NTNU – Trondheim
Norwegian University of
Science and Technology

Enhanced Similarity Matching by Grouping of Features

Andreas Ståleson Landstad

Master of Science in Computer Science
Submission date: July 2012
Supervisor: Agnar Aamodt, IDI

Norwegian University of Science and Technology
Department of Computer and Information Science

Assignment

Instance-Based Learning is a growing machine learning paradigm. When classifying a sample (or instance) in In k-Nearest-Neighbor (kNN) type of methods, the sample is compared to previously saved samples.

Verdande Technology is a company that applies Cased Based Reasoning in the oil drilling domain in order to improve the productivity and safety of drilling operations. The target of the DrillEdge software is to avoid faults by reusing past experience. Each input data stream can be thought of as a *feature*, but in order to be effectively used to index cases in a case base these features need to be abstracted.

It is believed that some of the input-features available during oil drilling are related locally and that abstract features that group these together therefore will be useful for detecting different anomalies. The hypothesis is that this might lead to improved prediction accuracy. This is a novel approach, which the company has just started to study, and the main focus of this thesis work. The idea behind the system in this assignment is to study the extraction of abstract features where one abstract feature is a group that contains one or more of these input-features.

The thesis work should combine theoretical investigations with the development of an experimental system, based on existing work in Verdande Technology. This also includes a study of the concept of *power-average*. A power-average of exponent n is the n-th root of the sum of the features in a group where each feature has exponent n. When n is large, the features with the highest values dominate and when n is small (negative), the smaller-valued features dominate. The key idea is to group features in different groups and then calculate the power average of each group with different exponents. A genetic algorithm should be applied to reveal the underlying tree structure of groups that gives the lowest classification-error.

Methods for boosting the accuracy of a learning algorithm should be given particular attention. This includes weighing the different features and application of dimensionality reduction methods such as Principal Component Analysis (PCA). It also includes methods for how similarity between features are measured and which classification rule that is used.

The results of this thesis work should include:

- A theoretical/experimental study of how accuracies may be boosted by use of dimensionality reduction methods and other relevant methods.
- A discussion of how classifiers can be evaluated.
- A presentation of the system outlined above.
- Tests performed on this system, which results should be benchmarked to results of for example k-Nearest-Neighbor classification.

Oppgavebeskrivelse

Instansbasert læring er et voksende maskinlæringsparadigme. I k-Nearest-Neighbor-lignende metoder blir en instans som skal klassifiseres sammenlignet mot tidligere lagrede instanser.

Verdande Technology er et selskap som bruker Case Based Reasoning i oljedrillingsfeltet for å forbedre produktivitet og sikkerhet i olje-drillingssammenheng. Målet til DrillEdge-programvaren er å unngå feil ved å gjenbruke tidligere erfaringer. Hver strøm av input-data som DrillEdge-programvaren bruker kan bli tenkt på som en *feature*, men for å kunne indeksere caser på en effektiv måte, må informasjon først bli abstrahert fra disse.

Man antar at flere av disse input-featurene er relaterte til hverandre lokalt og at abstrakte features som grupperer disse sammen vil være nyttige for å detektere ulike anomaliteter. Hypotesen er at dette kan føre til bedre prediksjoner. Dette er en ung metode som selskapet nylig har startet å forske på og hovedfokuset i denne oppgaven. Ideen bak systemet i denne oppgaven er å studere ekstrahering av abstrakte features hvor en abstrakt feature er en gruppe som inneholder flere input-features.

Oppgaven skal kombinere teoretiske undersøkelser sammen med utvikling av et eksperimentelt system, basert på jobben som er gjort i Verdande Technology. Dette inkluderer også en studie av konseptet *power-average*. En power-average med eksponent n er n-te roten av summen av features i en gruppe hvor hver feature har eksponent n. Nr n er høy vil features med høy verdi dominere, mens når n er liten (negativ) vil features med lav verdi dominere. Ideen er å gruppere features i ulike grupper og kalkulere power-averaget til de forskjellige gruppene med ulike eksponenter. En genetisk algoritme skal bli brukt til å finne ut hvilke trestrukturer av grupper som vil gi lavest klassifiseringsfeil.

Det bør også spesielt vies oppmerksomhet til metoder for å forbedre prediksjonsevnen til en klassifikator. Dette inkluderer å vekte features og å bruke dimensjonalitetsreduksjonsmetoder som for eksempel Principal Component Analysis (PCA). Andre viktige aspekter med tanke på prediksjonsevnen er hvordan similaritet blir kalkulert og hvilken klassifiseringsregel som blir brukt.

Resultatene av denne oppgaven burde inkludere:

- En teoretisk/eksperimentell studie av hvordan prediksjonsevnen til en klassifikator kan blir forbedret ved bruk av dimensjonalitetsreduksjonsmetoder og andre metoder.
- En diskusjon av hvordan klassifikatorer kan bli evaluerte.
- En presentasjon av systemet som er foreslått over.
- Tester gjort på systemet, hvis resultater skal bli sammenlignet med for eksempel resultater av k-Nearest-Neighbor-klassifisering.

Abstract

In this report we introduce a classification system named Grouping of Features (GoF), together with a theoretical exploration of some of the important concepts in the Instant Based Learning(IBL)-field that are related to this system.

A dataset's original features are by the GoF-system grouped together into abstract features. Each of these groups may capture inherent structures in one of the classes in the data. A genetic algorithm is used to extract a tree of such groups that can be used for measuring similarity between samples. As each class may have different inherent structures, different trees of groups are found for the different classes. To adjust the importance of one group in regards to the classifier, the concept of *power average* is used. A group's power-average may let either the smallest or the largest value of its group dominate, or take any value in-between. Tests show that the GoF-system outperforms kNN at many classification tasks.

The system started as a research project by Verdande Technology, and a set of algorithms had been fully or partially implemented before the start of this thesis project. There existed no documentation however, so we have built an understanding of the fields on which the system relies, analyzed their properties, documented this understanding in explicit method descriptions, and tested, modified and extended the original system.

During this project we found that scaling or weighting features as a data preprocessing step or during classification often is crucial for the performance of the classification-algorithm. Our hypothesis then was that by letting the weights vary between features and between groups of features, more complex structures could be captured. This would also make the classifier less dependent on how the features are originally scaled. We therefore implemented the Weighted Grouping of Features, an extension of the GoF-system.

Notable results in this thesis include a 95.48 percent and 100.00 percent correctly classified non-scaled UCI Wine dataset using the GoF- and WGoF-system, respectively.

Preface

This report was written as my master's thesis at the Department of Computer and Information Science (IDI) at the Norwegian University of Science and Technology (NTNU).

Verdande Technology is a cooperating company to the Artificial Intelligence group at IDI, NTNU that often includes master students in their research projects. Through Agnar Aamodt I got in touch with them regarding a new research project led by Sigve Hovda and this research project has set the theme for this master's thesis.

I want to thank Verdande Technology for including me in their research project and Sigve Hovda especially for his ideas, his support and for being a strong theoretical backbone to rely on throughout this project. I also want to thank Agnar Aamodt for providing new perspectives, guidance and suggestions which have improved this report at a large scale and Sigurd Fosseng for providing interesting ideas and insights with his work.

Contents

List of Figures

List of Tables

1

INTRODUCTION

1.1 Motivation

Verdande Technology does R&D and develops software for monitoring oil wells and oil drilling processes. The software analyzes time-series of data in real time and matches patterns in the data to previously saved data. Time-series in he previously saved data has been has been labeled if an anomaly happened as a result of this. If a new time series matches one of these labeled saved patterns, a user can be alerted and take action to prevent an unwanted outcome. The purpose of the system documented in this report is enhanced use of this information in order to better be able to predict such anomalies. The potential success of this work is a result which can be applied to classification tasks such as the ones used during oil drilling.

1.2 Problem Definition

This report presents a classification-system that is part of ongoing research led by Sigve Hovda in Verdande Technology. The system is in this report called Grouping of Features (GoF) and is based on theory from the field of Instance Based Learning (IBL). As backbone it uses a Genetic Algorithm with k-Nearest-Neighbor (kNN) as fitness-function in order to search for optimal solutions in a search space created

by custom modules.

IBL methods including kNN stores samples (instances) represented as vectors of features together with their classification. New samples are classified by comparing the new sample to the stored samples in order to find objects that in some sense are similar to the new sample. The GoF-system's goal is to improve classification accuracies in multi-class learning problems. Multi-class learning problems are learning problems where there exists a finite amount of classes and where a sample can be classified as belonging to precisely one of these classes (Dietterich, 1995).

Through the Grouping of Features system we suggest to building trees of features where features are grouped together. The purpose of this is the intuition that some features may become more meaningful to the classifier when combined than when they are alone. An example of this can be seen in figure 1.1. The blue class here is distributed in a gaussian along the line $y = 1 - x$ and the green class is uniformly distributed from 0 to 1 in both dimensions. Comparing the value of x of a new object to the stored objects alone will not help determining the class of the new object, as both classes are just as likely to take any value between 0 and 1 in either dimension.

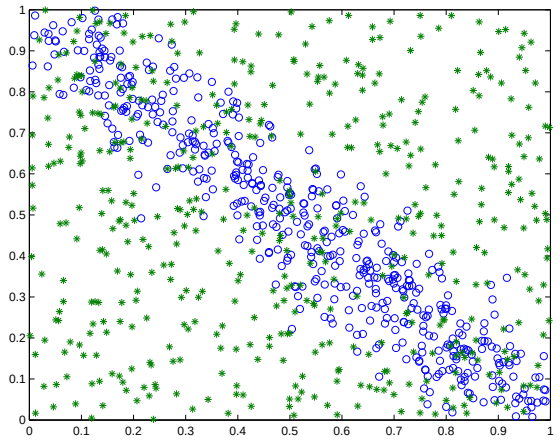

Figure 1.1: The 2f-dataset. One class uniformly distributed in the range [0,1] in both dimensions x, y, the other a gaussian with mean $y = 1 - x$ and a standard deviation of 0.1.

If one could group these two features (that is, dimension x and dimension y) together and have the classifier to classify an object as *blue* if the the sum of x and y is close to 1.0, and *green* if not, the classifier would perform well. If kNN was used to classify objects were the features were stored as the sum of x and y, all *blue* samples would be close to each other.

The GoF-system aims to capture different structures such as the one above. In addition, however, each class may have different inherent structures. Because of this the GoF-system lets every class represent the datapoints with their own tree-structure. The tree-structures are then used when measuring similarity between samples. As such a tree may be thought of as a distance-function. Optimal or sub-optimal parameters determining which features to group together, how the tree should look and how the distance should be calculated for each group are found using a genetic algorithm.

In order to be able to successfully create a well-performing system, it is im-

portant to understand key aspects within the field. Because of this, a thorough theoretical exploration of the field has been performed. Some of the findings in this exploration are presented in chapter 2.

1.3 Our Context in this Research Project

When we joined this research project, Verdande Technology had begun implementing the Grouping of Features system. The system was already able to do classifications, but some of the components that had been programmed were not very easily read and other components were not set up in the most intuitive fashion. As the system was not documented at all, we spent some time understanding this system by rewriting some of the components into more intuitive ones and writing tests for these components. Later, while documenting the system, we also replaced most components when we extended the system into the Weighted Grouping of Features System.

1.4 Research Goals and Methodology

The main research-questions supporting the theoretical exploration and the building and use of the GoF-system, were the following.

- Can the Grouping of Features system produce higher classification rates than for example basic kNN?
- Will a Genetic Algorithm be able to successfully find good parameters for these groups?
- Can the system be modified in any way to perform even better?
- What are the grouping of features-system's strengths and weaknesses?
- For what type of problems will this system be useful?

The methodological approach to find answers to these questions were guided by the following questions:

- What defines IBR-methods and k-Nearest-Neighbor?
- How can these be customized, and when is such customization useful?
- How can this usefulness be evaluated, that is, how should one evaluate a classifier?
- What are the key characteristics of a genetic algorithm, and how does one use it as a search heuristic?

Our high-level structured progress plan contains the following four steps:

1. An understanding of the field was sought by doing a theoretical exploration with supporting experiments
2. An understanding of the system was sought by documenting the system
3. The learnings from the theoretical exploration was used to try out different modifications on the original system
4. Tests were performed along the way in order to evaluate and understand different methods and the GoF-system.

1.5 Presentation of the Structure of this Report

The report is divided into six chapters. After this chapter, *Introduction*, comes the chapter *Theoretical Exploration*. That chapter provides an overview over some of the research and tests that have been performed as a theoretical base for the GoF-system. Instead of presenting this as a list of technologies and variations together with results from literature, a deeper understanding has been sought by performing experiments on many of these. After the stage has been set, the GoF-system is described in the chapter *Grouping of Features*. This presentation is followed by tests and results in the chapter *Tests and analysis*. The report and its main discoveries are then summed up in the *Conlusion*-chapter.

2

Theoretical Exploration

2.1 Introduction

In order to understand the possibilities of the Grouping of Features (GoF)-system presented in Chapter3, and in order to improve it and extend it, it was necessary to do a thorough theoretical exploration of the basic technologies and tools on which it is based. As k-Nearest-Neighbor is used by the GoF-system, the examples closely follow this method. In addition a somewhat deeper understanding of feature selection was sought, and for editing and selecting features one method was studied in particular, namely the popular Principal Component Analysis (PCA). In addition there is a brief presentation of the search heuristic genetic algorithm with an example in the end of this chapter. As a genetic algorithm is also used in the GoF-system, the search heuristic is further explained in chapter 3.

The theoretical exploration in this chapter is presented as a discussion of key elements together with supporting practical examples. Throughout this explorative process, some elements we have not found in literature have been discovered. We have implemented the mentioned experiments such as variants of k-Nearest-Neighbor, feature-scaling and -selection using PCA and a sample genetic algorithm in Java and/or Matlab with help of standard packages.

An introduction to kNN starts off this chapter together with a discussion of how classifiers are evaluated. This latter discussion is necessary as the other sections all evaluate experiments using these concepts.

2.2 K-Nearest-Neighbours Classification

The k-Nearest-Neighbor (kNN)-algorithm compares distances between a test-sample and all training-samples that have been stored on beforehand (Cover, 1967). As kNN is a supervised algorithm, the training examples all have known labels[1]. The k samples that have the shortest distance from the test sample are the k neighbors of the test-sample and the majority class among these is chosen as its label. Figure 2.1 is showing how kNN classifies a new sample from its $k = 3$ or $k = 5$ neighbors.

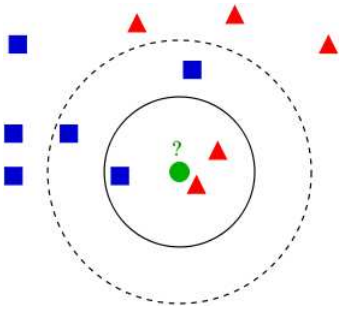

Figure 2.1: Example of k-nearest neighbour classification with $k = 3$ (solid line) and $k = 5$ (dashed line) from (Ajanki, 2007)

As there are two triangles among the three nearest neighbours, and only one square, the class triangle is predicted as the unknown sample's class. kNN does reasonably well at many classification tasks and is simple and easy to implement. One of its downsides is that all of the computational complexity is placed at execution time[2]. As such kNN is a lazy learning algorithm. In algorithm 1 is a high

[1] A label is an indicator of which class the sample belong to.

[2] If n is the number of training examples and m is the number of features in each training example, the complexity for classifying one sample is $O(nm)$ for regular kNN. For each test-example to be evaluated one has to iterate over all training-examples and measure the distance between all features.

level pseudocode for kNN that we have written.

Algorithm 1 The kNN algorithm

```
s_u is the sample which is to be classified
S is the vector of training examples
NN is vector of length k and is used to store nearest neighbors. Its initialized
with value Infinite for all members
for sample s_i in S do
    d_ui is the distance between s_u and s_i
    for sample s_n in NN do
        d_un is the distance between s_u and s_n
        if d_un < d_ui then
            replace s_n in NN with s_i and continue from 4.
        end if
    end for
end for
```

An example (or a sample) is a pair $(\mathbf{x}, y)$ where $\mathbf{x}$ is a set of features and y is a label. Russell and Norvig (1995) argues that a supervised learning method such as kNN formally is not a classifier per se as a classifier only is a function $\mathbf{X} \implies Y$ (Russell and Norvig, 1995). As kNN produces a prediction by comparing the new sample to all of the training data and no generalization is done on beforehand, kNN is instead a higher order function $(\mathbf{x} \implies Y)_* \implies (\mathbf{x} \implies y)_*$. Given a specific set of training data and a given sample x to be predicted, the classification-equation is $(X, Y)_{training-data} \implies (x \implies y)_{prediction}$. Nevertheless, the task one uses kNN for is classification. As such this thesis will follow common jargon in the classification-community and use the word classifier about supervised learning methods.

An Instance Based Learning-method such as kNN is a multi-class learning problem if there is a finite set of labels Y can take. It is especially this kind of task that is of interest in this report. If Y is only bound by $\mathbb{R}$, it is instead a regression-method where the label given is the average of its k nearest neighbors. There also

exist other supervised learning tasks besides multi-class learning which will not be discussed. Applications for kNN include searching in databases (Seidl, 1997), fraud detection (Ngai et al., 2011) and information retrieval(Yang and Liu, 1999).

kNN is one of the ten most influential data-mining algorithms (Wu et al., 2007) and has one strong guarantee, namely that with infinite training data, kNN will never perform worse than twice the Bayes error rate, and it is guaranteed to approach the Bayes error rate for some value of k (Cover, 1967)[3]. In the examples used in this report the Bayes error rate is the error rate one would obtain given optimal decision lines. In figure 2.2 the optimal decision line is at X_B, that is, this is the lowest possible error one can obtain. Moving the decision line in either direction will increase the classification error rate. The Figure 2.2 from Duda et al. (2000) shows how the non-optimal decision line at X^* will yield error rates similarly to the sum:

$$err = \int_{R_1} p(x|\omega_2)p(\omega_2)dx + \int_{R_2} p(x|\omega_1)p(\omega_1)dx \tag{2.1}$$

When trying to minimize the error rate, the goal of the classifier is to get as close to the optimal decision line as possible. The optimal decision line is at X_b, and using this line one would reduce the error by:

$$reduced\_err = \int_{X_B}^{X^*} p(x|\omega_2)p(\omega_2)dx - \int_{X_B}^{X^*} p(x|\omega_1)p(\omega_1)dx \tag{2.2}$$

2.2.0.1 Design Choices for kNN

For regular kNN there are four main design choices:

- Which classification rule to use.
- k, the number of neighbors to consider in the classification rule.
- Which distance function to use.

[3]Further refinements in regard to the Bayes error rate were formalized by Hostetler (1975).

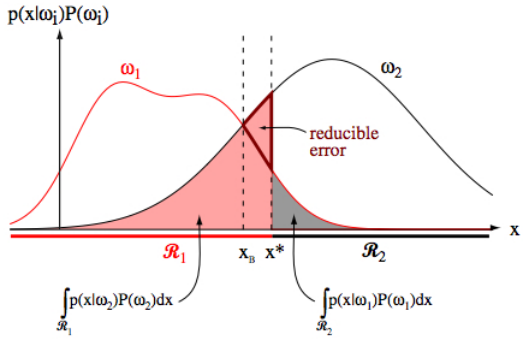

Figure 2.2: From Duda et al. (2000): Bayes error rate.

- What to do when there is a tie.

The choice of distance-function will be discussed in section 2.6 and the other three design choices are the main topics in the next section. The tie-breaker however, will only be mentioned briefly: If one has three classes or possible labels and one has chosen $k = 3$, a tie occurs when the three nearest neighbors each take different labels. For this example a reasonable tie-breaker might be to choose the closest one, or, if the distances are equal, choose one at random. For the case when one has two classes, choosing $k = 3$ avoids the tie-issue, and it is therefore a common choice of k. k=3 with the tie-breakers mentioned are used in this report if nothing else is mentioned.

2.2.1 Classification Rule of kNN

In Loftsgaarden (1965), the k-Nearest-Neighbor density function was formalized. The probability distribution for the label for an unknown sample x, is estimated as:

$$\hat{F}(x) = \frac{k-1}{nV_d r^d} \tag{2.3}$$

In this formula r is the sphere in which its k nearest neighbours lie, d is the dimensionality, n is the total number of samples in the dataset and V_d is the volume

of the mentioned sphere. The science-community later omitted the -1 in 2.3 without loss of consistency, and it is this model kNN uses. The conditional probability of sample x being in the sphere given that the label is y_i, is then:

$$\hat{F}(x|Y = y_i) = \frac{k_i}{n_i V_d r^d} \tag{2.4}$$

where k_i is the number of training-samples with label y_i in the sphere and and n_i is the total number samples with label y_i. With the kNN majority-rule, the prior probability of class y_i is $P(y_i) = \frac{n_i}{n}$, where n_i is the number of samples with class y_i in the dataset and n is the total number of samples in the dataset. The posterior probability for class y_i given a sample x is then given by:

$$P(Y = y_i|x) = \frac{\hat{F}(x|Y = y_i)P(y_i)}{\hat{F}(x)} \tag{2.5}$$

$$= \frac{k_i}{k} \tag{2.6}$$

This is the theoretical basis for the majority classification-rule used in regular kNN, which will be dwelled upon a little more in chapter 3. This rule finds the k nearest neighbors and the majority label among these are chosen as the test-sample's label. Increasing the k ultimately increases the sphere V_d, effectively reducing the effect of over-fitting as the decision lines become smoother, but also using samples further from the sample-point. The choice of k is alas an important choice and will be discussed in 2.2.3.

Albeit being the basis for regular kNN, the GoF-system does not generally use the majority rule. For reasons explained in 3.1.1.2, we expect to quicker obtain higher test results using a mean of the distances from k of the nearest neighbors from *each class* and choose the class with the smallest mean distance from the sample point. We call this the k-mean-rule.

2.2.2 Regression and kNN

kNN can be used as a regression-method by choosing the average of the k nearest neighbors as label for the new sample. In order to improve predictions, a scheme proposed by Dudani (1976) was to weigh the contribution of the different neighbors according to their distance. A typical weighting scheme for regression is to let the contribution of each neighbor be determined by the inverse of the distance from the sample: $\frac{1}{d}$, where d is the distance from the sample point. This is a generalization of linear interpolation (Kaur et al., 2012). Although important in regards to kNN, this is not applicable to multi-class classification tasks such as the ones the GoF is to solve. The GoF-system nevertheless uses a classification rule called *k-mean*. This method also uses a mean of neighbors in order to determine the label of a new sample, but should not be confused with the average-label chosen by the mean method used for regression.The k-mean classification rule will be presented in 3.1.1.2.

2.2.3 Choosing k

Choosing the k used by kNN is perhaps the largest design choice in the kNN-algorithm. The optimal choice of k depends on the data, and generally a small k will make the effect of noise larger, whereas large values of k make the decision boundaries smoother and less distinct. As a rule of thumb one may consider using larger k's when there are many data points, as the probability of having points close to the sample point increases, (Lange et al., 1995).

As an example consider the square-in-square dataset in figure 2.3:

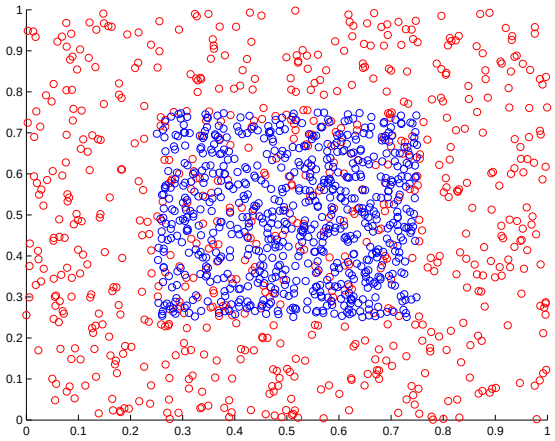

Figure 2.3: Square in Square. One square is uniformly distributed from 0 to 1 in two dimensions, the other from 0.25 to 0.75 in two dimensions. Both classes contain 1000 data points.

The larger square is an area of $\int_0^1 1\,dx = 1$ and the smaller area has an area of $\int_{0.25}^{0.75} 0.75 - 0.25\,dx = 0.3125$.

Because the blue class in the smaller square is more dense than the red, we would optimally like all points within this square to be classified as the red class. Outside the square we would like all the points to be classified as blue. With a small k, almost all points outside the smaller square will be classified correctly. This comes at a cost however, as a small k also will make the classifier classify several points within the small square as blue. A large k on the other hand, will make sure very few (if any) samples within the small square will be classified as blue. The cost is that the decision-boundary will grow and samples outside the small square that are close to the small square, are likely to be classified as red. Cross-Validation or Monte-Carlo(MC) simulations can be used to find a good value of k.

We implemented kNN in Java and Matlab for different test purposes. One

purpose was to see how the value of k changed the results when using kNN for classification. The results of cross-validation simulations show that for the square-in-square dataset, good values of k seem to be between ten and twenty as shown in figure 2.4:

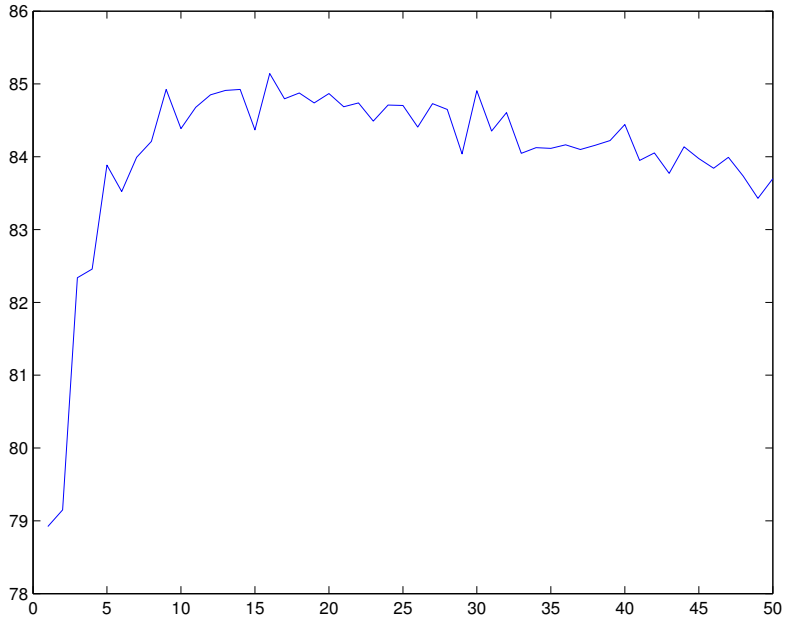

Figure 2.4: Results from running kNN with k varying from 1 to 50.

2.2.4 Different ks for Each Class

One thing we did not find in literature, but which seems promising, is to use a different k for each of the *classes* when using different classification rules. In this way, one can increase or decrease the probability of having the classifier select one particular class in cases of doubt, and a different k might account for different structures in the classes. This can also be used for enabling one to in some way use the Receiver Operator Characteristics (ROC) curve (presented in 2.3.1.2); If it was wanted to have the classifier select one class more often than it originally

did one could increase the k used for this class, and by having different k's for the different classes the decision boundaries might be smoothed in new ways. For kNN the theory is that using different k's will be especially useful when one class has a higher density than the other in a case of doubt. In such cases the decision boundary will be moved towards the lower-density class.

The GoF-system finds different ways of calculating the distances for the different classes, and also varying the k would be a natural extension. This is, however, saved for future work. What we did do, is test using different k's for the different classes on the square-in-square dataset. In table 2.1 are the results shown from this experiment. We varied k from 1 to 11 for the class contained in the large square and from 1 to 25 on the smaller square, both with increments of 2.

k_1	k_2	Result
1	1	77.18
1	3	83.94
1	5	85.98
1	7	87.00
1	9	87.62
1	11	88.38
1	13	88.30
1	15	88.75
1	17	88.42
1	19	88.85
1	21	88.90
1	23	88.06
1	25	87.07
3	1	78.28
3	3	82.17
3	5	83.54
3	7	85.53
3	9	86.17
3	11	86.30
3	13	86.63
3	15	87.11
3	17	87.52
3	19	86.95
3	21	87.43
3	23	87.49
3	25	87.77

k_1	k_2	Result
5	1	76.97
5	3	81.44
5	5	83.76
5	7	84.88
5	9	85.39
5	11	86.08
5	13	86.29
5	15	86.37
5	17	86.17
5	19	86.54
5	21	86.76
5	23	86.51
5	25	86.33
7	1	76.35
7	3	80.69
7	5	83.19
7	7	84.50
7	9	84.85
7	11	85.33
7	13	85.99
7	15	85.52
7	17	86.10
7	19	86.35
7	21	85.97
7	23	86.42
7	25	85.91

k_1	k_2	Result
9	1	75.54
9	3	80.50
9	5	82.79
9	7	84.60
9	9	84.88
9	11	85.25
9	13	85.11
9	15	85.37
9	17	85.27
9	19	86.01
9	21	85.86
9	23	85.52
9	25	86.03
11	1	75.37
11	3	80.52
11	5	82.86
11	7	83.49
11	9	84.13
11	11	84.58
11	13	85.03
11	15	85.49
11	17	85.10
11	19	85.61
11	21	85.30
11	23	85.61
11	25	85.27

Table 2.1: Results from classifying the square-in-square-set using kNN when using different ks for the two different classes.

When classifying using regular kNN, an improvement from 84.48 percent with k=9 to 88.90 percent was achieved when allowing the k's to be different for the different classes. The best results were obtained with $k_1 = 1$ for the red class and $k_2 = 21$ for the blue. The results with k_1 varying from 1-11 and k_2 varying from 1-25 are shown above in table 2.1, both with increments of 2 for the k_i's.

2.3 Evaluating Classifiers

A testing-environment for a classifier is not the same as using the classifier in the real world, and how good a classifier is might not best be evaluated by the intuitive concept "'accuracy"', that is: $\frac{correctly\ classified\ samples}{total\ number\ of\ samples}$. In this section several concepts will be briefly discussed, namely:

- The confusion matrix and special classes
- Training-/test-sets and overfitting
- Cross validation
- Monte Carlo-simulations and theoretical maximums

The next sub-section is a discussion about what accuracy is and why precision rate not always is the best way to measure accuracy.

2.3.1 The Confusion Matrix and Special Classes

Consider a dataset in which there are two possible labels for each sample: Positive p and Negative n. All of the information in regards to measuring the accuracy or usefulness of a classifier would then lie in the matrix in table 2.2:

Table 2.2: Confusion matrix

		Reality	
		p	n
Predicted	p	tp	fp
	n	fn	tn

In this table, known as a confusion matrix, tp, fp, tn and fn are short for True and False Positive and True and False Negative. The previous mentioned "'accuracy"'-measure is then simply $\frac{tp+tn}{tp+fp+tn+fn}$. Note that the denominator aggregates to the total number of samples. There are, however, two other measurements that are useful when evaluating some classifiers. These are "'precision"', $\frac{tp}{tp+fp}$ and "'recall"', $\frac{tp}{tp+fn}$. To exemplify the two an example is called for.

2.3.1.1 Precision/recall-example

The master-student Mary has received a bunch of data describing each student that went to her university ten years ago together with the average of the grades received during their time at the university. Her assignment is to create a classifier that predicts which students that receive the 1 percent highest grades (positive). In order to obtain a 99 percent accuracy, all her classifier has to do is to predict all students as not being in the top 1 percent, that is, as negatives:

$$accuracy = 100\frac{tp + tn}{tp + fp + tn + fn}\% = \frac{0 + 99}{100}\% = 99\% \tag{2.7}$$

Although the accuracy is 99 percent, the classifier does a very poor job of finding the students that are in the top 1 percent. If the classifier were to find most of the students in the top 1 percent, high recall is wanted. To get a 100 percent recall, she could make a classifier that returned only positives. This, like the classifier only returning negatives, is not a useful classifier either. In order to be useful, the classifier needs to have high precision in addition to high recall. Because the set of negatives is much larger than the set of positives, the concept of accuracy is much less meaningful than the other two measurements. An accuracy of 99 percent might be great if combined with a high recall, but without knowing the recall, a 99 percent accuracy might just be a classifier returning only negatives.

2.3.1.2 Receiver Operating Characteristic

For binary classification tasks a balance between the two - high recall vs high accuracy - can be found using a Receiver Operating Characteristic(ROC)-curve. An ROC-graph is a plot of true positive rate (tp/p) versus false positive rate (fp/n). An example of an ROC-curve is shown in figure 2.5.

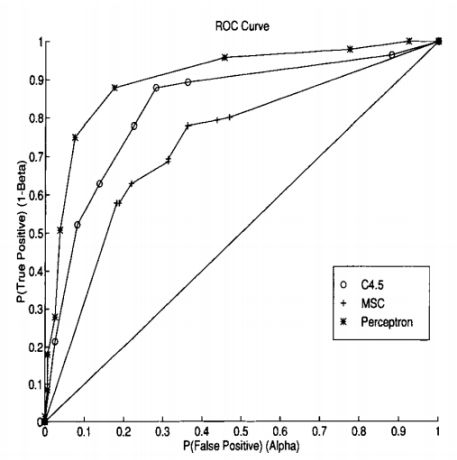

Figure 2.5: Example of a plot of three ROC-curves in one graph from Bradley (1997)

In order to take advantage of this however, one needs to be able to adjust the probability model in a way that increases the probability of choosing one class over the other. This is not something that can be trivially done with IBL-methods as there often are no parameters that can be tweaked in order to move the probabilities of choosing one class smoothly. A suggested method to in some sense increase the probability of one class is presented in the section about using different k's for different classes(2.2.4). If one has an explicit cost of misclassifying either class y_1,y_2 one can use a classification rule that minimizes the cost:

$$f_c(x) = \arg\min_i \sum_{j=1}^{n} C(i|j)P(Y = y_i|x) \tag{2.8}$$

Here the operator $C(i|j)$ is the cost of misclassifying a label as y_i if the true label were to be y_j. $P(Y = y_i|x)$ is the conditional (posteriori) probability of the label being y_i given the sample x. This is further presented in the section 2.2.1

2.3.1.3 Special Classes and This Report

When having special classes such as the one above it is therefore natural to consider the whole confusion matrix as all three evaluation-methods depend on all of the

numbers and one method alone might be meaningless. For most of the experiments and tests in this report however, the accuracy-measure is meaningful and considered adequate. Section 2.3.2 discusses another way in which a classifier that will perform poorly in the real world may obtain a 100 percent accuracy when an important step in the evaluation process are not being followed - namely splitting the data into exclusive training and test sets.

2.3.2 Training-/Test-Tets and Overfitting

How well a classifier performs in the real world might measured by how well it predicts labels or classes on data of which it does not know the label of. If one were to know the labels, the best classifier would just return these labels and no classification algorithm would be necessary[4]. In an academic- or research-environment where one actually knows the correct labels for all samples, it would therefore be unreasonable to test a algorithm on the same dataset that it was trained on.

As an example, imagine having a list of the heights of all students at a university, represented by numbers from $\mathbb{R}$. If all students' heights were randomly classified as Class A or Class B with a probability of 50 percent for each, a classifier should obtain a 0.5 precision rate when trying to predict these labels. Training and then testing on a fixed dataset of tuples of $(height, label)$ using kNN with k=1 however, will obtain a 100 percent precision rate - a sample's nearest neighbour would be itself.

Abstractly what has happened, is that the classifier has been fitted to noise as there are no structures in the training data. Fitting a classifier to the exact placements of the data points rather than structures the data was based on is called over-fitting. Because one tested on the same data as the classifier one trained the classifier on, the model was fitted to the training points and when testing on the same points, the accuracy was much higher than what is possible in regards to the

[4]This would be the same as table-lookup.

structure of the data.

2.3.2.1 Held-out data

In order to get results that reflect the performance the classifier will have in the real world, it is necessary to keep some data away from all training, and use this data for testing. By splitting the data into independent training- and testing-sets, overfitting to the training-data will be reflected by the results obtained by the classifier on the test-data. Data used for testing is also called held out data.

Sometimes it is desired to find optimal parameters for the classifier and one wants to perform tests to find these parameters. As using either the training data or the test data for this purpose would result in over fitting to either of the sets, it is necessary to split further into one training, one test and one validation set, where the separate validation set will be used to find good parameters. As such, different data is used for testing in chapter 4 where the GoF- and WGoF-systems are tested.

Holding out data however, has one down-side. By splitting up the data into two sets the amount of data used for either training or testing is also reduced. This is not an issue when one has an unlimited amount of data, but with a limited amount of data there is a trade-off between how how much data one should use for training and how much one should use for testing. More data for training is likely to yield a better classifier and more data for testing may give a more accurate approximation to how well the classifier performs. The latter is because a test-set of few samples makes the result more dependant on each of the individual samples rather than the structures among the samples. Cross-validation is a technique that reduces this issue of having less training- and testing-data:

2.3.3 Cross-Validation

When having a smaller dataset, cross-validation (CV) is an intuitive and easy way of capturing and avoiding overfitting to test-data while keeping the amount of

training-data high. CV randomly splits the data into k folds of approximately equal size. Then, for all k folds, one fold is held out for testing whereas the remaining k-1 folds are used for training. The accuracy of the classifier is then measured as the average of the k results. In this way as much training data as possible is used for training, while reducing random fluctuation of result on the test-set. In this report, 10-fold CV is used when nothing else is mentioned. One type of cross validation that is worth mentioning especially is Leave One Out Cross Validation (LOOCV) as this is used by the Grouping of Features-system. As the name suggests, this is k-fold CV where k equals the total number of samples in the dataset, that is, one uses all of the data except one sample as training-data and tests on the remaining sample. This is done for every sample in the database.

CV provides almost unbiased (Efron and Tibshirani, 1993) results and provides a fair estimate of how well a classifier might do with a limited amount of data.

2.3.3.1 Monte Carlo Simulations and Theoretical Maximums

The datasets used in this chapter are all generated in Matlab manually and it is therefore known exactly how the distributions look. Consider the square-in-square dataset that has been used in previous experiments. Because the class contained in the smaller square (class A) is more dense than the one in the larger, the optimal decision lines follow this square precisely. When Class A is chosen whenever a sample is inside the smaller square and vica versa, the recall of Class A is 100 percent and the recall of Class B is 75 percent. The theoretical maximum accuracy for this dataset is therefore $\frac{100\%+75\%}{2} = 87.75\%$.

In one of our experiments however, we were able to obtain an 88.9 percent accuracy on this dataset when using Cross Validation. The reason for this is simply that with the limited amount of data used for training and testing, there was a random tendency[5] in the data that was favorable for the classifier. Testing a clas-

[5]That is, the tendency does not stem from the distribution the data is picked from, but rather by coincidence because of random structures in the particular samples drawn from this

sifier more precisely, and thus avoiding such random tendancies, is possible using Monte Carlo Simulations.

Monte Carlo methods used for evaluating classifiers such as the ones in this report samples data randomly from the probability distribution the data stems from and splits this dataset into a training- and a test-set. The strength of this method however, is that Monte Carlo simulations draws new training- and test-data many times[6]. The results are then averaged and as the number of results grows, the precision of the average result becomes more accurate.

2.4 Normalization, Standardization and Scaling Data

This section and the one following this section both discuss ways to pre-process data. This section focuses on some of the simplest ways one can pre-process data which often yields higher classification accuracu, whereas the next section focuses on dimension-reduction techniques.

2.4.1 Normalization

Consider having a dataset with the following features representing basket players, together with a label indicating whether or not this basket player has performed any slam dunks the previous five games:

- Height in meters
- Weight in grams

For this example it is assumed that being tall and muscular (heavy) both are good indicator of whether a basket player has performed any slam dunks the previous games. If one were to use the data above for kNN-classification, an increase of 10 percent in height and weight for a basket player that is 1.85m and 85000grams,

distribution.

[6]Rather 10.000 times than for example 10 times.

would result in an increase of 0.185 in one dimension and 8500 in the other, - the change in height is drowned by the change in weight. In order to make changes in the two dimensions equally influential to the kNN-algorithm, one can normalize the dataset.

$$x = \frac{x_{old} - x_{min}}{x_{max} - x_{min}} \tag{2.9}$$

By doing this, all data points are scaled to take values between 0 and 1. If one has outliers, that is data points far from all other data points, these outliers will make the denominator much larger and the result would be that most data points would be distributed at a smaller interval in the 0-1 range.

2.4.2 Standardization

A solution to the problem mentioned above, that is, obtaining a very large denominator because of outliers, is to standardice the dataset. Standardization assumes the data points are distributed according to a gaussian with a certain mean and standard deviation, and after standardizing the dataset, each of the dimensions has a mean 0 and standard deviation of 1. After finding the vector of means for each of the dimensions μ and similarly the sigmas σ, each datapoint x is standardized as follows:

$$x_s = \frac{x_{old} - \mu}{\sigma} \tag{2.10}$$

2.4.3 Scaling kNN and Use of Weights

A new set of parameters one can introduce to kNN are weights that scale each of the features. Optimal or sub-optimal parameters can be found either by using knowledge about the data and brute force, or by using a search heuristic such as a genetic algorithm. An example of using a genetic algorithm to find weights for each feature is presented in subsection 2.7.1 and this is also done in the Weighted Grouping of Features system.

For some of the following experiments we have created another dataset. Because this dataset has three features, we have simply named it the 3f dataset. The 3f-set is shown in figure 2.6:

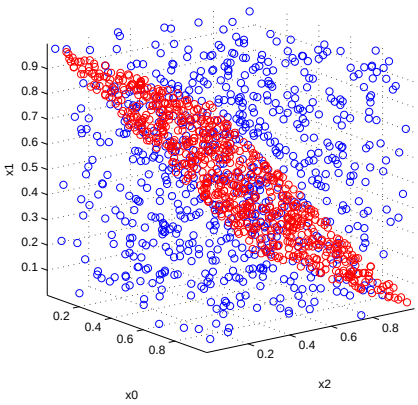

Figure 2.6: The 3f set. One class (blue) is randomly distributed uniformly over three dimensions, the other class (red) is randomly distributed in one dimension (x_3), but two of the dimensions are dependent as one is equal to one minus the other ($x_1 = 1 - x_2$)

Note that one dimension, dimension $x3$, is uncorrelated to both of the two classes. Using a brute force technique and kNN, we did an experiment where we adjusted the weights of the different dimensions. Each dimension was tested with weights (0, 0.33, 0.99) and the results obtained are shown in 2.3:

$Weight_1$	$Weight_2$	$Weight_3$	Result
0	0	0	50
0	0,33	0	49,88
0	0,66	0	49,88
0	0,99	0	49,88
0,33	0	0	49,86
0,33	0,33	0	*95,89*
0,33	0,66	0	94,93
0,33	0,99	0	94,03
0,66	0	0	49,86
0,66	0,33	0	94,77
0,66	0,66	0	*95,89*
0,66	0,99	0	95,41
0,99	0	0	49,86
0,99	0,33	0	93,94
0,99	0,66	0	95,34
0,99	0,99	0	*95,89*

$Weight_1$	$Weight_2$	$Weight_3$	Result
0	0	0,33	50,15
0	0,33	0,33	50
0	0,66	0,33	49,96
0	0,99	0,33	50
0,33	0	0,33	50,1
0,33	0,33	0,33	86,63
0,33	0,66	0,33	87,35
0,33	0,99	0,33	86,93
0,66	0	0,33	50,13
0,66	0,33	0,33	87,49
0,66	0,66	0,33	89,28
0,66	0,99	0,33	89,88
0,99	0	0,33	50,15
0,99	0,33	0,33	87,03
0,99	0,66	0,33	89,98
0,99	0,99	0,33	90,7

Table 2.3: Varying weights for dimensions x, y and z. Weighing the the non-informative z-dimension gave inferior results, none of which exceeded 90.00 percent.

We can observe that the results are best when each of the two dimensions containing information are equally large and the non-informative dimension has zero contribution. The latter has an obvious explanation [7], and if either of the two informative dimensions were to be given a higher weight than the other, some information from the other would be lost as one dimension alone cannot provide a classifier. The results are therefore as expected, and introducing weights improved the classification rate from 86.63 percent to 95.89 percent, an increase of 9.26 percent.

2.5 Dimension Reduction

This section aims to gain insight in regards to selecting features and extracting features and how this can improve classification rates. Feature selection is simply

[7] A non-informative dimension can only confuse the classifier.

to select a subset of the original set of features, whereas feature extraction transforms the original set of features into a new set of features of lower dimensionality. As there exist a large amount of methods used to extract and select features, we have chosen to focus on the basics and only investigate further on one particular method: Principal Component Analysis (PCA). Some of the insights gained from this section are used to enhance classification rates in the GoF-system.

K-Nearest-Neighbor's first step is to store the training examples that are to be used. This step is in itself trivial, but sometimes some of the features are plain noise and cannot be used in order to improve classification rates. Feature selection can then be performed in order to:

1. Reduce computational complexity

2. Improve classification rates by removing irrelevant and/or noisy features

kNN is suffering from the "'Curse of Dimensionality"'. This is partly because kNN is computationally slow in high-dimensional spaces, and partly because the performance tends to decrease drastically. The latter is because two samples from the same class often seldom are similar for most features when the dimensionality is high (Beyer et al., 1999). High dimensional similarity is discussed somewhat further in subsection 2.6, but in order to understand the effects of dimensions that do and do not contain information, two datasets are created, namely those in figures 2.3 and 2.6. Throughout this section the effects of removing features and editing features are shown using these two sets.

2.5.1 Manual Feature Selection

This section and section 2.6 is largely based on experiments performed on the square-in-square dataset.

Using regular kNN, the square-in-square dataset achieves a test-set precision rate of 84.43 percent with k=3. Note that somewhat better results were achieved with higher k's when we experimented with varying k's. The theoretical maximum for the square in square dataset is, - as mentioned in subsubsection 2.3.3.1, - 87.75 percent. This is confirmed by monte carlo testing where decision lines are formed around $0.25 <= x_i <= 0.75$. Because both dimensions in this set carry a similar amount of information and randomness, removing one dimension means that the theoretical limit would be reduced to 75%, a decrease of 12.75 percent. The dataset shown in figure 2.7 is the same dataset as the one in 2.3, but with dimension removed:

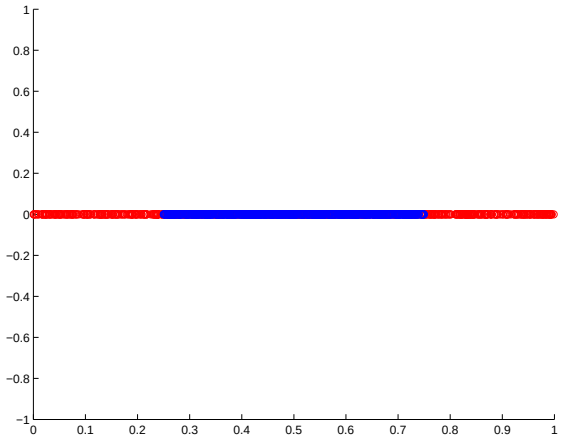

Figure 2.7: The Square-in-square dataset with one dimension reduced. Scatter of the same dataset as in figure 2.3 with one dimension removed.)

Classifying this dataset with kNN achieves a 72.07 percent accuracy, a decrease of 12.36 percent. The two dimensions carry synergies: If this was not the case, $(84.4 - 50)/2$ percent=17.2 percent accuracy would have been lost as this is the accuracy each dimension then would have had to carry. This synergy means that there exist cases where each of the one dimensions alone would have provided false

classifications, but where the two dimensions together correct this. This synergy is further confirmed by the theoretical limits that confirm that the actual synergy is of (87.75)/2 percent - 75/2 percent = 12.75 percent accuracy.

The 3f-set that was shown in figure 2.6 , showed a dataset where one dimension introduces nothing but noise. For this dataset it is therefore expected to achieve similar or better classification rates if the mentioned dimension is removed. The blue class in figure 2.6 is randomly distributed over three dimensions, and the red one is random at one dimension (x_3), but two of the dimensions are dependent as one is equal to one minus the other ($x_1 = 1 - x_2$). This dataset achieves a precision rate of 86.63 using kNN. Because one dimension contains no useful information however, and therefore only introduces noise, removing this dimension increases the precision rate. Figure 2.8 shows the dataset with x_3 removed:

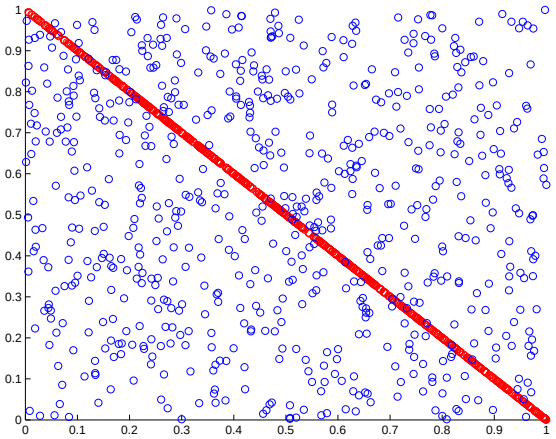

Figure 2.8: 3f reduced. Scatter of the same dataset as figure 2.6, but with dimension x_3 removed.

After removing the dimension, kNN now classifies the dataset 95.93 percent correctly. This improvement can be explained as follows:

- The removed dimension had no correlation to the classes, and could thus not

improve the accuracy.

- The removed dimension introduced information that was plain noise, $\frac{1}{3}$ of the information used by the classifier. This information could for some of the members:

    1. Increase the distance of one member of the class from all other members.

    2. Decrease the relative distance between one member from one class to members of the other class.

In the next section the accuracy on this dataset will be improved further.

Table 2.4: Removing features manually

	3f	Squares
All features	87.10	86.63
With removed feature	95.93	72.07

2.5.2 Automatic Dimension Reduction

In the previous subsection(section 2.5.1), features were removed manually because we knew something about the different dimensions. There are, however, several ways of reducing the dimensionality of a dataset automatically. In this part three ways of doing this will be mentioned:

1. Using a search heuristic to find which features that are best ignored.

2. Linear transformations of the data from a higher to a lower space (linear feature extraction).

3. Non-linear transformations of the data from a higher to a lower space (non-linear feature extraction).

The first listed strategy is an example of a *wrapper*-strategy as it uses a classifier to evaluate how well one set of parameters perform. This, or a generalization of this as discussed further in the 2.6-section, can be accomplished using a genetic algorithm. Section 2.7.1 shows an example of how this can be done. Using a genetic algorithm to accomplish this is also the strategy used by the Weighted Grouping of Features system (3.3). Strategy two and three are both *filter*-methods as they only use information that lies within the data. Principal Component Analysis (PCA) is a popular variant of the second listed strategy, and the rest of this subsection will take a glance at this method. For listed strategy number four, methods typically use kernel methods. This means that the data is mapped to higher dimensions before transformations similar to those of PCA are done in order to choose components that contain as much information as possible. One popular such method is Kernel Principal Component Analysis (Schölkopf and Smola, 1998).

2.5.2.1 Principal Component Analysis

Principal Component Analysis(PCA) was introduced in Pearson (1901). PCA maps the data to a lower dimension using an orthogonal transformation where the first dimension that maximizes the variance of the data. The new space is called *PCA-space* and is best shown using a figure, see figure 2.9.

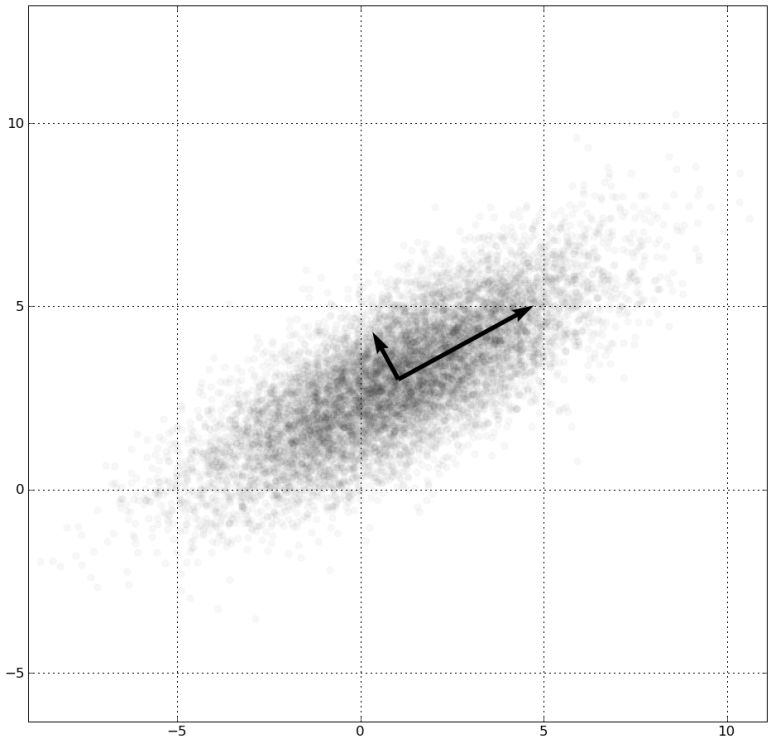

Figure 2.9: PCA-converted data from FrantzDale (2012)

In the figure the first principal component, pc_1, is the longest vector-arrow and we see that it covers the the largest variance in the data. The short vector-arrow, pc_2, is orthogonal to the first, and covers as much variance as possible given that it should be orthogonal to pca_1. As such PCA can be used to 1: transform the data into a new space where the underlying structure of the data is better explained, and 2: thereafter reduce the dimensionality by removing the least informative components. pc_1 is the most informative component, pc_2 is the second most informative and so on.

The reason why using PCA often works well is that it:

1. Transforms the data (linearly) into a space where the variance is maximized
2. Effectively finds the dominating components that will preserve as much information as possible

The following steps are performed in order to do a PCA transformation:

- The original data is normalized, that is, the mean of all datapoints is subtracted from each datapoint.
- The covariance matrix is found. This matrix shows covariance between different features.
- The eigenvalues and eigenvectors of this covariance matrix are calculated. These are necessary to do orthagonal transformations.
- The eigenvector with the largest adjoining eigenvalue is chosen as pc_1, and a number k of principal components less or equal to the number of features in the original data are chosen as dominant components, that is, $pc_1...pc_k$.
- The normalized original data is multiplied with the transpose of of the dominant components.
- The result is the original data tranformed into PCA-space.

Transforming the 3f-set Into PCA-space In order to perform PCA on data, we chose to use the Jama-package (Hicklin et al., 2005) for matrix-manipulations. Code from an example written by Gabe Johnson (Johnson, 2011) was also used. The 3f-set proved to be an interesting dataset to show both PCA's strengths and weaknesses:

Because of PCA's popularity and its guarantee of minimal information-loss, the first results on the 3f-set were somewhat discouraging. The data which we had hoped would neatly align the red class horizontally in two dimensions and be easy to classify, looked like figure 2.10:

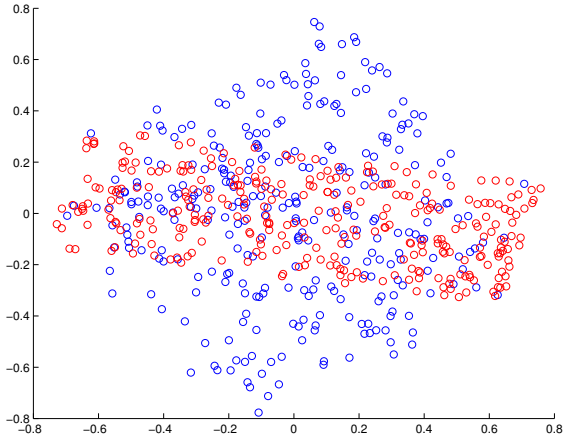

Figure 2.10: PCA3f reduced. Scatter of the same dataset as figure 2.6 after having transformed the data into PCA-space. Showing the two most dominant components.

As PCA is an unsupervised learning method and knows nothing about the classes in the data, PCA did not transform the red class [8] to align with the pca_1-axis. If the 3f-data had been transformed such that the red class was parallel to the x-axis, most of the information about the difference between the data in the x_2- and the x_3-axis would have been lost. Classifying this dataset using kNN yielded a

[8] The class where x_1=1-x_2 in the 3f-set

result of 62.2 percent, an $87.1 percent - 62.2 percent = 24.9 percent$ decrease. Doing a PCA-transformation to 3 dimension resulted in a classification rate by kNN of 78.6 percent, a decrease of 8.5 percent.

Removing the x_3-Axis Before the PCA-transformation If the x_3-axis was to be removed on beforehand however, there would be a great information gain when doing the PCA-transformation. Indeed, in 2.11 the variance of the red class is spread along the pca_1 axis and the two remaining edges of the square of the blue class along the other (pca_2) axis.

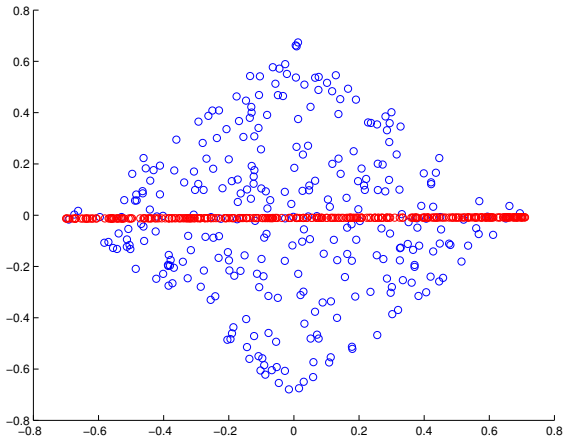

Figure 2.11: The 3f-set (2.6) with x_3 removed manually after having removed x_3 and then transformed the data into PCA-space with two dimensions.

This yielded far better classification-results at 95.0 percent correctly classified data points, results similar to those when the x_3 class was manually removed. However, it is first when two dimensions are removed PCA shows its real strength: Removing whichever two dimensions on the non-PCA-transformed 3f-set yields a maximum of 51.2 percent when classifying with kNN. The reason kNN barely performs better than random choice, is because the data in the red class are not

along one dimension and thus not meaningfully close to each other in one dimension. This is where PCA proves very useful on this dataset. PCA translates the data into what is seen in figure 2.12 and classifies the data 100% correctly:

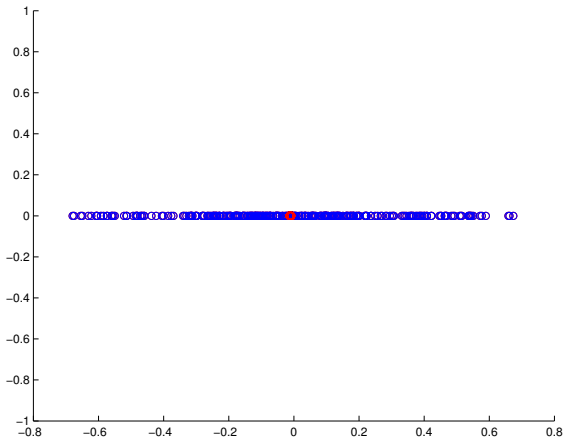

Figure 2.12: The 3f-set (2.6) with x_3 removed manually transformed into PCA-space with one dimension.

A table showing the results of removing features is shown below:

Table 2.5: Removing features automatically and manually. PCA3f is the same dataset in PCA-space, PCA3f-m is the 3f-set where one feature is removed manually before transforming the data into PCA-space. Showing with zero, one and two dimensions removed.

# of dim. removed	3f	PCA3f	PCA3f-m
0	87.1	78.6	
1	95.9	62.2	95.0
2	51.2	57.4	1.00

All possibilities and the complete usefulness of PCA have by no means shown by the example above. One could argue for example, that all of the work done in order to use PCA could have been exchanged by a much simpler tool, namely using

the rotation-matrix with $\theta = -45$ and then removing two features manually:

$$R = \begin{pmatrix} cos\theta & -sin\theta \\ sin\theta & cos\theta \end{pmatrix} \tag{2.11}$$

This would have been possible, but PCA can do this for all angles and will find a optimal angle if the data is not placed exactly along a line. The possibilities of PCA do not stop at this either, but covering PCA fully is beyond this report. Some important aspects we could learn from the 3f-example however, were that:

- PCA guarantees minimal loss of information.
- Although PCA guarantees this minimal loss of information, PCA is not necessarily optimal for classification as it is unsupervised and does not guarantee minimal loss of information in regards to differentiating classes.
- PCA can be useful when optimizing datasets before classification.

Before diving into a discussion about distance functions (section 2.6), it should be mentioned that there do exist successful attempts at creating supervised PCA as well (Chen et al., 2008; Bair et al., 2004).

2.6 Distance Functions

How distances between objects (or instances/samples) are calculated, is one of the major design choices that has to be taken when kNN is implemented. The first implementation of the Grouping of Features-system is somewhat special in the sense that it uses a different distance-function when comparing itself to the different classes. This will be further discussed in chapter 3.1.4. An understanding of what distance-functions are and especially an understanding of the Minkowski distance function, is important in order to understand the concept of power-average, also described in section 3.1.4. The most common distance-function $d(x, y)$ when x and y are vectors of real numbers, is the Euclidean distance:

$$d_E(x, y) = \sum_{i=1}^{n} \sqrt{(x_i - y_i)^2} \tag{2.12}$$

One kind of distance functions are metrics. They upholds the following intuitive criterias for distances (Cunningham, 2007).

1. $d(x, y) >= 0$; non-negativity
2. $d(x, y) = 0$ only if $x = y$; identity
3. $d(x, y) = d(y, x)$; symmetry
4. $d(x, z) >= d(x, y) + d(y, z)$; triangle inequality

As such the Euclidean distance is a metric, as is the more general Minkowski distance.

2.6.1 The Minkowski Distance Function

A generalization of the Euclidean distance is the Minkowski distance:

$$d_p(x, y) = \sqrt[1/p]{(x - y)^p}, p > 0 \tag{2.13}$$

This latter equation is the basis for the similarity function used in the Grouping of Features system. Using p=2 yields the Euclidean distance function and using power p=1 yields Manhattan distance. For high dimensions, (Aggarwal et al., 2001) found that using "'the Manhattan distance metric [was] consistently more preferable than the Euclidean distance metric [...] for high dimensional data mining applications. The results in table 2.6 from Aggarwal et al. (2001) show how using *fractions* as p's, that is, using $0 > p < 1$ consistently obtained higher test scores on high dimensional data.

Data Set	$L_{0.1}$	$L_{0.5}$	L_1	L_2	L_4	L_{10}	L_∞	Random
Machine	522	474	449	402	364	353	341	153
Musk	998	893	683	405	301	272	163	140
Breast Cancer (wdbc)	5299	5268	5196	5052	4661	4172	4032	3021
Segmentation	1423	1471	1377	1210	1103	1031	300	323
Ionosphere	2954	3002	2839	2430	2062	1836	1769	1884

Table 2.6: From Aggarwal et al. (2001): Results of using different p's in the Minkowski distance-equation on high dimensional data. L_p is the Minkowski distance-function of power p. The datasets are all datasets from the UCI Machine Learning Repository.

As none of the datasets presented in this chapter are of higher dimension, we instead try to vary p on a lower-dimension dataset. In the table in figure 2.6.1 are results of varying p on the square-in-square dataset. We see that the performance is best with p a little larger than 1.

From the figure in figure 2.13, we can see that lower p's allow one feature to have a larger distance as long as the other are close. Close means near the center of the figure, as distances are measured from origo in this figure. In fact, for $p < 1$, the unit circle becomes an asterix, that is, the convex sides become concave. The smaller the p, the further away one feature can be from the sample point without adding to the distance as long as the other features are close. In the square-in-square set, it makes sense that reasonably low p's perform better: When close to the optimal decision line a higher p would create a larger possible area on the other side of the line where a random higher concentration of of data points could be. If the p was very small however, the information from one dimension would be lost, requiring only one feature to be close to the sample point.

To sum up, the Minkowski-distance for low p's produces a low resulting distance as long as at least one feature is close to the sample point. On the other hand, when p is large, a low distance requires all of the features to be close to the sample point's features.

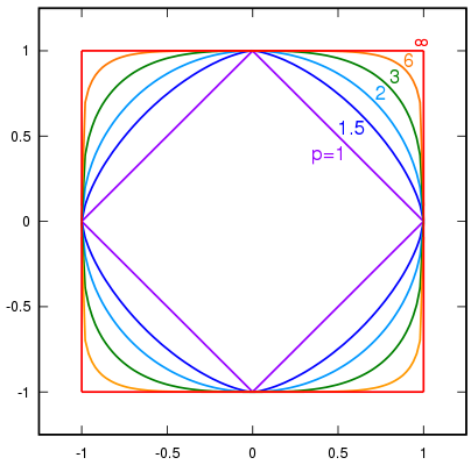

Figure 2.13: From Quartl (2011): The unit circle of different p's, $1 <= p < \infty$ (left) and table of results from varying the p in the Minkowski distance function of kNN on the square-in-square dataset (right).

p	Result
0.1	79.05
0.4	83.50
0.7	83.89
1.0	84.84
1.18	*84.96*
1.3	84.93
1.6	84.82

When p approaches infinity, the Minkowski distance function becomes the Chebyshev-distance function: For vectors of features $\mathbf{x}$ and $\mathbf{y}$, the Chebyshev-distance between those are:

$$d_C(x, y) = \max_i |x_i - y_i| \tag{2.14}$$

In other words the feature which has the largest distance will dominate.

These properties of the Minkowski distance function are important to the concept of power-average, which again is an important part of the GoF-system.

2.7 Genetic Algorithm

A Genetic Algorithm(GA) is a search algorithm that models natural evolution. In natural evolution individuals with high fitness survive and reproduce. In this way their genes are passed on and, informally, good genes are passed on to the next generation. In the next generation the set of genes are somewhat modified either by reproduction where the new chromosome is a mixture of genes from the two parents, or by mutation. In nature mutation is caused by radiation, chemicals or during cell-replication where mistakes sometimes are made when a cell copies its DNA-sequence before the cell-division(Center, 1969)

The genetic algorithm was chosen as search heuristic because it is robust and likely to find good parameters i large and complex search spaces (Wright and Ali Alajmi, 2005).

The idea of using evolution as a model in order to find optimal solutions to problems, was presented in the early 70s by John Holland (Holland, 1975) and Ingo Rechenberg (Rechenberg, 1971) who begun two paradigms which eventually have somewhat converged as many use blends of the two. Ingo Rechenberg introduced *Evolution Strategies* and John Holland's work introduced the *Genetic Algorithm*. Some explicitly combine the two like Cordon and Herrera (1996) and others like Wu et al. (2007) implicitly makes hybrid systems by incorporating elements from one algorithm in the other. An in-depth analysis between the two is beyond the scope of this report, but some differences include:

- ES selects parents at random and chooses the fittest children for the next generation, whereas GA is more likely to select fitter parents to be used for reproduction, that is, the parents are selected according to a probability distribution where fitter parents are more likely to be picked.
- ES uses vectors of real values to represent chromosomes, whereas genetic algorithms uses vectors of integers.

- GA mutates by flipping bits or using different parts of the cromosomes of its parents to represent the child. ES encodes mutation-schemes in its chromosome.

The remainder of this report will focus on the genetic algorithm in its purest form as this is what is used in the GoF-system 3. A genetic algorithm is a search heuristic with optimization as its goal. The degree of optimality, or fitness, is defined by some custom fitness function. In order to find better chromosomes [9], many evolutions are performed, where new generations are formed in every evolution by doing reproductions and mutations.

Some terminology is useful:

- Fitness-function: A custom-made function used to evaluate the fitness for each of the individuals. If the task is classification it may for example use the parameters from the individual on a classification task and return the number of correctly classified samples.
- Fitness: A value used to compare optimality between individuals. This is what the fitness-function returns.
- Individual: One set of parameters.
- Chromosome: The vector of genes that represents an individual.
- Gene: A part of a chromosome.
- Allele: A set of genes that represent one trait of an individual. Might be thought of as a parameter in our case.
- Generation: A set of individuals that exist at the same time.
- To evolve: Using the genetic operators so that a generation n becomes generation $n + 1$.

[9] In our case this would mean parameters that produce better classification rates

- Genetic operator: Operators that alters the individuals in the population, that is, crossover and mutation.
- Crossover: "'Reproduction"', a new individual is created by combining genes from two or more from the previous generation.
- Mutation: By some probability some random change a individual is altered by randomly altering some of the genes of the individual.

A genetic algorithm's flow is to first set up the chromosome and default parameters, then run evolutions, each evolution hopefully improving the next generation of individuals. After a certain amount of evolutions, the individual that obtained the highest fitness is returned. This individual can then be tested against a held-out test-set. The genetic algorithm is explained somewhat further in chapter 3, but before that the presentation of a genetic algorithm-example in 2.7.1 aims to provide a basic understanding of how GA's kan be used to improve the performance of the kNN-algorithm.

2.7.1 Genetic Algorithm Example

Using a Genetic Algorithms(GA)/Evolution Strategy(ES)-hybrid as search heuristic in order to improve the kNN-algorithm in some way has been done by several, and a straightforward method that successfully found optimal weights for each of the features in the kNN algorithm, is done by He et al. (1999).

2.7.1.1 Chromosome Description

The chromosome used by He et al. (1999) is a vector of real values, each value representing the weight for its adjoining feature. Offspring were created in a steady state-manner, that is, they kept the population stable in both size and in regards to variation by only having two individuals reproduce exactly two offsprings. The two individuals where selected randomly, but with a higher probability for individuals with higher fitness.

2.7.1.2 Crossover

With n being the number of genes in a chromosome, a number $n1$ was chosen at random, and the first $n1$ genes were chosen from the first individual and the $n-n1$ from the other in order to do crossover. After these offspring were created mutation was performed.

2.7.1.3 Mutation

The mutation was small, subtractions or additions of a value v that were performed on each feature with a certain probability. The value v was chosen from a normal probability distribution.

2.7.1.4 Fitness-function

As fitness function they used the inverse of the number of mis-classified examples plus a regularization-term. The regularization-term was necessary in order to prevent inflation of the weights.

This system effectively found near-optimal weights for the features and obtained results better than the ones achieved by other approaches He et al. (1999) compared themselves to.

2.8 Other Methods and What This Report Does Not Cover

2.8.1 Other Classifiers

There are a very large number of classifiers and no one classifier is considered the best one for all problems. Classifiers like Support Vector Machines (SVM), Random Forests, classifiers using neural networks and Naive Bayes may perform better at some tasks than kNN will[10]. In the GoF-system presented in the next chapter however, none of the above could easily be used by the system as elegantly as kNN

[10]Note that this is not always is the case; Kuramochi (2001) for example obtained better results using kNN than SVM on some problems.

could. Because of this, other classification methods are considered out of the scope of this report.

2.8.2 Reducing the Expense of Classifying a Sample

The expense of classifying a sample is one of the main drawbacks of the kNN algorithm. Oftentimes approximative algoritms are used and only representative samples from the training set are used during classification. There exist many such methods, but the only complexity-reducing method used in the GoF-system is dimensionality reduction.

2.8.3 Other Dimensionality Reduction Techniques

A representative set of methods one can use in order to perform dimensionality reduction has been presented in this chapter: Removing features manually, removing features using a genetic algorithm and extracting features using PCA. These are all related to the GoF system. There also exist other dimensionality reduction techniques that are not as closely related to the GoF-system, some of which have been mentioned briefly in this chapter. There also exist methods that may enhance the nature of the data from the classifiers viewpoint that does not reduce dimensions. GoF in a way does this by itself by grouping features. Other methods include Independent Component Analysis (ICA) which is a generalization of PCA. ICA does not find uncorrelated components, but instead finds "'independent"' components, that is, components that among other things has minimal mutual information. Doing this or using other methods to alter the data before classification using the GoF-system would be interesting. This, however, is also outside the scope of this report.

3

GROUPING OF FEATURES

3.0.4 Context

When we started working on this thesis project, Verdande Technology through Sigve Hovda had implemented most of the basic GoF-system and was doing research in regards to the theory on which it is based. Our contribution to the implementation of this system specifically, has been to:

- refactor and enhance code
- implement modules such as different classification rules
- write unit and integration tests
- replace units such as the chromosome and fitness-function used in the genetic algorithm
- experiment with other extensions of the system
- prepare and test datasets on the system

In order to prevent this thesis from being too lengthy, only the most interesting modifications and extensions are included in this report. The most mentionable extension we created is the Weighted Grouping of Features (WGoF) system. In the

previous chapter it was shown that weighing the different features in a dataset could improve classification rates significantly. The WGoF system extends the genetic algorithm to find optimal weights for each of the features within their group. How this was done will be described in its own section, section 3.3.

3.0.5 The Structure of this Chapter

This chapter is divided into three major parts. The first part presents the GoF-specific theory and compares this to the theory of the previous chapter. The second part gives a walk-through of the implementation of the GoF-system and provides an example of how distances are calculated in this system. Lastly the third part explains which changes that were done to the system in order to extend it to use weights.

3.1 GoF-Specific Theory

The previous chapter provided a theoretical backbone that enables one to understand how classification systems such as the GoF-system can work and how classification accuracy can be improved by using different techniques. This first section of the current chapter presents the theory that is specific to the GoF-system, and compares some of this to the theory presented in the previous chapter. The section starts off with a discussion of the classification rule the GoF-system uses and how and why the GoF-system uses class-specific distance functions when calculating distances.

3.1.1 The Classification Rule of the GoF System

In the previous chapter we presented the following posterior probability estimate for class y_i given as sample x:

$$P(Y = y_i|x) = \frac{\hat{F}(x|Y = y_i)P(y_i)}{\hat{F}(x)} \tag{3.1}$$

$$= \frac{k_i}{k} \tag{3.2}$$

As the denominator would be equal for every class, the predicted class y_i with this probability estimate would be the one with the largest adjoining k_i. The probability estimate was derived from using the following two equations:

$$\hat{F}(x) = \frac{k}{nV_d r^d} \tag{3.3}$$

$$\hat{F}(x|Y = y_i) = \frac{k_i}{n_i V_d r^d} \tag{3.4}$$

The GoF system, however, uses different distance functions when calculating distances for the different classes. As the V_d is the sphere in which the k nearest neighbours lie, each class therefore has its own volume sphere V_{d_i}. This will be further explained in section 3.1.2. For now, note that with different V_{d_i}s for different classes, the two previous equations have to be combined differently than what was done in 3.2. The most concise formula for $\hat{F}(x|y_i)$ is thus found in this way:

$$\begin{aligned}
\hat{F}(x|y_i) &= P(x|y_i)P(y_i) && (3.5)\\
&= \frac{k_i}{n_i V_{d_i} r_i^d}\frac{n_i}{n} && (3.6)\\
&= \frac{k_i}{V_{d_i} r_i^d}\frac{1}{n} && (3.7)\\
&= \frac{k_i}{nV_{d_i} r_i^d} && (3.8)
\end{aligned}$$

Here we have used that $p(x, y_i) = p(x|y)p(y)$, and combined this with equation 3.4. Prediction is now done by choosing the class which has the highest estimated probability:

$$\arg\max_i \frac{k_i}{nV_{d_i} r_i^d} \tag{3.9}$$

3.1.1.1 Different Sphere Volumes and the GoF System

With different volume spheres for the different classes, the correct way to calculate the probabilities would require one to calculate the V_{d_i}s and r_i for each of the classes and compare distances for each class according to the distance functions of

all classes. Ongoing research on how to do this efficiently is performed, but this research is in its early stages and will not be presented in this report.

Instead another classification rule is used and we expect that a genetic algorithm will adjust for the differences in volume spheres. The reasons for why this is expected will be presented in the next section.

3.1.1.2 The k-Mean Classification Rule[1]

In the classification rule used by the GoF system, the k nearest neighbors are chosen from each class according to the different classes' *own distance functions*. It is the *mean* distance from the k neighbors found from *each of the classes* that is used for prediction. The predicted class is the class whose mean distance, according to its own distance function, is smallest.

This might at first seem preposterous: With each class using different distance-functions, in what way can it be reasonable to compare these distances to each other? The answer is a combination of what is presented underneath, where the flexibility one has by the use of the genetic algorithm is essential:

1. As the V_{d_i}'s in this system might be combined to be an average of several V_{d_i}'s for other features further down the tree, the V_{d_i}'s are somewhat smoothed.

2. The classification rule we have chosen to use in the GoF-system, uses an average of distances - which again is an average of V_ds - within each class. The V_ds are in this way smoothed further.

3. Distances of the distance function are monoton increasing with increased distances of the features.

4. A genetic algorithm optimizes the combinations from #1 and #2 and is able to do this well because of #3.

[1] Not to be confused with the k-means clustering algorithm.

With monoton increasing distances also with the new distance-functions, the genetic algorithm is able to search well. As the genetic algorithm searches for distance functions which makes the classifier obtain as high accuracies as possible, the genetic algorithm will adjust the distance functions to find V_{d_i}s to this goal. #1 and #2 (smoothing the V_ds) makes this search space somewhat smaller, and further helps the genetic algorithm in its search.

We want to again emphasize the flexibility one gains by using a genetic algorithm: Although we believe the genetic algorithm is assisted by #1 and #2, we hypotethize that the genetic algorithm will find distance functions that performed well even if these were not present. An experiment that supports this was performed:

3.1.1.3 Using Majority Voting instead of K-Mean for Classification

Even though the majority voting rule used by regular kNN would not do any smoothing of the V_ds by combining several of them, we wanted to see wether or not the system could produce good classification results regardless.

When comparing the use of majority voting to using the k-mean rule in preliminary tests on the GoF-system[2] we observed the following pattern:

- More iterations (evolutions) were needed in order to converge to optimal solutions.
- The algorithm ran slower.
- The results were similar to those of the GoF-system with the K-Mean rule.

As using the mean of distances from each class is hypothesized as an aid to the genetic algorithm that enables it to search more efficient, we expected the number

[2]The GoF-system will be further in subsequent sections, the preliminary test here is included to show the robustness obtained by using a genetic algorithm.

of evolutions needed to be increased. The complexity was somewhat higher in the majority vote rule than in the k-mean rule, resulting in longer running time for each evolution. Lastly the final results were similar, which is expected to some extent as the search space is similar. We do, however, believe that k-mean rule might be less prone to getting stuck in sub-optimal local optimums which are not very good, and the majority voting rule had significantly higher running times.

For the remainder of this thesis we have therefore chosen to use the k-mean rule.

Now that we have presented the reasons and justifications for our choice of classification rules when using different distance functions for different classes, we will in the next section explain why we hypothesize that it might be useful to use different distance functions for different classes.

3.1.2 Using Different Distance Functions for Different Classes

In this section we will show that one Distance Function (DF) that is useful for predicting one class in a dataset, might be useless for predicting another. We will also show that a predictor might be able to perform well given very simple rules if one allows the distance functions for the different classes to be different.

Consider the following dataset "easySet" of objects $O_i = (f_1, f_2)$ where f_1 and f_2 are features:

$Class_A$:
$(f_1, f_2) : (3, 0)(2, 1)(1, 2)$.
$Class_B$:
$(f_1, f_2) : (3, 0)(2, 0)(1, 0)(1, 0)(1, 0)(0, 0)$.

Table 3.1: EasySet

An easy way of predicting Class A would be to give the predictor the following DF: "find the average of the features" and the following rule. "If 1.5, predict class

A."

This seems like a reasonable rule to use for predicting class A as it would get most samples correct[3]. Using an "average of"-DF for class B however, will not help predicting this class at all. The average of the features of the objects in class A is not a constant. A rule that would predict class B from the data above would be to use DF: "find f_2" and rule "if 0, predict class B".

Just as using class B's DF for classifying class A did not make sense, using class B's DF which looks at the value of f_2 will not produce good predictions for class A. This shows that very simple rules may be used to obtained good classification results when using simple, but different DF's for the different classes. This is precisely what is done withe the GoF-system: A genetic algorithm searches for different DFs to use for the different classes.

3.1.3 The Grouping of Features System

The Grouping of Features(GoF) classification system represents objects[4] as trees where a tree is made up of of groups of features as shown in figure 3.1:

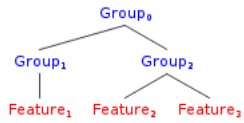

Figure 3.1: Grouping of Features tree. An object with three features.

A tree created by the GoF-system always consists of the Top_Group node. This node may have $Features$ [5] and $Sub-Groups$ as children in the tree. A $Sub-Group$ cannot have other $Sub-Groups$ as children. As such the tree can maximum have

[3]Note that the classes are not completely separable as both contain the sample (3,0), so one erroneous prediction is expected.

[4]Also known as as instances/samples.

[5]which are leaf nodes and represents features from the dataset.

three levels.

Similarity between an test-sample and another sample can be chosen to be measured at group level for the different groups, or not:

- If similarity is measured at a group's level, the power-average of the test-sample's group's children is calculated and the difference between this result and the result of the power-average of the sample-point becomes this node's value.
- If similarity is not measured at the group's level, the differences between the group's children of the two instances are calculated. The value(s) from this are then aggregated and becomes this group's value.

The resulting value at the top-node is the one that finally is used for comparison between classes. kNN is used for this, and lower values are shorter distances.

Several optimal or sub-optimal parameters for the system are found using a genetic algorithm. These parameters determine the structure of the trees found, which parameters to group together and how the calculations are done. Effectively these parameters determine the resulting distance functions used by the different classes. More on the distance functions will be presented after a brief discussion of how the GoF-system fits into the IBL-field.

3.1.3.1 Grouping of Features and Instance Based Learning (IBL)

As this thesis has had some focus on IBL-methods, it is natural to see how the GoF-system fits into this context.

The GoF-system uses a genetic algorithm to find trees that in effect work as distance functions that can be used when classifying the different classes. As such there is quite a lot of generalization done by the GoF system, and the GoF-system

is as such not an IBL-method itself. The GoF-system uses an IBL-method (kNN) as the basis of its fitness-function, and the distance-functions the GoF-system produces are to be used in IBL-methods.

As such GoF is a *supporting system* that may *enhance IBL-methods*.

The tests performed on the GoF- and the later introduced WGoF-systems, are done by classifying the datasets using a modified version of kNN where the majority voting rule has been replaced with the k-mean rule presented in 3.1.1.2 and where the Euclidean distance function replaced with distance functions found using the GoF- and WGoF-systems.

3.1.4 The Distance Function in the GoF-system and the concept of Power Averages

In this section and the sub-sequent sub-sections, the distance function used by the GoF-system will be presented from the functional side, the theoretical side and the technical side. In the end an actual equation representing the distance function used in the GoF system is presented.

The GoF-system allows for the following simple group-structures and mixtures between them:

1. The groups value is the average of the features
2. The groups value is the largest of the values of the group
3. The groups value is the lowest of the values of the group

These can be thought of as:

1. *mean*-groups
2. *or*-groups

3. *and*-groups

The function chosen which enables the system to create such structures is the *power-average*:

3.1.4.1 Power Average

The distance-function used in the GoF-system uses a concept called *power average*. This concept is similar to the Minkowski distance function presented in section 3.1.1.2. If one were to relax the Minkowski-functions limitation of having the exponent $p > 0$, the function would be:

$$\hat{pa}(p) = \sqrt[p]{\sum_{i=1}^{n} d_i^p} \tag{3.10}$$

This function has some interesting properties. Figure 3.2 shows a plot of function 3.10:

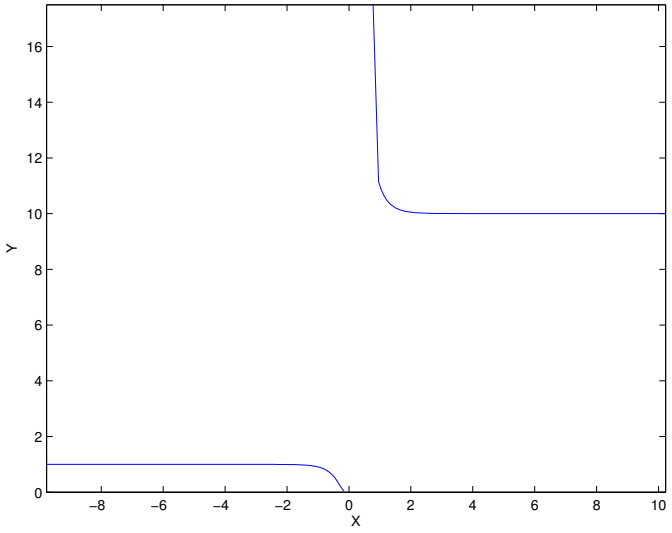

Figure 3.2: Power-average-function without geometric mean adjustment for distances d_1=1, d_2=10.

In the plot, two example-distances, d_0=1 and d_1=10. As p approaches infinity, $\hat{pa}(p)$ approaches the larger distance, d_1. As p approaches minus infinity, the function approaches the smaller distance d_2. This is always the case, also when the number of distances are larger: The larger value will dominate as the power grows and the smaller will dominate as the power decreases.

$\hat{pa}(p)$ is undefined for p=0, and as we can see from the graph, when zero is approached from the negative side, $\hat{pa}(p)$ approaches zero and as zero is approached from the positive side, it approaches infinity. In our case this is not wanted behavior, and following norms in the science community we have chosen to define $\hat{pa}(0)$ as the geometric mean as suggested by Yager (2001). The geometric mean is, for two distances d_1 and d_2, calculated as follows:

$$pa(0) = \sqrt{d_1 \times d_2} \tag{3.11}$$

The equation for the power-average $pa(\mathbf{d}, p)$ where $\mathbf{d}$ is a vector of distances and p is an exponent therefore:

$$\hat{pa}(\mathbf{d}, p) = \sqrt[p]{\sum_{i=1}^{n} d_i^p},\ pa(\mathbf{d}, 0) = \sqrt[n]{\prod_{i=1}^{n} d_i} \tag{3.12}$$

With two distances the square root is used, for three the cube et cetera. In this way, for the example with distances 1 and 10, $pa(\{0, 10\}, 0) = \sqrt{0 + 10}$ 3.16, and the function is monotonly increasing in the range $[-\infty, \infty]$.

Another thing that can be observed in the figure, is that as the function approaches -3 and 3, it has already almost converged to either the smaller or the largest value. The genetic algorithm is given a vector of values it can choose as the power for the different groups. This vector of values uses a standard vector of possible powers with values mostly in the range [-3,3].

Using power-average-groups one could classify the dataset presented in the previous section (dataset 3.1) each on their own easily: For class A an *average*-group

containing both features would work well. For class B a high-powered f_2-group and a low-powered f_1-group would work well.

3.1.4.2 The GoF-Distance Function

With the concepts of groups and the power-average presented, we can show the actual distance function used by kNN. Note that distances are measured bottom-up in the tree and that a sub-group has the option of either measuring similarity on group level or at feature level. If distances are measured at group level for a sub-group, the power-average is measured here. If not, the group just aggregates (sums) the distances calculated at feature-level. Similarly the top group can either measure similarity at its node (using the power-average), or just sum the distances of the sub-groups.

Given the following:

- $pa(G_i, p)$ is the power-average of θ_i for the set of features in group G_i.
- G_i, i=[1..n-1] represents groups that measures similarity on group level
- G_i, i=[n..m] represents groups that measures similarity on feature level
- f_j, i=[1..o] represents each of the original features

Then, in the case where similarity is not measured on group level for the top group, the distance function looks as follows:

$$d(x,y)_{sim=false} = \sum_{i=1}^{i=n-1} pa_x(G_i,p) - pa_y(G_i,p)) + \sum_{i=n}^{i=m}\sum_{i=1}^{i=o}(f_{i_x} - f_{i_y}) \quad (3.13)$$

If similarity *is* measured on group-level for the top group, the distance function becomes:

$$d(x,y)_{sim=true} = pa(d(x,y)_{sim=false}, p) \quad (3.14)$$

where θ_0 is the power-average-exponent used for the top-group.

3.2 The GoF System's Implementation

Throughout the upcoming section the reader is provided an understanding of how our implementation of the genetic algorithm is set up and works, and how the trees (distance functions) of the different classes are found.

We created a simple, high level class diagram that shows the main modules used by the GoF-system. In figure 3.3 we can see that the GoF-system, given an input and parameters to be used by the genetic algorithm, uses the GoF-genetic algorithm to produce an output. The GoF-genetic algorithm uses the GoF-Chromosome and the parameters provided by the Gof-system to find optimal parameters. Based on the chromosome, the genetic algorithm's fitness-function builds the necessary trees to calculate distances.

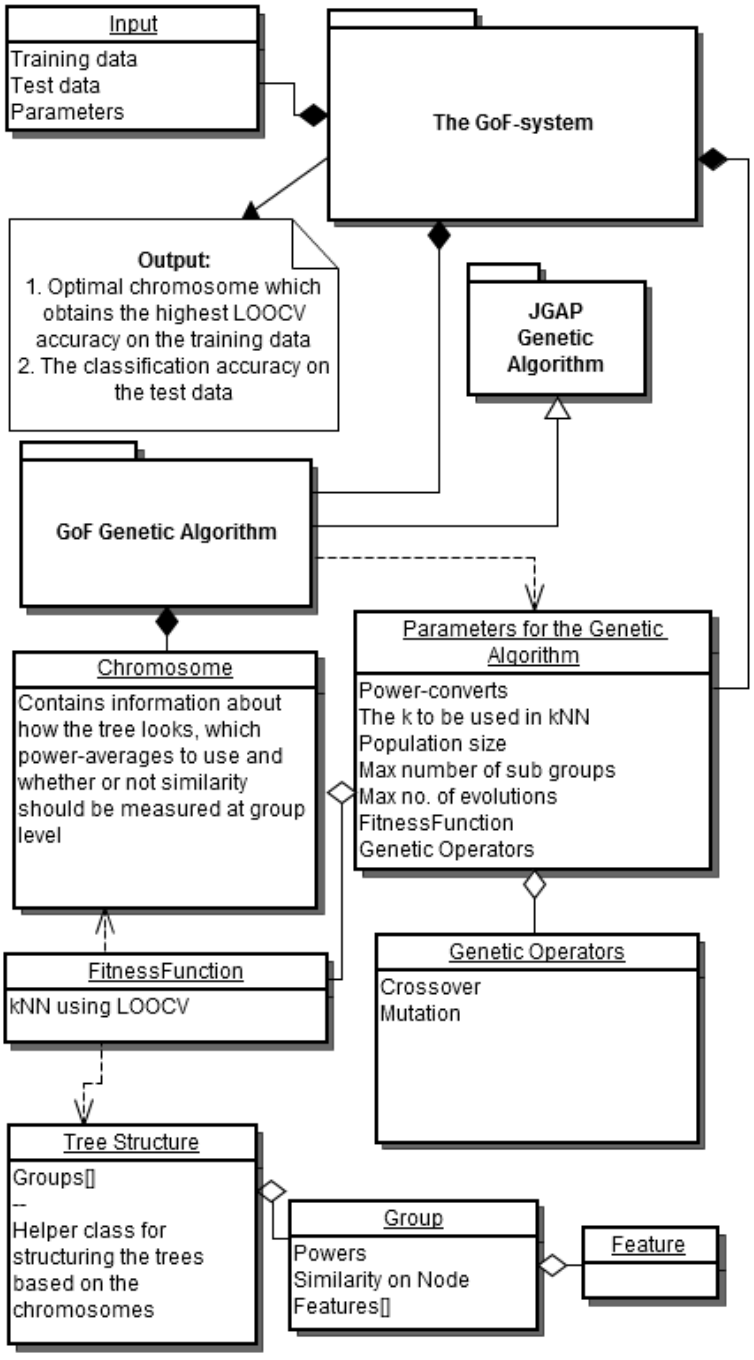

Figure 3.3: Showing the main parts of the GoF-system

3.2.1 Recap of the Genetic Algorithm

Before going into the details of the implementation, a quick recap of how a genetic algorithm works is called for. The genetic algorithm An evolution of a genetic algorithm is performed in the following way:

- In the beginning of an evolution, the least fit subjects are removed from the population.
- Genetic operators are performed on a subset of those subjects. These alter the subjects and in effect create new ones.
- When this is done, all subjects are evaluated using the fitness function chosen.

The genetic operators used, work in the following way:

3.2.2 Genetic Operators

Genetic Operator 1: Crossover

On 35 percent of the previous generation, crossover is performed. Crossover (reproduction) is performed in the following way:

- Two subjects are chosen randomly.
- The two chosen subjects are duplicated.
- Two duplicated subjects' genes are randomly swapped between each other.

Genetic Operator 2: Mutation

1/12th of all genes in the population are altered randomly (that is, a parameter is given a random value in the range accepted by that parameter).

3.2.3 Forming a new generation

After using the genetic operators, the fitness of each subject in the altered population is evaluated. In order to keep the population size constant the n fittest subjects are chosen to become the new generation.

3.2.4 Parameters used by the Genetic Algorithm

In addition to the genetic operators, the following parameters are used as parameters for the genetic algorithm the GoF-system uses unless otherwise stated:

- Fitness-function: Leave One Out Cross Validation(LOOCV) using kNN.
- The vector of power-converts: [-Infinity, -3, -2, -1, 0, 1, 2, 3, Infinity].
- The k to be used by the classification rule: 3.
- Max number of subgroups: 1
- Population size: 200
- The number of evolutions to be performed: 200

3.2.5 Chromosome

The chromosome used by the genetic algorithm in the basic GoF-system, uses the following structure:

- For each class in the dataset a tree is represented in the chromosome.
- Each tree in the chromosone consists of the following three parts
  - Group indicators for each feature.
  - Indicators whether distances are to be calculated the different groups' level.
  - Power-converts for each group used by the distance function.

The functions of the different parts of the tree are as follows:

- The first part of the tree are integers that tell the distance function which group each of the features belong to. This vector has hence length n where n is the total number of features.
- The second part are booleans that tell the distance function whether it should calculate distances at group level or at feature-level. This vector is of length m where m is the maximum number of subgroups. If there are only ones

in this set, this means that distances are to be calculated at feature level. If there is a zero in this set, the features belonging to the group this one is indicator for is considered a group and the power-average of the members is taken.

- The third part consists provide the indices in the power-vector where the powers of the groups are.

As an example, consider a tree that looks like this that can have maximum one subgroup:

$$[0\ 1\ 1][0][0\ 1]$$

In this case there are three features. One belongs to the topgroup and two belong to $Sub-Group_1$. From the second part of the chromosome we can see that the similarity for $Sub-Group_1$ should be measured on group level. The third part of the chromosome shows that $Sub-Group_1$ should use the power from the 0th index of the power-convert-vector, whereas the Top Group should use the power from the 1st index. In the next section is a walk-through of how the distance between two samples is calculated in the GoF-system.

3.2.6 Calculating a Distance, an Example

Using the example tree from 3.2.5 ([0 1 1][0][0 1]), and the power-convert-vector $[2, 1]$, the distance for the class the tree belongs to will be calculated in the following manner:

We name the three features in the first part of the chromosome [0,1,1] f_1, f_2 and f_3. Because f_1 has a group-indicator of zero it belongs to the $Top-Group$. The other two belong to $Sub-Group_1$. Looking at the second part of the chromosome ([0]) we see that the similarity for $Sub-Group_1$ should be calculating at group-level.

This means that the distance between the power-average of the training sample and the power-average of the test-sample should be measured in the following way:

$$d(x,y)_{Sub-Group_1} = | \sqrt[p_1]{\left(\frac{1}{n_1}\sum_{i=1}^{n_1} x_i\right)^{p_1}} - \sqrt[p_1]{\left(\frac{1}{n_1}\sum_{i=1}^{n_1} y_i\right)^{p_1}} | \tag{3.15}$$

Here $d(x,y)_{Sub-Group_1}$ is the value for $Sub-Group_1$, that is, the distance between sample x and sample y at this node. The p_1 is the power to be used for $Sub-Group_1$ and n_1 is the size of the group, that is, the amount of children this node has. The top-group is not to be calculated at this level. Instead the distance found in 3.15 is aggregated with the distance between the f_1 in the training- and the test-sample. The total distance is therefore calculated as in 3.16:

$$dist(x,y)_{TopGroup} = |f_{1_x} - f_{1_y}| + d(x,y)_{Sub-Group_1} \tag{3.16}$$

p_1 can be found in the power-convert-vector ([2 1]) by looking up the indexes provided by the third part of the chromosome. We see that p_1=1. The total distance is therefore:

$$\begin{aligned} dist(x,y) &= |f_{1_x} - f_{1_y}| + | \sqrt[p_1]{\left(\frac{1}{n_1}\sum_{i=1}^{n_1} x_i\right)^{p_1}} - \sqrt[p_1]{\left(\frac{1}{n_1}\sum_{i=1}^{n_1} y_i\right)^{p_1}} | \\ &= |f_{1_x} - f_{1_y}| + |\frac{1}{2}\sum_{i=1}^{2} x_i - \frac{1}{2}\sum_{i=1}^{2} y_i| \end{aligned}$$

The theoretical foundation for the Grouping of Features-system has thus been provided and so has a presentation of how the GoF-system works using a genetic algorithm. The next section shows one extension of the system; namely adding weights to the power-average function.

3.3 Weighted Grouping of Features

In the theoretical exploration of chapter 2, a major finding was the importance of the scaling of features in regards to classification rates. Scaling features could be

done by weighing the features, and this is part of what the WGoF-system does.

In order to make the system able to capture more complex structures, the power-average concept was extended to one one using normalized weights for each of the features and for each of the groups; *weighted power-averages*. By letting a genetic algorithm find the optimal weights for each feature in the different groups and the optimal contribution of each group, it was believed classification rates would be increased. Now not only would optimal tree-structures be found, but the contribution of each weight in each of the groups and the optimal contribution of each group would be optimized as well.

Each node in the tree, that is, both groups and features, have weights:

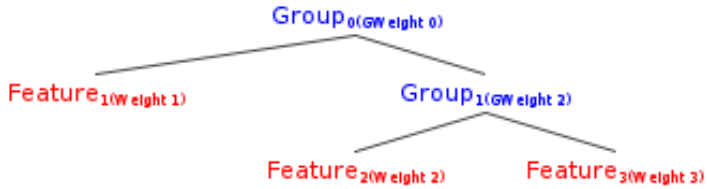

Figure 3.4: Tree that shows each node in the tree has its own weight.

3.3.1 Weighted Power Average

The logical change made without regards to the genetic algorithm was to change the power-average function used in the fitness-function,

$$pa(\mathbf{d}, p) = \sqrt[p]{\sum_{i=1}^{n} d_i^p}, \qquad pa(\mathbf{d}, 0) = \sqrt[n]{\prod_{i=1}^{n} d_i} \tag{3.17}$$

with the *weighted power-average function*:

$$wpa(\mathbf{d}, p) = \frac{1}{\sum_{i=1}^{n} w_i} \sqrt[p]{\sum_{i=1}^{n} w_i d_i^p}, \qquad wpa(\mathbf{d}, 0) = \sqrt[n]{\prod_{i=1}^{n} d_i}, \quad \sum_{i=1}^{n}(w_i) = 1 \tag{3.18}$$

Note that because the weights are normalized within the group, that is, $\sum_{i=1}^{n} w_i = 1$, the value of a group can now be considered a weighted average of its members. The scale used for each of the features are all in the range [0,1], and the sum of them is one. The limitation that any group's value is in the range $[f_min, f_max]$ where f_min is the feature with the lowest value in the group and f_max is the largest value in the group is therefore kept.

The final value of the $Top - Group$ is now a weighted power average of the sub-groups and a sub-group's value is a weighted power-average of its members.

3.3.2 Changes in Regards to the Genetic Algorithm

In order to have the genetic algorithm find the optimal weights, the following changes were made to the system:

- In the chromosome, one new gene was added for each feature. This gene represented the weight of a feature.
- Similarly one gene was added for each of the potential sub-groups.

  On the creation of a group when a new chromosome is created, the gene representing a feature, having a value between 0 and 1, is divided by the sum of weight-genes for this group. The weights are in this way normalized; the weights of the children of one group add to one.

  Similarly as in the case for the feature, the group-weights are normalized on the creation of the three, that is; a group's weight equals the weight it is given by the chromosome divided by the sum of the weights of the children of its parent.

3.3.3 Effect of adding weights in regards to performance

The search space grew quite a bit with the weight-extension. Each feature and each group now has a weight represented in the the trees representing each class

in the chromosome. Underneath is a comparison between the search space of the GoF- and the WGoF-system with a given dataset.

The search space in GoF if one has a dataset with two classes, ten features and parameter max number of sub-groups=3, would result in the following amount of genes in each chromosome:

$$\begin{aligned} \#ofGenes &= \#ofClasses \times \#genesPerClass && (3.19)\\ &= 2(\#ofFeatures + 2 \times \#ofSubGroups + 1) && (3.20)\\ &= 2(10 + 2 \times 3 + 1) && (3.21)\\ &= 27 && (3.22)\\ & && (3.23) \end{aligned}$$

In WGoF the search space is:

$$\begin{aligned} \#ofGenes &= \#ofClasses \times \#genesPerClass && (3.24)\\ &= 2(2 \times \#ofFeatures + 3 \times (\#ofSubGroups + 1)) && (3.25)\\ &= 2(2 \times 10 + 3 \times (3 + 1)) && (3.26)\\ &= 44 && (3.27)\\ & && (3.28) \end{aligned}$$

In this case the search space became approximately 1.63 times larger. As the trees found by the GoF-system also can be found by the WGoF-system [6], the WGoF-system should in theory never perform worse than the GoF-system given a large enough population and enough evolutions[7].

[6]The weighted power-average is the same as the power average if all weights are equal to 1.

[7]Note that the WGoF-system, as the GoF-system, may get stuck in a local maximum and therefore not find solutions that are as good as it would otherwise

However, because of the larger search space, more searching is needed, and when using the WGoF-system it is expected that larger population-sizes are needed. Complexity might therefore be a larger issue.

4

TESTS AND ANALYSIS

In this chapter, results from tests performed on the Grouping of Features(GoF)-system and the Weighted Grouping of Features (WGoF) are presented. A wide variety of tests have been performed, and only the more interesting test-results are shown here.

This chapter first describes which tests were performed, on which datasets they were performed, and how the tests were performed. After that results from the different datasets are provided with adjoining analysis of the results.

Before this however, a motivational retrospect on the dataset presented in the introduction of this report is provided. A hypothesis was made, namely that the simple structures of the 2f-set would be captured by the GoF-system and that the GoF-system therefore would obtain better classification rates than those of kNN.

The results from classifying this dataset with KNN and the GoF-system is presented in table 4.1:

When the project on making this report started out, the 2f-set was used in order to get an increased understanding of the field and the possibilities and challenges of the GoF-system. Obtaining a 2.5 percent classification improvement over kNN on this set is therefore encouraging. The reason for why this is the case is further

Classifier	Percent
kNN	76.33
GoF	78.93

Table 4.1: Results on the 2f-set using kNN and GoF. k=3, one sub-group in the GoF-system.

analyzed in its own section of this chapter.

4.1 Presentation of Tests and Datasets

4.1.1 Tests

There are three setups or systems that are tested on this chapter, namely:

1. The basic GoF-system
2. The Weighted GoF-system (WGoF)
3. The GoF-system used on PCA-transformed data

4.1.2 Datasets

The datasets tested on are the following five:

1. The 2f-set: A simple two-class set in two dimensions, also used in chapter 2.
2. The square-in-square-set: Another simple two class set in two dimensions, also used in 2.
3. The 3f-set: A quite simple two-class set in three dimensions, also used in chapter 2.
4. The UCI Wine-set: A three-class set in 13 dimensions that can be classified well, but usually needs some kind of transformation and scaling in order to obtain good results.

5. DigitsSmall: A subset of the UCI Digits-set. A nine-class set in 64 dimensions where each attribute (dimension) can take values in the range [1,2,..,15,16].

These datasets are all further described in their own sections.

4.1.3 Test-Schemes for Different Datasets

Because the datasets are of very different nature, different testing-schemes are used:

For the datasets for which an infinite number of data points are available (that is: 2f, 3f and square-in-square), a special 5-fold Cross Validation (CV) is performed: As evaluating a chromosome is much less computationally expensive than training one is, and as several training- and test-sets can be created, five training and test-sets are created. Each training set is then evaluated by all test sets and the mean of all these means are shown as the result. We name this the "mean-of-means"-result. For the kNN-benchmark, the same scheme using mean-of-means is used.

The DigitsSmall-set showed to need multiple restarts in order to obtain good parameters. The reason is discussed in the Digits-section 4.2.5. Because of this, a (one) training-set was chosen and restarted 50 times. The five trees that produced the highest fitness during these 50 runs were then tested by five test sets and the mean of means of these are the results shown. As the top five runs are the "top five" in regards to performance on the training data without having seen the test data, these are not over-fitted results. The same mean-of-means-scheme is used for the kNN-benchmark.

The UCI Wine-dataset is quite small and is because of this it is usually evaluated using Leave One Out Cross Validation (LOOCV). This is therefore done for both the GoF-system and for the kNN-benchmark. This system was also tested by a 70/30 splitted training-/test-dataset as an extra confirmation of the validity of the results.

All tests are done by classifying the test set using kNN with the metrics produced by the GoF- and WGoF-systems.

4.1.4 Determining Parameters

As each dataset is different, the parameters chosen reflect this and are different for each dataset. Hereby the standard parameters are the following:

- k: 3
- Number of sub-groups: 3
- Power-convert-vector: [-Infinity, -3, -2, -1, 0, 1, 2, 3, Infinity]
- Fitness function: LOOCV kNN with the kmean classification rule

The standard parameters above are chosen because of the following reasons: 1. $k = 3$ prevents ties when two classes. 2. The difference between 3 and ∞ as power is very small. 3. Even though enabling the system to use more than 3 sub-groups, these sub groups are more likely to be over fitted to the training data and will be difficult to understand. 4. The parameters have shown to be parameters that often yield good results without requiring large population sizes and/or many evolutions in preliminary tests.

There are a few more parameters, namely the population size, the number of evolutions performed and the number of restarts done. A hypothesis is that all of these should be increased when the complexity of the dataset grows. The hypothesis is grounded in the following: As the complexity of the dataset grows, there are more possible local maximas and increasing those three parameters increases the chances of finding a better maxima as more of the search space is evaluated. The obvious drawback of this is a possible over fitting to the training data because increasing all of these parameters might allow the model to be fitted to a very complex structure that is special for the training set.

4.1.5 Structure of Presentation

For each dataset the following structure is used to present the results:

1. Presentation of dataset

2. Results and Analysis

Onward to the actual tests:

4.2 Tests and Analysis

4.2.1 The 2f-set

4.2.1.1 Dataset description

The 2f-set (figure 4.1) consists of 2000 datapoints, 1000 belonging to each of the classes. One class is uniformly distributed in the range [0,1] in both directions, the other class is a gaussian of mean $y = 1 - x$ and variance 0.1.

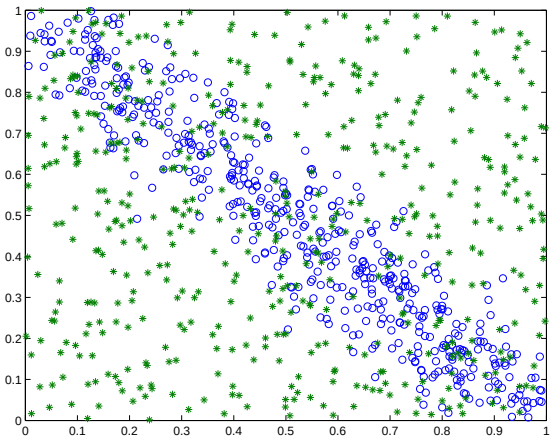

Figure 4.1: 2f-set

Classifier	Percent correct
kNN	76.33
GoF	78.93

Table 4.2: Results on the 2f-set using kNN and GoF. k=3, one sub-group in the GoF-system.

4.2.1.2 Results and Analysis

The Basic GoF-result in 4.2 resulted from the following trees:

Tree, uniform class:

TopGroup: Features [1], Power: 0, Group-similarity: False

Tree, class where $y = 1 - x$:

TopGroup: Features []

SubGroup: Features [0,1], Power: 1, Group-similarity: True

These trees obtained the highest fitness and were therefore chosen in order to classify the test-set. We can see that for the $y = 1 - x$-class, the manhattan-distance, $x + y$, is the metric used as the power for this group is $p = 1$. This is exactly as hypothesized as this means x and y are summed together before distances are calculating. Because the sum adds up to 1 with a standard deviation of 1/10th, the classification task thus becomes easy for the classifier. As for the uniform class, the power of -1 is not well understood, and most likely this exact power is due to over-fitting to the training set. The hypothesized best value for this parameter was $p = 0$ as the class is uniformly distributed with a mean 0.5.

4.2.2 The 3f-Set

The 3f-set has two classes and three dimensions. Each class has 1000 data-points. One of the dimensions is independent and uncorrelated to any of the classes. For the other dimensions one class (blue) is randomly distributed uniformly over both whereas the other class (red) has the dependance that $x_1 = 1 - x_2$:

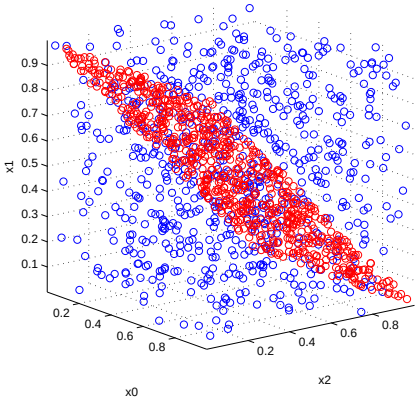

Figure 4.2: The 3f set.

4.2.3 Results and Analysis

In chapter 2 the 3f-set was first classified using regular kNN and obtained 86.63 percent correctly classified samples. Removing features manually obtained 95.9 percent accuracy and kNN on a PCA-reduced dataset obtained 62.20 percent accuracy. By removing one feature manually and then running PCA and choosing the first principal component, a 100% accuracy was obtained. 100% was achievable, but somewhat cumbersome. With the IBR-system 100% accuracy was obtained right away with no modifications to the dataset.

Classifier	Percent correct
kNN	86.63
kNN with removed feature	95.9
kNN on PCA transformed data (d=2)	62.20
kNN with removed feature on PCA data	100
Basic GoF	100

Table 4.3: Results on the 3f-set from the previous chapter and the results on the GoF-system.

This result is encouraging. The GoF-system created one group of the uncorre-

lated dimension and in effect ignored this one. In addition a and-group was made of the other two features and as the average of these in this set are exactly equal, the classification task then became trivial.

4.2.4 The Square-in-Square-Set

4.2.4.1 Dataset description

The square-in-square set has 2000 data points, 1000 in each of its two classes. One class is uniformly distributed in the range [0,1] in two dimensions, whereas the other is uniformly distributed in the range [0.25,0.74] in two dimensions.

4.2.4.2 Results and Analysis

Classifier	Percent correct
kNN	82.17
Basic GoF	84.38
WGoF	85.64
WGoF with 10 sub groups	84.21

Table 4.4: Results on square-in-square-set classified by kNN, GoF and WGoF.

The GoF-test obtained results between 83.82 and 84.64 percent with a mean of 84.38. The tree that performed the best looked as follows:

Tree, large square

Topgroup features [0,1] Power ∞, Group-similarity: True

Tree, small square:

Topgroup features [1], Power: 1, Group-similarity: True
- Subgroup 1 features [0], Power 1

The second tree varied quite a lot from each run and we could not find a pattern. The tree representing the large square, however, always used ∞-groups. In effect

the largest of the features of the features is found and this is quite understandable. Consider the following table:

-,+	avg,+	+,+
-,avg	avg,avg	+,avg
-,-	avg,-	+,-

Table 4.5: Results on square-in-square-set classified by kNN and on the GoF-system.

Here + means large value, − a small value and avg an average value. The +, − and avg represent the (x,y)-values a samples in the respective sub-areas have. The optimal decision lines are around the avg, avg-sub-area.

As we can see, for the larger square-class, five of the eight sub-areas have either its x- or its y-value as large(+. All of these sub-areas are considered the same IBR-system as it made an infinity-group yielding only the largest value. Therefore the boundaries around these are easy to classify correctly. The (−,−)-sub-area will also be classified correctly given nearby sample points.

Having this correct and classifying the remaining samples as the smaller class, the classification rate would be $(5/9 + 1)/2$ percent=83.34percent. Both kNN and the GoF-system performs a little better than this.

4.2.4.3 Weighted GoF-Results and Overfitting

The WGoF-test obtained higher scores, averaging one percentage better than GoF and kNN. As there are few structures to fit a model to, a hypothesis was that increasing the number of sub-groups on such a dataset would decrease the classification rate due to overfitting. Indeed, while the correctly classified on the training set increased from 97.67 percent to 99.50 percent, the test-set performance decreased to 84.21 percent.

The WGoF-test obtained higher fitness on the training data (99.75 percent correct), but lower results on the test-data with an average of 84.21. The difference from the GoF-system quite small, but the reason for WGoF performing worse is probably due to overfitting: With not very many structures to fit to and several parameters used to fit to the data, the WGoF-system probably fitted a model that captured

4.2.5 The DigitsSmall-Set

4.2.5.1 Dataset description

The original UCI Digits-dataset contains 44*250 data points originated from 44 people who have written 250 digits each which in turn have been converted into digital 16-featured representations. The dataset is therefore pretty large, the whole matrix including labels has 44*250*(16+1)=187 000. Because of the limited computational efficiency of the MacBook Pro (2Ghz i7, 4gb RAM) and the size of the dataset, for these preliminary tests, a subset of the dataset is used. We call this subset DigitsSmall and it uses 100 data points for each of the digits (0 through 9) in the training sets and 40 data points for each of the digits in the test sets.

4.2.5.2 Results and Analysis

With many classes and and attributes, this dataset is the most complex one. Of 35 runs, only eight runs obtained a satisfactory fitness on the training data. These runs were done using 150 evolutions and a population of 400 individuals. It was believed that increasing the population size would reduce the multiple restarts necessary, but of five runs, each spending 15 hours on the MacBook Pro i7, only one of these obtained satisfactory fitness when using a population of 1000 individuals.

The GoF-system performed well on a few of the runs, but the genetic algorithm did not find good parameters for this dataset easily. Because of the large search space, it is expected that a larger population would result in high classification rates more consistently. Because of limited time and resources, extensive testing on the WGoF system was not an option. From time-consuming runs with the same

Classifier	Percent correct
kNN	89.36
GoF	91.25
WGoF	low 60s

Table 4.6: Results on the DigitsSmall-set using kNN and GoF. k=3, one sub-group in the GoF-system.

setup as above, the best resulting tree produced classification rates in the low 60s. This was not prioritized to test further.

This dataset showed one important thing, namely that large datasets with many classes [1] may require a very large population and run time in order to find fitting trees. As soon as the trees are produced by the system however, the trees may be used for classification in a much less computationally expensive way. Because of this the complexity-issue may not be a large con for Verdande Technology as we believe they do have the resources for computing the optimal trees. For future work it may be interesting to try higher mutation rates in the genetic algorithm in order to cover broader patches of the search space in less evolutions.

4.2.6 The UCI Wine-Set

4.2.6.1 Dataset description

The Wine-set consists of 178 data points with 59/71/48 data points in each of the three classes. When transformations and scaling is done to the dataset, classificators can achieve good results on this set, up too 100% correctly classified. Even kNN can perform In its original form however, only 75.71 percent samples are correctly classified using kNN. It is interesting to see how well GoF will perform on this set, both in its original form and when PCA-transformed. The results were obtained using a population of 100 individuals and 200 evolutions.

[1] Remember that the search space grows almost linearly with the number of classes

4.2.6.2 Results and Analysis

Classifier	Percent correct
kNN	76.27 orig
kNN with PCA-transformed data	76.96
GoF	92.09
GoF with PCA-transformed data	95.48
WGoF	100.00

Table 4.7: Results on the UCI Wine set classified by kNN and using the GoF- and WGoF-systems.

From the table we see that without altering the data in any way, GoF clearly outperformed kNN. We believe the high improvement over kNN is mainly due to how the GoF-system may give more and less significance to different features by assigning different powers and in this way scale the dataset. Scaling the dataset has shown to be essential - Aeberhard et al. (1992) obtained 96.10 percent correctly classified on the set after transforming the dataset.

Even though the scaling of features can be assigned much of the credit for the improvements over kNN, the GoF-system also outperforms kNN performed on scaled data. Hence we expect that some of the improvement is derived from capturing structures using groups.

As kNN is known to obtain much higher classification rates on the UCI Wine-dataset when the dataset has been transformed , it was interesting to try to: 1. Transform the dataset using PCA and 2. Present the dataset to the WGoF system, as this system can scale the dataset further within groups using its weights.

In table 4.7 we can see that PCA-transforming the set to some extent improved classification-rates for kNN (0.69 percent) and increased classification rate of the GoF-system by 3.45 percent to 95.48 percent.

The increase of 0.69 percent for the kNN set may not be significant and other transformations are known to enable kNN to classify the dataset 96.10 percent correctly (Aeberhard et al., 1992). The PCA-transformation did in other words not seem to be a good transformation for this dataset.

As further scaling across groups is possible using the WGoF-system this was very interesting: Our hypothesis when implementing the WGoF-system was that this system often would even be less dependent on how the dataset is scaled on beforehand as it scales the features within the groups itself.

Only one method before has been able to classify the set 100% correctly, namely Aeberhard et al. (1992) when using Regularized Discriminant Analysis(RDA).

For the WGoF-system a population of 500 individuals was created, and after between 20 and 30 iterations a 100% classification rate is obtained on the dataset.

The WGoF system thus pars the best results obtained on this algorithm (RDA, 100 percent) and beats the second and third best algorithms (Quadratic Discriminant Analysis[QDA] 99.43 percent and Linear Discriminant Analysis[LDA] 98.9 percent).

It is interesting that convergence was reached so fast, using less than half of the evolutions needed for reaching convergence using the non-weighted GoF-system. The reason is probably a combination of having a large population size and a quite small dataset.

After obtaining such great results, the premises were double checked. The dataset was split into 70/30 percent in training-/test-set was made and tested on to be sure there was no extra over fitting. This dataset obtained 98.33 percent correctly classified (that is, only one wrongly classified) with a smaller population

size and 30 evolutions. As this result also is very good and is obtained using only 70 percent of the already small dataset, we are because of this confident the 100 percent classified as per the LOOCV-test are fair results.

4.3 Summary of results

There are especially three things that the tests performed in this chapter has taught us:

1. The GoF-system does indeed perform well on many classification tasks and is robust in regards to the scaling of the data.

2. On complex datasets with many classes where the search space becomes very large, the GoF-system requires large computational resources in order to find optimal trees.

3. The WGoF-system performs even better than the GoF-system and did obtain results that par with the very best on the UCI Wine-dataset.

5

CONCLUSION

Throughout the theoretical exploration the methodology-questions presented in the introduction were examined. This exploration and a study of the Grouping of Features (GoF)-system built an understanding of what the GoF-system is and the fields on which it grounds. This understanding enabled us to document the GoF-system and to for example extend it to include weights.

One of the key research question was whether this system could be successfully implemented and perform certain classification tasks better than for example k-Nearest-Neighbor (kNN). Chapter 4 showed that this indeed was the case: The GoF-system outperformed kNN on most classification tasks, and when extending the GoF-system with weights, the Weighted Grouping of Features-system improved the accuracy from 92.09 to 100 percent on the non-transformed UCI Wine-dataset, a dataset which kNN only classifies 76.27 percent correct on.

In the theoretical exploration it was made clear that scaling and transformations done on the dataset may be crucial for obtaining high classification rates. By removing features and using Principal Component Analysis (PCA) for further dimension reduction, a three-dimension set was reduced to one dimension and the classification accuracy was increased from 87.1 percent to 100 percent correctly

using the GoF-system.

When studying PCA we also learned that PCA-transforming a dataset not always increased classification rates, but sometimes indeed could make the results worse. On the 3f-set, classification rates using kNN were reduced by 8.5 percent when PCA-transformed into the original number of dimensions, the reason being that PCA had no knowledge of the different classes and therefore included an uncorrelated dimension when maximizing the variance of the data.

One of the GoF-system's strengths therefore is that it seems less reliant on the scaling of the features in the training data. Through tests performed on the UCI Wine set, it was shown that the GoF-system could perform well on the non-scaled dataset even though for example kNN cannot.

As both the GoF- and, even more, the WGoF-systems outperformed kNN's performance on scaled data as well, scaling is not all the GoF-systems do. Another strength is that the systems capture structures in the data which for example kNN cannot. This was shown by outperforming kNN on several datasets and especially through testing on the 2f dataset. On this set the GoF-system grouped the members of one class together and by this was able to make members of one class relatively closer to each other compared to the other class. This resulted in increased classification accuracy.

The largest challenge when testing the GoF- and WGoF-systems was in regards to computational performance. As the datasets grew in complexity, and especially when there were many classes involved as in the UCI Digits-subset, the computational resources needed to find optimal trees were substantial. The subset of the UCI Digits dataset showed that for such datasets higher population sizes are important and that multiple restarts may be useful for finding optimal trees.

Regarding the complexity-issue it should be noted that requiring large computational resources for finding the optimal trees is far from making the system useless: Optimal trees can be found using super-computers on beforehand, and therefore companies such as Verdande Technology may do this and then, in a real-time system, perform the much less computationally expensive task of making use of the trees for classification.

Nevertheless the GoF and WGoF performed the best on smaller datasets without a large amount of classes.

There are many existing classifiers for multi-class learning problems. The GoF-system and its extension WGoF has in this thesis shown to outperform one of the ten most popular classifiers in the world on most datasets, and the WGoF-system classified the well-known UCI Wine dataset 100 percent correctly on non pre-processed data.

5.1 Future work

As mentioned there is already ongoing research in regards to be able to compute the V_d's in probability-model of the GoF-system. Initial tests have given promising results, and hopefully this will be presented in future work of Verdande Technology.

In addition to this, one might want to take measures ton dampen the complexity-issue. Ideas include trying different types of dimension reduction techniques or training-set simplification techniques and altering the mutation-rate of the genetic algorithm.

There is also ongoing research in regards to using different k for different classes. This was shown to be interesting for datasets where the densities of the different classes varied. Testing this obtained good results in the theoretical exploration,

but it is believed that the improvements of doing this might be even higher in the GoF-system.

Finally, as the GoF-system proved to be successful with its three levels of nodes - top-group, sub-groups and features, a generalization of this system would be interesting. A suggested generalization would be to let any group have sub-groups so that the tree could have more than three levels. By doing this one might be able to capture even more complex structures.

Bibliography

Aeberhard, S., Coomans, D., and de Vel, O. (1992). Comparison of Classifiers in High Dimensional Settings. *Tech. Rep. no. 92-02.*

Aggarwal, C. C., Hinneburg, A., and Keim, D. A. (2001). On the Surprising Behavior of Distance Metrics in High Dimensional Space. *Lecture Notes in Computer Science*, pages 420–434.

Ajanki, A. (2007). Example of k-nearest neighbour classification.

Bair, E., Hastie, T., and Paul, D. (2004). Prediction by supervised principal components. *Journal of the American Statistical Association*, pages 1–35.

Beyer, K., Goldstein, J., and Ramakrishnan, R. (1999). When is Nearest Neighbors Meaningful? *In Int. Conf. on Database Theory.*

Bradley, A. P. (1997). The use of the area under the ROC curve in the evaluation of machine learning algorithms. *Pattern Recognition*, 30(7):1145–1159.

Center, G. S. L. (1969). What Causes DNA Mutations?

Chen, X., Wang, L., Smith, J. D., and Zhang, B. (2008). Supervised principal component analysis for gene set enrichment of microarray data with continuous or survival outcomes. *Bioinformatics (Oxford, England)*, 24(21):2474–81.

Cordon, O. and Herrera, F. (1996). A Hybrid Genetic Algorithm-Evolution Strategy Process for Learning Fuzzy Logic. *Evolutionary Computation.*

Cover, T. (1967). Nearest Neighbour Pattern Classification. *Information Theory, IEEE Transactions on.*

Cunningham, P. (2007). k-Nearest neighbour classifiers. *Multiple Classifier Systems*, pages 1–17.

Dietterich, T. G. (1995). Solving Multiclass Learning Problems via Error-Correcting Output Codes. *Journal of Artificial Intelligence Research*, 2.

Duda, R. O., Hart, P. E., and Stork, D. G. (2000). *Pattern Classification (2nd Edition)*. Wiley-Interscience.

Dudani, S. A. (1976). The Distance-Weighted k-Nearest-Neighbor Rule. *Man and Cybernetics, IEEE Transactions on*, pages 325–327.

Efron, B. and Tibshirani, R. J. (1993). *An Introduction to the Bootstrap.*

FrantzDale, B. (2012). PCA figure.

He, H., Graco, W., and Yao, X. (1999). Application of Genetic Algorithm and k-Nearest. *Knowledge Acquisition*, pages 74–81.

Hicklin, J., Moler, C., Webb, P., Boisvert, R. F., Miller, B., Pozo, R., and Remington, K. (2005). JAMA: A Java Matrix Package.

Holland, J. H. (1975). Adapation in Natural and Artificial Systems. *SIAM Review*, 18(3):287–299.

Hostetler, L. D. (1975). k-Nearest-Neighbor Bayes-Risk Estimation. *IEEE Transactions on Information Theory*, I(x).

Johnson, G. (2011). PCA example six11utils r108.

Kaur, A., Cheema, R. S., and Sandhu, P. S. (2012). Identification of Reusable Procedure Based Modules using kNN Approach. *International Conference on Latest Computational Technologies*, (Cc).

Kuramochi, M. (2001). Gene classification using expression profiles: A feasibility study. *2001. Proceedings of the IEEE 2nd.*

Lange, N., Bishop, C. M., and Ripley, B. D. (1995). *Neural Networks for Pattern Recognition.*, volume 92.

Loftsgaarden, D. (1965). A Nonparametric Estimate of a Multivariate Density Function. *The Annals of Mathematical Statistics.*

Ngai, E. W. T., Hu, Y., Wong, Y. H., Chen, Y., and Sun, X. (2011). The application of data mining techniques in financial fraud detection: A classification framework and an academic review of literature. *Decision Support Systems*, 50(3):559–569.

Pearson, K. (1901). On lines and planes of closest fit to systems of points in space. *The London, Edinburgh, and Dublin Philosophical*, pages 559–572.

Quartl (2011). p-norms.

Rechenberg, I. (1971). No Optimierung technischer Systeme nach Prinzipien der biologischen Evolution. *Stuttgart: frommann-holzbog.*

Russell, S. and Norvig, P. (1995). *Artificial intelligence: a modern approach.*

Schölkopf, B. and Smola, A. (1998). Nonlinear Component Analysis as a Kernel Eigenvalue Problem. *Neural computation.*

Seidl, T. (1997). Efficient User-Adaptable Similarity Search in Large Multimedia Databases. *Processing*, pages 506–515.

Wright, J. and Ali Alajmi (2005). The Robustness of Genetic Algorithms in Solving Unconstrained Building Optimization Problems. *International, Ninth Conference, Ibpsa*, pages 1361–1368.

Wu, C., Tzeng, G., Goo, Y., and Fang, W. (2007). A real-valued genetic algorithm to optimize the parameters of support vector machine for predicting bankruptcy. *Expert Systems with Applications*, 32(2):397–408.

Yager, R. (2001). The power average operator. *Systems, Man and Cybernetics, Part A: Systems*.

Yang, Y. and Liu, X. (1999). A re-examination of text categorization methods. *Proceedings of the 22nd annual international ACM SIGIR conference on Research and development in information retrieval SIGIR 99*, pages(Berkeley, CA):42–49.