i\}$$

for all $i \in I_{ku}$. Set

$$x^{ij} = x^{i,j-1,*}, y_{ij} = y_{i,j-1,*}, i \in I_{ku}, j = s_{k+1,u}^i.$$

2. *If $f_{2,ku}^* > \varepsilon_1$ then set*

$$r_{k+1,u} = r_{ku} + 1, I_{k+1,u} = I_{ku} \cup \{r_{k+1,u}\}, J_{r_{k+1,u}}^{k+1,u} = J_{r_{ku}}^{ku}.$$

Set

$$x^{ij} = x^{i-1,j,*}, y_{ij} = y_{i-1,j,*}, i = r_{k+1,u}, j \in J_{r_{ku}}^{ku}.$$

*7: (*New starting point*) Set*

$$X^{k+1,u} = (X^{ku*}, x_{ij},\ i \in I_{k+1,u},\ j \in J_i^{k+1,u}),$$

$$Y_{k+1,u} = (Y_{ku*}, y_{ij},\ i \in I_{k+1,u},\ j \in J_i^{k+1,u}),$$

$$l_{k+1,u} = \sum_{i \in I_{k+1,u}} |J_i^{k+1,u}|.$$

4.2.2 Explanations to the algorithm

The following explains Algorithm 1 in more details. In Step 1 we initialize the starting points, the number of hyperplanes for each class and the collection C^1 of sets to be separated. Step 2 is the stopping criterion verifying that the collection C^k contains at least two sets to separate. In the third step Algorithm 2 is called and returns piecewise linear boundaries for each set, the subsets of points not yet separated by these piecewise linear boundaries and updated starting points for the next iteration of Algorithm 1. In Step 4 we refine the set C^k by removing sets fully separated using piecewise linear boundaries from previous and current iterations.

The following explanations clarify Algorithm 2. In Step 1 we compute a piecewise linear function with a preselected number of hyperplanes using the starting point provided by Algorithm 1. It also computes the separation error rate between a given class u and the rest of the data set. The algorithm contains three stopping criteria which are given in Steps 2, 3 and 4.

- The algorithm terminates if both values $f^*_{1,ku}, f^*_{2,ku}$ for class u are less than a given tolerance $\varepsilon_1 > 0$. The last piecewise linear function for this class is accepted as a boundary between this class and the rest of the data set (Step 2).

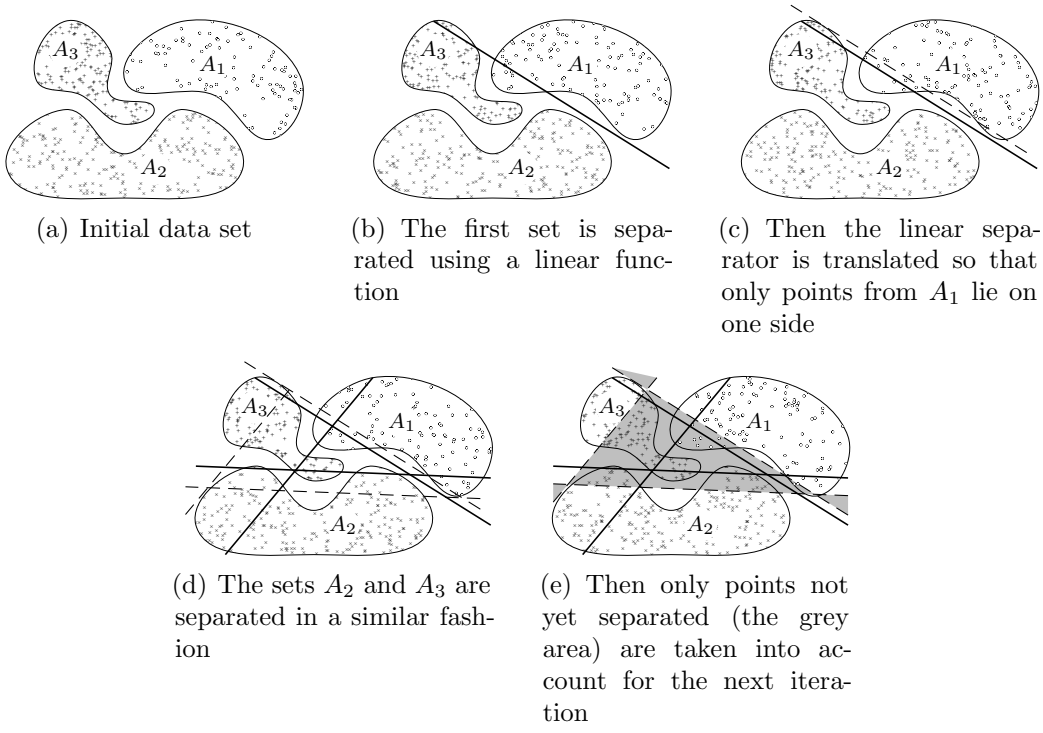
- If $k \geq 2$ and the difference between values of the error function (for class u) in two successive iterations is less than a given tolerance $\varepsilon_2 > 0$ then the algorithm terminates. The piecewise linear function from the previous iteration is accepted as the boundary between this class and the rest of the data set (Step 3).

- Finally, if the error rate is less than a threshold $\varepsilon_3 > 0$ then the algorithm terminates and the piecewise linear function from the last iteration is accepted as the boundary between this class and the rest of the data set (Step

4).

If none of these stopping criteria is met, then in Step 5 we refine the set of undetermined points by removing points easily separated using the piecewise linear functions from the current iteration. In Step 6, depending on the values of the error function on both sets, we may add new hyperplanes. Finally in Step 7 we update the starting point and the number of hyperplanes.

As an illustration Figure 4.1 shows the result of the first iteration of Algorithm 1 for a data set with three classes A_1, A_2 and A_3. At this iteration we compute one hyperplane for each set. The data set in its original form is illustrated in Figure 4.1(a). We select any starting point in Step 1 of Algorithm 1 and then call Algorithm 2 in Step 3. Algorithm 2 computes one linear function for each class using one vs all strategy. A hyperplane given in Figure 4.1(b) presents the linear function separating the class A_1 from the rest of the data set with the minimum error function value. This hyperplane is computed in Step 1 of Algorithm 2. Then in Step 5 of Algorithm 2 we compute a hyperplane (with dashed lines in Figure 4.1(c), here $\sigma = 1$) by translating the best hyperplane so that beyond this dashed line only points from the class A_1 lie. We remove all points from the class A_1 which lie beyond this line before the next iteration (Step 5 of Algorithm 2) and do not consider them in the following iterations. These data points can be easily classified using linear separation. We repeat the same computation for other classes A_2 and A_3 and remove all data points which can be classified using linear functions (see Figure 4.1(d)). Then we compute all data points which lie in the grey area in Figure 4.1(e). These points cannot be determined by linear separators and we use only these points to compute piecewise linear boundaries in the next iteration of Algorithm 1.



(a) Initial data set

(b) The first set is separated using a linear function

(c) Then the linear separator is translated so that only points from A_1 lie on one side

(d) The sets A_2 and A_3 are separated in a similar fashion

(e) Then only points not yet separated (the grey area) are taken into account for the next iteration

Figure 4.1: The first iteration of Algorithm 1 for three sets A_1, A_2 and A_3.

4.3 Classification rules

At each iteration k, $k \geq 1$, Algorithm 1 generates a piecewise linear boundary (X^{ku}, Y_{ku}), the set A_u^{k+1} of "undetermined" points and the set Q_u^{k+1} of easily separated points. After the algorithm stops all final piecewise linear boundaries $\varphi_1, \ldots, \varphi_q$ have been obtained.

If the new point belongs to a set $Q_u^k \setminus \bigcup\{Q_k^t,\ t = 1, \ldots, q,\ t \neq u\}$ then it is classified in set u. Otherwise it is associated with one function φ_u per class. In this case a new point v is attributed a set of function values $\{\varphi_1(v), \ldots, \varphi_q(v)\}$. We classify this point to the class associated with the minimum function value: $i = \text{argmin}\{\varphi_1(v), \ldots, \varphi_u(v)\}$.

Figure 4.2 shows this classification rule in the case of the separation between three sets: the easily separated areas at the first iteration are unshaded. The light shaded area represents the points easily separated using the piecewise linear function from the second iteration of Algorithm 1, and the dark shaded area

represents the points separated using the final piecewise linear separating function returned by Algorithm 1.

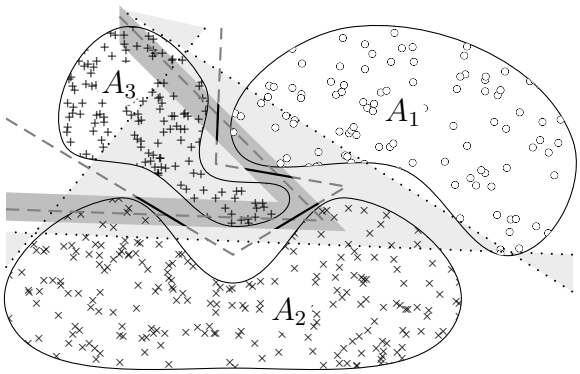


Figure 4.2: Classification rule between three sets A_1, A_2 and A_3 using Algorithm 1

4.4 Implementation of the algorithm

In this section we describe conditions for the implementation of the classification algorithm.

In Algorithm 1 we choose $\varepsilon_0 = 0.01$ for the value of the tolerance ε_0.

The following conditions have been chosen for the implementation of Algorithm 2.

1. The values of tolerances $\varepsilon_1 > 0, \varepsilon_2 > 0$ and $\varepsilon_3 > 0$ are:

$$\varepsilon_1 = 0.005,\ \varepsilon_2 = f_1^*/100,\ \varepsilon_3 = 0.001,$$

   where f_1^* is the optimal value of the objective function for linear separation. The number $\sigma = 1.25$.

2. In the proposed algorithm we restrict the number of hyperplanes to 10.

3. In Step 1 of Algorithm 2 we solve minimization problem (3.17). We use the discrete gradient method of [7, 8] as modified in [11].

In all data sets we apply our algorithm on a training set and test our classification rules on a test set. In our experiments we use data sets with known training and test sets.

We implemented the algorithm in Fortran 95 and compiled it using the Lahey Fortran compiler on an Intel Pentium IV 1.83GHz CPU with 1GB of RAM running Windows XP.

4.5 Numerical Experiments

We tested the proposed algorithm (Algorithm CIMMS - Classification through incremental max-min separability) on real world data sets readily available from the UCI machine learning repository [6]. The data sets were selected as follows: they have either continuous or integer attributes and no missing values. Table 4.1 contains a brief description of the characteristics of the data sets. The number of attributes in these tables also includes the class attribute.

In our experiments we used some classifiers from WEKA (Waikato Environment for Knowledge Analysis) for comparison.

We selected classifiers with fast testing time and/or based on separating functions: Naive Bayes (with kernel), Logistic Regression based classifier Logistic, Multi-Layer Perceptron (MLP), Linear LibSVM (LIBSVM (LIN)), SMO with normalized polynomial kernel (SMO (NPOL)), SMO (PUK), decision tree classifier J48 (which is an implementation of the C4.5 algorithm) and the rule based classifier PART. In addition we tested the classifier based on polyhedral separability [5]. Since the number of hyperplanes in polyhedral separability is not known a priori, we tested this algorithm with 2 to 5 hyperplanes and report only the best results on test sets with the corresponding CPU time.

We apply all algorithms with default values of parameters. For most classifiers it is possible, but not always easy, to find better parameters for each data set

Table 4.1: Brief description of data sets

Data sets	(train,test)	No. of attributes	No. of classes
Abalone (AB)	(3133,1044)	9	3
DNA	(2000, 1186)	180	3
Image segmentation (SEG)	(1848,462)	20	7
Landsat satellite image (LSI)	(4435,2000)	37	6
Letter recognition (LET)	(15000,5000)	17	26
Optical recognition of handwritten digits (OD)	(3823, 1797)	65	10
Pen-based recognition of handwritten digits (PD)	(7494,3498)	17	10
Phoneme_CR (PHON)	(4322, 1082)	6	2
Shuttle control (SH)	(43500,14500)	10	7
Texture_CR (TEXT)	(4400, 1100)	41	11
Vehicle (VEH)	(679,167)	19	4
Yeast (YEAST)	(1191, 293)	9	10
Isolet (ISO)	(6238, 1559)	618	26
Page blocks (PB)	(4000,1473)	11	5
Spambase (SB)	(3682,919)	58	2

which will produce a better accuracy than the one we report. We put limits of 3 hours (for training and testing) and 1GB of RAM. A dash line in the tables shows that the corresponding algorithm exceeded one of these limits.

Results of numerical experiments are presented in Tables 4.2-4.4. In these tables we present accuracy for test sets.

Table 4.2: Results of numerical experiments: test set accuracy.

Algorithm	AB	DNA	SEG	LSI	LET
NB(kernel)	57.85	93.34	85.71	82.10	74.12
Logistic	64.27	88.36	96.75	83.75	77.40
MLP	63.51	93.68	97.40	88.50	83.20
LIBSVM (LIN)	60.73	93.09	94.37	85.05	82.40
SMO (NPOL)	60.25	95.36	94.81	79.60	82.34
SMO (PUK)	64.18	57.93	97.19	91.45	-
J48	60.15	92.50	96.97	85.35	87.70
PART	57.95	91.06	96.75	85.25	87.32
Polyhedral	65.23	94.10	96.10	87.00	88.68
CIMMS	65.80	93.42	97.19	88.15	91.90

Table 4.3: Results of numerical experiments: test set accuracy (cont).

Algorithm	OD	PD	PHON	SH	TEXT
NB(kernel)	90.32	84.13	76.53	98.32	81.00
Logistic	92.21	92.85	74.58	96.83	99.64
MLP	96.55	89.85	81.52	99.75	99.91
LIBSVM (LIN)	96.55	95.00	77.54	-	99.18
SMO (NPOL)	96.66	96.86	78.74	96.81	97.27
SMO (PUK)	96.61	97.88	83.27	99.50	99.55
J48	85.75	92.05	85.67	99.95	93.91
PART	89.54	93.65	82.72	99.98	93.82
Polyhedral	96.05	97.03	79.02	99.29	99.91
CIMMS	94.27	96.63	81.05	99.84	99.82

Table 4.4: Results of numerical experiments: test set accuracy (cont).

Algorithm	VEH	YEAST	ISO	PB	SB
NB(kernel)	59.88	57.34	-	88.39	76.17
Logistic	77.84	58.02	-	91.72	92.06
MLP	82.04	56.66	-	92.80	92.06
LIBSVM (LIN)	71.86	54.95	96.02	87.03	90.97
SMO (NPOL)	72.46	54.95	-	89.48	92.60
SMO (PUK)	74.25	60.75	-	88.53	93.04
J48	73.05	56.31	83.45	93.55	92.93
PART	74.85	54.61	82.81	92.46	91.40
Polyhedral	86.23	56.66	-	87.98	92.71
CIMMS	82.63	53.24	95.19	87.10	93.80

Based on these results we can draw the following conclusions:

- The proposed algorithm obtained the best or close to the best accuracy on 7 data sets: Abalone, Image segmentation, Letter recognition, Shuttle control, Texture_CR, Spambase and Isolet.

- On 5 data sets (DNA, Landsat satellite image, Pen-based recognition of handwritten digits, Phoneme_CR and Vehicle) the proposed algorithm is among the classifiers with the best accuracy.

- On the other 3 data sets (Page blocks, Optical Recognition of handwritten digits and Yeast) the algorithm did not perform well compared with most of the other classifiers.

In Figure 4.5 we present pairwise comparison of the proposed algorithm with existing ones graphically. For each algorithm we calculated the following value:

$$R = \frac{\text{Acc}}{\text{Acc(CIMMS)}} - 1.$$

Here Acc is a test set accuracy obtained by an algorithm and $Acc(CIMMS)$ is a test set accuracy obtained by the proposed algorithm. The values R are displayed for all the data sets, in the same order they are in Table 4.1. The horizontal line represents a threshold: if the curves lie below this line, the proposed algorithm outperformed the existing algorithm for this data set. These graphs demonstrate that on most of data sets the proposed classifier outperforms other classifiers.

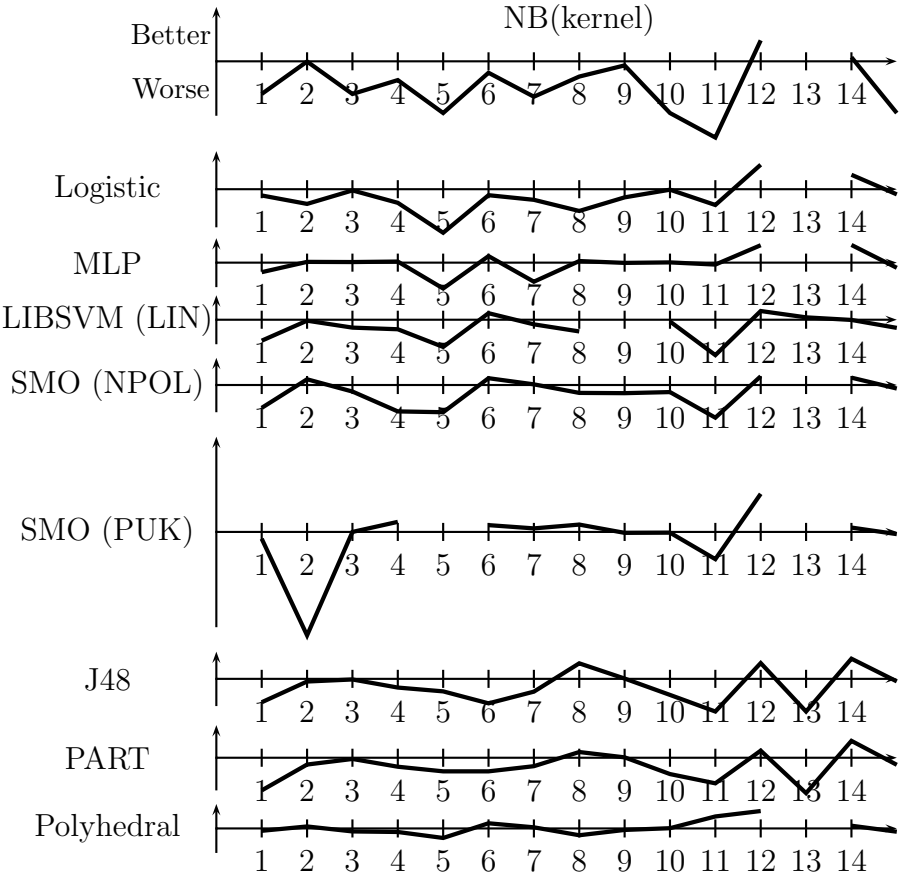


Table 4.5 presents training time required by two classifiers: Polyhedral and CIMMS. We do not include training time of other classifiers because they were

implemented on different platform.

Table 4.5: Results of numerical experiments: CPU time for Polyhedral and CIMMS algorithms.

Data set	Polyhedral	CIMMS	Data set	Polyhedral	CIMMS
AB	12.83	27.22	SH	1247.02	782.47
DNA	90.23	32.06	TEXT	128.41	47.28
SEG	39.97	17.20	VEH	58.09	17.25
LSI	1231.13	523.28	YEAST	20.19	73.67
LET	8941.81	9941.34	ISO	-	3927.36
OD	524.66	81.88	PB	51.48	27.63
PD	729.92	203.02	SB	228.75	295.23
PHON	0.89	34.75			

In comparison with most classifiers, and on most data sets, the proposed algorithm requires a longer training time. However, the comparisons with the Polyhedral classifier show that on some data sets the proposed classifier requires significantly less training time.

Overall, we can observe that the proposed classifier achieves consistently good classification accuracy on the test set. CPU time is reasonable, the testing time and memory usage is very low.

4.6 Conclusion

In this chapter we have developed an incremental algorithm for the computation of piecewise linear boundaries of finite point sets. At each iteration of this algorithm a new piecewise linear boundary is constructed for each class, using a starting point constructed from the boundaries obtained at previous iterations.

The new boundaries are used to eliminate points that they can easily separate. Other "undetermined" points are identified for separation at further iterations. This allows us to significantly reduce the computational effort, while still reaching a near global minimizer of the classification error function. This piecewise linear boundary separates the sets with as few hyperplanes as needed with respect to a given tolerance.

We tested the new algorithm on 15 real world data sets. Computational results demonstrate that the new classifier achieves good classification accuracy while requiring reasonable training time. This classifier can be used for real-time classification and has a low memory requirement, so it can be used on many portable devices.

Chapter 5

A piecewise linear classifier based on polyhedral conic and max-min separabilities

5.1 Introduction

The previous piecewise linear classifier proposed in chapter 4 presents an incremental algorithm. As we recall, that algorithm finds piecewise linear boundaries between pattern classes by gradually adding new hyperplanes until a separation of classes is obtained with respect to some predefined tolerance.

The aim of this piecewise linear classifier, as also in chapter 4 is to reduce computational effort and improve the separation quality. In the first stage of this algorithm polyhedral conic functions, introduced in [57], are used to identify data points which are on or close to the boundary between classes. In the second stage max-min separability is applied to find a piecewise linear boundary using only those data points. This boundary is constructed incrementally starting with linear separation and adding hyperplanes as needed.

Furthermore, as shown in the previous chapter the advantage of an incre-

mental approach is that at each iteration of the process we refine our scope by removing the points which are furthest from the piecewise linear boundary. This in turn reduces the complexity of the error function and thus allows for better separation accuracy and less time for the underlying optimization algorithm to converge to a solution.

In this chapter we present a classifier based on polyhedral conic functions. Firstly, we give a brief description of polyhedral conic functions. Then, using polyhedral conic functions and the incremental algorithm presented in section 4.2 we solve the max-min separability problem.

We report results of numerical experiments using 12 real-world data sets. We also provide the comparison of the new piecewise linear classifier with 9 other classifiers from the WEKA suite. We finally conclude the chapter presenting our results.

5.2 Polyhedral conic sets and max-min separability

As previously mentioned, the algorithm proposed in this chapter consists of two main stages. In the first stage we use polyhedral conic functions to identify points which lie on or close to the boundary between pattern classes. These functions are formed as an augmented l_1-norm with a linear part added. A graph of this type of function is a polyhedral cone with a level set containing an intersection of at most 2^n half spaces. Due to this property we formulate a linear programming problem to effectively and easily identify the boundary points between classes. A more detailed description of polyhedral conic functions and max-min separability can be found in [57] and [9, 11], respectively. In this section we briefly describe polyhedral conic functions.

Let A and B be given disjoint sets in $\mathbb{R}^n$ containing m and p points, respec-

tively:

$$A = \{a^1, \ldots, a^m\}, a^i \in \mathbb{R}^n, i = 1, \ldots, m,$$
$$B = \{b^1, \ldots, b^p\}, b^j \in \mathbb{R}^n, j = 1, \ldots, p.$$

5.2.1 Separation via polyhedral conic functions

Polyhedral conic functions (PCFs) have recently been proposed to construct a separation function for the sets A and B [57]. Definition 2 and Lemma 1 quoted below are given in [57].

Definition 2. *A function $g : \mathbb{R}^n \to \mathbb{R}$ is called* polyhedral conic *if its graph is a cone and all its level sets*

$$S(\alpha) = \{x \in \mathbb{R}^n : g(x) \leq \alpha \},$$

for $\alpha \in \mathbb{R}$, are polyhedrons.

Given $w, c \in \mathbb{R}^n, \xi, \gamma \in \mathbb{R}$ a polyhedral conic function $g_{(w,\xi,\gamma,c)}$: $\mathbb{R}^n \to \mathbb{R}$ is defined as follows:

$$g_{(w,\xi,\gamma,c)}(x) = \langle w, (x - c)\rangle + \xi \|x - c\|_1 - \gamma, \tag{5.1}$$

where $\|x\|_1 = |x_1| + \cdots + |x_n|$ is an l_1-norm of the vector $x \in \mathbb{R}^n$ and $\langle \cdot, \cdot \rangle$ is an inner product in $\mathbb{R}^n$.

Lemma 1. *A graph of the function $g_{(w,\xi,\gamma,c)}$ defined in equation (5.1) is a polyhedral cone with a vertex at $(c, -\gamma) \in \mathbb{R}^n \times \mathbb{R}$. We call this cone a polyhedral conic set and c its center.*

The sets A and B are polyhedral conic separable if there exist a finite number

of PCFs $g_l = g_{(w^l,\xi^l,\gamma^l,c^l)},\ l = 1,\ldots,L$ such that

$$\min_{l=1,\ldots,L} g_l(a) \leq 0 \ \ \forall\ a \in A$$

and

$$\min_{l=1,\ldots,L} g_l(b) > 0 \ \ \forall\ b \in B.$$

An error function can be formulated as follows:

$$\Phi(w^1,c^1,\xi^1,\gamma^1,\ldots,w^L,c^L,\xi^L,\gamma^L) = \frac{1}{m}\sum_{a\in A}\max\left\{0,\min_{l=1,\ldots,L} g_l(a)\right\} + \frac{1}{p}\sum_{b\in B}\max\left\{0,-\min_{l=1,\ldots,L} g_l(b)\right\}. \quad (5.2)$$

Then the problem of finding polyhedral conic functions separating sets A and B is reduced to the following mathematical programming problem:

$$\text{minimize } \Phi(w^1,c^1,\xi^1,\gamma^1,\ldots,w^L,c^L,\xi^L,\gamma^L) \quad (5.3)$$

subject to $w^i, c^i \in \mathbb{R}^n,\ \xi^i,\gamma^i \in \mathbb{R},\ i = 1,\ldots,L.$

An algorithm generating a polyhedral conic separating function, therefore called a PCF algorithm, is developed in [57].

The way a PCF can separate two sets A and B in $\mathbb{R}^2$ is shown in Figure 5.1. In this figure three different situations are shown according to the level sets of the corresponding functions. For each situation point c is placed at the origin (see (5.1)). Even though A and B are linearly inseparable, the constructed polyhedral conic functions can completely separate these sets. In (a), the level set of the obtained function is the intersection of three halfspaces: $x_1 \geq -1$, $x_2 \leq 1$ and $-x_1 + x_2 \leq 1$. In (b) and (c), new points $(1,-6)$ and $(7,-4)$, respectively, are added to their set B. Figure 5.2 shows how a PCF can be used to approximate

classes in three class data set.

In the next section we combine the incremental algorithm presented in section 4.2 with PCF separability to propose a new classification algorithm.

Figure 5.3 shows the result of the first iteration of the algorithm for three classes A_1, A_2 and A_3. At this iteration we compute one hyperplane for each class. The solid lines represent the linear separating functions for each class computed by solving Problem (3.17). The dashed lines represent their translations beyond which only points from one class lie. Points from the grey region are used to compute piecewise linear boundaries in the next iteration of the algorithm.

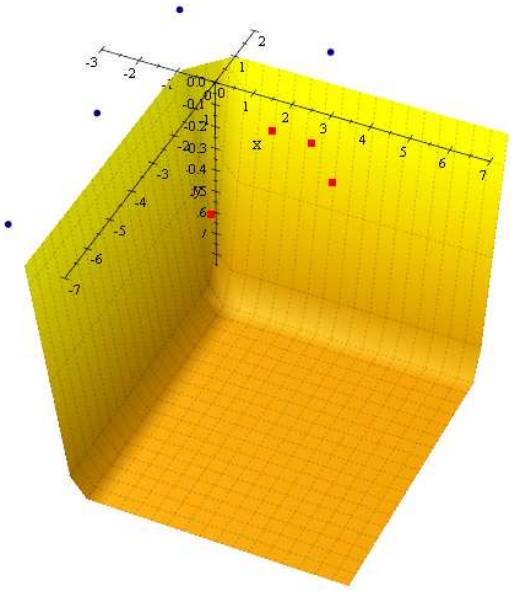
5.2.2 Explanations to the algorithm

The following explanations clarify Algorithm 2. In Step 1 we compute a piecewise linear function with a preselected number of hyperplanes using the starting point provided by Algorithm 1. It also computes the separation error rate between a given class u and the rest of the data set. The algorithm contains three stopping criteria which are given in Steps 2, 3 and 4.

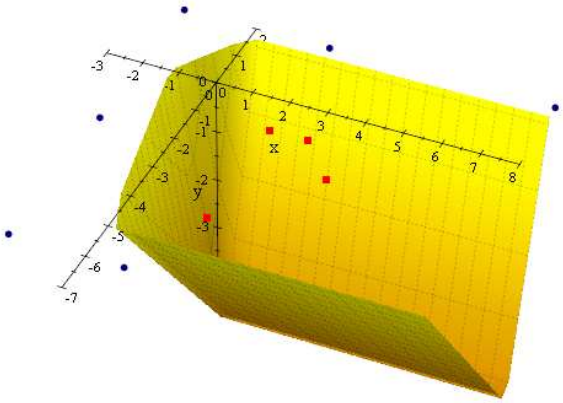
- The algorithm terminates if both values $f^*_{1,ku}, f^*_{2,ku}$ for class u is less than a given tolerance $\varepsilon_1 > 0$. The last piecewise linear function for this class is accepted as a boundary between this class and the rest of the data set (Step 2).

- If $k \geq 2$ and the difference between values of the error function (for class u) in two successive iterations is less than a given tolerance $\varepsilon_2 > 0$ then the algorithm terminates. The piecewise linear function from the previous iteration is accepted as the boundary between this class and the rest of the data set (Step 3).

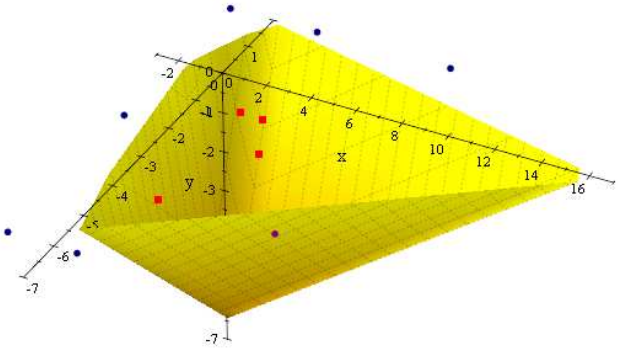
- Finally, if the error rate is less than a threshold $\varepsilon_3 > 0$ then the algorithm terminates and the piecewise linear function from the last iteration is ac-



$A = \{(2,-1),(2,-4),(3,-1),(4,-2)\}$
$B = \{(-2,2),(-2,-2),(-2,-6),(2,2),(8,2),(1,-6)\}$
$g(x_1,x_2) = -0.5x_1 + 0.5x_2 + 0.5(|x_1| + |x_2|) - 1$
(a)



$B = \{(-2,2),(-2,-2),(-2,-6),(2,2),(8,2),\mathbf{(1,-6)}\}$
$g(x_1,x_2) = -2x_1 + x_2 + 2(|x_1| + |x_2|) - 5$
(b)



$B = \{(-2,2),(-2,-2),(-2,-6),(2,2),(8,2),(1,-6),\mathbf{(7,-4)}\}$
$g(x_1,x_2) = -1.9x_1 + 1.1x_2 + 2.3(|x_1| + |x_2|) - 6.6$
(c)

Figure 5.1: 3d graph of polyhedral conic functions with different level sets

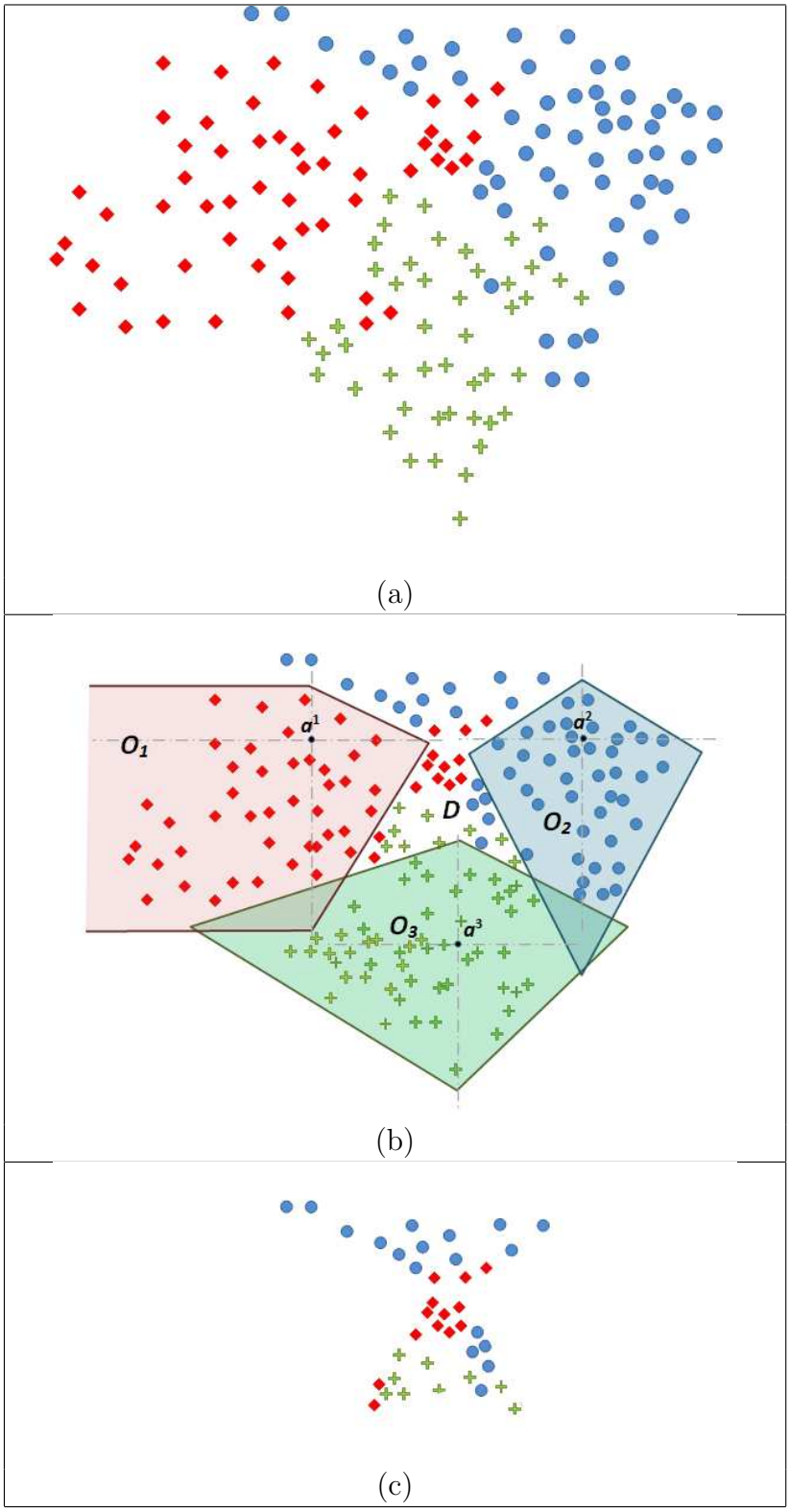
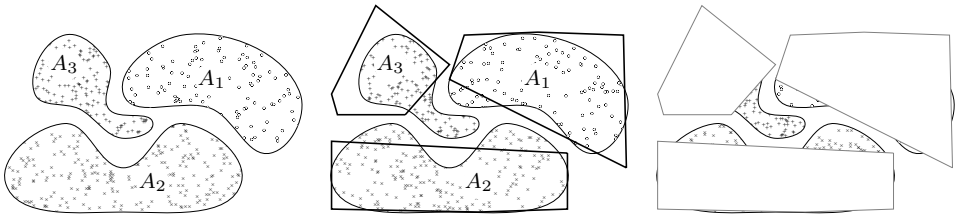


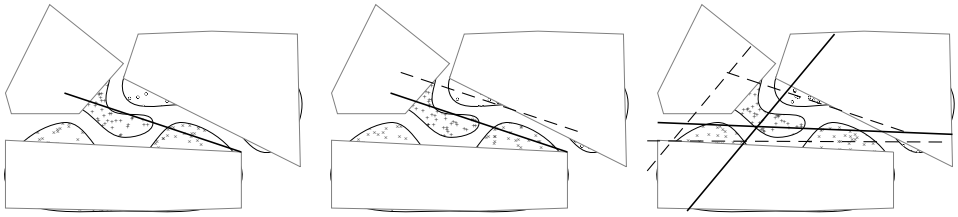
Figure 5.2: Approximations of classes for three class data set in $\mathbb{R}^2$



(a) Initial dataset

(b) Approximating the sets using PCF

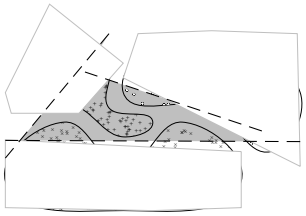
(c) Removing the points using PCF



(d) The first set is separated using a linear function

(e) Then the linear separator is translated so that only points from A_1 lie on one side

(f) The sets A_2 and A_3 are separated in a similar fashion



(g) Then only points not yet separated (the grey area) are taken into account for the next iteration

Figure 5.3: The first iteration of Algorithm 1 for three sets A_1, A_2 and A_3.

cepted as the boundary between this class and the rest of the data set (Step 4).

If none of these stopping criteria is met, then in Step 5 we refine the set of undetermined points by removing points easily separated using the piecewise linear functions from the current iteration. In Step 6, depending on the values of the error function on both sets, we may add new hyperplanes. Finally in Step 7 we update the starting point and the number of hyperplanes.

5.3 The hybrid polyhedral conic and max-min separability algorithm

Algorithm 1 allows one to find piecewise linear boundaries between pattern classes. At each iteration function (3.16) is minimized. The complexity of the computation of this function depends on the number of data points. However, in large data sets many data points lie far away from other classes. Therefore they are not relevant to the computation of the boundary between their class and other classes. We propose to identify these points using one PCF in order to eliminate them before applying Algorithm 1.

If we fix the center c, then the problem of minimizing function (5.2) with one PCF is a linear programming problem. Furthermore, if this center is not fixed, then the PCF may eliminate points which are close to other classes. It is preferable to take as a center a data point which is far away from the boundary of its associated class. In order to find such a point we propose to use hyperboxes approximating classes. Then we select a center lying inside only one hyperbox when possible (Step 1).

Using these centers we minimize error function (5.2). As a result we find a PCF approximating the interior of the classes (Step 2). Then we eliminate those points from the data set and apply Algorithm 1 to the remaining points (Step 3).

In the sequel we explain each step of this algorithm in more detail.

5.3.1 Computation of centers of polyhedral conic sets

In Step 1 we approximate each class by one hyperbox.

Assume that we are given data set A with $q \geq 2$ classes $A_1, \ldots, A_q$. For each class A_i we compute:

$$\bar{\alpha}_j^i = \min_{a \in A_i} a_j, \quad \bar{\beta}_j^i = \max_{a \in A_i} a_j, \ j = 1, \ldots, n, \ i = 1, \ldots, q$$

Figure 5.4: Identification of the centers of the polyhedral conic sets for the two classes A_1 and A_2 using hyperboxes

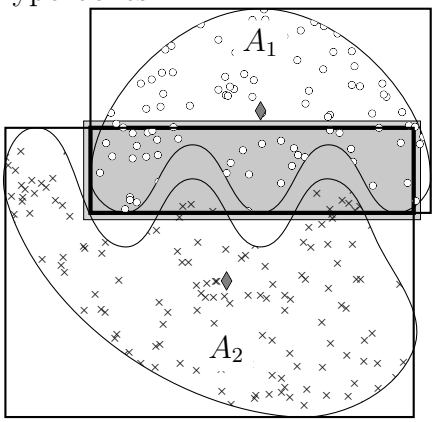
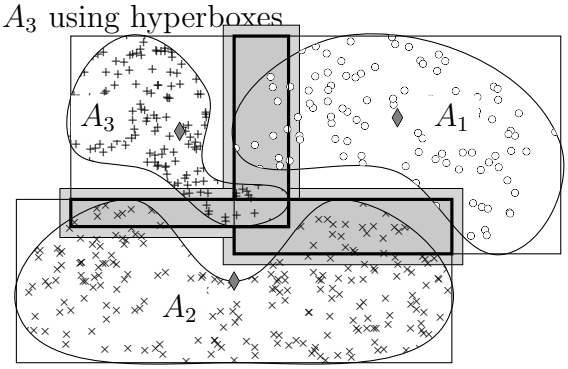


Figure 5.5: Identification of the centers of the polyhedral conic sets for the three classes A_1, A_2 and A_3 using hyperboxes



and define vectors $\bar{\alpha}^i = (\bar{\alpha}^i_1, \ldots, \bar{\alpha}^i_n)$, $\bar{\beta}^i = (\bar{\beta}^i_1, \ldots, \bar{\beta}^i_n)$, $i = 1, \ldots, q$ which in turn define the following hyperboxes in n-dimensional space $\mathbb{R}^n$ for $i = 1, \ldots, q$:

$$\bar{H}(A_i) = [\bar{\alpha}^i, \bar{\beta}^i] \equiv \left\{x \in \mathbb{R}^n : \ \bar{\alpha}^i_j \leq x_j \leq \bar{\beta}^i_j, \ j = 1, \ldots, n\right\}.$$

All points from the i-th class belong to the hyperbox $\bar{H}(A_i)$.

To ensure that the centers of the polyhedral conic sets lie inside the classes we take a sufficiently small $\eta > 0$ and consider the following extended hyperbox for each class A_i, $i = 1, \ldots, q$:

$$H(A_i) = [\alpha^i, \beta^i] \equiv \left\{x \in \mathbb{R}^n : \ \alpha^i_j \leq x_j \leq \beta^i_j, \ j = 1, \ldots, n\right\},$$

where for $j = 1, \ldots, n$

$$\alpha_j^i = \bar{\alpha}_j^i - \eta(\bar{\beta}_j^i - \bar{\alpha}_j^i), \quad \beta_j^i = \bar{\beta}_j^i + \eta(\bar{\beta}_j^i - \bar{\alpha}_j^i).$$

The hyperbox $H(A_i)$ can be described as

$$H(A_i) = \{x \in \mathbb{R}^n : \ \psi_i(x) \leq 0\},$$

where the piecewise linear function $\psi_i(x)$ is defined as follows:

$$\psi_i(x) = \max\left\{\alpha_j^i - x_j, x_j - \beta_j^i, \ j = 1, \ldots, n\right\}$$

In order to find the center for the i-th polyhedral conic set we define the set

$$R_i = \left\{a \in A_i : \min_{k=1,\ldots,q, \ k \neq i} \psi_k(a) > 0\right\}.$$

This set contains all points from the i-th class which are outside hyperboxes of all other classes. First we consider the case when the set $R_i \neq \emptyset$. Figures 5.4 and 5.5 illustrate this case. We compute

$$\bar{Q}_1 = \min_{a \in R_i} \psi_i(a)$$

and choose the center c^i as follows:

$$c^i \in R_i, \psi_i(c^i) = \bar{Q}_1.$$

If $R_i = \emptyset$ then for any $a \in A_i$

$$\min_{k=1,\ldots,q, \ k \neq i} \psi_k(a) \leq 0.$$

In this case we compute

$$\bar{Q}_2 = \max_{a \in A_i} \min_{k=1,\ldots,q,\ k \neq i} \psi_k(a)$$

and choose the center c^i as follows:

$$c^i \in A_i, \min_{k=1,\ldots,q,\ k \neq i} \psi_k(c^i) = \bar{Q}_2.$$

5.3.2 Identification of boundary points

To identify boundary points we minimize function (5.2) with $L = 1$ and using center c^i for each class $i = 1, \ldots, q$. This is a linear programming problem and after solving it we find the values for the vector w^i and scalars $\xi_i,\ \gamma_i$ which define polyhedral conic functions g_i for each class $i = 1, \ldots, q$. Then for a given class $i \in \{1, \ldots, q\}$ we compute the following:

$$\bar{\delta}_i = \min_{j=1,\ldots,q, j \neq i} \min_{b \in A_j} g_i(b).$$

Consider the level sets of the function $g_i,\ i = 1, \ldots, q$:

$$S_i(\delta) = \{x \in \mathbb{R}^n :\ g_i(x) \leq \delta\},\ \delta \in \mathbb{R}.$$

If $\bar{\delta}_i > 0$ then the set $S_i(0)$ does not contain points from any other classes, it only contains points from the i-th class. If $\bar{\delta}_i \leq 0$ then the set $S_i(0)$ also contains points from other classes. Therefore we replace $\bar{\delta}_i$ by $\hat{\delta}_i = \min\{0, \bar{\delta}_i\}$. To ensure that boundary points are not removed, $\hat{\delta}_i$ is again replaced by the following number:

$$\delta_i = \hat{\delta}_i - \theta(|\gamma_i| - \hat{\delta}_i)$$

where $\theta > 0$ is a sufficiently small number. For each class i we can define the following sets:

$$D_i = \{a \in A_i : g_i(a) \leq \delta_i\}, \ i = 1, \ldots, q. \tag{5.4}$$

The set D_i approximates the interior of the i-th class and it does not contain points from other classes. We then define the set of boundary points as follows:

$$B_i = A_i \setminus D_i, \ i = 1, \ldots, q. \tag{5.5}$$

Let $\sigma \in (0, 1)$ be a sufficiently small number. For each class $i = 1, \ldots, q$ we introduce the following number:

$$r_i = |B_i|/|A_i|$$

and then we consider the set

$$P = \{i = 1, \ldots, q : \ r_i > \sigma\}. \tag{5.6}$$

If $P = \emptyset$ then classes $A_i, \ i = 1, \ldots, q$ can be approximated by their corresponding sets D_i with the accuracy σ. Otherwise we can apply Algorithm 1 over sets $B_i, \ i \in P$ to find piecewise linear boundaries between classes.

5.3.3 Outline of the algorithm

In summary an algorithm for finding piecewise linear boundaries between classes $A_i, \ i = 1, \ldots, q$ can be formulated as follows:

Algorithm 3. *Computation of piecewise linear boundaries*

1: *(*Finding a center of a polyhedral conic set*) Approximate each class* $i = 1, \ldots, q$ *with the hyperbox* $H(A_i)$ *and compute the center* c^i *of the corresponding polyhedral conic set (see Subsection 5.3.1).*

*2: (*Identifying boundary points*) Compute polyhedral conic sets by minimizing function (5.2) and using the centers* $c^i,\ i = 1, \ldots, q$. *Find the sets* $D_i,\ i = 1, \ldots, q$ *using equation (5.4) and the sets* $B_i,\ i = 1, \ldots, q$ *of boundary points using equation (5.5) (see Subsection 5.3.2). Compute the set* P *using (5.6). If* $|P| \leq 1$ *then stop. Otherwise go to Step 3.*

*3: (*Finding piecewise linear boundaries*) Apply Algorithm 1 over sets* $B_i,\ i \in P$ *to find piecewise linear boundaries between classes (see Section 4.2).*

We call this algorithm the Hybrid Polyhedral Conic and Max-min Separability (HPCAMS) algorithm. This algorithm generates one PCF g_i for each class $i = 1, \ldots, q$. It also generates piecewise linear functions φ_i for classes $i \in P$ when $|P| > 1$. If $i \notin P$ then we set $\varphi_i(x) \equiv +\infty$. If $|P| \leq 1$ then we set $\varphi_i(x) \equiv +\infty$ for all $i = 1, \ldots, q$. The function Ψ_i separating the i-th class from the rest of the data set can be computed as follows:

$$\Psi_i(x) = \min \{g_i(x),\ \varphi_i(x)\},\ i = 1, \ldots, q. \tag{5.7}$$

5.4 Implementation of the algorithm

In this section we describe the conditions for the implementation of Algorithm 3. As mentioned earlier this algorithm consists of two stages. In the first stage we compute polyhedral conic functions approximating classes (Steps 1 and 2). There are three tolerances in this stage, in Step 1 $\eta > 0$ and in Step 2 $\theta > 0$ and $\sigma > 0$. We take $\eta = 0.1,\ \theta = 0.05$ and $\sigma = 0.01$.

In the second stage we apply Algorithm 1 to find piecewise linear boundaries (Step 3). This algorithm contains one tolerance $\varepsilon_0 \geq 0$. We choose $\varepsilon_0 = 0.01$.

The following conditions have been chosen for the implementation of Algorithm 2.

1. The values of tolerances $\varepsilon_1 > 0, \varepsilon_2 > 0$ and $\varepsilon_3 > 0$ are:

$$\varepsilon_1 = 0.005,\ \varepsilon_2 = f_1^*/100,\ \varepsilon_3 = 0.001,$$

   where f_1^* is the optimal value of the objective function for linear separation.

2. We restrict the number of hyperplanes to 10.

3. In Step 1 of Algorithm 2 we use the discrete gradient method of [7, 8] as modified in [11] to solve minimization problem (3.17).

The classification rule is as follows. For each class i Algorithm 3 generates the separating function Ψ_i defined in (5.7). If the new point v belongs to the set D_i then we classify it to the i-th class. If this point does not belong to any of the sets $D_i,\ i = 1, \ldots, q$ then we compute the values $\Psi_1(v), \ldots, \Psi_q(v)$ and classify this point to the class i associated with the minimum function value: $i = \text{argmin}\{\Psi_1(v), \ldots, \Psi_q(v)\}$.

We implemented the algorithm in Fortran 95 and compiled it using the Lahey Fortran compiler on a 1.83GHz Intel Pentium IV CPU with 1GB of RAM running Windows XP.

5.5 Numerical Experiments

We tested the HPCAMS algorithm on medium sized and large scale real world data sets readily available from the UCI machine learning repository ([6]). The selected data sets contain either continuous or integer attributes and have no missing values. Table 5.1 contains a brief description of the characteristics of the data sets. This table contains the number of data points in training and test sets. The class attribute is included in the the number of attributes in this table.

In our experiments we used some classifiers from WEKA for comparison. We chose the following classifiers: Naive Bayes (with kernel) (NB kernel), Logistic,

Table 5.1: Brief description of data sets

Data sets	(train,test)	No. of attributes	No. of classes
Shuttle control (SH)	(43500,14500)	10	7
Letter recognition (LET)	(15000,5000)	17	26
Landsat satellite image (LSI)	(4435,2000)	37	6
Pen-based recognition of handwritten digits (PD)	(7494,3498)	17	10
Page blocks (PB)	(4000,1473)	11	5
Optical recognition of handwritten digits (OD)	(3823, 1797)	65	10
Spambase (SB)	(3682,919)	58	2
Abalone (AB)	(3133,1044)	9	3
DNA	(2000, 1186)	180	3
Isolet (ISO)	(6238, 1559)	618	26
Phoneme_CR (PHON)	(4322, 1082)	6	2
Texture_CR (TEXT)	(4400, 1100)	41	11

Multi-Layer Perceptron (MLP), Linear LIBSVM (LIBSVM (LIN)), support vector machines classifier SMO with normalized polynomial kernel (SMO (NPOL)), SMO (PUK), a decision tree classifier J48 (which is an implementation of the C4.5 algorithm) and a rule based classifier PART.

We apply all algorithms from WEKA with the default parameter values. We put the following limits: 3 hours of CPU time (for training and testing) and 1GB of memory usage. In the tables a dash line shows that an algorithm exceeded one of these limits.

Results of numerical experiments are presented in Tables 5.2, 5.4 and 5.3. Table 5.2 contains test set accuracy on different data sets using different classifiers. One can see that in most of the data sets (except Optical recognition of handwritten digits, Phoneme_CR and Page blocks) the classification accuracy achieved over the test set by the HPCAMS algorithm is either the best or comparable with the best accuracy.

Table 5.4 presents pairwise comparison of the HPCAMS classifier with other classifiers using test set accuracy. The table contain the number of data sets and

Data set	AB	DNA	LSI	LET	OD	PD
Classifier						
NB(kernel)	57.85	93.34	82.10	74.12	90.32	84.13
Logistic	64.27	88.36	83.75	77.40	92.21	92.85
MLP	63.51	93.68	88.50	83.20	96.55	89.85
LIBSVM (LIN)	60.73	93.09	85.05	82.40	96.55	95.00
SMO (NPOL)	60.25	95.36	79.60	82.34	96.66	96.86
SMO (PUK)	64.18	57.93	91.45	-	96.61	97.88
J48	60.15	92.50	85.35	87.70	85.75	92.05
PART	57.95	91.06	85.25	87.32	89.54	93.65
HPCAMS	66.09	94.18	87.15	91.04	93.10	96.57

Table 5.2: Test set accuracy for different classifiers.

Data set	PHON	SH	TEXT	ISO	PB	SB
Classifier						
NB(kernel)	76.53	98.32	81.00	-	88.39	76.17
Logistic	74.58	96.83	99.64	-	91.72	92.06
MLP	81.52	99.75	99.91	-	92.80	92.06
LIBSVM (LIN)	77.54	-	99.18	96.02	87.03	90.97
SMO (NPOL)	78.74	96.81	97.27	-	89.48	92.60
SMO (PUK)	83.27	99.50	99.55	-	88.53	93.04
J48	85.67	99.95	93.91	83.45	93.55	92.93
PART	82.72	99.98	93.82	82.81	92.46	91.40
HPCAMS	80.13	99.86	99.36	93.52	89.55	93.47

Table 5.3: Test set accuracy for different classifiers.

their proportion where the HPCAMS algorithm achieved better testing accuracy. These results demonstrate that the HPCAMS algorithm performs well on test set in comparison with other classifiers.

Table 5.5 presents training and testing time for the HPCAMS algorithm. The use of polyhedral conic functions allow us to significantly reduce training time on all data sets under consideration. Moreover, the use of PCFs allows one to reduce training time by 3-10 times compared to the max-min separability algorithm from [9]. However, the HPCAMS algorithm requires a much longer training time than most of the other classifiers. The proposed algorithm is very fast in testing phase for all data sets. Results show that testing of the new algorithm is similar to that of Neural Network classifier MLP, Logistic classifier. Decision tree and rule-based classifiers use more testing time than the proposed algorithm. SVM algorithms

Classifier	No. of data sets	Proportion
NB(kernel)	12	100%
Logistic	10	83.33%
MLP	7	58.33%
LIBSVM(LIN)	10	83.33%
SMO (NPOL)	9	75.00%
SMO (PUK)	7	58.33%
J48	9	75.00%
PART	8	75.00%

Table 5.4: Pairwise comparison of the HPCAMS classifier with others using testing accuracy.

Dataset	Training time	Testing time
AB	37.03	0.00
DNA	42.27	0.03
LSI	451.67	0.03
LET	7389.73	0.16
OD	106.31	0.05
PD	158.91	0.03
PHON	39.13	0.00
SH	731.70	0.03
TEXT	55.33	0.02
ISO	2994.64	1.86
PB	89.55	0.02
SB	240.20	0.02

Table 5.5: Training and testing time for the the HPCAMS algorithm (in seconds).

use 1-2 order more testing time than the HPCAMS algorithm.

It should be noted that in order to implement the HPCAMS classifier it is sufficient to save in memory one polyhedral conic and one piecewise linear functions for each class. Therefore the memory usage of the HPCAMS classifier is very low.

5.6 Conclusion

In this chapter we developed another new algorithm for the computation of piecewise linear boundaries between pattern classes. This algorithm consists of two main stages.

In the first stage we compute one polyhedral conic function for each class in order to identify data points which lie on or close to the boundaries between classes. In the second stage we apply the max-min separability algorithm to find piecewise linear boundaries using only those data points.

Such an approach allows us to reduce the training time of the max-min separability algorithm on large data sets by 3-10 times. This new algorithm requires almost instantaneous testing time and has a low memory usage.

The results of the numerical experiments demonstrate that the proposed algorithm consistently produces a good test set accuracy on most data sets when comparing with a number of other mainstream classifiers. However, the proposed algorithm can require more training time than most of the other classifiers.

Chapter 6

An incremental piecewise linear classifier based on hyperboxes and max-min separation

6.1 Introduction

The previous piecewise linear classification technique proposed in chapter 5 uses polyhedral conic functions to initially eliminate points. In this chapter we consider hyperboxes and a similar incremental algorithm presented in chapter 4. As with the classifier presented in chapter 5 a scheme to initially remove points that do not contribute to the decision boundary is presented. However, in this case the first stage of point elimination is done using hyperboxes.

This new algorithm, consists of two main stages. In the first stage we approximate each class using one hyperbox. Then we divide all data points into two categories:

1. Data points which belong **"obviously"** to their respective classes;

2. Data points which are in the **"indeterminate"** regions [59].

The intersections of the hyperboxes define the "indeterminate" regions. Data points from the first category are points which are not in these intersections. They are easily classified using the hyperboxes. We use piecewise linear functions to separate points from different classes in the "indeterminate" regions. It reduces computational effort and also avoids possible overfitting. Since hyperboxes can be considered as a special case of continuous piecewise linear functions, such an approach allows us to find a piecewise linear boundary of classes.

The second stage is the incremental construction of the class boundaries. The piecewise linear functions used to separate classes in the indeterminate region are built incrementally, that is we start with one hyperplane and keep adding new hyperplanes while the separation accuracy is improving sufficiently. Such an approach allows us to find good starting points for minimization of nonconvex error functions. It computes as many hyperplanes as necessary for separating classes and avoids overfitting.

Another advantage of the incremental approach is that at each iteration of the process we refine the indeterminate region so as to remove the points which are furthest from the piecewise linear boundary. The complexity of the error function is thus further reduced.

In this chapter we present the first stage of the algorithm, of which explains the operation of hyperboxes. Next we present the incremental algorithm and then finally we present the classification rules.

We report results of numerical experiments on 10 publicly available data sets. We also provide the comparison of the new piecewise linear classifier with 9 other classifiers from the WEKA suite. Finally we present a conclusion of our results.

6.2 Piecewise linear separability

The approach proposed in this chapter finds piecewise linear boundaries of classes. Firstly we approximate the classes using hyperboxes, which allow us to identify the "indeterminate" regions where points from several classes are intertwined. We then focus on these regions to find more precise boundaries between the classes, using continuous piecewise linear functions to do so. These boundaries are determined using max-min separability, a concept which was introduced in [9] (see also [11]). For the proofs of the propositions we refer to the paper [9].

6.3 Identification of indeterminate regions using hyperboxes

In this section we will describe a scheme to select points belonging to indeterminate regions. Firstly, we approximate each class by one hyperbox, then we define the indeterminate regions as the intersections of these hyperboxes and finally we select data points belonging to these regions for further separation by piecewise linear functions. All other points belong to only one hyperbox and therefore are classified simply using hyperboxes (see Figs. 6.1 and 6.2).

Figure 6.1: Identification of indeterminate region between two sets A_1 and A_2 using hyperboxes: grey region

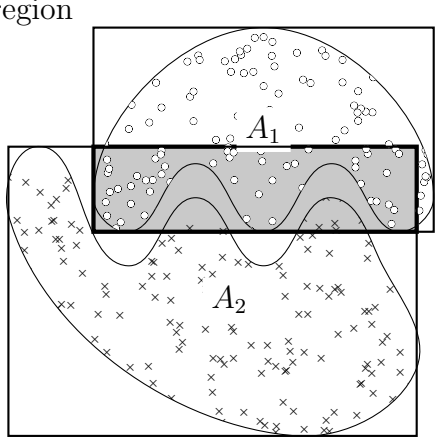
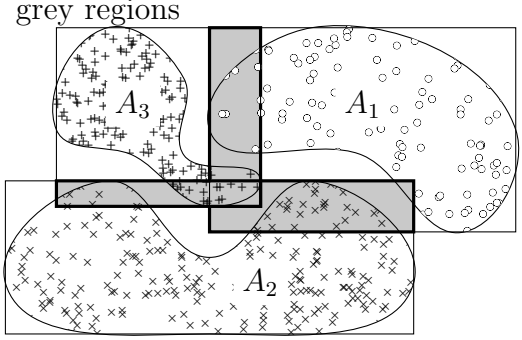


Figure 6.2: Identification of indeterminate regions between three sets A_1, A_2 and A_3 using hyperboxes: grey regions



Assume that we are given data set A with $q \geq 2$ classes $A_1, \ldots, A_q$ and the i-th class contains m_i data points. Then for each class we compute:

$$\bar{\alpha}_j^i = \min_{a \in A_i} a_j, \quad \bar{\beta}_j^i = \max_{a \in A_i} a_j, \ j = 1, \ldots, n, \ i = 1, \ldots, q.$$

Then we define vectors $\alpha^i = (\alpha_1^i, \ldots, \alpha_n^i)$, $\beta^i = (\beta_1^i, \ldots, \beta_n^i)$, $i = 1, \ldots, q$ as follows:

$$\alpha_j^i = \bar{\alpha}_j^i - \gamma(\bar{\beta}_j^i - \bar{\alpha}_j^i),$$

$$\beta_j^i = \bar{\beta}_j^i + \gamma(\bar{\beta}_j^i - \bar{\alpha}_j^i),$$

where $\gamma \geq 0$ is a sufficiently small number. These vectors define the following hyperboxes in n-dimensional space $\mathbb{R}^n$:

$$H(A_i) = [\alpha^i, \beta^i] \ \equiv \ \left\{ x \in \mathbb{R}^n : \ \alpha_j^i \leq x_j \leq \beta_j^i, \ j = 1, \ldots, n \right\}, \ i = 1, \ldots, q.$$

All points from the i-th class belong to the hyperbox $H(A_i)$.

In order to obtain the indeterminate region between the class A_i and the rest of the data set we need to compute the intersection between the hyperbox $H(A_i)$ and other hyperboxes $H(A_j)$, $j = 1, \ldots, q$, $j \neq i$. This is done by computing pairwise intersections between hyperbox $H(A_i)$ and other hyperboxes.

The intersection between two hyperboxes is also a hyperbox, which is obtained

using the following:

$$H(A_i, A_j) = [c^{ij}, d^{ij}]$$

where

$$c_k^{ij} = \max\{\alpha_k^i, \alpha_k^j\}, \quad d_k^{ij} = \min\{\beta_k^i, \beta_k^j\}, \ k = 1, \ldots, n.$$

If $d_k^{ij} < c_k^{ij}$ for at least one $k \in \{1, \ldots, n\}$ then the hyperboxes $H(A_i)$ and $H(A_j)$ do not intersect and the hyperbox $H(A_i, A_j)$ is empty. The indeterminate region D_i between the i-th class and the rest of the data set is defined as:

$$D_i = \bigcup_{j=1, j\neq i}^{q} H(A_i, A_j). \tag{6.1}$$

Then the whole indeterminate region D is

$$D = \bigcup_{i=1}^{q} D_i.$$

The set of obviously classified points in class A_i is:

$$O_i = A_i \setminus D_i.$$

These points can be simply classified using hyperboxes, and therefore are removed from the data set before the next stage where we calculate a piecewise linear boundary between the classes.

6.4 Incremental algorithm

In this section we describe the second stage of our method. It is an incremental algorithm which differs from algorithm 1 by the way the incremental algorithm is constructed. In this algorithm a different indeterminate region is calculated for each class.

We will now present an algorithm for separating class $A_u, \ u \in \{1, \ldots, q\}$ from

the rest of the data set. We assume that the indeterminate region D_u has been obtained using formula (6.1). One of the following cases will occur:

1. The set D_u is empty or only contains points from class u. In this case we use the hyperbox $H(A_u)$ to separate the class u from the rest of the data set.

2. The set D_u does not contain any points from class u and contains points from the rest of the data set. In this case we use the obvious region O_u to separate the class u from the rest of the data set.

3. The set D_u contains points from both class u and the rest of the data set. In this case we compute a piecewise linear boundary between the class u and the rest of the data set within D_u.

We denote $\overline{A_u} = A_u \cap D_u$ and $\underline{A_u} = D_u \cap (\cup_{t=1,t\neq u}^{q} A_t)$. In Case 4 $\overline{A_u} \neq \emptyset$, $\underline{A_u} \neq \emptyset$. Our aim is to find a piecewise linear function separating the sets $\overline{A_u}$ and $\underline{A_u}$.

Let $\varepsilon_1 > 0, \varepsilon_2 > 0, \varepsilon_3 > 0$ be given tolerances and $\delta > 0$ be a sufficiently small number.

Algorithm 4. *An incremental algorithm*

*1: (*Initialization*) Set*

$$D_u^1 = D_u,\ \overline{A_u}^1 = \overline{A_u},\ \underline{A_u}^1 = \underline{A_u}.$$

Select any starting point

$$(X^1, Y_1) = (x^1, y_1), x^1 \in \mathbb{R}^n, y_1 \in \mathbb{R}^1.$$

Set

$$I_1 = \{1\}, J_1^1 = \{1\}, f_1 = f(x^1, y_1),$$

$$r_1 = |I| = 1, s_1 = |J_1^1| = 1,$$

the number of hyperplanes $l = 1$ *and iteration counter* $k = 1$.

2: *(*Computation of piecewise linear function*) Solve problem (3.17) over the set* D_u^k *starting from the point* $(X^k, Y_k) \in \mathbb{R}^{(n+1)l}$ *. Let* (X^{k*}, Y_{k*}) *be the solution to this problem,* f_k^* *be the corresponding objective function value, and* $f_{1,k}^*$ *and* $f_{2,k}^*$ *be values of functions* f_1 *and* f_2*, respectively. Let* E_k *be the error rate at iteration* k *over the set* D*, that is*

$$E_k = \frac{|\{a \in \overline{A_u} : \varphi^k(a) > 0\} \cup \{b \in \underline{A_u} : \varphi^k(b) < 0\}|}{|D|},$$

where

$$\varphi^k(a) = \max_{i \in I_k} \min_{j \in J_i^k} \left(\langle x^{ij*}, a \rangle - y_{ij*} \right).$$

3: *(*The first stopping criterion*) If* $\max\{f_{1,k}^*, f_{2,k}^*\} \leq \varepsilon_1$ *then stop.* (X^{k*}, Y_{k*}) *is the final solution.*

4: *(*The second stopping criterion*) If* $k \geq 2$ *and* $f_{k-1}^* - f_k^* \leq \varepsilon_2$ *then stop.* $(X^{k-1,*}, Y_{k-1,*})$ *is the final solution.*

5: *(*The third stopping criterion*) If* $E_k < \varepsilon_3$ *then stop.* (X^{k*}, Y_{k*}) *is the final solution.*

6: *(*Refinement of indeterminate regions*) Compute*

$$f_{k,max} = \max_{a \in \overline{A_u}^k} \max_{i \in I_k} \min_{j \in J_i^k} \left(\langle x^{ij*}, a \rangle - y_{ij*} \right),$$

$$f_{k,min} = \min_{a \in \underline{A_u}^k} \max_{i \in I_k} \min_{j \in J_i^k} \left(\langle x^{ij*}, a \rangle - y_{ij*} \right)$$

and the following sets:

$$C_1 = \left\{ a \in \overline{A_u}^k : \max_{i \in I_k} \min_{j \in J_i^k} \langle x^{ij*}, a \rangle - y_{ij*} \leq (1 + \delta) f_{k,min} \right\},$$

$$C_2 = \left\{ a \in \underline{A_u}^k : \max_{i \in I_k} \min_{j \in J_i^k} \langle x^{ij*}, a \rangle - y_{ij*} \geq (1+\delta) f_{k,max} \right\}$$

Refine the indeterminate region as follows:

$$D_u^{k+1} = D_u^k \setminus \{C_1 \bigcup C_2\},$$

$$\overline{A_u}^{k+1} = A_u \cap D_u^{k+1}, \quad \underline{A_u}^{k+1} = D_u^{k+1} \cap (\cup_{t=1, t \neq u}^{q} A_t).$$

*7: (*Adding new hyperplanes*)*

1. *If* $f_{1,k}^* > \varepsilon_1$ *then set*

$$s_{k+1} = s_k + 1, J_i^{k+1} = J_i^k \cup \{s_{k+1}\}$$

for all $i \in I_k$. *Set*

$$x^{ij} = x^{i,j-1,*}, y_{ij} = y_{i,j-1,*}, i \in I_k, j = s_{k+1}.$$

2. *If* $f_{2,k}^* > \varepsilon_1$ *then set*

$$r_{k+1} = r_k + 1, I_{k+1} = I_k \cup \{r_{k+1}\}, J_{r_{k+1}}^{k+1} = J_{r_k}.$$

Set

$$x^{ij} = x^{i-1,j,*}, y_{ij} = y_{i-1,j,*}, i = r_{k+1}, j \in J_{r_k}^k.$$

*8: (*New starting point*) Set*

$$X^{k+1} = (X^{k*}, x_{ij}, \ i \in I_{k+1}, \ j \in J_{k+1}^i),$$

$$Y_{k+1} = (Y_{k*}, y_{ij}, \ i \in I_{k+1}, \ j \in J_{k+1}^i),$$

$$l = \sum_{i \in I_{k+1}} |J_i^{k+1}|, k = k + 1$$

and go to Step 2.

The following explains Algorithm 4 in more details. In Step 1 we set the initial indeterminate region as in (6.1), and select one hyperplane and starting point for the computation of that hyperplane. In Step 2 we compute a piecewise linear function with a preselected number of hyperplanes using the previous solution to construct the starting point and to also compute the error rate over the indeterminate region (see Fig 6.3). The algorithm contains three stopping criteria which are given in Steps 3, 4 and 5.

- The algorithm terminates if the values of the error function over the given class and the rest of the data set is less than a given tolerance $\varepsilon_1 > 0$ and the last piecewise linear function is accepted as a boundary between classes (Step 3).

- If $k \geq 2$ and the difference between values of the error function in two successive iterations is less than a given tolerance $\varepsilon_2 > 0$ then the algorithm terminates and the piecewise linear function from the previous iteration is accepted as the boundary between the classes (Step 4).

- Finally, if the error rate is less than some threshold $\varepsilon_3 > 0$ then again the algorithm terminates and the piecewise linear function from the last iteration is accepted as the boundary between classes (Step 5).

If none of these stopping criteria is met, then in Step 6 we refine the indeterminate region by removing the data points which are far from the piecewise linear boundary. In Step 7, depending on the values of the error function on both sets, we add new hyperplanes. Finally in Step 8 we update the starting point and the number of hyperplanes.

In summary, the classification algorithm proposed in this chapter consists of two main stages. In the first stage the data set is divided into two regions using hyperboxes. The first region consists of the points that can be easily classified using the hyperboxes and therefore can be excluded from further investigation. The remaining points belong to the indeterminate region. In the second stage of our algorithm we incrementally construct a piecewise linear function to separate the classes within this indeterminate region using Algorithm 4. Since hyperboxes can be represented as piecewise linear functions we then get piecewise linear boundaries of all the classes. Both the processes of computing the hyperboxes and of eliminating the obviously classified points can be implemented linearly with the number of instances in the data set.

6.5 Classification rules

To compute piecewise linear boundaries between classes we use the one vs. all strategy, that is, for each class i we consider this class as one class and the rest of the data set as a second class. Then we apply the proposed algorithm to separate the i-th class from the rest of the data set in the indeterminate region. This means that for each class we get one piecewise linear function taking into account that hyperboxes can also be considered as piecewise linear functions.

The following classification rule is applied to classify new data points (observations). A new point can belong to one of three distinct sets:

1. This point belongs to one of the sets $O_i,\ i = 1, \ldots, q$.

2. This point belongs to the set

$$Q = \mathbb{R}^n \setminus \left(\bigcup_{i=1}^{q} H(A_i) \right).$$

3. This point belongs to the indeterminate region

$$D = \bigcup_{i=1}^{q} D_i.$$

If the new point belongs to the set O_i then we classify it to the i-th class. If this point belongs to the set Q we classify it to the class associated with the closest hyperbox.

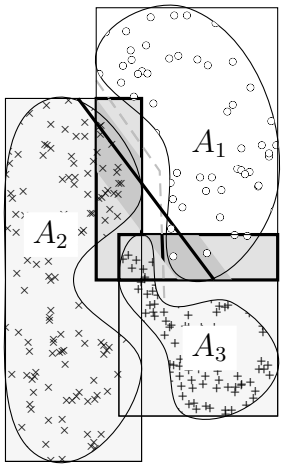
If the new point belongs to the indeterminate region we use the following classification rule. First we find all hyperboxes $H(A_{i_1}), \ldots, H(A_{i_t}),\ 0 < t \leq q$ containing this point. At each iteration $k,\ k \geq 1$ of the incremental algorithm we refine the region D_u^k and associate a piecewise linear function φ^k to it. When the algorithm terminates at iteration $k_{i,max}$ another piecewise linear function $\varphi_i^{k_{i,max}}$ is found. If the new point belongs to a set $D_u^k \setminus D_u^{k+1}$ then it is associated with function φ_i^k. Otherwise it is associated with function $\varphi_i^{k_{i,max}}$. Therefore a new point v is associated with one piecewise linear function per class, that is with a set of function values $\{\varphi_1(v), \ldots, \varphi_t(v)\}$. We classify this point to the class associated with the minimum value among these function values: $i = \text{argmin}\{\varphi_1(v), \ldots, \varphi_t(v)\}$.

Figure 6.3 shows this classification rule in the case of three-set separation: the obviously separated areas are unshaded. The light shaded area represents the points separated using linear separation (obtained at the first iteration of Algorithm 4), and the dark shaded area represents the points separated using the final piecewise linear separating function returned by Algorithm 4.

6.6 Implementation of the algorithm

In this section we describe conditions for the implementation of the classification algorithm. As we mentioned above this algorithm consists of two main stages. In the first stage we calculate hyperboxes containing classes. Here we choose

Figure 6.3: Classification rule to separate sets A_1, A_2 and A_3 using hyperboxes and max-min separability



$\gamma = 0.05$. The following conditions have been chosen for the implementation of Algorithm 4.

1. The values of tolerances $\varepsilon_1 > 0, \varepsilon_2 > 0$ and $\varepsilon_3 > 0$ are:

$$\varepsilon_1 = 0.005,\ \varepsilon_2 = f_1^*/100,\ \varepsilon_3 = 0.001,$$

   where f_1^* is the optimal value of the objective function for linear separation.

2. The value of $\delta > 0$ is: $\delta = 0.25$.

3. In the proposed algorithm we restrict the number of hyperplanes to 10.

4. Since for linear separation Problem (3.17) is convex, any initial point leads us to the global minimum. In Step 1 we select $x^1 = 0, y_1 = 0$.

5. In Step 2 of Algorithm 4 we solve minimization problem (3.17). We use the discrete gradient method of [7, 8] as modified in [11].

6. In all data sets we apply our algorithm on a training set and test our classification rules on a test set. In our experiments we use data set for which the division into training and test sets is given.

We implemented the algorithm in Fortran 95 and compiled it using the Lahey Fortran compiler on an Intel Pentium IV 1.83GHz CPU with 1GB of RAM running Windows XP.

6.7 Numerical Experiments

We tested the proposed algorithm on medium sized and large scale real world data sets readily available from the UCI machine learning repository [6]. The data sets were selected as follows: they have either continuous or integer attributes and no missing values. Table 6.1 contains a brief description of the characteristics of the data sets. The number of attributes in these tables also includes the class attribute.

Data sets	(train,test)	No. of attributes	No. of classes
Shuttle control (SH)	(43500,14500)	10	7
Letter recognition (LET)	(15000,5000)	17	26
Landsat satellite image (LSI)	(4435,2000)	37	6
Pen-based recognition of handwritten digits (PD)	(7494,3498)	17	10
Abalone (AB)	(3133,1044)	9	3
Spambase (SB)	(3682,919)	58	2
Isolet (ISO)	(6238, 1559)	618	26
Covertype(CT)	(15120, 565892)	55	7
Magic telescope (MT)	(15025, 3995)	11	2
Texture_CR (TEXT)	(4400, 1100)	41	11

Table 6.1: Brief description of data sets

In our experiments we used a number of classifiers from WEKA for comparison.

In order to test our algorithm we used the following classifiers. We chose representatives of each type of classifier from WEKA: Naive Bayes (with kernel), Instance-based algorithm IBk (with $k = 5$), Logistic Regression based classifier Logistic, Multi-Layer Perceptron (MLP), support vector machine classifiers

Linear LibSVM (LIBSVM (LIN)) and SMO (PUK), decision tree classifier J48 (which is an implementation of the C4.5 algorithm) and the rule based classifier PART. The classifiers chosen produced an overall better accuracy than others from WEKA. In addition we tested a classifier based on polyhedral separability [5]. Since the number of hyperplanes in polyhedral separability is not known a priori, we tested this algorithm with 2 to 5 hyperplanes and report only the best results on test sets.

We apply all algorithms with the default values of parameters from WEKA. We set limitations of 3 hours for training and testing CPU time and memory usage of 128MB of RAM. A dash line in the tables shows that the corresponding algorithm exceeded one of these limits.

Table 6.2 contains test set accuracy on the data sets for all classifiers. Since Covertype is too large for WEKA given the restrictions on the memory we do not include results on this data set. Test set accuracy for the proposed algorithm on the Covertype data set is 74.30.

Data set	SH	LET	LSI	PD	AB	SB	ISO	MT	TEXT
Algorithm									
NB(kernel)	98.32	74.12	82.10	84.13	57.85	76.17	-	76.02	81.00
IBk(k=5)	99.88	94.96	89.85	97.60	60.35	90.10	91.08	84.33	98.82
Logistic	96.83	77.40	83.75	92.85	64.27	92.06	-	79.25	99.64
MLP	99.75	83.20	88.50	89.85	63.51	92.06	-	86.41	**99.91**
LIBSVM (LIN)	-	82.40	85.05	95.00	60.73	90.97	**96.02**	79.35	99.18
SMO (PUK)	99.50	-	**91.45**	**97.88**	64.18	93.04	-	**86.93**	99.55
J48	99.95	87.70	85.35	92.05	60.15	92.93	83.45	85.48	93.91
PART	**99.98**	87.32	85.25	93.65	57.95	91.40	82.81	85.08	93.82
Polyhedral	99.29	88.68	87.00	97.03	65.23	92.71	-	84.93	**99.91**
Max-min	99.92	**91.82**	89.30	96.43	**66.38**	**95.65**	93.01	86.36	98.27

Table 6.2: Test set accuracy for different classifiers.

Table 6.3 presents pairwise comparisons of the proposed classifier with other classifiers using test set accuracy. We do not count the Covertype data set. The table contains the proportion of the data sets where the proposed algorithm achieved better or similar testing accuracy. These results demonstrate that the

Classifier	Proportion
NB(kernel)	100%
IBk (k=5)	55.56%
Logistic	88.89%
MLP	88.89%
LIBSVM(LIN)	77.78%
SMO (PUK)	55.56%
J48	88.89%
PART	88.89%
Polyhedral	77.78%

Table 6.3: Pairwise comparison of the proposed classifier with others using testing accuracy

Data set	Training Time	Testing time
SH	76.84	0.03
LET	6124.38	0.19
LSI	739.83	0.03
PD	168.22	0.03
AB	108.73	0.00
SB	359.53	0.00
ISO	93.61	1.78
CT	957.73	5.44
MT	352.38	0.02
TEXT	26.41	0.02

Table 6.4: Training and testing time for the proposed algorithm (in seconds)

proposed algorithm performs well in comparison with other classifiers.

Table 6.4 presents training and testing time for the proposed algorithm. The use of hyperboxes allow us to significantly reduce training time on data sets such as Shuttle control, Pen-based recognition of handwritten digits, Abalone, Isolet, Covertype and Texture_CR. The proposed algorithm is very fast in classifying new observations. Results show that testing time of the proposed algorithm is similar to that of the Neural Network classifier MLP and the Logistic classifier. Decision tree and rule-based classifiers use more testing time than the proposed algorithm. SVM and lazy classifiers use significantly more testing time than the proposed algorithm.

It should be noted that the proposed piecewise linear classifier needs only to store one hyperbox and one piecewise linear function per class in memory.

Therefore, the memory usage of this classifier is very low. Results presented in this section also demonstrate that the proposed classifier achieves consistently good classification accuracy over the test set and its training time is reasonable. All these make the proposed classifier suitable for the application areas listed in the Introduction of this chapter.

6.8 Conclusion

In this chapter we developed another new algorithm for the computation of piecewise linear boundaries of finite point sets. This algorithm consists of two main stages. In the first stage we compute a hyperbox that approximates each class. These hyperboxes allow us to identify the so-called indeterminate regions, where data points from different classes are mixed. Data points which belong to only one hyperbox are called "obviously" classified points and are removed from further examination. Thus, in most cases the use of hyperboxes allows us to significantly reduce the number of data points and consequently computational effort.

In the indeterminate region we use more complex piecewise linear functions than hyperboxes to find boundaries between classes. Since the number of linear functions or hyperplanes necessary for separating classes is not known a priori we suggest to use an incremental approach to find such boundaries. This approach allows us: (i) to refine the indeterminate region and therefore to further reduce the number of data points in this region; (ii) to compute as many hyperplanes as necessary to separate classes with respect to a given tolerance; (iii) to find good starting points for global minimization of the error functions; (iv) to formulate a better classification rule for new data points.

We tested the proposed algorithm on medium to large scale real world data sets. Computational results demonstrate that the new classifier achieves good classification accuracy while requiring reasonable training time. This classifier

can be used for real-time classification.

Chapter 7

Conclusion and further work

7.1 Conclusion

Classifiers using Piecewise linear decision boundaries offer many advantages over current main stream techniques as real time classifiers. These include classification in real time, their simplicity, not needing parameters and their modest memory and processing requirements due to classification based on a piecewise linear function. However, they are not immune to the complexities in dealing with large volumes of data.

In this thesis three different real time piecewise linear classifiers were developed. Our main approach in building the piecewise linear boundaries is based on the concept of max-min separability. Unlike other existing approaches, our approach allows one to simultaneously compute boundaries while removing data points that do not contribute to the construction of this boundary. The problem is formulated as a non-smooth optimization problem which can be solved by efficient algorithms such as bundle methods, the discrete gradient method etc. We proposed these three approaches to reduce the computational effort for the training of the piecewise linear classifiers.

All three approaches aim to identify data points close to, or on the decision

boundary. The first technique used an incremental algorithm to compute piecewise linear boundaries. At each iteration a new piecewise linear boundary is constructed for each class, using a starting point constructed from the boundaries obtained at previous iterations. Each new boundary is used to eliminate easily separated points from further calculations. The points not easily separated, called "undetermined" points, are identified for separation at further iterations. This piecewise linear boundary separates the sets with as few hyperplanes as needed, with respect to a given tolerance.

The second technique uses one polyhedral conic function for each class to identify data points which lie on or close to the boundaries between classes. This again allows for all other points to be eliminated from further calculations. On large data sets, this approach can have a reduction in training time by 3-10 times.

The final technique uses hyperboxes to identify the so-called indeterminate regions (undetermined points), where data points from different classes are mixed. This simultaneously identifies data points belonging to only one hyperbox, which are the "obviously" classified points that can be removed from further examination.

The classifiers, in most cases, had a significant reduction in the number of data points and as a consequence computational effort was reduced. Furthermore, this also allowed for:

- computation of only as many hyperplanes as necessary to separate classes;
- ability to find good starting points for global minimization of the error functions;
- formulation of a better classification rule for new data points.

We tested the proposed algorithms on medium to large scale real world data sets. Computational results demonstrate that the new classifiers achieve good

classification accuracy while requiring reasonable training time. The results further show that a single optimization solution using a piecewise linear function allows for real time classification, i.e. fractions of a second, and that the final function requires very little memory space for storage.

Therefore, the above features make the classifiers ideally suited for real time embedded system applications that need to respond to incoming stimuli from an external environment. Furthermore, as the classifiers have low memory and processing requirements, they are also ideally suited to embedded systems applications with size, power, processing and memory limitations. With many examples in a growing industry, that include small reconnaissance robots, autonomous mobile robots, intelligent cameras, monitoring systems etc., current and further research of real time classification techniques are of great importance.

7.2 Further research

Our results on piecewise linear classifiers presented in this thesis clearly demonstrate that they are efficient classifiers for solving data classification problems. Further research in this area may include:

- the development of more accurate algorithms that could utilize techniques such as Condensed Nearest Neighbour (CNN), for the identification of points close to, or on the boundaries between classes;
- the extension of these classifiers for very large data sets, including dynamic and streaming data. Application of piecewise linear classifiers in this area is especially important because of their low memory requirements and very low testing time;
- the use of the notion of a margin, similar to the concept found in SVM like algorithms, that improve the efficiency of the classifier and help to avoid the problem of overfitting the data.

Bibliography

[1] C. C. Aggarwal, A. Hinneburg, and D. A. Keim. On the surprising behavior of distance metrics in high dimensional space. *Lecture Notes in Computer Science*, 1973:420–434, 2001.

[2] E. L. Allwein, R. E. Schapire, and Y. Singer. Reducing multiclass to binary: A unifying approach for margin classifiers. In *Proceedings of the 17th International Conference on Machine Learning (ICML-2000)*, pages 9–16. Morgan Kaufmann, 2000.

[3] R. Anand, K. Mehrotra, C. Mohan, and S. Ranka. Efficient classification for multiclass problems using modular neural networks. *Neural Networks, IEEE Transactions on*, 6(1):117 –124, jan 1995.

[4] J. A. Anderson and E. Rosenfield. Talking nets: An oral history of neural networks. *IEEE Transactions on Neural Networks*, 9(5):1054–1054, 1998.

[5] A. Astorino and M. Gaudioso. Polyhedral separability through successive lp. *Journal of Optimization Theory and Applications*, 112(2):265–293, 2002.

[6] A. Asuncion and D. Newman. UCI machine learning repository, 2007.

[7] A. Bagirov. Minimization methods for one class of nonsmooth functions and calculation of semi-equilibrium prices. *Progress in Optimization:Contribution from Australasia*, pages 147–175, 1999.

[8] A. Bagirov. A method for minimization of quasidifferentiable functions. *Optimization Methods and Software*, 17(1):31–60, 2002.

[9] A. Bagirov. Max-min separability. *Optimization Methods and Software*, 20(2-3):271–290, 2005.

[10] A. Bagirov. Modified global k-means algorithm for minimum sum-of-squares clustering problems. *Pattern Recognition*, 41:3192–3199, 2008.

[11] A. Bagirov and J. Ugon. Supervised data classification via max-min separability. In Jeyakumar and Rubinov, editors, *Continuous Optimisation: current trends and modern applications*, chapter 6, pages 175–208. Springer, Berlin, 2005.

[12] Y. Bao and Z. Liu. A fast grid search method in support vector regression forecasting time series. In *IDEAL*, pages 504–511, 2006.

[13] G. Batista and M. Monard. An analysis of four missing data treatment methods for supervised learning. *Applied Artificial Intelligence*, 17:519–533, 2003.

[14] O. Bennet, K.P.and Mangasarian. Robust linear programming discrimination of two linearly inseparable sets. *Optimization Methods and Software*, 1:23–34, 1992.

[15] K. Bennett and O. Mangasarian. Bilinear separation of two sets in n-space. *Computational Optimization and Applications*, 2:207–227, 1993.

[16] J.-M. Berge. *High-Level System Modeling: Specification Languages*. Kluwer Academic Publishers, Norwell, MA, USA, 1995.

[17] J. Bloch. *Effective Java programming language guide*. Sun Microsystems, Inc., Mountain View, CA, USA, 2001.

[18] L. Bobrowski. Design of piecewise linear classifiers from formal neurons by a basis exchange technique. *Pattern Recognition*, 24(9):863–870, 1991.

[19] A. Bose, X. Hu, K. G. Shin, and T. Park. Behavioral detection of malware on mobile handsets. In *MobiSys '08: Proceeding of the 6th international conference on Mobile systems, applications, and services*, pages 225–238, New York, NY, USA, 2008. ACM.

[20] R. R. Bouckaert, E. Frank, M. A. Hall, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten. WEKA–experiences with a java open-source project. *Journal of Machine Learning Research*, 11:2533–2541, 2010.

[21] A. P. Bradley. The use of the area under the roc curve in the evaluation of machine learning algorithms. *Pattern Recognition*, 30(7):1145 – 1159, 1997.

[22] L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone. *Classification and Regression Trees*. Chapman & Hall, New York, NY, 1984.

[23] G. Brightwell, C. Kenyon, and H. Paugam-Moisy. Multilayer neural networks: One or two hidden layers? In *Advances in Neural Information Processing Systems 9, Proc. NIPS*96*, pages 148–154. MIT Press, 1996.

[24] C. E. Brodley and P. E. Utgoff. Multivariate decision trees. *Mach. Learn.*, 19(1):45–77, 1995.

[25] B. Brumen, I. Golob, H. Jaakkola, T. Welzer, and I. Rozman. Early assessment of classification performance. In *ACSW Frontiers '04: Proceedings of the second workshop on Australasian information security, Data Mining and Web Intelligence, and Software Internationalisation*, pages 91–96, Darlinghurst, Australia, Australia, 2004. Australian Computer Society, Inc.

[26] C. J. C. Burges. A tutorial on support vector machines for pattern recognition. *Data Mining and Knowledge Discovery*, 2(2):121–167, 1998.

[27] A. Burns and A. J. Wellings. *Real-time systems and their programming languages*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1990.

[28] L. S. Camargo and T. Yoneyama. Specification of training sets and the number of hidden neurons for multilayer perceptrons. *Neural Comput.*, 13(12):2673–2680, 2001.

[29] L. S. Camargo and T. Yoneyama. Specification of training sets and the number of hidden neurons for multilayer perceptrons. *Neural Comput.*, 13(12):2673–2680, 2001.

[30] R. Caruana and A. Niculescu-Mizil. Data mining in metric space: an empirical analysis of supervised learning performance criteria. In *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '04, pages 69–78, New York, NY, USA, 2004. ACM.

[31] R. Caruana and A. Niculescu-Mizil. An empirical comparison of supervised learning algorithms. In *Proceedings of the 23rd international conference on Machine learning*, ICML '06, pages 161–168, New York, NY, USA, 2006. ACM.

[32] B. Chai, T. Huang, X. Zhuang, Y. Zhao, and S. J. Piecewise linear classifiers using binary tree structure and genetic algorithm. *Pattern Recognition*, 29(11):1905–1917, 1996.

[33] C.-C. Chang and C.-J. Lin. Training ν-support vector classifiers: Theory and algorithms. *Neural Computation*, 13(9):2119–2147, 2001.

[34] P.-H. Chen, C.-J. Lin, and B. Schlkopf. A tutorial on ν-support vector machines. *Applied Stochastic Models in Business and Industry*, 21(2):111–136, 2005.

[35] S. Chen, C. F. N. Cowan, and P. M. Grant. Orthogonal least squares learning algorithm for radial basis function networks. *IEEE Transactions on Neural Networks*, 2(2):302–309, 1991.

[36] J. Cheng and R. Greiner. Learning Bayesian belief network classifiers: Algorithms and system. *Lecture Notes in Computer Science*, 2056:141–148, 2001.

[37] C. Chou, B. Kuo, and F. Chang. The generalized condensed nearest neighbor rule as a data reduction method. In *Proceedings of the International Conference of Pattern Recognition*, pages II: 556–559, 2006.

[38] W. W. Cohen. Fast effective rule induction. In *In Proceedings of the Twelfth International Conference on Machine Learning*, pages 115–123. Morgan Kaufmann, 1995.

[39] M. Cosnard, P. Koiran, and H. Paugam-Moisy. A step towards the frontier between one-hidden-layer and two-hidden-layer neural networks. In *Neural Networks, 1993. IJCNN '93-Nagoya. Proceedings of 1993 International Joint Conference on*, volume 3, pages 2292–2295, Oct 1993.

[40] T. Cover and P. Hart. Nearest neighbor pattern classification. *Information Theory, IEEE Transactions on*, 13(1):21 – 27, Jan 1967.

[41] G. B. Dantzig. *Linear programming and extensions.* Princeton, N.J., Princeton University Press, 1963.

[42] A. Dempster, N. Laird, and D. Rubin. Maximum likelihood from incomplete data via the em algorithm. *J. Royal Statistical Society, Series B*, 39(1):1–38, 1977.

[43] J. Demšar. Statistical comparisons of classifiers over multiple data sets. *J. Mach. Learn. Res.*, 7:1–30, 2006.

[44] R. Desai, J. Buckey, and J. Pearce. Timing specifications and accuracy of the real-time 3d echocardiographic reconstruction system. In *Image Processing, 1994. Proceedings. ICIP-94., IEEE International Conference*, volume 1, pages 895–899, nov. 1994.

[45] L. Devroye, L. Gyorfi, and G. Lugosi. *Probabilistic Theory of Pattern Recognition.* Springer-Verlag, 1996.

[46] R. O. Duda, P. E. Hart, and D. G. Stork. *Pattern Classification.* Wiley, New York, 2. edition, 2001.

[47] J. Einbinder, K. Scully, R. Pates, J. Schubart, and R. Reynolds. Case study: a data warehouse for an academic medical center. *J Healthc Inf Manag*, 15(2):165–175, Summer 2001.

[48] S. E. Fahlman. An empirical study of learning speed in back-propagation networks. Technical Report Computer Science Technical Report, Carnegie-Mellon University, 1988.

[49] E. Fix and J. Hodges. Discriminatory analysis, non-parametric discrimination: consistency properties. Technical report, USAF Scholl of aviation and medicine, Randolph Field, 1951. 4.

[50] E. Forgy. Cluster analysis of multivariate data: efficiency versus interpretability of classifications. *Biometrics*, 21:768–780, 1965.

[51] E. Frank and I. H. Witten. Generating accurate rule sets without global optimization. In *ICML '98: Proceedings of the Fifteenth International Conference on Machine Learning*, pages 144–151, San Francisco, CA, USA, 1998. Morgan Kaufmann Publishers Inc.

[52] J. H. Friedman. Another approach to polychotomous classification. Technical report, Department of Statistics, Stanford University, 1996.

[53] J. Fürnkranz. Separate-and-conquer rule learning. *Artif. Intell. Rev.*, 13(1):3–54, 1999.

[54] J. Fürnkranz. Round robin classification. *J. Mach. Learn. Res.*, 2:721–747, 2002.

[55] D. T. G. Approximate statistical tests for comparing supervised classification learning algorithms. *Neural Comput.*, 10(7):1895–1923, 1998.

[56] A. Gambier. Real-time control systems: a tutorial. In *Control Conference, 2004. 5th Asian*, volume 2, pages 1024–1031, Jul 2004.

[57] R. N. Gasimov and G. Ozturk. Separation via polihedral conic functions. *Optimization Methods and Software*, 21(4):527–540, 2006.

[58] D. Grossman and P. Domingos. Learning bayesian network classifiers by maximizing conditional likelihood. In *ICML '04: Twenty-first international conference on Machine learning*. ACM Press, 2004.

[59] J. Grzybowski, D. Pallaschke, and R. Urbanski. Data pre-classification and the separation law for closed bounded convex sets. *Optimization Methods and Software*, 20(2-3):219–229, 2005.

[60] H. Guo and H. L. Viktor. Learning from imbalanced data sets with boosting and data generation: the databoost-im approach. *SIGKDD Explor. Newsl.*, 6(1):30–39, 2004.

[61] J. Han and M. Kamber. *Data Mining: Concepts and Techniques*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2005.

[62] P. Hart. The condensed nearest neighbor rule. *Information Technology*, 14(5):515–516, May 1968.

[63] T. Hastie and R. Tibshirani. Classification by pairwise coupling. In *NIPS '97: Proceedings of the 1997 conference on Advances in neural information*

processing systems 10, pages 507–513, Cambridge, MA, USA, 1998. MIT Press.

[64] D. Hebb. *The organization of Behaviour*. Wiley, New-York, 1949.

[65] G. Herman and K. Yeung. On piecewise-linear classification. *IEEE Trans. Pattern Anal. Mach. Intell.*, 14(7):782–786, 1992.

[66] V. Hodge and J. Austin. A survey of outlier detection methodologies. *Artificial Intelligence Review*, 22(2):85–126, 2004.

[67] R. C. Holte, L. E. Acker, and B. W. Porter. Concept learning and the problem of small disjuncts. In *IJCAI'89: Proceedings of the 11th international joint conference on Artificial intelligence*, pages 813–818, San Francisco, CA, USA, 1989. Morgan Kaufmann Publishers Inc.

[68] T. Hou, W. Liu, and L. Lin. Intelligent remote monitoring and diagnosis of manufacturing processes using an integrated approach of neural networks and rough sets. *Journal of Intelligent Manufacturing*, 14:239–253, 2003.

[69] C.-W. Hsu, C.-C. Chang, and C.-J. Lin. A practical guide to support vector classification. Technical report, Department of Computer Science, National Taiwan University, 2003.

[70] C.-W. Hsu and C.-J. Lin. A comparison of methods for multiclass support vector machines. *IEEE Transactions on Neural Networks*, 13(2):415–425, 2002.

[71] K. Huang, I. King, and M. Lyu. Learning maximum likelihood semi-naive bayesian network classifier, 2002.

[72] S.-C. Huang and Y.-F. Huang. Bounds on the number of hidden neurons in multilayer perceptrons. *Neural Networks, IEEE Transactions on*, 2(1):47–55, jan 1991.

[73] X. Huang and W. Pan. Linear regression and two-class classification with gene expression data. *Bioinformatics*, 19(16):2072–2078, 2003.

[74] E. B. Hunt, P. J. Stone, and J. Marin. *Experiments in induction / Earl B. Hunt, Janet Marin, Philip J. Stone.* Academic Press, New York, 1966.

[75] T. I. Two modifications of cnn. *IEEE Transactions on Systems, Man and Cybernetics*, 6(11):769–772, 1976.

[76] A. Isar, S. Moga, and D. Isar. A new denoising system for sonar images. *J. Image Video Process.*, 2009:1–1, 2009.

[77] S. J. and W. G.S. *Pattern Classifiers and Trainable Machines.* Springer-Verlag, New York, 1979.

[78] K. Jackowski and M. Wozniak. Algorithm of designing compound recognition system on the basis of combining classifiers with simultaneous splitting feature space into competence areas. *Pattern Anal. Appl.*, 12(4):415–425, 2009.

[79] A. Jayawardena, D. Fernando, and M. Zhou. Comparison of multilayer perceptron and radial basis function networks as tools for flood forecasting. In *Proceedings of the Conference at Anaheim, CA, IAHS Destructive Water: Water-Caused Natural Disaster, their Abatement and Control*, pages 173–181, 1997.

[80] S. Jesus and D. Dias. Embedded systems architecture, 2008.

[81] G. H. John, R. Kohavi, and K. Pfleger. Irrelevant features and the subset selection problem. In *International Conference on Machine Learning*, pages 121–129, 1994. Journal version in AIJ, available at http://citeseer.nj.nec.com/13663.html.

[82] M. Kantardzic. *Data Mining: Concepts, Models, Methods and Algorithms*. John Wiley & Sons, Inc., New York, NY, USA, 2002.

[83] C. Kenyon and H. Paugam-Moisy. Multilayer neural networks and polyhedral dichotomies. *Annals of Mathematics and Artificial Intelligence*, 24(1-4):115–128, 1998.

[84] R. Kimball. *The data warehouse toolkit: practical techniques for building dimensional data warehouses*. John Wiley & Sons, Inc., New York, NY, USA, 1996.

[85] R. Kohavi. A study of cross-validation and bootstrap for accuracy estimation and model selection. In *IJCAI'95: Proceedings of the 14th international joint conference on Artificial intelligence*, pages 1137–1143, San Francisco, CA, USA, 1995. Morgan Kaufmann Publishers Inc.

[86] H. Kopetz. *Real-Time Systems: Design Principles for Distributed Embedded Applications*. Kluwer Academic Publishers, Norwell, MA, USA, 1997.

[87] A. Kostin. A simple and fast multi-class piecewise linear pattern classifier. *Pattern Recognition*, 39:1949–1962, 2006.

[88] S. B. Kotsiantis. Supervised machine learning: A review of classification techniques. *Informatica*, 31(3):249–268, 2007.

[89] S. Kumar, J. Ghosh, and M. M. Crawford. A hierarchical multiclassifier system for hyperspectral data analysis. In *MCS '00: Proceedings of the First International Workshop on Multiple Classifier Systems*, pages 270–279, London, UK, 2000. Springer-Verlag.

[90] L. I. Kuncheva. Clustering and selection model for classifier combination, 2000.

[91] R. Labib and K. Khattar. Mlp bilinear separation. *Neural Comput. Appl.*, 19(2):305–315, 2010.

[92] P. W. Langley and J. Carbonell. Language acquisition and machine learning. In B. MacWhinney, editor, *Mechanisms of language acquisition*. Lawrence Erlbaum, Hillsdale, N.J., 1987.

[93] T.-F. Lee, M.-Y. Cho, C.-S. Shieh, and F.-M. Fang. Particle swarm optimization-based svm application: Power transformers incipient fault syndrome diagnosis. *Hybrid Information Technology, International Conference on*, 1:468–472, 2006.

[94] G. Leen and D. Heffernan. Expanding automotive electronic systems. *Computer*, 35(1):88–93, 2002.

[95] Y. Li, M. Pont, N. Jones, and J. Twiddle. Applying mlp and rbf classifiers in embedded condition monitoring and fault diagnosis applications. *Transactions of the Institute of Measurement & Control*, 23(3):313–339, December 2001.

[96] A. Likas, N. Vlassis, and J. Verbeek. The global k-means clustering algorithm. *Pattern Recognition*, 36(2):451–461, 2003.

[97] C. Linnaeus. *Systema Naturae*. Holmiae, 10th edition, 1758.

[98] H. Liu and H. Motoda. *Instance Selection and Construction for Data Mining*. Kluwer Academic Publishers, Norwell, MA, USA, 2001.

[99] F. Lotte, M. Congedo, A. Lcuyer, F. Lamarche, and B. Arnaldi. A review of classification algorithms for eeg-based brain-computer interfaces. *Journal of Neural Engineering*, 4(2):R1, 2007.

[100] J. B. Macqueen. Some methods for classification and analysis of multivariate observations. *Proceedings of 5-th Berkeley Symposium on Mathematical Statistics and Probability*, 1(281-297), 1967.

[101] S. Markovitch and D. Rosenstein. Feature generation using general constructor functions. In *MACHINE LEARNING*, pages 59–98. The MIT Press, 2002.

[102] P. Marwedel. *Embedded System Design*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.

[103] B. Mendil and K. Benmahammed. Simple activation functions for neural and fuzzy neural networks. In *Circuits and Systems, 1999. ISCAS '99. Proceedings of the 1999 IEEE International Symposium on*, volume 5, pages 347 –350 vol.5, 1999.

[104] R. Michalski and J. Larson. Selection of most representative training examples and incremental generation of vl1 hypotheses: the underlying methodology and the description of programs ESEL and AQ11. UIUCDCS-R 78-867, Computer Science Department, Univ. of Illinois at Urbana-Champaign, 1978.

[105] D. Michie, D. Spiegelhalter, and C. Taylor. *Machine Learning, Neural and Statistical Classification*. Ellis Horwood, Hemel Hempstead, 1994.

[106] M. Minsky and P. S. *Perceptrons*. MIT Press, Cambridge, MA, 1969.

[107] T. Mitchell. Generative and discriminative classifiers: Naive bayes and logistic regression. http://www.cs.cmu.edu/ tom/mlbook/NbayesLogReg-2-05.pdf.

[108] S. K. Murthy, S. Kasif, and S. Salzberg. A system for induction of oblique decision trees. *J. Artif. Int. Res.*, 2(1):1–32, 1994.

[109] R. Niemann. *Hardware/Software CO-Design for Data Flow Dominated Embedded Systems*. Kluwer Academic Publishers, Norwell, MA, USA, 1998.

[110] N. J. Nilsson. *Learning Machines*. McGraw-Hill, New York, 1965.

[111] K. Ord. Outliers in statistical data : V. barnett and t. lewis, 1994, 3rd edition, (john wiley & sons, chichester), 584 pp., [uk pound]55.00, isbn 0-471-93094-6. *International Journal of Forecasting*, 12(1):175–176, March 1996.

[112] C. Orsenigo and C. Vercellis. Accurately learning from few examples with a polyhedral classifier. *Computational Optimization and Applications*, 38(2):235–247, 2007.

[113] M. A. Oskoei and H. Hu. Support vector machine-based classification scheme for myoelectric control applied to upper limb. *IEEE Transactions on Biomedical Engineering*, 55(8):1956–1965, 2008.

[114] G. Ou and Y. L. Murphey. Multi-class pattern classification using neural networks. *Pattern Recogn.*, 40:4–18, January 2007.

[115] A. Owens. Empirical modeling of very large data sets using neural networks. In *IJCNN '00: Proceedings of the IEEE-INNS-ENNS International Joint Conference on Neural Networks (IJCNN'00)-Volume 6*, page 6302, Washington, DC, USA, 2000. IEEE Computer Society.

[116] C. P and S. Delany. k-nearest neighbour classifiers. Technical report, University College Dublin, School of Computer Science and Informatics, 2007.

[117] G. Pagallo and D. Haussler. Boolean feature discovery in empirical learning. *Mach. Learn.*, 5(1):71–99, 1990.

[118] H. Palm. A new piecewise linear classifier. In *ICPR90*, volume 1, pages 742–744, 1990.

[119] D.-C. Park and D.-M. Woo. Prediction of network traffic using dynamic bilinear recurrent neural network. In *ICNC (2)*, pages 419–423, 2009.

[120] Y. Park and J. Sklansky. Automated design of multiple-class piecewise linear classifiers. *Journal of Classification*, 6:195–222, 1989.

[121] Z. Pawlak, J. Grzymala-Busse, R. Slowinski, and W. Ziarko. Rough sets. *Commun. ACM*, 38(11):88–95, 1995.

[122] K. Philip. Design constraints on embedded real time control systems. In *Systems Design & Network Conference proceedings*, 1990.

[123] J. Quinlan. Induction of decision trees. *Machine Learning*, 1(1):81–106, 1986.

[124] J. Quinlan. *C4.5:Programs for Machine Learning*. Morgan Kaufmann, 1993.

[125] B. Raducanu and J. Vitrià. Online nonparametric discriminant analysis for incremental subspace learning and recognition. *Pattern Anal. Appl.*, 11(3-4):259–268, 2008.

[126] T. Reinartz. A unifying view on instance selection. *Data Min. Knowl. Discov.*, 6(2):191–210, 2002.

[127] R. E. Reinke and R. S. Michalski. Incremental learning of concept descriptions: A method and experimental results. *Machine intelligence 11 proceedings*, pages 263–288, 1988.

[128] J. Rennie, L. Shih, J. Teevan, and D. Karger. Tackling the poor assumptions of naive bayes text classifiers. In *Proceedings of ICML-03, 20th International Conference on Machine Learning*, Washington, DC, 2003. Morgan Kaufmann Publishers, San Francisco, USA.

[129] R. Rifkin and A. Klautau. In defense of one-vs-all classification. *J. Mach. Learn. Res.*, 5:101–141, 2004.

[130] J. Rissanen. A universal prior for integers and estimation by minimum description length. *The Annals of Statistics*, 11(2):416–431, 1983.

[131] F. Rosenblat. The perceptron: a probabilistic model for information storage and organisation in the brain. *Psychological Review*, 65:386–408, 1958.

[132] D. E. Rumelhart, G. Hinton, and R. J. Williams. Learning internal representations by error propagation. In D. E. Rummelhart and J. L. McClelland, editors, *Parallel Distributed Processing*, volume 1. MIT Press, Cambridge MA, 1986.

[133] K. S. *Information Theory and Statistics*. Wiley, New York, 1959.

[134] Y. Saeys, I. Inza, and P. Larraaga. A review of feature selection techniques in bioinformatics. *Bioinformatics*, 23(19):2507–2517, 2007.

[135] S. N. Sanchez, E. Triantaphyllou, J. Chen, and T. W. Liao. An incremental learning algorithm for constructing boolean functions from positive and negative examples. *Comput. Oper. Res.*, 29(12):1677–1700, 2002.

[136] W. E. Saris. A technological revolution in data collection. *Quality and Quantity*, 23(3):333–349, 09 1989.

[137] J. C. Schlimmer and R. H. Granger Jr. Incremental learning from noisy data. *Mach. Learn.*, 1(3):317–354, 1986.

[138] B. Schulmeister and F. Wysotzki. The piecewise linear classifier dipol92. *In Proceedings of the European Conference on Machine Learning (Catania, Italy). F. Bergadano and L. De Raedt, Eds. Springer-Verlag New York,Secaucus, NJ*, pages 411–414, 1994.

[139] L. Shalabi, Z.Shaaban, and B. Kasasbeh. Data mining: A preprocessing engine. *Journal of Computer Science*, 2(9):735–739, 2006.

[140] J. Shlens. A tutorial on principal component analysis. http://www.snl.salk.edu/ shlens/pub/notes/pca.pdf, December 2005.

[141] J. Sklansky and G. G.S. Wassel. Pattern classifiers and trainable machines. *Springer, Berlin*, 1981.

[142] J. Sklansky and L. Michelotti. Locally trained piecewise linear classifiers. *IEEE Trans. Pattern Anal. Mach. Intell.*, 2(2):101–111, 1980.

[143] C. Soares and P. B. Brazdil. Selecting parameters of svm using meta-learning and kernel matrix-based meta-features. In *SAC '06: Proceedings of the 2006 ACM symposium on Applied computing*, pages 564–568, New York, NY, USA, 2006. ACM.

[144] Y. Sun, S. Shimada, and M. Morimoto. Visual pattern discovery using web images. In *MIR '06: Proceedings of the 8th ACM international workshop on Multimedia information retrieval*, pages 127–136, New York, NY, USA, 2006. ACM.

[145] J. A. K. Suykens, G. Horváth, S. Basu, C. Micchelli, and J. Vandevalle, editors. *Advances in Learning Theory: Methods, Models and Applications*, volume 190. IOS Press, NATO Science Series, 2003.

[146] R. Takiyama. A two-level committee machine: a representation and a learning procedure for general piecewise linear discriminant functions. *Pattern Recognition*, 13(3):269 – 274, 1981.

[147] D. M. J. Tax and R. P. W. Duin. Using two-class classifiers for multiclass classification. In *ICPR (2)*, pages 124–127, 2002.

[148] H. Tenmoto, M. Kudo, and S. M. Piecewise linear classifiers preserving high local recognition rates. *Kybernetika*, 34(4):479–484, 1998.

[149] H. Tenmoto, M. Kudo, and M. Shimbo. Piecewise linear classifiers with an appropriate number of hyperplanes. *Pattern Recognition*, 31(11):1627–1634, 1998.

[150] M. Towsey, D. Alpsan, and L. Sztriha. Training a neural network with conjugate gradient methods. *Neural Networks*, 1:373 – 378, 1995.

[151] B. Tudu, B. Kow, N. Bhattacharyya, and R. Bandyopadhyay. Comparison of multivariate normalization techniques as applied to electronic nose based pattern classification for black tea. In *ICST 2008. 3rd International Conference on Sensing Technology, 2008.*, pages 254–258, Nov 2008.

[152] V. Vapnik. *Theory of Pattern Recognition.* Nauka, Moscow, 1974. In Russian.

[153] V. Vapnik. *Estimation of Dependences Based on Empirical Data: Springer Series in Statistics (Springer Series in Statistics).* Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1982.

[154] V. Vapnik. *The Nature of Statistical Learning Theory.* Springer, New York, 1995.

[155] V. N. Vapnik. *Statistical Learning Theory.* Wiley-Interscience, 1998.

[156] K. Veropoulos, C. Campbell, and N. Cristianini. Controlling the sensitivity of support vector machines. In *Proceedings of the International Joint Conference on AI*, pages 55–60, 1999.

[157] M. W. *Computer Oriented Approaches to Pattern Recognition.* Academic Press, New York, 1972.

[158] P. H. Winston. Learning structural descriptions from examples. Technical report, Massachusetts Institute of Technology, Cambridge, MA, USA, 1970.

[159] I. Witten and E. Frank. *Data Mining: Practical machine learning tools and techniques*. Morgan Kaufmann, San Francisco, 2nd edition, 2005.

[160] J. R. Wolpaw. Brain-computer interfaces (bcis) for communication and control. In *Assets '07: Proceedings of the 9th international ACM SIGACCESS conference on Computers and accessibility*, pages 1–2, New York, USA, 2007. ACM.

[161] H.-S. Wong, K. K. Cheung, C.-I. Chiu, Y. Sha, and H. H. Ip. Hierarchical multi-classifier system design based on evolutionary computation technique. *Multimedia Tools Appl.*, 33(1):91–108, 2007.

[162] M. W.S. and P. W. A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, 5:115–133, 1943.

[163] L. Xu and C. Li. Multi-objective parameters selection for svm classification using nsga-ii. In P. Perner, editor, *Advances in Data Mining*, volume 4065 of *Lecture Notes in Computer Science*, pages 365–376. Springer Berlin / Heidelberg, 2006.

[164] H. Yu, J. Yang, and J. Han. Classifying large data sets using svms with hierarchical clusters. In *KDD '03: Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 306–315, New York, NY, USA, 2003. ACM Press.

[165] L. Yu and H. Liu. Efficient feature selection via analysis of relevance and redundancy. *J. Mach. Learn. Res.*, 5:1205–1224, 2004.

[166] L. Z.-P. and B. B. Comparison of a neural network and a piecewise linear classifier. *Pattern Recognition Letters*, 12(11):649–655, 1991.

[167] Z. Zakaria, N. A. M. Isa, and S. A. Suandi. A study on neural network training algorithm for multiface detection in static images. In *World Academy Of Science, Engineering and Technlogy*, 62, pages 170–174, February 2010.

[168] S. Zhang, C. Zhang, and Q. Yang. Data preparation for data mining. *Applied Artificial Intelligence*, 17:375–381, 2002.

[169] Y. H. Zweiri, J. F. Whidborne, and L. D. Seneviratne. A three-term backpropagation algorithm. *Neurocomputing*, 50:305 – 318, 2003.