Umeå University
Department of Computer Science
Master Thesis

Feature Extraction using Genetic Algorithms

Patrik Norman
c99pno@cs.umu.se

June 12, 2003

Abstract

An evolutionary algorithm, genetic algorithm, searches for weights for an example-based learning system, a k-nearest neighbor classifier. The weighting of the features gives the system a possibility to extract useful features in order to increase the accuracy and the understanding of data sets. The classifier works for both numeric and symbolic features. The training of the classifier, i.e. searching for combination of weights, is computational expensive and various ways to reduce the computational cost are studied.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

A combined k-nearest neighbors classifier[1] (knn) and genetic algorithm[2] (GA) is used to increase the prediction accuracy on data sets. A classifier inputs a pattern and outputs a prediction. It maps a pattern into one of several predefined classes; finding the properties that are shared within a class from the accidental properties that could differ between members in the class. The pattern consists of dimensions, each represented by a feature. One example can be fruits; the shape, size, and color are three features. The classes can be orange, banana, or apple.

Knn is a traditional classification technique. It has a database of examples that are already classified. Each example consists of features along with the classification. Given an unclassified pattern, knn compares the pattern against the examples. The unclassified pattern and the examples have the same dimension. The pattern is assumed to have the same class as the k nearest examples. The number of classes from the k nearest examples is counted. The most frequent class determines the prediction. The distance to the neighbors is usually calculated using the Euclidian distance. Knn has been proved to work good on many data sets and it is quite simple to implement.

The accuracy of classifiers including knn depends on the selection of relevant features; irrelevant features can make the classifier behave poorly. The selection can also increase the efficiency and reduce the cost of feature

[1]The reader should have a basic knowledge of pattern classification, examples of literature in this area are "Pattern Classification" [DHS01], and "Machine learning, neural and statistical classification" [DJC94].

[2]As genetic algorithms are briefly described in this report, the reader is suggested to take a look on some of the literature that exists in this area. A very good book is "An Introduction to Genetic Algorithms" [Mit98].

measurement. Feature selection reduces the number of features while keeping the classification accuracy. A more general method is feature extraction that transforms a set of features into a new set of features. Feature selection and extraction are also important in data mining, where the understanding of a data set is the main goal and not the prediction as in machine learning and pattern classification. This observation has led to many variants of knn where weights are assigned to the features in order to increase the accuracy. The problem is to find relevant weights. With many features (high dimension), a large number of different combination of weights must be searched.

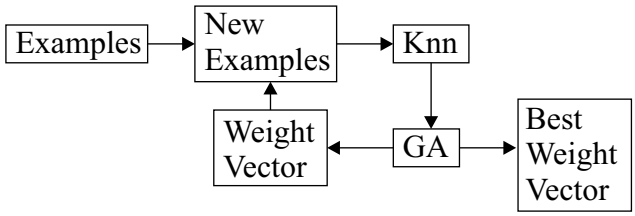

Figure 1.1: The knn is combined with a GA (the hybrid). The GA tests various weight configurations, using feedback (classification accuracy) from the knn.

The GA is loosely based on concepts from the theory of evolution in biology and have been successfully applied in many areas including pattern recognition tasks [BBM93a]. The feature selection and extraction are problems that are well suited for GA. Kelly et al. [KD91] applies knn combined with GA (hybrid) for feature extraction for three data sets. In figure 1.1 the hybrid is shown.

However, the computational time required by the GA searching for feature weights is large. In large databases with many examples and features this is a limit. Some various methods to reduce the computational time are surveyed.

1.1 Objectives

Basically this work has two objectives:

- Verify that the hybrid works. Test it in feature extraction tasks using an implemented or existing system.

- Time efficiency is an overall problem for GAs. Test the hybrid with a limited number of training and test examples. Test what impact the

population size has on the performance of the hybrid. Survey some research made for speeding up GAs, and how it could be applied for the hybrid.

1.2 Report Structure

In the next chapter, a review of literature is made. K-nearest neighbor, genetic algorithms, the hybrid, and time efficient approaches to genetic algorithms are reviewed. In chapter 3, an implementation of the hybrid is described and some experiments are made. The last chapter gives some conclusions about the most interesting results.

Chapter 2

Literature Review

2.1 K-Nearest Neighbor

Two approaches in statistical pattern recognition are parametric and non-parametric. In the parametric approach the class-conditional density function of the features is assumed to be known in advance. The class-conditional density function, $p(x|c_i)$, expresses the probability density of a feature value x given that the pattern belongs to class c_i. The density function is often assumed to have a Gaussian distribution.

In non-parametric methods such as k-nearest neighbor (knn) the decision is entirely based on the examples in the data set. There is no assumption made of the probability density of the feature values. The name non-parametric is used to distinguish it from parametric methods and the name is a bit of a misnomer, since non-parametric methods do usually depend on some parameter.

The popularity of knn is due to its simplicity and the fact that the underlying class-conditional density need not be known. This is often the case with other methods. Knn is well-known and has shown to give good performance on real-world data sets. It is competitive with other methods, such as decision trees and statistical discriminant functions [DJC94][WAM97].

2.1.1 K-Nearest Neighbor Classifier

A classifier predicts the class of a query pattern. Knn matches the query pattern against already classified examples. The k nearest examples form a subset. The query pattern's class is predicted as the class that occurs most frequently in that subset.

More formally, a pattern is a point, $\boldsymbol{x} = \{x_1, x_2, ..., x_{|\boldsymbol{F}|}\}$ in a multi-

dimensional space created from a set of features $\boldsymbol{F}$. The set of classes that the patterns can have is denoted by $\boldsymbol{C}$. The knn classifier holds a database of classified examples, the training examples. The examples consist of a pattern together with its classification, $x_e = (\boldsymbol{x}, x_c)$.

Usually, a classifier's objective is to minimize the risk of misclassifications for each class $c \in \boldsymbol{C}$. For example, misclassifying a patient suffering from a disease as healthy can be much worse than the other way around. Knn normally treats all misclassifications with equal cost.

In order to let knn determine the nearest examples, some kind of metric is used. For numeric features the Euclidian distance measure is the most commonly used. Numeric features can be temperature readings or peoples weights. The distance between two patterns $\boldsymbol{x}$ and $\boldsymbol{y}$ are,

$$d(\boldsymbol{x}, \boldsymbol{y}) = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2}. \tag{2.1}$$

A knn with two numeric features can be visualized on a 2-dimensional surface. In figure 2.1 the 3 nearest neighbors is used to determine the class of the unknown pattern. A circle is centered on the pattern that shall be classified. The radius is increased until the circle captures k examples. If the surface has few examples in that area, this will only lead to a circle with larger radius.

Algorithm 1: The knn classification procedure.

Data : A query pattern p, training examples $\boldsymbol{E}$, and k.
Result : A prediction of the query pattern's class c_p.
begin
  foreach *training example* e *in* $\boldsymbol{E}$ **do**
    calculate the distance to the pattern p
  let the k training examples with shortest distances form the set $\boldsymbol{S}$
  find the class c that occurs most frequently in $\boldsymbol{S}$
  $c_p \longleftarrow c$
end

A more detailed description over the knn classification procedure is shown in algorithm 1. The captured examples form a set. The pattern is assigned a class that appears most frequently in the set. If there is a tie between the numbers of classes, one of the most probable classes is randomly selected.

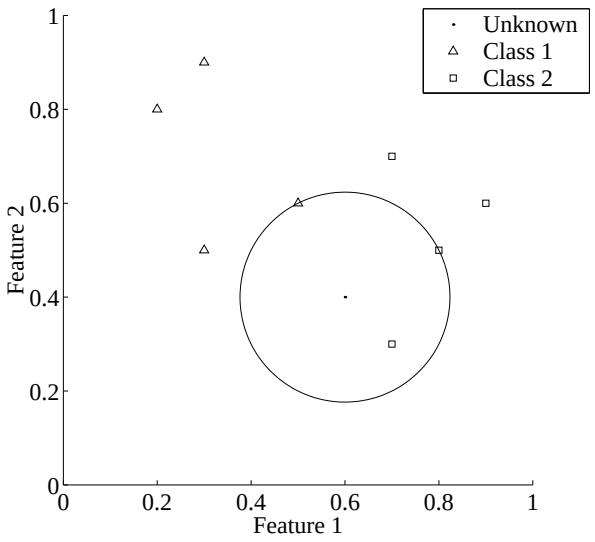

Figure 2.1: The knn predicts the class of the unknown pattern as class 2, using the majority class of the three nearest neighbors.

One example on how to handle a situation where a tie occurs can be where $k = 5$. In the selected examples, the three different classes sums to; $c_0 = 2$, $c_1 = 2$, and $c_2 = 1$. Then there is a tie between c_0 and c_1. In the training examples, say there are 100 examples, the class c_0 occurs 50 times and class c_1 25 times. The class is then predicted by the randomly selected class c_0 with a probability of 2/3, and class c_1 with a probability of 1/3.

Normalization of feature values is important in order to give each feature the same influence on the distance measure. A feature with values ranging from 0 to 100 will otherwise easily overpower a feature with values ranging from 0 to 1. Every value for each feature is divided by the range of that feature, that is

$$normalize(x_f) = \frac{x_f}{max(f) - min(f)}, \tag{2.2}$$

where $min(f)$ and $max(f)$ are the minimum and maximum values for that feature. Those values are determined simply by going through all examples and store the minimum and the maximum value for that feature.

2.1.2 Symbolic Features

Some data sets include symbolic features. The values of such features have no given order among themselves. One example can be a feature that consists of the colors green, yellow, brown to determine the ripeness of bananas.

To handle symbolic features the distance function is extended [WM96] and defined as,

$$difference(x_f, y_f) = \begin{cases} |x_f - y_f| & \text{if } f \text{ is numeric,} \\ 0 & \text{if } f \text{ is symbolic and } x_f = y_f, \\ 1 & \text{if } f \text{ is symbolic and } x_f \neq y_f. \end{cases} \quad (2.3)$$

And the distance is a sum of the difference between two patterns,

$$distance(\boldsymbol{x}, \boldsymbol{y}) = \sqrt{\sum_{i=1}^{n} (difference(x_i - x_i))^2}. \quad (2.4)$$

The root never changes the order of the nearest neighbors and adds an extra computational burden so it is not calculated in practice.

2.1.3 Weighting Features

The accuracy of knn is good on most data sets but it can behave poorly. Irrelevant, interacting, or noisy features can have much effect on the distance calculation. This observation has led to many variants where weights are assigned to the features in order to increase the accuracy

Irrelevant feature appears as noise on in the distance measure. The effect of interacting features is more subtle. Interacting features are features that can be derived from other features. For example, one feature is the square of some other. The simplest case is when one feature is a copy of another one. If a feature is repeated five times in the data set, its contribution to the distance will be five times greater. If that feature is more useful for classification this can be good, but otherwise the interacting features decrease the accuracy. Another difference against irrelevant features is that redundant features cannot be skipped; they are actually meaningful for the classification. This has given rise to the idea of using feature weights because some of the features are more useful for a classification while others are not. Irrelevant features are assigned low weights (near 0) and give a low influence on the distance measure. The method serves another purpose as well. Interacting features can be fine-tuned together and lead to an increase in performance [WM96]. The weighted distance measure between two patterns $\boldsymbol{x}$ and $\boldsymbol{y}$ is defined as

$$distance(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{w}) = \sqrt{\sum_{i=1}^{n} (w_i \times difference(x_i - y_i))^2}, \quad (2.5)$$

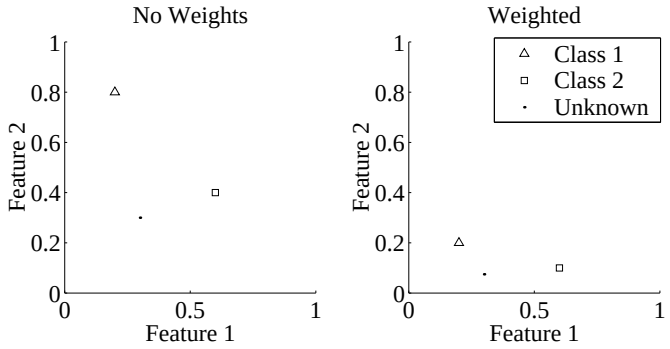

Figure 2.2: Weighting the features scales the axis.

where $\boldsymbol{w}$ is the weight vector for the features.

The weights scale each axis in the space as shown in figure 2.2. On the right the effect of weighting can be seen. The example of class 1 is closer the unknown pattern than the example of class 2. The weight for feature 2 is 0.25 instead of 1.0.

2.1.4 Evaluation of Performance

The accuracy is the hit rate on a data set using the knn classifier. In order to determine the accuracy of knn it must be tested on unseen data. The database is usually divided into two sets, one for training and one for testing, without any shared examples. Then the knn predicts each of the test examples using all the training examples. The predicted class of each test example is checked against its correct class. The hit rate h is the number of correct predictions, $hits$, divided by the total number of predictions, the same as the number of test examples, $|\boldsymbol{E_{test}}|$,

$$h = \frac{hits}{|\boldsymbol{E_{test}}|}. \tag{2.6}$$

The number of training and test examples will further on be referred as the test size and the training size respectively. A more detailed description of the procedure is shown in algorithm 2. The hit rate is only an estimate of the true hit rate unless we have tested the classifier against the all possible examples. It can be impossible test the classifier against the all possible examples because they can be infinitely many. Conversely, the error rate is defined as the number of wrong predictions divided by the total number of predictions.

Algorithm 2: Estimation of accuracy of a knn classifier.

Data : Examples $\boldsymbol{E}$.
Result : The accuracy in form of hit rate h.
begin
  separate the examples $\boldsymbol{E}$ into two sets, $\boldsymbol{E_{training}}$ and $\boldsymbol{E_{test}}$
  $hits \longleftarrow 0$
  foreach *test example in* $\boldsymbol{E_{test}}$ **do**
    perform the knn classification using $\boldsymbol{E_{training}}$
    if *the predicted class is the correct class* **then**
      $hits \longleftarrow hits + 1$
  $h \longleftarrow hits/|\boldsymbol{E_{test}}|$
end

The classifier can behave differently for different values of k. The knn works well if there are many examples very close the pattern. A larger value of k includes more examples and gives a more reliable estimate. But with a larger value of k the examples tend to be farther away from the sample. A compromise must be made and the optimal value of k is usually larger than 1.

Overfitting is when the classifier gives a higher appeared hit rate then the true hit rate. If the classifier overfits it has become too specialized on those training examples and it does not generalize well for the unseen data.

In order to have a good estimate of the true hit rate (and avoid overfitting) many tests are made on the examples, known as resampling. One resampling method is k-fold cross-validation where the examples is divided in k parts or folds. Each fold is used for testing and the rest (k-1) for training. K tests are made and the overall accuracy is the average of all tests. When k equals the number of examples it is called leave-one-out. Each example is used for test and the rest for training. N number of tests are made, and the overall accuracy is the average of all tests.

To maintain the accuracy of the classifier, the number of examples has to increase exponentially with the number of features. This is called *curse of dimensionality*. Having a fixed number of examples and then increasing the number of features (increasing the dimensionality) makes the examples appear more sparsely distributed. This can be seen in figure 2.3. If the examples are distributed more sparsely, then the accuracy of the knn usually drops [JC82]. The remedy is to use more examples.

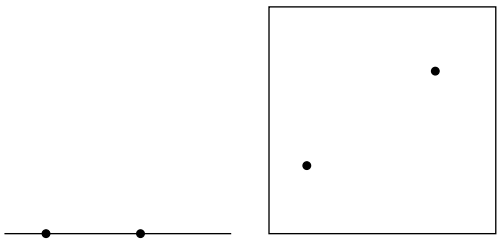

Figure 2.3: If two points are randomly spread out on a straight line and in a square, then the two points are more sparsely distributed in the square.

Later in this report, some experiments are performed on data sets with a limited number of examples. The purpose behind this is to speed up the evaluation of weight configurations, refer to the hybrid figure 1.1.

With a finite training size and using knn, one cannot make any general statements about the hit rate. The hit rate does not have to increase monotonically with an increased training size [DHS01]. That is, adding more training examples can have a negative impact on the hit rate. The knn classifier can have a lower hit rate with an increased training size, but in general the hit rate will increase with an increased training size. Since more training examples sometimes can have a negative impact, the GA will have a harder task in finding promising weight configurations.

2.1.5 Computational Complexity

The computational complexity is measured in both time and memory. To reduce the search time for knn, various methods have been suggested [DHS01].

In *partial distance*, the distance between the query pattern and one training example is calculated in steps for each feature. Each step adds the distance between the query pattern and the training example for that feature to a sum. Also, in each step this sum is compared against already calculated distances. If it is greater than the k greatest distances, then the training example cannot be one of the k nearest neighbors. Then it is meaningless to continue the distance calculation for that training example. This reduces the search time.

Editing is elimination of unnecessary examples. This is simplest when k is 1. If there is a group of several examples with the same classification, most of the examples could be eliminated. The classification would be the same. This reduces the storage needs.

In *search trees*, the examples are linked together in a tree structure so

that examples that are neighbors have a direct link. This reduces search time but some work must be done before the classification.

2.2 Genetic Algorithms

Genetic algorithms (GAs) are loosely based on concepts from biological evolution. GAs are used to solve search and optimization problems and can be applied to different areas including machine learning.

GAs has an analogy with the principles of natural selection. In nature individuals compete with each other. Successful individuals will in successive generations have a relatively larger number of offspring than less successful ones. This is often referred to as "survival of the fittest".

Algorithm 3: A basic genetic algorithm.

```
begin
    create the initial population
    repeat
        calculate fitness of each individual
        select individuals for recombination and mutation
        insert the offspring into the next population
    until best individual's solution is acceptably good
end
```

GAs can never guarantee that a sufficient solution is found within a certain time. But it can, under the right circumstances, find acceptably good solutions for problems within an acceptably short period of time. The search is done by having a population of individuals. The individuals are expressions of solutions. The solution can for example be weights in an artificial neural network.

Each individual has a fitness score determined by the fitness function (section 2.2.3). The fitness score should reflect the ability to solve the problem. Depending on the fitness score, parents are selected (section 2.2.4) for recombination or slightly changed to create offspring. This is known as reproduction (section 2.2.5). The methods, recombination and mutation, used for creating new individuals are known as *genetic operators*. The search is stopped when the fitness value for the fittest individual score is greater than or equal to a predetermined value. The search can be also run for a number of generations or be stopped when the population converges. The convergence

of a population can briefly be said to be when the population consists to a large degree of identical individuals. A basic GA is shown in algorithm 3.

Next, other methods, the basic principles of GAs and other aspects are treated in more detail.

2.2.1 Overview

Solving problems by searching for solutions is common in computer science. The *search space* refers to candidate solutions and the distance between them. Other techniques similar to GAs are gradient and iterated search. Gradient search uses a derivative of a continuous function to guide the search towards a peek, known as *hill-climbing*. If the function has several peeks, the first peek will be climbed and it may often not be the highest peek. Iterated search is gradient search restarted several times at different locations in order to find the global maximum.

Simulated annealing is a method inspired from physics and it simulates cooling, where one wants to find the global minimum. It uses random moves, moves in all directions. The probability that upward (bad) moves are accepted, is described by a function $p(t)$, where t is the time. In the beginning $t = 0$ and $p(0) = 1$, that is, all moves are accepted, both downward and upward moves. But after some time, the result from $p(t)$ gradually goes towards 0, and eventually only good moves are accepted. This is the analogy with cooling. The idea behind this is that the risk of getting stuck in a local minimum is reduced.

GAs differ from gradient search, iterated search, and simulated annealing in that GAs use many candidate solutions at once. In this way, they use information from many different locations in the search space. They also use information that has been accumulated over several generations.

An important question is when GAs is preferable to the other methods. Mitchell [Mit98] describes some points where GAs is thought to be preferable.

- A search space that is so large that searching all combinations takes too long.

- There is no knowledge what the search space looks like, or it is known to be discontinuous.

- A sufficiently good solution is acceptable and therefore the global maximum must not be found.

- It is competitive with other weak learning methods. Weak learning

methods are methods that do not use domain specific knowledge. However, GA can incorporate domain specific knowledge.

- GA is robust for noisy problems when it uses several individuals and accumulates information over time.

These points are only hints when the GAs can be successful. The performance of GAs depend on many things. More details are mentioned under section 2.2.7.

2.2.2 Encoding of Parameters

The solution for a problem requires a set of parameters. The parameters are represented in a string, *chromosome*. One approach is a string of binary digits, each digit is called a *gene*. One example can be a solution for a problem that requires 5 parameters. Each parameter consists of 10 binary digits. Then they are placed in a string of 50 binary digits.

Those parameters are called the *genotype* and can be seen as a blueprint. The solution, called the *phenotype*, is built using the blueprint. In nature the phenotype represents the physical organism. From the phenotype, various characteristics are measured.

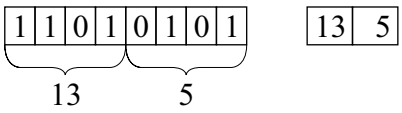

Figure 2.4: A binary and a real-valued representation. The real-valued representation is somewhat simplified, it uses integer numbers instead.

The classical approach is a binary representation. The parameters are encoded and put in to a string of binary valued digits. There are some disputes whether binary approach is the best, as other representations can have benefits [Mit98]. In a real-valued representation, each parameter is represented by a floating point number. If the parameters are floating point numbers, then this representation is of course preferable as there is no need for converting the binary representation to floating point numbers. In figure 2.4 a binary and a real-valued representation are shown.

2.2.3 Fitness Function

A fitness function assigns a score to an individual. The score reflects the ability to solve a problem. The fitness function is considered to be the most

crucial part of the GA. For a hybrid system such as the knn classifier, refer to section 2.1.4, the fitness score can be the hit rate of the classifier,

$$h = \frac{hits}{|\boldsymbol{E_{test}}|}. \tag{2.7}$$

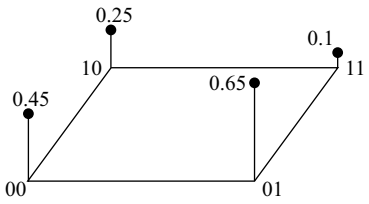

Figure 2.5: A simple fitness landscape.

For many problems there are often many different things to consider. They must be weighted against each other and sometimes they are contradictory. The *fitness landscape* is the plane of all genotypes along with their fitness, the term comes from biology. Figure 2.5 shows a simple fitness landscape. The genotypes are $\{00, 01, 10, 11\}$.

2.2.4 Selection

The selection part in GAs selects individuals in order to create new individuals. The GA has a set of individuals, a population. The composition of individuals in this population changes over time. One way to change the composition of individuals is to form a completely new population using the current population. This is called *generational* GA as the current generation is used as material for the next generation. One can also keep the population and only replace a smaller part of the population. Usually the least fit individuals are replaced. This type of GA is called *steady-state* GA. In order to select individuals, various selection methods (*selection schemas*) are used.

The selection schemas select individuals based on their fitness and the selection is usually stochastic, that is every individual has a probability of being selected. The idea is to simulate natural selection.

The selection schemes have different strength. Two contradictory search strategies are used; exploration and exploitation. Exploring the search space can be seen as spreading out individuals at many places in order to get close to a global maximum. Exploitation is when individuals take advantage of locations already visited, refining the search to find the global maximum. A successful selection schema should be a tradeoff between exploration and

exploitation. Exploration can be useful in the beginning of the search and at the end exploitation can be more important.

Fitness proportional selection selects an individual with a probability that is relative to its fitness and the fitness of all the individuals. The probability that an individual is selected is

$$P_i = \frac{f_i}{\sum_{j=1}^{n} f_j}, \tag{2.8}$$

where P_i is the probability that individual i is chosen, f is the fitness score, and n is the number of individuals in the population. The problem with this selection scheme is that individuals with low fitness may not be selected at all. This problem is often solved with different fitness scaling methods.

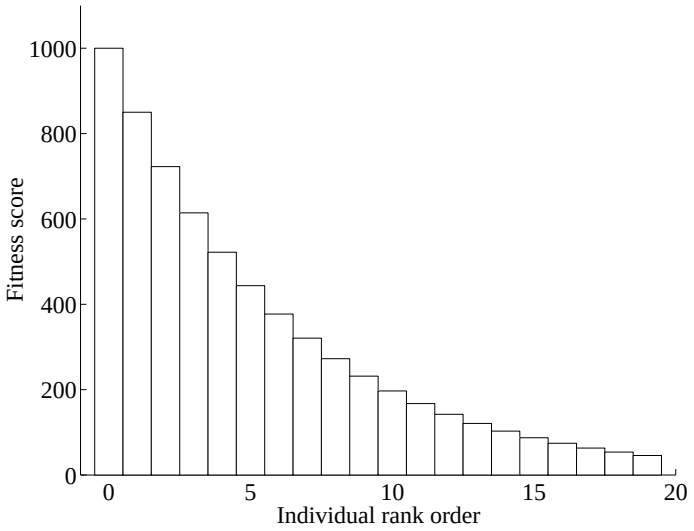

Figure 2.6: The figure shows an exponential rank selection where the individual with rank order 0 has the highest fitness.

Rank selection does not use the fitness of the individuals directly. Instead it sorts the individuals in increasing or decreasing order according to the fitness. A new fitness score is calculated from the rank of an individual. The rank selection can be linear or exponential. One example can be a population of 20 individuals, their fitness values (from hit rate) are sorted with the best (highest hit rate) individual first. Each individual is assigned a new fitness score $1000c^r$, where r is the rank $\{0, 1, ..., 19\}$, and c is a constant with the value 0.85. In figure 2.6, the fitness score of the individuals can be seen. The constant c controls the *selection pressure* and a lower value on c will favor more fit individuals.

Another selection scheme is tournament selection; it is inspired from biology. Two or more individuals are randomly chosen from the population and put into a set. From this set only the best (highest fitness score) individual is used; it is the winner of that pack. After the selection, all individuals in the set are put back and can be selected next time. In this way individuals are selected for reproduction.

Elitist selection keeps the best individuals in successive generations. This selection scheme is usually combined with other selection schemas. Steady-state GAs use this kind of selection. Refer to the beginning of this section for a description of generational and steady-state GAs.

The *convergence* is the advance towards uniformity. The convergence is controlled by selection with exploration and exploitation. A gene has converged when some degree of the population share the same value for that particular gene. 95% has been used [Mit98]. When all genes have converged, then the population has converged.

2.2.5 Reproduction

In order to create new individuals from the current population, individuals are first selected, then special operations (called genetic operators) are applied on the chromosomes in the selected individuals. There are two basic operators, crossover and mutation.

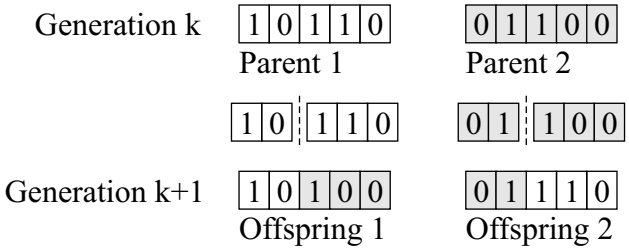

Figure 2.7: The one-point crossover creates two new chromosomes by recombining the parents.

Crossover recombines chromosomes by cutting the chromosomes at a randomly chosen position. In figure 2.7, one-point crossover and two chromosomes are used. The two tail segments are swapped to create two new chromosomes. There is a variety of different crossover operators. They can be applied to more than two chromosomes at the same time or by using several cutting points.

A uniform crossover swaps (with a certain probability) each binary digit

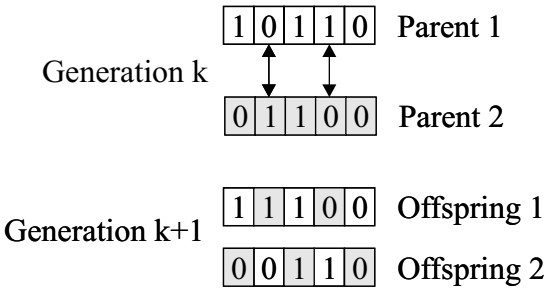

Figure 2.8: A uniform crossover swaps (with a certain probability) each binary digit with the corresponding digit on another chromosome.

with the corresponding (same position) digit on another chromosome as shown in figure 2.8.

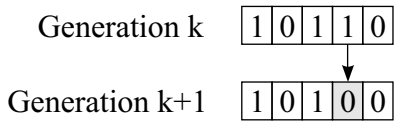

Figure 2.9: A mutation of one of the binary digits in a chromosome.

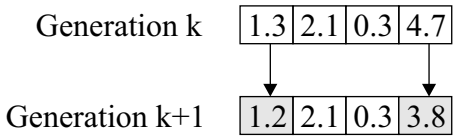

Figure 2.10: For a real-valued chromosome other operations are possible; a mutation could add or subtract some random value to one or more of the parameters, called creep.

Mutation makes random changes to a chromosome, figure 2.9. In a string of binary digits it can invert one or more of the digits. For a real-valued chromosome other operators are possible; a mutation could add or subtract some random value to one or more of the parameters, as shown in figure 2.10. This genetic operation is called *creep*.

2.2.6 Mutation or Crossover

Crossover is often considered to be the main force driving the search in GAs, but nature shows that sophisticated creatures can be evolved without crossover, only by using selection and mutation. Biologist calls this *naïve*

evolution. Selection and mutation only performs a hill-climb-like search. Spears [Spe93] compares mutation and crossover and concludes that they have different properties.

Crossover and mutation are important in different stages of the search [BBM93b]. Crossover speeds up the construction of good individuals. This is good in the beginning with a population that has a large diversity. When a population converges, mutation becomes more effective than crossover. When a population is near convergence, the individuals have almost identical chromosomes. Using crossover on those individuals creates chromosomes that already exist in the population.

2.2.7 Why GAs work

There is no generally accepted theory behind the workings of GAs. But there are some hypotheses that try to explain the properties [Mit98].

One of them is the *schema theorem* that uses string templates so called schemas (or schemata). The schemas are meant to be a formalization of the chromosomes. The schemas correspond to partial solutions and they consists of the values 0, 1, or #. The # is a wildcard matching both 0 and 1. One example can be the chromosome $(0, 1, 0, 1)$ where $(0, \#, \#, 1)$ and $(\#, 1, \#, \#)$ are valid schemas. The schema $(0, \#, \#, 1)$ has 2 defined positions and is said to be of *order* 2. The *defining length* of a schema is the distance between the two outermost non-# symbols. For the schema $(0, \#, \#, 1)$, that is 4.

The schema theorem can be interpreted as there are some schemas whose fitness is above average, those are also called *building blocks*. These are more likely to be transferred to the next generation and thus exponentially increase in frequency in the population. The properties for these schemas are that they are short (the defining length is small) and of low-order (few defined positions) and have a fitness that is above average. The short and low-order schemas have less chance of being disrupted by crossover or mutation. Individuals with high fitness are assumed to consist of these schemas.

While the schema theorem describes the change in frequency of schemas it does not directly give predictions about the convergence rate or the compositions of individuals. Other models have given more insight, but they often make the assumption of having an infinite population. The models are made simpler when having an infinite population, while the real GAs suffer from finite-sample problems.

However, the behavior of GAs has not yet been fully understood and empirical tests are often used to control the parameters. The theoretical work has given some insights but there is much more to do.

2.3 The Hybrid Algorithm

In this section the GA combined with knn, also called the hybrid, is described. The performance of the hybrid is a tradeoff between the classifier accuracy (obtained hit rate from the evaluation of weight vectors) and the computational cost (time used by the GA). The knn weights each feature using a vector of weights in order to increase the classification accuracy, refer to section 2.1.3 for details. The GA is employed in the search for a good weight vector. In figure 2.11, the hybrid is shown.

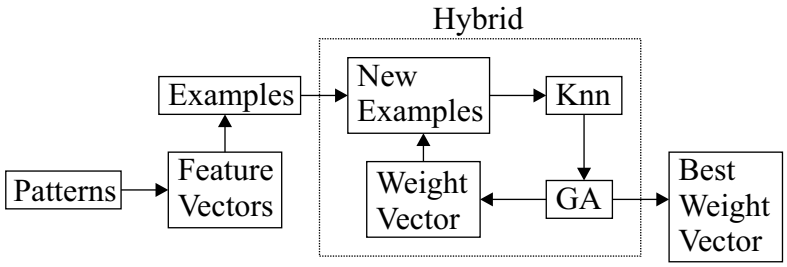

Figure 2.11: The knn is combined with GA in order to search for a weight vector that increases the classification accuracy.

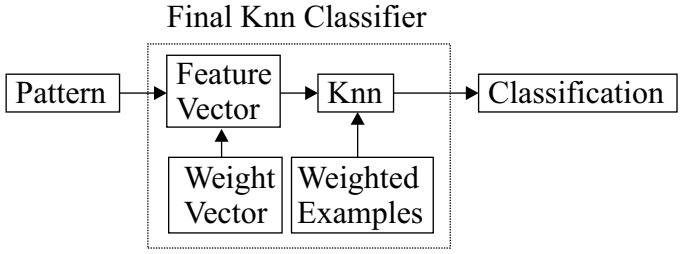

Figure 2.12: The resulting knn classifier can use already weighted examples directly.

When the hybrid has found a good weight vector, which is a weight vector that increases the classification accuracy, it can be used to weight the examples in advance for direct use in a knn classifier. This is shown in figure 2.12.

The feature weighting can also be used to reduce the cost of feature measurement. Weights with values near 0.0 are less important for the accuracy of the weighted knn classifier and can be left out of the classification procedure. The data set could be preprocessed in this manner (ignoring some of the features) and be used with some other classification method than knn. Feature

extraction could also be used in data mining, where the understanding of a data set is the main goal and not the prediction; as in machine learning and pattern classification. A small survey is made over different approaches for feature selection and extraction.

There are many methods for feature selection and extraction. Wettschereck et al. [WAM97] reviews some methods. In feature (subset) selection the search for a subset of relevant features involves an *objective function.* It will measure how "good" the subset is. The hybrid uses feedback from the knn classifier using the predicted hit rate as an objective function. There are other approaches that rates the subset's information content, which can be information-theoretic or statistical dependence. Those methods use no feedback and are usually faster.

2.3.1 Earlier Work

Kelly et al. [KD91] describe their hybrid, similar to the one in figure 2.11. They use feature weighting where the distance between two patterns, $\boldsymbol{x}$ and $\boldsymbol{y}$, are weighted using a weight vector $\boldsymbol{w}$, refer to equation 2.5. They also use additional methods to improve the knn classification and help the GA search. As this is not so important for this report, it is left out.

In the initial stages of the search for weights, the GA creates a number of weight vectors. The vectors are initialized to random values in some predetermined interval. Each weight vector, or individual to use GA terms, is evaluated using the knn classifier. Each individual will then have a hit rate that is used to rank the individuals in the population. The GA proceeds by creating new individuals and testing these, according to algorithm 3 in section 2.2.

Genetic Operator	Probability
Real Number Mutation	0.20
Small Creep Mutation	0.29
Large Creep Mutation	0.15
Average	0.06
Uniform Crossover	0.30

Table 2.1: The probabilities of the genetic operators in the hybrid from Kelly et al. [KD91].

In the GA, they use three different types of mutation. Real number mutation replaces a real number on a chromosome. The two creep operators

(small and large) add or subtract a random value from a real number. Small creep changes a real number with the probability 0.1 with a value randomly generated in the interval 0 to 0.25. Small creep changes a real number with the probability 0.05 with a value randomly generated in the interval 0 to 0.1. Uniform crossover is used. They also included an average operator. The average operator produces one offspring with its values set to the average of its two parents. Table 2.1 shows a summary over the genetic operators.

The selection scheme uses rank selection. The individuals are ranked with one of the two evaluation functions and assigned a fitness from the series $\{1000c^0, 1000c^1, ..., 1000c^{n-1}\}$, also described in section 2.2.4. The individuals with fitness below 1 are replaced with 1. The c was set to 0.95 in the beginning. Under the search process the c is decreased in gradually and finally set to 0.7. Different settings of c will control the exploring and the exploitation in the search. The population size is 50 and the 600 new individuals are evolved in a steady-state manner. They tested 3 data sets from 150 to 463 examples, the largest data set had 17 features, and their result shows that the hybrid outperforms knn. The data sets they used have few examples and using a single 5-fold cross-validation can give very inaccurate results.

Raymer et al. [RPG+00] also use knn combined with GA. Their work focuses on dimensionality reduction but also includes feature extraction. One difference is that they use a masking vector for the features, i.e. all weights are binary, having the values of either 0 or 1. First a subset of the features is selected with help of the masking vector. The selected features are used in an ordinary weighted knn. The idea behind a masking vector is to let the GA faster find a subset of relevant features. Their hybrid shows good results on two data sets.

2.4 Time Efficient Genetic Algorithms

The time effectiveness of GAs are important when they are used practically. In practice the GA is only allowed to run a limited amount of time and the final solution is the best individual from the ending generation. Next, some research for time efficient GAs is surveyed.

2.4.1 Noisy GAs

Noise is often present in the fitness function and can be due to many reasons. Noisy information can effect the fitness evaluation. Noisy data (the data itself

contains noise), sensor input (random noise), and human error (a mistyping by a human) are examples of noisy information. Also, there may be no known accurate fitness evaluation, so instead an approximate fitness function must be used.

In order to have a more accurate estimation of the true value, several measurements (samples) can be made to reduce the noise. It can be used everywhere where the fitness function has a random noise component. One example of an application where sampling has been used is; estimating the difference between two images [Mil97].

In the case with the knn classifier, the noise comes from the fact that a limited training and test size are used to evaluate the classifier. The limited number of examples does not describe all the possible cases perfectly, refer to section 2.1.4.

In the case with the hybrid, the fitness of an individual is based on its ability to classify patterns. If it accurately classifies the whole data set with patterns (all test examples that are infinitely many), then it will not have any noise. An evaluation of the individual will always return the same hit rate and thus it will not be noisy. Instead, if the individual only classifies a smaller subset of the data, the variances of the hit rate are very much a function of the size on that subset.

Miller [Mil97] defines a *noisy fitness function*. The noisy fitness function,

$$f_i^{'} = f_i + rnd[N(0, \sigma_E^2)], \tag{2.9}$$

consists of a fitness function plus a random noise component. Miller makes the assumption that the noise is randomly drawn from a Gaussian distribution with a variance of σ_E^2. The noise is unbiased, that is the noise has a mean of zero.

The noisy fitness function is repeatedly sampled by a *sampling fitness function*. The number of samples made is called the sample size. The evaluation of an individual using the sample size n can be expressed as the sampling fitness function

$$f_{n,j}^{*} = \frac{1}{n} \sum_{i=1}^{n} f_{i,j}^{'}, \tag{2.10}$$

where $f_{i,j}$ is the ith noisy fitness evaluation of individual j. If the fitness function $f^{'}$ has noise variance σ_E^2 and the computational cost of β. Then, the sampling fitness function f_n^* has noise variance σ_E^2/n and the computational cost of $\beta \times n$. The sample size is simply number of samples used by the fitness function and it determines the speed and accuracy of the fitness evaluations.

For the hybrid, the sample size n is thought to represent the test set size. The higher n, the more accurate estimate of the fitness is made. Ideally one would like to have as high value on n as possible, but both the test size and computational time are limited.

Miller looks into the question of; given a bounded time, what sample size maximizes the performance of a GA? The optimal sample size maximizes the performance of the GA within a fixed time bound. Miller uses "proportion of building blocks" in his performance measure of the GA-runs. The chromosome consists of several genes. Some of them, the building blocks, provide the individual with a higher fitness then the other genes. The proportion of building blocks is the number of such genes divided by the total number of genes in the chromosome. The "proportion of building blocks" can be roughly translated to the fitness. And when dealing with the hybrid, the fitness is the hit rate.

Noise from the fitness evaluation will directly affect the selection mechanism in the GA (section 2.2.4). When using a rank selection, for instance, individuals with higher fitness will mix in rank with individuals with lower fitness. For example, if the true fitness evaluation ranks some individuals in the following order; $\{i_0, i_1, i_2, i_3, i_4, i_5\}$. Then with noise, the ranking could be $\{i_3, i_2, i_0, i_1, i_5, i_4\}$ instead. The selection mechanism will then make bad building block decisions. Increased selection noise requires a larger population in order prevent premature convergence. In a rank selection, the convergence rate is controlled by the parameter c, see section 2.2.4. Here, convergence rate is supposed to be dependent on selection noise also.

Miller draws a number of conclusions regarding the behavior of noisy GAs. Also, a directed graph over the aspects for the behavior of noisy GAs is shown in figure 2.13.

- The sample size affects fitness noise.

- Fitness noise affects selection noise.

- Selection noise affects convergence rate and population size. Selection noise can be seen as the individuals are shuffled to some degree in GA's rank selection.

- Population size affects premature/slow convergence and computational requirements. A small population converges faster and a larger population converges slower.

- Computational requirements affect performance.

- Convergence rate affect performance.

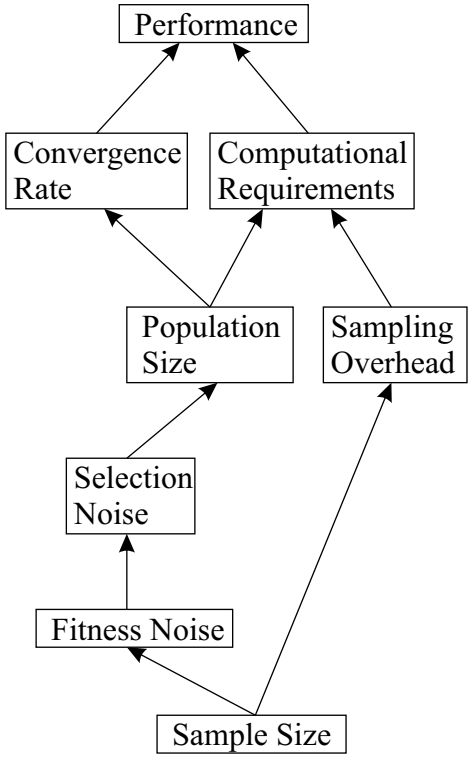

Figure 2.13: A directed graph over the dependencies in noisy GAs.

The performance is the best fitness from a population within a bounded time. The optimal performance is the best performance obtained within a bounded time. The optimal performance is dependent on both the population and the sample size, and in order to predict it, Miller makes models over the behavior of the GA.

Even simple problems results in complex models. Also, many assumptions are made; like the fitness noise has a normal distribution. A model that Miller uses is the well understood Onemax domain. This domain is a type of optimizing problem that sums all ones in a string of digits where each digit consists of either zero or one. The fitness is equivalent to the sum of the ones in the string where a higher value is better.

The model uses information about of the building blocks. Also, the variance of the populations real fitness (fitness variance) and the noise variance for each individual must be supplied. From this, the optimal population size and sample size can be determined. The results show empirically that

the models are good approximations. Most of the settings are unfortunately domain dependent and cannot easily be used directly in other domains.

One of the settings is domain independent; that is the lower bound on the optimal sample size. For this, both the variance of the fitness and the noise variance must be supplied.

The effect of noise on time bounded GAs has not been extensively studied [Mil97], and it is often difficult when designing a knn classifier to determine how many training and test examples that gives a "safe" estimate of the true hit rate [JC82]. These two facts combined is not an easily solved problem when trying to optimize the hybrid in terms of time efficiency.

In [Mil97] there is a practical guide to applicability (section 5.2). The guide applies to making predictions to optimal population sizing and sample sizes when using sampling fitness functions. The equations and parameters used for those predictions are described close to the experiments.

2.4.2 Scheduling

Aizawa et al. [AW93] use what they call scheduling in an attempt to speed up noisy GAs. The scheduling consists of allocating computational time (or samples) for each individual or generation. The GA search is modeled as a statistical process. The methods are too lengthy to describe in detail here, but the underlying ideas are good. In all their trials the population size is held constant.

The first is *duration-scheduling* which samples all individuals equally in each generation, but differently across generations. The last generations take longer time than the first with this scheduling. The idea is that, when the population converges, the fitness values of the individuals are distributed denser than earlier in the search. In order to distinguish the individuals in a better way towards the end of the GA search, more samples are made in the last generations.

The other is *sample-allocation* that involves the procedure of allocating a fixed sample size for each individual in every generation. All generations take the same time but some individuals are sampled more than others. The idea is that it is more important to distinguish individuals with higher fitness. Spending more time sampling those individuals gives better efficiency in terms of computational time.

In order to apply the scheduling both the noise variance of the fitness function and the variance of the fitness of individuals must be known. The calculation of those takes some overhead time. The results show some speed up in most cases. But the speed-up was only a few percent and some runs

were even slower using scheduling than without.

Varying the population size and combining duration-scheduling with sample-allocation can give more possibilities. However, the difficulties, especially when predicting the behavior of the GA increase.

Chapter 3

Experimentation and Results

The motivation for the experimentation is; verifying that the hybrid works and examine how the hybrid can be made more time efficient. Kelly et al. [KD91] mention that performance issues should be addressed, the search for weights is computationally expensive. Wilson et al. [WM96] says that reducing the training time of the hybrid, fewer test examples could be used as an approximate value. Controlling parameters adaptively or predicting the behavior of the GA is also of great importance.

The experiments can be summarized to the following points.

- Verify that the hybrid successfully weights features in order to increase the accuracy of the knn classifier.

- How do various types of GAs (like mutation only and crossover only) compare with completely random searches?

- How does the hybrid perform if the training and test examples are reduced?

- How do various population sizes affect the performance?

3.1 Description of the System

The implementation of the hybrid was made in ANSI C++. The system implemented has mostly the same features as the one described by Kelly et al. [KD91] in section 2.3.1. The implementation was made for educational purposes. It could also have been possible to use parts from other already implemented systems.

The design decisions made for the implementation are summarized in the following points.

- The knn uses numeric and symbolic feature values (section 2.1.2), and the weighting of features (section 2.1.3).

- In the evaluation of accuracy of the classifier, different methods can be used such as leave-one-out, k-fold cross-validation, or a separate training and test set (section 2.1.4). The hit rate from the knn classifier is used as a fitness value from an individual.

- Rank selection with the selection pressure $c = 0.85$ is used (section 2.2.4 and 2.3.1). The selection pressure controls the convergence rate of the GA.

- The population is generational (a new population is created when individuals are evolved (section 2.2.4).

- The GA-operators and probabilities of GA-operators are the same as in Kelly et al.'s system (section 2.3.1).

- Each new GA run has its random seed set by the current time so that no run should be the same.

- The k for the k-nearest neighbors in the knn classifier is set to 3 throughout all the trials for the sake of consistency.

The parameters such as selection pressure and others can be modified. In all experiments the parameters were as above (unless stated otherwise).

The database (a vector in memory) holds all the examples loaded from a file. From this database a number of examples are selected randomly without replacement into a subset. When calculating the hit rate, with leave-one-out for instance, the examples from this subset are used. The normalization of values uses min and max values from the training examples. This is to ensure that the classifier should not be aware of the unseen test examples.

In order to resolve ties within knn, the method described in section 2.1.1 is used. There was no support implemented to deal with missing feature values or highly uneven class distributions.

In order to measure time the program used a simple counter. When evaluating the hit rate of the knn classifier, a number of comparisons between training and test examples must be done. Leave-one-out compares each test example against each of the remaining $n - 1$ training examples n

times, and that gives a total of $(n^2 - n)$ comparisons. The hybrid simply counts each comparison of feature vectors and used the sum as a measure on computational time. A separate training and test set with n and m examples respectively gives a total sum of $n \times m$ comparisons.

The sum was also used to stop the hybrid within a certain training time. It is essentially the same thing as using computational time. It was discovered that the fitness function evaluations took much more time than the time spent for the rest of the GA. The time spent for the rest of the GA is therefor neglected.

3.2 Data Sets

All data sets used in the experiments comes from the UCI repository [BM98]. The UCI repository is commonly used for machine learning. There were difficulties with some of the data sets; some examples haves missing feature values and in some data sets the class distribution is highly uneven. The data sets used in the experiments have no missing feature value. The class distributions are equal in the LED24 and Chess data sets, but somewhat uneven in the Glass data set. The data sets used in the experiments should still give fair results as the difficult properties are mostly a task for the knn itself and not the hybrid. Table 3.1 gives a summary over the three data sets used in the trials.

Data Set	No. Examples	No. Classes	Feature Values
LED24	1000	10	Symbolic
Glass	214	6	Numeric
Chess	3196	2	Symbolic

Table 3.1: A summary of the data sets used in the trials.

The data set in the UCI repository has few data sets with irrelevant or interacting features. So in order to test the classifiers on data sets with such properties, existing data sets are modified [WM96]. One exception is the LED24 data set described below that has many irrelevant features.

In the experiments three data sets were used. The LED24 data set consists of 7 features (forming the numbers 0 to 9) and an additional set of 17 irrelevant features. There are 10 classes with the same distribution (theoretically). Each of the first 7 features has a 10% chance of being inverted. The

data set is artificial and created by a small C program. In the experiments, the LED24 data set consists of a data set of 1000 examples.

The Glass data set consists of glass samples from the scene of an accident. There are 9 features, 6 classes, and 214 examples. In this data set, two of the classes have quite few examples.

The Chess data set consists of 36 features and two classes determining whether white can or cannot win. Each pattern represents a board position. There are 3196 examples in this data set and the two classes are almost equally represented.

The data sets used in the experiments have either numeric or symbolic features. The implementation (section 3.1) has support for both numeric and symbolic features in the same data set. None of the data has interacting features (as far as I know).

3.3 Comparing GAs

As GAs are stochastic and each run is different in terms of the final solution, the convergence. Measuring speed-up can be done in two ways. The first way is letting the GA search converge, fully or to some degree. Convergence is defined as when the population consists of nearly identical individuals. The comparison is then made based on the run time.

The other is letting the GA run until some fitness value (such as the hit rate) has reached a pre-set threshold. The problem is that in time-bounded GAs, the search is stopped after a predetermined time and the search may not get the chance to converge. Also, the best individual's fitness value at each generation can be very approximate, so that it cannot be used in order to stop the GA.

Another way to measure speed up (or more generally time effectiveness) is using a finite time instead and then compare the GAs using the final solution what ever the value may be. In the case of the hybrid this means; run the hybrid for a bounded time. When it has been stopped, the final solution is the best individual's weight vector at the ending generation. The knn classifier accuracy obtained from that weight vector is the performance of the hybrid.

This is more natural when considering that there is a bounded time. In the experiments this approach has been chosen. So, the performance of the hybrid is the accuracy of the final knn classifier within a bounded time (computational cost).

Here, time is measured by counting the number of comparisons between

examples internally in the hybrid (in the knn classifier part). The counting of the comparisons is described further in section 3.1. In the experiments that should run a bounded amount of time (the sections 3.6 and 3.7), the time chosen was a time that is somewhat lower than the average time needed to for the population to converge. It was seen by previous experimentation that some runs found good solutions using that amount of time.

The training and test size used in the trials were chosen to sizes that were shown to work good in earlier trials.

3.4 Feature Extraction

In this experiment the hybrid is tested against the three data sets. The progress of one of the searches can be seen in figure 3.1. The plot of the best individuals fitness varies more than the average fitness score in the population.

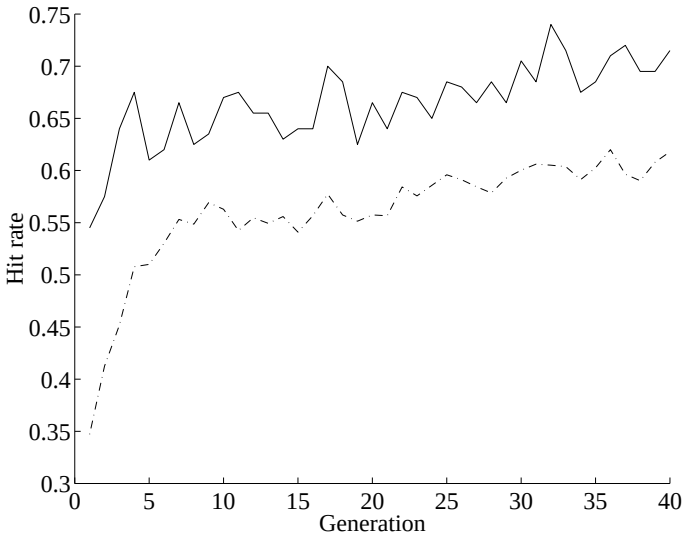

Figure 3.1: The solid plot shows the best individual in each successive generation, the dashed plot shows the average fitness score of the population.

From the final weights, figure 3.2, it can be seen that the irrelevant features (last 17 ones) has lower weights assigned to them. An ordinary knn (without weights) had the hit rate 0.484, and the best weight configuration achieved the hit rate 0.682. More statistics are shown in the appendix, in the sections A.1 and A.2.

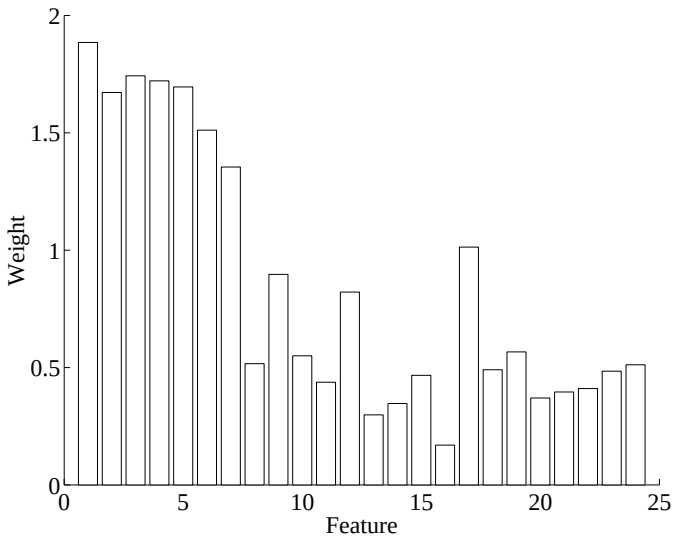

Figure 3.2: The first 7 weights (1-7) are higher than the last 17 ones (the irrelevant features).

Data Set	Non-weighted	Hybrid	Other Classifier
LED24	0.484	0.684	0.70 (CART)
Glass	0.682	0.754	0.639 (C4.5)
Chess	0.958	0.988	0.995 (C4.5)

Table 3.2: A comparison between the non-weighted (all weights are 1.0) and the hybrid.

Each weight configuration has been tested against the full data set and to reduce statistical variation the hit rate is the mean of 25 leave-one-out tests. Leave-one-out was used because other estimation methods for the hit rate did not work on the data set Glass. The hybrid has found weights that give better classification accuracy for all three data sets.

For the data set Glass all 214 examples were used. From the Chess data set (containing 3196 examples) 320 examples were randomly selected. A comparison between the non-weighted (all weights are 1.0), the hybrid, and other classification methods is shown in table 3.2. The other classification methods are CART and C4.5 which are decision tree algorithms. The values from CART and C4.5 comes from [BM98] and [KF94] respectively.

The Glass data set stands out when even a non-weighted knn yields better results than C4.5. Kelly et. al [KD91] also get a lower hit rate, 0.588, with a

non-weighted knn on this data set. One possibility for this difference is that the Glass data set has relatively few examples of two of its classes and the estimation of the accuracy could be overfitting the data set.

3.5 Random Search and Naïve Evolution

In section 2.2.6 the difference between mutation and crossover is discussed. It is not really clear how they perform on a task like this. Also, how well does a completely random search perform. Often in literature this type of questions is omitted.

This gives the motivation for an experiment that tests four search algorithms; random search, mutation only, crossover only, and GA with mutation and crossover. The random search has no feedback at all; it creates completely new random weight vectors and keeps the best weight vector. Mutation only and crossover only uses exactly the same algorithm as the hybrid only that in mutation only the crossover and average operators are disabled. In crossover the mutation and average operators are disabled. The fourth algorithm, labeled GA, is the hybrid used in all the other experiments.

The experiment is performed using 1600 evaluations of the fitness function for each of the four methods. For mutate only, crossover only, and GA 40 generations and a population size of 40 is used. For random search 1600 new weight vectors is created. Each algorithm is run 20 times and, at the end of the training, the best weight configuration was evaluated against the full data base of 1000 examples 25 times to reduce statistical variation. The mean and other statistics is presented in table 3.3.

Algorithm	Mean	Min	Max
Random Search	0.591	0.522	0.639
Mutation Only	0.660	0.627	0.694
Crossover Only	0.666	0.642	0.692
GA	0.667	0.645	0.690

Table 3.3: A summary over the performance for four search algorithms in the LED24 data set.

From table 3.3 the hybrid and crossover only performs best (when looking at their mean values) but mutate only is not far behind, random search performs worst. Notably, the random search sometimes can come up with competitive solutions (when looking at the max value of random search).

More statistics are shown in the appendix, section A.3, table A.4.

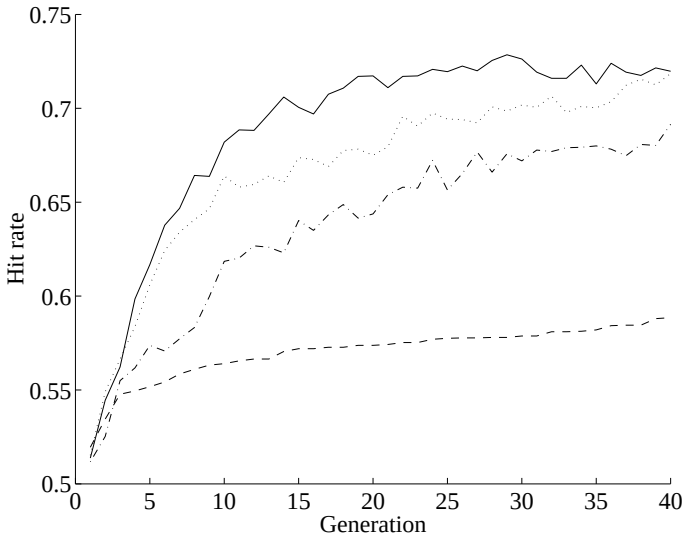

Figure 3.3: The plots shows the best individuals or solution found at each generation. The solid plot shows the crossover only approach, the dash-dotted plot shows the mutation only, the dotted shows the GA, and the dashed plot shows random search.

Plotting the hit rate from the best individual or solution found throughout the searches confirms that crossover and mutation have different properties, as mentioned in section 2.2.6; crossover is more efficient early in the search and mutation is more efficient early when the population begins to converge. The plots are shown in figure 3.3. As random search did not have any generations, the best solution for every new 40 evaluations is used instead. The crossover only search is more effective in the beginning of the search than the GA on this data set. It can be due to LED24 contains either meaningful or irrelevant features so that the average operator used in the GA for the hybrid becomes nearly useless.

3.6 Population Size

The population size affects convergence (section 2.4.1). A small population gives premature convergence and a large population gives slow convergence. The aim with the experiment was to empirically test the effect of various population sizes. The settings are shown in table 3.4. The population size was varied from 5 to 800 individuals. For each population size, 30 GA runs

were made. In the experiments, a separate training and test set is used to determine the accuracy of knn.

Data Set	Training	Test	Time
LED24	200	100	20M
Glass	150	64	15M
Chess	75	75	15M

Table 3.4: The training and test size, and time used in the three data sets.

Population Size	LED24	Glass	Chess
5	0.568	0.692	0.933
10	0.614	0.705	0.956
25	0.648	0.707	0.964
50	0.650	0.710	0.970
75	0.654	0.705	0.969
100	0.641	0.706	0.969
150	0.642	0.701	0.967
200	0.630	0.702	0.967
400	0.621	0.700	0.959
800	0.603	0.699	0.957

Table 3.5: The performance of the hybrid using various population sizes over three data sets.

As the GA could not be stopped within one generation, more fitness evaluations were made for some of the runs. The runs with the highest number of fitness evaluations were with the largest population sizes. So, the performance of the hybrid using population sizes of 400–800 individuals are somewhat higher than it should be. This does not change any of the interpretations of the values.

It can be seen from table 3.5 that the population size has some impact on the performance of the hybrid. The hybrid performs best using 25–75 individuals for the data sets LED24 and Glass. For the Chess data set, the hybrid performs best using 50–200 individuals. In the figure 3.4, it can be seen that the performance of the hybrid quickly increases with the size of the population, then reaches a maximum, and then slowly sinks. More statistics are shown in the appendix, in section A.4, in the tables A.5, A.6, and A.7.

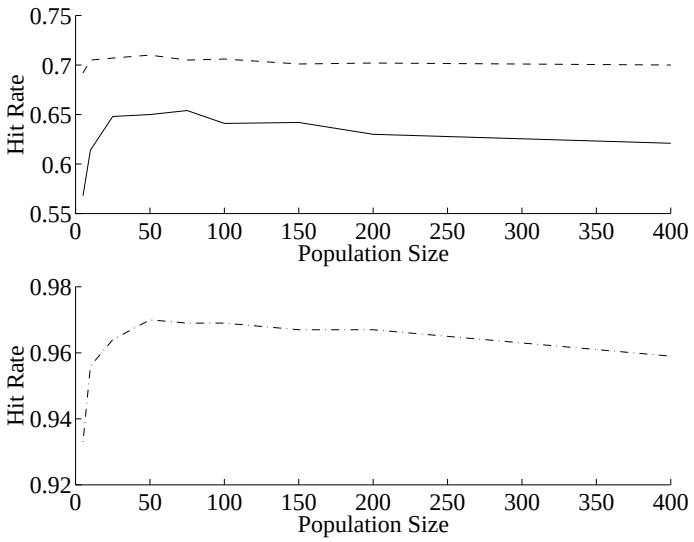

Figure 3.4: Three plots over the performance of the hybrid using various population sizes. The solid plot shows LED24, the dashed shows Glass, and the dash dotted shows Chess.

Unfortunately, the population sizing models described in [Mil97] are problematic to apply because the lack of domain knowledge. Also, the models are not made for a real-number representation of the chromosomes. One trial is made using a "thumb of rule" population sizing approximation. It is defined as,

$$N' = -\frac{2^{k_{max}-1} ln(\psi)\sqrt{\pi}}{d_{min}}\sqrt{\sigma_F^2 + \sigma_N^2}, \tag{3.1}$$

where,
N' is the population size,
σ_F^2 is the population real fitness variance,
σ_N^2 is the noise fitness variance,
k_{max} is an estimate of the maximal length of the building blocks,
d_{min} is the minimum building block signal that the GA is able to detect,
ψ is the failure rate, that is the probability that an individual fails to converge to the optimal solution.

The equation 3.1 is derived from a random walk model of the GA. In the OneMax domain, both k_{max} and d_{min} are 1, and the failure rate ψ is 0.01.

The failure rate, ψ, is in practice often set to a small value, such as 0.01. It represents the probability that an individual fails to converge to the optimal solution. However, the as the time bound used in the experiments for the hybrid were limited, a higher value of that parameter is more likely.

The ψ was set to 0.02.

The d_{min} needs some further explanation. Any chromosome within d_{min} of the highest achievable hit rate (true optimal) is considered an acceptable solution. For the hybrid, d_{min} is set to 0.02 for the LED24 data set.

The k_{max} is an estimate of the maximal length of the building block. One heuristic is to take the maximum number of variables that are interdependent and set k_{max} to the number of bits that represents the variables. For the real number encoded chromosomes used in the hybrid, none of the variables (features) could be said to be interdependent. As the one feature is represented by one value, one guess is to use the value 1 for k_{max} in this case.

The sum, $\sigma_F^2+\sigma_N^2$, can be determined directly from the initial population, using the variance of the fitness of the individuals. For the LED24 data set, $\sigma_F^2 + \sigma_N^2$ was found to be 0.007. This gives $N' = 29$ in the LED24 data set using $\psi = 0.02$, $k_{max} = 1$, and $d_{min} = 0.02$. In table 3.5, using 25 individuals is not bad, but the performance is slightly higher using 75 individuals.

The same calculation is made for the Chess and Glass data sets. The only difference is that d_{min} is set 0.01 for the Glass data set and to 0.005 for the Chess data set. The reason for this is that d_{min} controls the width of the interval where the solution is accepted and for these two data sets that width was decreased. In the Glass and Chess data sets, the $\sigma_F^2 + \sigma_N^2$ was found to be 0.003 and 0.004 respectively. In the Glass data set, using $\psi = 0.02$, $k_{max} = 1$, $d_{min} = 0.01$, N' was calculated to 38. For the Chess data set, using $\psi = 0.02$, $k_{max} = 1$, $d_{min} = 0.005$, results in $N' = 88$.

The values for N' agrees with both the data sets. The hybrid performance is best for the Glass data set using 50 individuals. In the Chess data set, the hybrid performance is best using 50–100 individuals.

Even though equation 3.1 seems to give good predictions, most of the settings of the parameters are guesses. The ψ was set to 0.02 throughout the calculations. That parameter should probably have a much higher value.

3.7 Varying the Number of Examples

One way to speed up the evaluation of individuals is simply to use fewer examples from the data set. Refer to the methods described in 2.4.1. In the following experiments the performance of the hybrid is empirically tested using limited training and test size.

In the first section, experiments are performed with varying test sizes. This is closely related to varying sample sizes described in described in 2.4.1.

In the second section, experiments are performed varying the training sizes instead. It is interesting to see how the hybrid handles a varying training size as they can give a more random impact on the accuracy from the knn classifier. There are no general answers on how the training size affects the accuracy of knn [DHS01].

In the experiments, a separate training and test set is used to determine the accuracy of the knn. The leave-one-out changes both the training and test size. Here, we want to distinguish the effects from various sizes on the training and test set.

3.7.1 Test Examples

The aim of this experiment is to test how the test size affects the performance of the hybrid using a fixed training time. The settings are shown in table 3.6.

Data Set	Training	Time
LED24	50	20M
Chess	75	15M

Table 3.6: The training size, and time used in the two data sets.

Test Examples	LED24	Chess
10	0.620	0.950
20	0.641	0.965
30	0.647	0.968
40	0.657	0.969
50	0.660	0.964
75	0.669	0.969
100	0.669	0.966
150	0.667	0.968
200	0.667	0.968
400	0.658	0.964
800	0.644	0.958

Table 3.7: The performance of the hybrid with various test sizes.

The results from the experiment is presented in table 3.7 and in figure 3.5. The Glass data set has few examples, so that varying the test size also

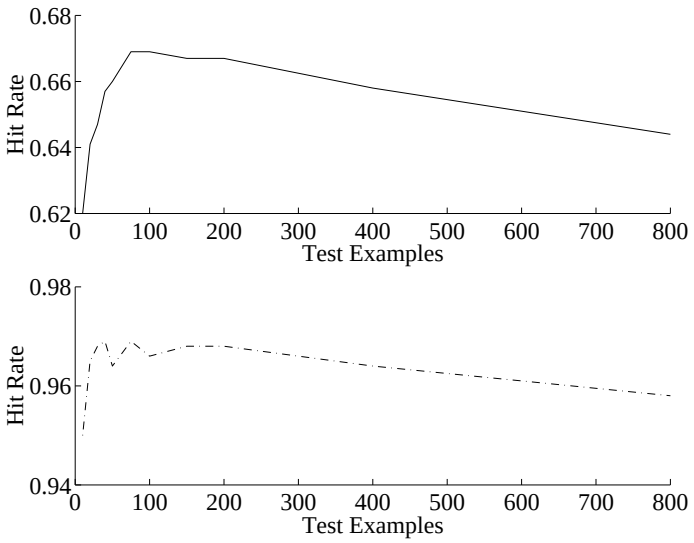

Figure 3.5: Two plots over the performance of the hybrid using various test sizes. The solid plot shows LED24, and the dash dotted shows Chess.

changes the training size. The results of the hybrid using the Glass data set is shown in section 3.7.2. The values are the average of 30 runs.

With LED24 data set, 75–200 test examples give the best hybrid performance. With Chess data set, 30–200 test examples give the best hybrid performance. However, on the Chess data set the performance sometimes drops (using 50 test examples). This can be due to statistical variations. More statistics are shown in the appendix, in section A.5, in the tables A.9, and A.10.

Unfortunately, predicting the optimal sample size using the methods described in[Mil97] were hard. Both the fitness variance and the noise variance must be supplied. In the population sizing approximation the sum of these two could be estimated from the population, section 3.6. Here, the ratio between the two variances is used instead, so both of them must be known. It has not been possible to calculate them.

3.7.2 Training Examples

The aim of this experiment is to test how the training size affects the performance of the hybrid using a fixed training time.

The settings; test size and time, is shown in table 3.8. The Glass data set only had 214 examples, so the examples in that data set that were not used

Data Set	Test	Time
LED24	100	20M
Glass	$214 - t$	15M
Chess	100	15M

Table 3.8: The test size and time used in the three data sets. The test examples of the Glass data set are $214 - t$, where t is the training size.

Training Examples	LED24	Glass	Chess
5	0.526		
10	0.602	0.692	0.956
20	0.651	0.694	0.963
30	0.659	0.695	0.967
40	0.669	0.699	0.967
50	0.670	0.697	0.970
75	0.664	0.701	0.969
100	0.663	0.701	0.970
150	0.653	0.707	0.965
200	0.653	0.706	0.967
400	0.635		0.962

Table 3.9: The performance of the hybrid using various training sizes.

for training were used as test examples instead. The values are the average of 30 runs. More statistics are shown in the appendix, in section A.6, in the tables A.12, A.13, and A.14.

The results from the experiment is presented in table 3.9, and in figure 3.6. With LED24, the results show that the GA is quite robust when it comes to the training size. 20–200 training examples give about the same performance. In the Chess data set, the hybrid performs best with 50–100 training examples. However, the differences in performance are very small. In LED24 and Chess data sets, it can be seen that the performance of the hybrid quickly increases with the training size and then it slowly decreases.

The Glass data set gives different results; the hybrid performs slowly better using more training examples. As the test size varied with the training size, it cannot be compared against the runs with the data sets LED24 and Chess.

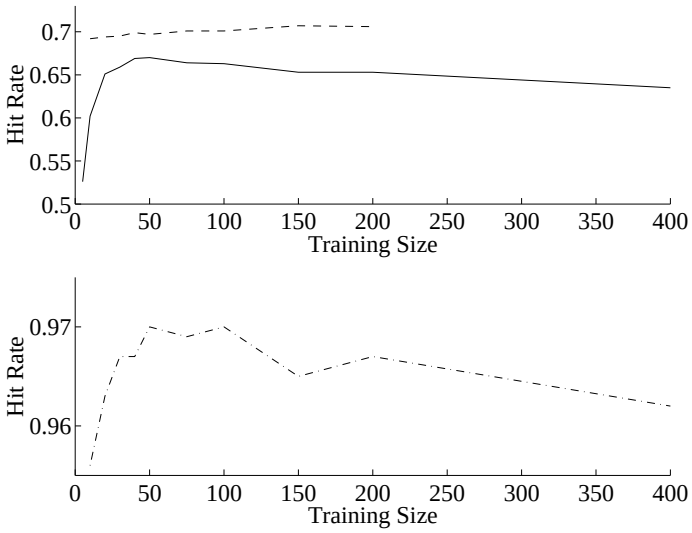

Figure 3.6: Three plots over the performance of the hybrid using various training sizes. The solid plot shows LED24, the dashed shows Glass, and the dash dotted shows Chess.

Chapter 4

Conclusions

The conclusions from the results obtained in the experiments are discussed in the following sections.

4.1 Feature Extraction

The hybrid was successful in feature extraction (section 3.4). The classification accuracy for the three data sets tested was increased relative to standard knn and comparable to other classification methods. The hybrid can also be used in feature selection tasks where features with lower weights are less important than the ones with higher weights. Also, data mining is one other area where the hybrid can be used.

4.2 Random Search and Naïve Evolution

The naïve evolution, using mutation only, compared to crossover only confirms what is mentioned in section 2.2.6, that they have different properties. Crossover is more effective early in the search and mutation more effective at the end of the searches. It also shows that completely random search is far behind in comparison with GAs, even though it sometimes can come up with good solutions.

It should also be said that algorithms tailored for a specific task (such as traveling salesman) can be more effective than GAs. Further comparisons with other techniques than GA would have been interesting to do, both in terms of the time efficiency and classifier accuracy.

4.3 Time Efficiency

The time efficiency is an overall problem for GAs. In order to reduce the time, methods such as noisy GAs (section 2.4.1) and scheduling (section 2.4.2) has been used when dealing with noisy fitness functions. These methods use models of the GA. In order to apply the methods, some domain knowledge must be known in order to set some of the parameters.

The empirical test on population size shows that it has a big impact on the performance of the hybrid, especially time bounded GAs as it affects early/slow convergence. One trial is made in predicting the optimal population size for the three data sets used in the experiments. Even though the predictions are close, they are dependent on parameters whose values are mostly guessed.

The experiments were using various training and test set sizes. They showed that there are a number of examples that give an optimal performance of the hybrid. The plots over the performance of the hybrid using various test sizes has the same appearance as the ones in Millers work, where the sample size is compared against GA performance measures [Mil97]. The optimal test sizes are hard to predict because the lack of knowledge for fitness and noise variance. The experiments with various training sizes were done in order to study how the hybrid behaves using various training sizes. It is shown that there are an optimal training size for the data sets Glass and Chess.

Even with extensive domain knowledge, the problem of deciding which sampling size, which controls the accuracy of the fitness function, is complex. As there are tradeoffs between solution qualities (the hit rate obtained) and additional overhead in time of the fitness function.

In the section 2.4.2, the method of sample-allocation is described; where it is more important to distinguish between more fit individuals than the less fit ones. The methods described are interesting. However, applying them to the hybrid would take both some hard work and time. It is the same problem here as in determining the optimal test size, namely the lack of knowledge for fitness and noise variance.

In further work, applying the research for noisy GAs, binary encoding in the hybrid could a better choice because then the models for noisy GAs are easier to apply. In order to apply the research for noisy GAs, the GA should be designed (the same choice of selection scheme and so forth) as in the experiments using the Onemax domain.

Bibliography

[AW93] Akiko N. Aizawa and Benjamin W. Wah. Scheduling of genetic algorithms in a noisy environment. In Stephanie Forrest, editor, *Proceedings of the Fifth International Conference on Genetic Algorithms*, pages 48–55, San Mateo, CA, 1993. Morgan Kaufman. `http://citeseer.nj.nec.com/aizawa94scheduling.html`.

[BBM93a] David Beasley, David R. Bull, and Ralph R. Martin. An overview of genetic algorithms: Part 1, research topics, 1993. `http://citeseer.nj.nec.com/16527.html`.

[BBM93b] David Beasley, David R. Bull, and Ralph R. Martin. An overview of genetic algorithms: Part 2, research topics, 1993. `http://citeseer.nj.nec.com/beasley93overview.html`.

[BM98] C.L. Blake and C.J. Merz. UCI repository of machine learning databases, 1998. `http://www.ics.uci.edu/~mlearn/MLRepository.html`.

[DHS01] Duda, Hart, and Stork. *Pattern Classification.* Wiley, 2001.

[DJC94] Michie D., Spiegelhalter D. J., and Taylor C. C. *Machine Learning, Neural and Statistical Classification.* Ellis Horwood, New York, 1994. `http://www.amsta.leeds.ac.uk/ charles/statlog/`.

[JC82] A. Jain and B. Chandrasekaran. Handbook of statistics, volume 2. In P. R. Krishnaiah and L. N. Kanal, editors, *Case–Based Reasoning, Research and Development, First International Conference*, pages 835–855, Amsterdam, Holland, 1982. North–Holland.

[KD91] James D. Kelly and Lawrence Davis. A hybrid genetic algorithm for classification. In Morgan Kaufmann, editor, *Proceedings of the*

Twelfth International Joint Conference on Artificial Intelligence, 1991.

[KF94] R. Kohavi and B. Frasca. Useful feature subsets and rough set reducts, 1994. `citeseer.nj.nec.com/kohavi94useful.html`.

[Mil97] Brad L. Miller. Noise, sampling, and efficient genetic algorithms. Technical Report 1997, Urbana, Illinois, 1997. `http://citeseer.nj.nec.com/miller97noise.html`.

[Mit98] Melanie Mitchell. *An Introduction to Genetic Algorithms*. MIT Press, Cambridge, Massachusetts, 1998.

[RPG+00] Raymer, Punch, Goodman, Kuhn, and Jain. Dimensionality reduction using genetic algorithms. *IEEETEC: IEEE Transactions on Evolutionary Computation, A Publication of the IEEE Neural Networks Council*, 4, 2000. `http://citeseer.nj.nec.com/raymer00dimensionality.html`.

[Spe93] William M. Spears. Crossover or mutation? In L. Darrell Whitley, editor, *Foundations of Genetic Algorithms 2*, pages 221–237. Morgan Kaufmann, San Mateo, CA, 1993. `http://citeseer.nj.nec.com/spears92crossover.html`.

[WAM97] Dietrich Wettschereck, David W. Aha, and Takao Mohri. A review and empirical evaluation of feature weighting methods for a class of lazy learning algorithms. *Artificial Intelligence Review*, 11(1-5):273–314, 1997. `http://citeseer.nj.nec.com/wettschereck97review.html`.

[WM96] D. Wilson and T. Martinez. Instance-based learning with genetically derived attribute weights, 1996. `http://citeseer.nj.nec.com/wilson96instancebased.html`.

Appendix A

Results from the Hybrid

The E in some of the following tables is a 95%-confidence interval. One example can be; in table A.1, the LED24 has a mean of 0.4836. The 95%-confidence interval is then, 0.4836±0.0047. The 95%-confidence interval expresses the probability that the value, in this case the estimation of the hit rate for the LED24 data set, is in the interval [0.4836-0.0047, 0.4836+0.0047] with a probability of 0.95. The Std. in some of the following tables is the sampled standard deviation.

The k in knn was set to 3 throughout the experiments.

A.1 Feature Extraction

The values are calculated from 25 leave-one-out estimations of the hit rate.

Data Set	Mean	Std.	Min	Max	E
LED24	0.4836	0.0130	0.4580	0.5110	0.0047
Glass	0.6817	0.0047	0.6729	0.6916	0.0017
Chess	0.9582	0.0021	0.9540	0.9625	0.0008

Table A.1: The accuracy of the knn classifier using non-weighted features.

In table A.3, the final evaluation of the hit rate used the full data set "Size". In the training of the hybrid, the "Eval. Set" were used instead.

Data Set	Mean	Std.	Min	Max	E
LED24	0.6842	0.0043	0.6760	0.6940	0.0015
Glass	0.7535	0.0056	0.7430	0.7664	0.0020
Chess	0.9875	0.0000	0.9875	0.9875	0.0000

Table A.2: The accuracy of the knn classifier using weighted features.

Data Set	Size	Eval. Set	Generations	Pop. Size
LED24	1000	200	40	40
Glass	214	214	40	40
Chess	3196	320	40	40

Table A.3: The settings for the hybrid in feature extraction (the runs in table A.2).

A.2 Weight Vectors

Those are the weight vectors that were shown to give the best knn classifier accuracy.

LED24:
{1.88513, 1.67205, 1.74281, 1.72123, 1.69561, 1.51154, 1.3542, 0.516513, 0.896897, 0.549906, 0.437511, 0.821521, 0.298298, 0.346756, 0.46686, 0.16972, 1.01322, 0.490502, 0.566567, 0.3702, 0.39595, 0.410363, 0.484484, 0.5116}

Glass:
{0.022022, 0.155405, 1.42949, 1.80287, 0.11549, 1.88213, 0.0397304, 1.78482, 1.56993}

Chess:
{1.38201, 0.980731, 1.28948, 0.538038, 0.283133, 1.00946, 1.11856, 0.299675, 0.270958, 1.52953, 0.568474, 0.544858, 0.264264, 0.731568, 1.10512, 1.32974, 1.51471, 0.424424, 0.416416, 0.540541, 2.06882, 0.55956, 0.326326, 0.217342, 1.17565, 0.533934, 1.19309, 0.762401, 0.22973, 1.08941, 0.404404, 1.57021, 2.13002, 0.578704, 0.929195, 0.236737}

A.3 Random Search

The results are from 20 runs, each run's last population's best individual has its value from 25 evaluations against the full LED24 data set. The evaluation method of the weight vectors in the search methods were leave-one-out on 200 examples.

Search Method	Mean	Std.	Min	Max	E
GA	0.6674	0.0125	0.6450	0.6900	0.0055
Crossover	0.6661	0.0124	0.6421	0.6920	0.0054
Mutate	0.6596	0.0154	0.6274	0.6944	0.0067
Random	0.5912	0.0327	0.5220	0.6390	0.0143

Table A.4: The performance for four search algorithms (types of GAs and random search) in the LED24 data set.

A.4 Varying Population

The values are calculated from 30 runs. Evaluation method of the individuals in the hybrid was separate test and train sets.

Population Size	Mean	Std.	Min	Max	E
5	0.5677	0.0475	0.4500	0.6580	0.0170
10	0.6136	0.0311	0.5350	0.6710	0.0111
25	0.6476	0.0215	0.5880	0.6800	0.0077
50	0.6502	0.0174	0.6110	0.6810	0.0062
75	0.6544	0.0127	0.6350	0.6770	0.0045
100	0.6406	0.0245	0.5990	0.6840	0.0088
150	0.6420	0.0226	0.5980	0.6750	0.0081
200	0.6299	0.0276	0.5570	0.6580	0.0099
400	0.6210	0.0207	0.5660	0.6570	0.0074
800	0.6028	0.0345	0.5090	0.6660	0.0124

Table A.5: Varying population sizes in the hybrid for the LED24 data set.

Population Size	Mean	Std.	Min	Max	E
5	0.6919	0.0138	0.6636	0.7196	0.0049
10	0.7045	0.0153	0.6776	0.7290	0.0055
25	0.7067	0.0184	0.6682	0.7430	0.0066
50	0.7098	0.0134	0.6822	0.7336	0.0048
75	0.7050	0.0131	0.6776	0.7290	0.0047
100	0.7053	0.0140	0.6776	0.7336	0.0050
150	0.7008	0.0122	0.6682	0.7336	0.0044
200	0.7016	0.0122	0.6822	0.7196	0.0044
400	0.7003	0.0162	0.6636	0.7290	0.0058
800	0.6991	0.0165	0.6682	0.7290	0.0059

Table A.6: Varying population sizes in the hybrid for the Glass data set.

Population Size	Mean	Std.	Min	Max	E
5	0.9330	0.0274	0.8661	0.9618	0.0098
10	0.9556	0.0142	0.9105	0.9743	0.0051
25	0.9644	0.0096	0.9406	0.9862	0.0034
50	0.9701	0.0069	0.9524	0.9819	0.0025
75	0.9686	0.0072	0.9543	0.9825	0.0026
100	0.9691	0.0067	0.9518	0.9806	0.0024
150	0.9671	0.0058	0.9543	0.9775	0.0021
200	0.9674	0.0061	0.9562	0.9793	0.0022
400	0.9591	0.0079	0.9418	0.9718	0.0028
800	0.9574	0.0090	0.9337	0.9712	0.0032

Table A.7: Varying population sizes in the hybrid for the Chess data set.

Data Set	Training Size	Test Size	Time
LED24	200	100	20M
Glass	150	64	15M
Chess	75	75	15M

Table A.8: The settings for the hybrid in varying population sizes for the three data sets.

A.5 Varying Test Size

The values are calculated from 30 runs. Evaluation method of the individuals in the hybrid was separate test and train sets.

Test Size	Mean	Std.	Min	Max	E
10	0.6197	0.0372	0.4910	0.6660	0.0133
20	0.6407	0.0254	0.5720	0.6780	0.0091
30	0.6468	0.0188	0.6100	0.6810	0.0067
40	0.6567	0.0177	0.6060	0.6780	0.0063
50	0.6602	0.0238	0.5540	0.6870	0.0085
75	0.6685	0.0124	0.6360	0.6910	0.0044
100	0.6689	0.0112	0.6410	0.6900	0.0040
150	0.6671	0.0201	0.5760	0.6900	0.0072
200	0.6673	0.0095	0.6450	0.6890	0.0034
400	0.6576	0.0130	0.6240	0.6820	0.0047
800	0.6438	0.0132	0.6170	0.6710	0.0047

Table A.9: Varying the test size used in the hybrid for the LED24 data set.

Test Size	Mean	Std.	Min	Max	E
10	0.9502	0.0190	0.8874	0.9775	0.0068
20	0.9646	0.0090	0.9456	0.9819	0.0032
30	0.9683	0.0077	0.9531	0.9831	0.0027
40	0.9691	0.0091	0.9406	0.9800	0.0033
50	0.9643	0.0078	0.9468	0.9750	0.0028
75	0.9685	0.0065	0.9481	0.9800	0.0023
100	0.9664	0.0081	0.9468	0.9793	0.0029
150	0.9676	0.0066	0.9512	0.9781	0.0024
200	0.9683	0.0079	0.9556	0.9806	0.0028
400	0.9641	0.0070	0.9524	0.9793	0.0025
800	0.9584	0.0096	0.9374	0.9737	0.0034

Table A.10: Varying the test size used in the hybrid for the Chess data set.

Data Set	Training Size	Population Size	Time
LED24	50	50	20M
Chess	75	50	15M

Table A.11: The settings for the hybrid in varying test size used for the data sets LED24 and Chess.

A.6 Varying Training Size

The values are calculated from 30 runs. Evaluation method of the individuals in the hybrid was separate test and train sets.

Training Size	Mean	Std.	Min	Max	E
5	0.5261	0.0487	0.3540	0.5990	0.0174
10	0.6021	0.0334	0.4980	0.6490	0.0119
20	0.6511	0.0193	0.5960	0.6810	0.0069
30	0.6587	0.0132	0.6310	0.6840	0.0047
40	0.6687	0.0133	0.6250	0.6930	0.0048
50	0.6695	0.0127	0.6420	0.6960	0.0045
75	0.6637	0.0144	0.6060	0.6820	0.0052
100	0.6629	0.0156	0.6150	0.6980	0.0056
150	0.6533	0.0118	0.6330	0.6790	0.0042
200	0.6526	0.0144	0.6130	0.6800	0.0051
400	0.6347	0.0142	0.6090	0.6590	0.0051

Table A.12: Varying the training size used in the hybrid for the LED24 data set.

Training Size	Mean	Std.	Min	Max	E
10	0.6919	0.0182	0.6449	0.7243	0.0065
20	0.6941	0.0161	0.6636	0.7196	0.0058
30	0.6949	0.0125	0.6729	0.7196	0.0045
40	0.6994	0.0151	0.6776	0.7430	0.0054
50	0.6967	0.0111	0.6729	0.7196	0.0040
75	0.7012	0.0155	0.6682	0.7430	0.0056
100	0.7014	0.0107	0.6822	0.7196	0.0038
150	0.7072	0.0136	0.6822	0.7383	0.0049
200	0.7058	0.0128	0.6729	0.7336	0.0046

Table A.13: Varying the training size used in the hybrid for the Glass data set.

Training Size	Mean	Std.	Min	Max	E
10	0.9558	0.0091	0.9355	0.9681	0.0032
20	0.9630	0.0073	0.9474	0.9743	0.0026
30	0.9674	0.0090	0.9406	0.9812	0.0032
40	0.9674	0.0074	0.9468	0.9793	0.0026
50	0.9700	0.0069	0.9543	0.9850	0.0025
75	0.9689	0.0078	0.9449	0.9800	0.0028
100	0.9698	0.0065	0.9549	0.9806	0.0023
150	0.9654	0.0157	0.8949	0.9812	0.0056
200	0.9670	0.0078	0.9468	0.9856	0.0028
400	0.9621	0.0086	0.9431	0.9781	0.0031

Table A.14: Varying the training size used in the hybrid for the Chess data set.

Data Set	Test Size	Population Size	Time
LED24	100	50	20M
Glass	214 - training size	50	15M
Chess	100	50	15M

Table A.15: The settings for the hybrid in varying the training size used for the three data sets.