Feature Selection for Support Vector Machines by Means of Genetic Algorithms

-Diploma Thesis in Computer Science-

Holger Fröhlich
Wilhelm-Heusel-Str. 10
72074 Tübingen
email: H.Froehlich@gmx.de

Supervisors:

Prof. Dr. A. Ultsch
Department of Mathematics and Computer Science
Philipps-University Marburg

Prof. Dr. B. Schölkopf
Max-Planck-Institute for biological Cybernetics

21st December 2002

Contents

Chapter 1

Introduction

This Diploma Thesis deals with the problem of feature selection for Support Vector Machines by means of Genetic Algorithms. The problem of feature selection is an important issue in Machine Learning and of great practical importance, e.g. in bioinformatics. Similar to a human being a machine can increase its estimated learning performance by filtering out irrelevant features of the data to learn. For example if one wants to learn to discriminate tables from chairs in general the color is completely irrelevant and thus the feature “Color” should not be taken into consideration for the classification of an unknown object. Another problem evolves in bioinformatics: Suppose we want to detect cancer cells by means of their genetic properties. Notice that the human DNA has millions of genes. So the arising question is: Which of these genes are really relevant for the classification of a cell as “cancer” or “not cancer”? Do we need 10, 50, 100 or 10,000 genes to solve this task? The question which is behind that is, what the term “relevance” actually means and how relevant features can be detected. These are the two fundamental problems of feature selection. It will turn out that the detection of relevant features is a difficult combinatorial task. Feature **selection** should not be confused with feature **extraction** as it is done e.g. in Principal Component Analysis (e.g. [Duda73]) or Independent Component Analysis (e.g. [KwaChoCho01]).

The problem of feature selection is deeply connected with Machine Learning in general. An important question in feature selection is, how one can estimate the learning performance of a machine, i.e. how well the machine can *generalize*. An often used metaphor (see e.g.[RusNor95, p. 528]) to explain what is meant by that is the following: A learner always acts within an environment from which he can observe certain things. Of course the learner is aware of what he is observing right now and what he observed in the past, but not what he will observe in future. Every action the learner is taking causes a certain result, which itself might be perceptible or not. The goal of learning is to construct a model or a *hypothesis* of the environment based on received perceptions, such that he can predict the result of future actions. Thus the hypothesis is a *generalization* of perceptions made in past to any point in time.

A Support Vector Machine (SVM [BosGuyVap92]) is a learning algorithm which has become quite popular during the last years. The concept of SVMs evolved from insights of Statistical Learning Theory which was mainly developed by V. Vapnik and A. Chervonenkis [VapChe64, VapChe74, VapChe79, VapLer63, Vapnik79, Vapnik95, Vapnik98]. An important property of SVMs is that there exist theoretical upper bounds on the generalization error for them. These bounds can be used to estimate the generalization performance of a SVM. Compared to general estimation methods like cross-validation or bootstrapping they are less accurate, but computationally attractive. Additionally, when used within a Genetic Algorithm, they reduce the potential danger of *overfitting*, i.e. the machine does not generalize well, because the learned hypothesis is somehow to special.

Genetic Algorithms have been developed in the 70s by John Holland and his students [Holland75, DeJong75]. They have been established as a general purpose tool for difficult optimization problems and are thus applicable to the feature selection problem. This has been done before, e.g. in combination with decision trees and cross-validation [FerKadKit93, BriBroMar92, RicLan96, RayPunGooKuhJai00]. As in this thesis SVMs are used for classification, Genetic Algorithms will be used in combination with the theoretical bounds on the generalization error. This new approach can be compared to the traditional one with cross-validation as well as to other existing algorithms for feature selection on artificial and real life data sets. The real life data sets are DNA micro array data. As it was already pointed out above, especially for these very high dimensional data the problem of feature selection is of great practical interest.

This thesis is organized as follows: In the next chapter the theoretical background of this thesis is clarified. First the central terms “generalization” and “overfitting” are defined which are necessary to understand the rest of this thesis. For this purpose we have to take a short view on pattern recognition in the light of Statistical Learning Theory. Afterwards a short introduction into Support Vector Machines is given and different methods to estimate the generalization capability of a SVM are described. Having this knowledge we can put our focus on the question of feature selection. Finally Genetic Algorithms are introduced.

In chapter 3 three existing algorithms for feature selection are described, before in chapter 4 we move on to Genetic Algorithms for feature selection. The different Genetic Algorithms are compared to each other and to existing algorithms in chapter 5. Chapter 6 contains a discussion and a summary of the experimental results, before in chapter 7 there is a general conclusion of this work.

Chapter 2

Theoretical Background

2.1 General Prerequisites — Pattern Recognition in Statistical Learning Theory

In Machine Learning generally one distinguishes between *supervised* and *unsupervised learning*. Supervised learning is distinguished into *corrective learning* and *reinforcement learning* (e.g. [RusNor95, Rojas96]). In *corrective learning* the learner has full access to both, received observations and the obtained results of every action. This is often compared to a friendly teacher who is providing you with exercises and the right answers. In *reinforcement learning* the learner is only provided with an evaluation of every action he is taking and not with the correct results. This is comparable to a teacher who doesn't tell the correct answers, but is only giving grades on how well you are doing. In unsupervised learning the learner doesn't have access to the results of his actions at all. That means he only can learn relationships between observations.

In Statistical Learning Theory (see e.g. [SchSmo02, pp. 6 - 11, pp. 125 - 146]) one is primarily interested in corrective learning. Hence the learner receives a set of training examples $\mathcal{E} = \{\,(x_i, y_i)|\, x_i \in \mathcal{X}, y_i \in \mathcal{Y}, i = 1, ..., n\}$, where $\mathcal{X}$ is some nonempty set of *patterns* (*cases, inputs, instances* or *observations*) and $\mathcal{Y}$ is a nonempty set of *targets* (*outputs*). Here $\mathcal{Y}$ will be considered as a finite set of *class labels*. That means we are trying to learn a *classification* of the training patterns $x_1, ..., x_n$. This situation is also commonly referred to as *pattern recognition.*

The standard assumption in Statistical Learning Theory is, that the data $\mathcal{E}$ is generated independently from some unknown (but fixed) probability distribution $\mathcal{P}(x, y)$. The goal is to construct a *decision function* (or *hypothesis*) f based on the training data $\mathcal{E}$ that will correctly classify as many unseen examples (x, y) as possible, which are drawn from the same distribution $\mathcal{P}(x, y)$, such that $f(x) = y$[1]. This is called *generalization.* We are trying to infer f from our training samples such that the hypothesis learned from these samples can be generalized to any unseen sample drawn from the same distribution. For this purpose one needs to make use of an *induction principle.* The problem is that f could be chosen from any class of functions, and we have only our data $\mathcal{E}$ which is given. We could easily choose f such that it fits perfectly all training points (e.g. in a 1D classification problem by means of a polynomial function of degree $n - 1$ — provided that $x_i \neq x_j$ for $y_i \neq y_j$ for all $i, j \in \{1, ..., n\}$). That means $f(x_i) = y_i$ for $i = 1, ..., n$. But that doesn't guarantee that for some unseen example (x, y) $f(x) = y$ (figure 2.1).

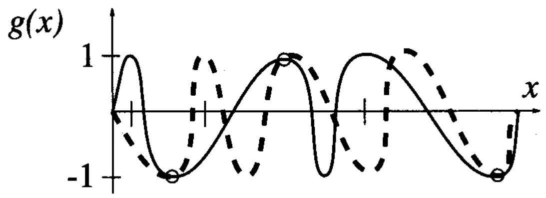

Figure 2.1: A 1D classification problem with a set of three training points (marked by circles) and test points (marked as lines on the x-axis): disriminate class -1 from class +1. Classification is performed by thresholding a real valued function g such that $f(x) = \text{sgn}(g(x))$. *Both* functions (dotted line, solid line) perfectly explain the training data, but they give opposite predictions on the test inputs. Lacking further information, there is no way to decide, which function to take (source: [SchSmo02]).

[1] i.e. $f(x) = \arg\max_y \mathcal{P}(x, y)$

To put it more formally, let us consider we have some nonnegative *loss function* $\ell : \mathcal{X} \times \mathcal{Y} \times \mathcal{Y} \to [0, \infty[$ with the property $\ell(x, y, y) = 0$ for all $x \in \mathcal{X}, y \in \mathcal{Y}$ which gives us some measure for the error we make by predicting some sample (x, y) (drawn from $\mathcal{P}(x, y)$) by means of our classifier f [SchSmo02, pp. 62 and following]. The simplest choice for a loss function would be

$$\ell(x, y, f(x)) \quad = \quad \begin{cases} 0 & \text{if } y = f(x) \\ 1 & \text{otherwise} \end{cases} \tag{2.1}$$

A crucial insight of Statistical Learning Theory is that minimizing the average *training error* (or *empirical risk*, see e.g. [Vapnik98])

$$R_{emp}[f] = \frac{1}{n} \sum_{i=1}^{n} \ell\left(x_i, y_i, f(x_i)\right) \tag{2.2}$$

does not imply a small expected *generalization error* (or *risk*) over all possible patterns drawn from the underlying distribution $\mathcal{P}(x, y)$

$$R[f] = \int_{\mathcal{X} \times \mathcal{Y}} \ell\left(x, y, f(x)\right) d\mathcal{P}(x, y) \tag{2.3}$$

Note that expression (2.3) is actually not computable, since we don't know $\mathcal{P}$. The term *overfitting* refers to the situation where R_{emp} is very low or maybe 0, but R is very high. That means that the learned hypothesis covers very well all special cases provided by our training examples $\mathcal{E}$, but is somehow not general enough to cover other data. On the other hand we have to avoid a hypothesis which is so vague that it can explain neither our training examples nor unseen test data sufficiently, i.e. we get a high empirical risk (*underfitting*). To avoid this Vapnik [Vapnik95, Vapnik98] suggested to restrict the set of functions (or *concepts*) $\mathcal{C}$ from which f is chosen to one that has a suitable *capacity* for the amount of data available. One of the best known capacity concepts is the so called *VC (Vapnik-Chervonenkis) dimension* (e.g. [Vapnik98, SchSmo02, Vapnik95]). The VC dimension is defined as the largest number l, such that there exists a set of l training points which a given class of concepts can *shatter*, and ∞ if no such l exists. Here a set of l points is said to be shattered by a class of functions $\mathcal{C}$, if one can realize all possible separations of the points, which are induced by any possible combinations of labeling the patterns, by means of a function from $\mathcal{C}$. For example if there are two classes +1 and -1, there are $2^3 = 8$ possible ways to label 3 training patterns. Suppose all training patterns are in $\mathbb{R}^2$ and are non-collinear, each possibility can be realized using a separating hyperplane. That means the class of hyperplanes is able to shatter 3 points. This would not work for 4 points, no matter how we placed them. Therefore the VC dimension of the class of hyperplanes in $\mathbb{R}^2$ is 3. A class of concepts is *learnable* only, if its VC dimension is finite.

Let $h < n$ be the VC dimension of the class of functions that the learning machine can implement, then for all functions of that class, independent of the underlying distribution $\mathcal{P}(x, y)$, with a probability of at least $1 - \delta$ over the drawing of the training sample the following bound holds [Vapnik98, Vapnik95]:

$$R[f] \leq R_{emp}[f] + \phi(h, n, \delta) \tag{2.4}$$

where ϕ is a *confidence* (or *capacity*) *term* which is defined as

$$\phi(h,\, n,\, \delta) = \sqrt{\frac{1}{n}\left(h\left(\ln\frac{2n}{h} + 1\right) + \ln\frac{4}{\delta}\right)} \tag{2.5}$$

Now for each finite set of examples $\mathcal{E}$ we can always come up with a function f such that we have zero training error (provided we have no examples contradicting each other), i.e. $R_{emp}[f] = 0$. This will even be the case if all training patterns x and all labels y are statistically independent, i.e. $\mathcal{P}(x, y) = \mathcal{P}(x)\mathcal{P}(y)$, and the labels are equally likely. In this situation there is no way of making a good guess about a label of a test pattern. However, to achieve zero training error we will necessarily require a large VC dimension h. As the confidence term (2.5) increases monotonically with the VC dimension this will lead to a high risk R, i.e. overfitting. That means the class of functions $\mathcal{C}$, from which f is taken, has to be restricted such that its capacity (e.g. VC dimension) is on one hand as small as possible (in relation to the available amount of data) to provide a good generalization performance (i.e. avoid overfitting) and on the other hand large enough to model the dependencies hidden in $\mathcal{P}(x, y)$ (i.e. to avoid underfitting). Hence Vapnik and Chervonenkis [Vapnik79, VapChe74] proposed minimizing the right hand side of (2.4), rather than just minimizing the empirical risk. This leads to the *Structural Risk Minimization* principle. The main idea is to build a nested sequence of function classes (or *structures*) $S_1 \subset S_2 \subset ... \subset S_h \subset ...$ of increasing size (and thus, of increasing capacity) and minimize the right hand side of (2.4) over the choice of the structure. This way a function f_* is picked which has a small training error and is an element of a structure that has a low capacity h_*(figure 2.2).

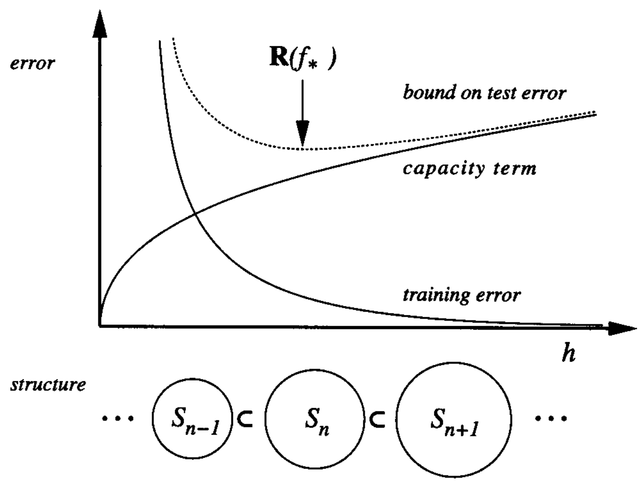

Figure 2.2: Structual Risk Minimization principle (source: [SchSmo02])

Note that (2.5) also depends on the amount n of data we have. The higher the VC dimension h of the class of concepts $\mathcal{C}$ we are taking into account the higher is the number of training examples we need to avoid overfitting. On the other hand, if we have an unlimited supply of training examples, we can avoid overfitting by just taking some class of functions $\mathcal{C}$ with a finite VC dimension.

It should be mentioned that the approach of Statistical Learning Theory presented here is not the only one to the problem of pattern recognition. Another statistical approach would be e.g. Bayesian learning. In the Bayesian view learning is nothing more than a subproblem of the more fundamental problem of making predictions [RusNor95, p. 588]. The idea is to use hypotheses as intermediaries between data and prediction. First the probability of each hypothesis, given the data, is estimated. Then predictions are made from the hypotheses using their posterior probabilities. That means predictions are weighted by the probability of the underlying hypothesis.

A completely other approach came out of the field of Artificial Intelligence. Here the goal is to find *rules* behind the data which can be represented in a symbolic fashion, e.g. predicate logic. That means the learned hypothesis can be seen as knowledge which can be communicated to human beings in a symbolic way. An example of this approach would be learning by means of decision trees [RusNor95, p. 531]. In contrast to this approach SVMs as well as Artificial Neural Networks — which are motivated by biology — represent our hypothesis in a "subsymbolic" way.

Additionally one could try not to infer a hypothesis from the whole training data at once by means of an induction principle, but to start with some simple hypothesis and refine it gradually whenever a new example arrives. This approach is called *incremental learning* [RusNor95, p. 529]. An example of such an approach is the version-space learning algorithm as described e.g. in [RusNor95, p. 549].

2.2 What are Support Vector Machines?

2.2.1 The Idea behind it — Optimal Margin Hyperplanes

In the last section the importance of capacity control for any learning algorithm was described. Therefore one needs to come up with a class of functions whose capacity can be computed. Vapnik et al. [VapLer63, VapChe74, VapChe79] considered the class of hyperplanes in some vector space $\mathcal{H}$ with a defined dot product $\langle \cdot, \cdot \rangle$:

$$\{\mathbf{x} \in \mathcal{H} | \langle \mathbf{w}, \mathbf{x} \rangle + b = 0\}, \ \mathbf{w} \in \mathcal{H}, b \in \mathbb{R} \tag{2.6}$$

As any hyperplane divides the space into two half spaces (and thus induces a classification into a class -1 and a class +1) this corresponds to decision functions

$$f(\mathbf{x}) = \operatorname{sgn}(\langle \mathbf{w}, \mathbf{x} \rangle + b) \tag{2.7}$$

The *margin* of a hyperplane is the distance between the hyperplane and the point closest to it. The important point is that one can prove that the capacity (e.g. the VC dimension) of the class of separating hyperplanes with a given margin decreases with increasing margin (e.g. [SchSmo02, p. 142]). Looking back at the last section that means the bigger the margin the lower becomes the right hand side of (2.4). This is exactly what we want. By making the margin as large as possible we are minimizing the structural risk. Hence, Vapnik et al. proposed to find the so called *optimal* hyperplane which induces the maximum margin by separating the two classes. This optimal hyperplane is unique. It can be constructed in the following way:

Let $(\mathbf{x}_1, y_1), ..., (\mathbf{x}_n, y_n) \in \mathcal{H} \times \{-1, +1\}$ be a set of training examples. Then any hyperplane of the form (2.6) can be scaled such that

$$\min_{i=1,\dots,n} |\langle \mathbf{w}, \mathbf{x} \rangle + b| = 1$$

which means that the point closest to the hyperplane has a distance of $1/\left\|\mathbf{w}\right\|$ (figure 2.3).

By considering this normalization, the optimal hyperplane can be constructed by solving the optimization problem (see e.g. [SchSmo02, pp. 196 and following])

$$\begin{array}{l} \min_{\mathbf{w} \in \mathcal{H}, b \in \mathbb{R}} \tau(\mathbf{w}) = \frac{1}{2} \left\|\mathbf{w}\right\|^2 \\ \text{subject to } y_i \left(\langle \mathbf{x}_i, \mathbf{w} \rangle + b\right) \geq 1 \text{ for all } i = 1, ..., n \end{array} \tag{2.8}$$

This is called the so called *primal optimization problem*. Constrained optimization problems like this are the subject of optimization theory and can be solved by means of the *Karush-Kuhn-Tucker (KKT)* theorem (see e.g.

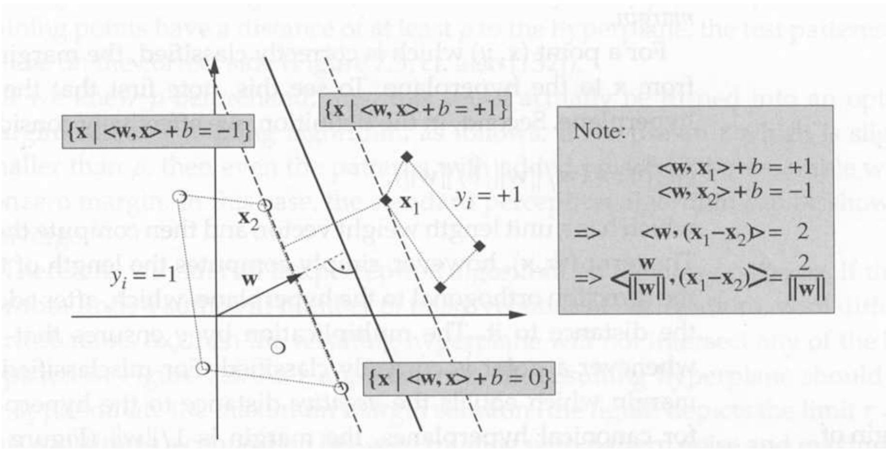

Figure 2.3: A binary classification problem: separate balls from diamonds. The *optimal* hyperplane is shown as a solid line. As the problem is linearly separable, there exists a weight vector $\mathbf{w}$ and a threshold b such that $y_i\left(\langle \mathbf{w}, \mathbf{x}_i \rangle + b\right) > 0$ $(i = 1, ..., n)$. Rescaling $\mathbf{w}$ and b such that the point(s) closest to the hyperplane satisfy $|\langle \mathbf{w}, \mathbf{x}_i \rangle + b| = 1$, we obtain a canonical form of the hyperplane satisfying $y_i\left(\langle \mathbf{w}, \mathbf{x}_i \rangle + b\right) \geq 1$. That means the margin in this case equals $1/\|\mathbf{w}\|$. This can be seen by considering two points $\mathbf{x}_1, \mathbf{x}_2$ on opposite sides of the hyperplane, which exactly satisfy $|\langle \mathbf{w}, \mathbf{x}_i \rangle + b| = 1$, and projecting them onto the hyperplane normal vector $\mathbf{w}/\|\mathbf{w}\|$. (source: [SchSmo02])

[SchSmo02, pp. 165 and following]). Instead of solving (2.8) one deals with the *dual problem*

$$\begin{array}{l} \max_\alpha W(\alpha) = \sum_{i=1}^n \alpha_i - \frac{1}{2}\sum_{i,j=1}^n \alpha_i\alpha_j y_i y_j \langle \mathbf{x}_i, \mathbf{x}_j \rangle \\ \text{subject to } \alpha_i \geq 0 \text{ for all } i = 1, ..., n, \text{ and } \sum_{i=1}^n \alpha_i y_i = 0 \end{array} \tag{2.9}$$

where $\alpha = (\alpha_1, ..., \alpha_n)$ is an n-tuple of Lagrangian multipliers. The KKT theorem guarantees that the solution of (2.9) is exactly the same as of (2.8) and has the form

$$\mathbf{w} = \sum_{i=1}^n \alpha_i y_i \mathbf{x}_i \tag{2.10}$$

which is leading to a decision function of the form

$$f(\mathbf{x}) = \operatorname{sgn}\left(\sum_{i=1}^n \alpha_i y_i \langle \mathbf{x}, \mathbf{x}_i \rangle + b\right) \tag{2.11}$$

Obviously $\mathbf{w}$ only depends on those training vectors $\mathbf{x}_i$, for which the corresponding α_i is non-zero. These patterns are called *support vectors.* All remaining examples are irrelevant for the decision function. According to the KKT theorem only the Lagrangian multipliers α_i which are non-zero at the saddle point of (2.9) correspond to an exact equality in the constraints. That means that all support vectors are lying exactly on the margin, since only they can fulfill $y_i (\langle \mathbf{x}_i, \mathbf{w} \rangle + b) = 1$. This nicely captures our intuition of the problem: As the hyperplane is completely determined by the training points closest to it, the solution does not depend on other examples.

2.2.2 Support Vector Machines (SVMs)

Until now we have only focused on the case where the training data is perfectly linearly separable by a hyperplane. However, one can show that there are problems which are not linearly separable (e.g. the XOR problem [Rojas96, p. 62]). If we still want to use optimal margin hyperplanes, which are attractive due to the available capacity bounds, we need a method to increase the capacity of our set of decision functions another way. The idea is to construct a mapping $\phi : \mathcal{X} \rightarrow \mathcal{H}$ from the set of inputs $\mathcal{X}$ to our dot product space $\mathcal{H}$ (also called *feature space*), such that the data lying in $\mathcal{H}$ is linearly separable. To reach this goal, one uses a function $\mathrm{k} : \mathcal{X} \times \mathcal{X} \rightarrow \mathbb{R}$ such that $\mathrm{k}(x, x') = \langle \phi(x), \phi(x') \rangle$. This function is called *kernel* function. The goal of the "kernel trick" is to avoid the explicit computation of the mapping ϕ. Instead we define a dot product in feature space by means of the kernel function. The kernel can be interpreted as a similarity measure between the vectors $\phi(x)$ and $\phi(x')$ in space $\mathcal{H}$. This becomes more clear, if we consider the canonical dot product between two vectors $\mathbf{x} = (x_1, ..., x_d)^{\mathbf{T}}, \mathbf{y} = (y_1, ..., y_d)^{\mathbf{T}} \in \mathbb{R}^d$ $\langle \mathbf{x}, \mathbf{y} \rangle = \sum_{i=1}^{d} x_i y_i$. Then $\langle \mathbf{x}, \mathbf{y} \rangle$ is proportional to the cosine of the angle between $\mathbf{x}$ and $\mathbf{y}$ and thus an indicator for the similarity between both vectors. The idea of kernels is to generalize this concept by first mapping our training data into a higher dimensional feature space and then defining some dot product (and thus similarity measure) in that space.

The question is how this can be done in practice. We somehow need

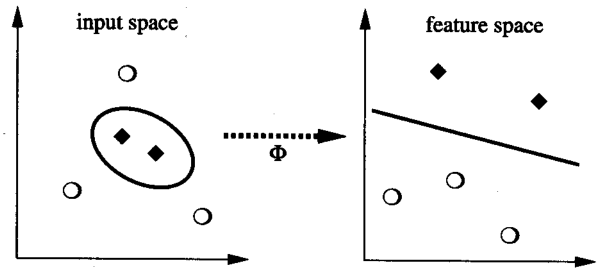

Figure 2.4: The idea of kernels: map the training data into a higher dimensional feature space, such that the mapped data is linearily separable with a maximum margin hyperplane in this space. This induces a nonlinear decision boundary in the original input space. By the use of a kernel function, it is possible to compute the separating hyperplane without explicitly computing the mapping ϕ (source: [SchSmo02, p. 15])

to check whether some function we chose defines in fact a kernel and hence implicitly a feature space or not. Because of the properties of the dot product any function k which is a kernel has to be symmetric. Thus the *Gram-matrix* or *kernel matrix* $\mathbf{K} = (k(x_i, x_j))_{ij}$ has to be symmetric as well. Furthermore it can be shown [CrisSha00, p. 35] that k is a kernel function (i.e. it fulfills $\mathrm{k}(x, x') = \langle \phi(x), \phi(x') \rangle$) if and only if $\mathbf{K}$ is positive definite (i.e. has non-negative eigenvalues).

Popular kernel functions are e.g. the polynomial kernels of degree $g > 0$ $\mathrm{k}(x, x') = (\langle x, x' \rangle + c)^g,\ c \in \mathbb{R}$, or the radial basis functions (RBF) of width $\sigma > 0$ $\mathrm{k}(x, x') = \exp\left(-\frac{\|x-x'\|^2}{2\sigma^2}\right)$. These kernel functions require $\mathcal{X}$ to be a dot product vector space, but there exist also kernels for non-vectorial spaces, e.g. for strings (see e.g. [LesEskNob02]). It is also possible to build new kernels from existing ones. If for example k_1 and k_2 are kernels over $\mathbb{R}^n$, then also $k = k_1 + k_2$ is a kernel. This can be seen easily by considering the corresponding Gram matrices $\mathbf{K}_1, \mathbf{K}_2$ and some vector $\mathbf{v} \in \mathbb{R}^n$. Then the matrix $\mathbf{K}_1 + \mathbf{K}_2$ has to be positive semi-definite, because $\mathbf{v^T}(\mathbf{K}_1 + \mathbf{K}_2)\mathbf{v} = \mathbf{v^T}\mathbf{K}_1\mathbf{v} + \mathbf{v^T}\mathbf{K}_2\mathbf{v} \geq 0$. There is also some on going work in estimating the kernel directly from the training data (e.g. [CriShaEliKan02]). This is done by optimizing the *alignment* $\frac{\langle \mathbf{K}, \mathbf{yy^T} \rangle_F}{\sqrt{\langle \mathbf{K}, \mathbf{K} \rangle_F \langle \mathbf{yy^T}, \mathbf{yy^T} \rangle_F}}$ of the unknown kernel matrix

$\mathbf{K}$ with the matrix $\mathbf{y}\mathbf{y}^{\mathbf{T}}$, where $\mathbf{y} \subset \mathcal{Y}^n$ is the vector of the class labels of the training data.[2]

Let's go back to our Optimal Margin Hyperplanes from the last section now. The idea is to replace the dot product in (2.9) by a kernel function which allows us to induce a nonlinear decision boundary in the original input space. This is leading to the following formulation of the dual optimization problem (see e.g. [SchSmo02, pp. 202 and following])

$$\begin{array}{l} \max_\alpha W(\alpha) = \sum_{i=1}^n \alpha_i - \frac{1}{2}\sum_{i,j=1}^n \alpha_i\alpha_j y_i y_j \mathrm{k}(x_i, x_j) \\ \text{subject to } \alpha_i \geq 0 \text{ for all } i = 1, ..., n, \text{ and } \sum_{i=1}^n \alpha_i y_i = 0 \end{array} \tag{2.12}$$

which induces a decision function

$$f(\mathbf{x}) = \operatorname{sgn}\left(\sum_{i=1}^n \alpha_i y_i \mathrm{k}(x, x_i) + b\right) \tag{2.13}$$

This is the so called hard margin Support Vector Machine (see figure 2.5). Note that our optimization problem is now formulated in the original input space $\mathcal{X}$, which is not necessarily a vector space.

One open point is what happens, if the two classes are not perfectly separable, because of outliers or noise in the data. In this case a separating hyperplane in feature space doesn't exist. Cortes and Vapnik ([CorVap95]) solve this problem by introducing so called slack variables $\xi_i \geq 0,\ i = 1, ..., n$ and relaxing the separation constraints in (2.8) by just requiring

$$y_i\left(\langle \mathbf{x}_i, \mathbf{w}\rangle + b\right) \geq 1 - \xi_i,\ i = 1, ..., n \tag{2.14}$$

This is commonly referred to as a *soft margin hyperplane*. Obviously these constraints could always be met by just making all ξ_i large enough. To compensate this, the objective function (2.8) is modified as

$$\min_{\mathbf{w}\in\mathcal{H}, b\in\mathbb{R}} \tau(\mathbf{w}, \xi) = \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^n \xi_i \tag{2.15}$$

[2] $\langle \mathbf{K}_1, \mathbf{K}_2\rangle_F$ is the Frobenius dot product which is defined as $\langle \mathbf{K}_1, \mathbf{K}_2\rangle_F = \sum_{i,j}(\mathbf{K}_1)_{ij}\left(\mathbf{K}_2^{\mathbf{T}}\right)_{ij}$.

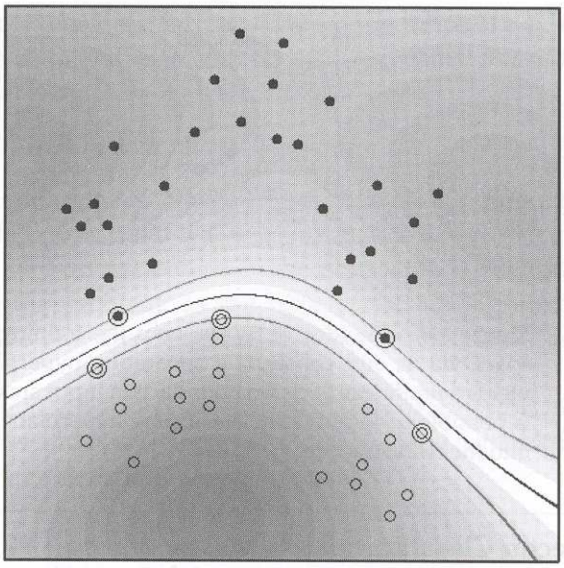

Figure 2.5: Example of a hard margin SVM using a RBF kernel. Circles an disks are two classes of training examples. The lines show the decision boundary within the margin. The SVs (marked with extra circles) are lying exactly on the margin. (source: [SchSmo02, p. 16])

where the constant $C > 0$ determines the trade-off between margin maximization and error minimization. Incorporating a kernel and the dual problem formulation, one has to maximize (2.12) subject to the constraints

$$0 \leq \alpha_i \leq C \text{ for all } i = 1, ..., n, \text{ and } \sum_{i=1}^{n} \alpha_i y_i = 0 \tag{2.16}$$

The only difference from the separable case is the upper bound C on the Lagrangian multipliers α_i. This way the influence of the individual patterns (which could be outliers) is limited. The solution takes the same form as (2.13). This is the so called C-SVM, which is the most commonly used SVM. Note that for $C \to \infty$ the solution converges against the solution of the hard margin SVM.

Another formulation is the so called ν-SVM (e.g. [SchSmo02, pp. 206 and following]). Instead of using the parameter C, one uses a parameter $\nu \in [0, 1]$ which can be shown [SchSmo02] to provide a lower bound on the fraction of examples that will be support vectors and upper bounds for the fraction of points with $\xi_i > 0$ (*margin errors*). Patterns which are margin errors are either misclassified or lie within the margin. The primal optimization problem for the ν-SVM is formulated as

$$\begin{array}{l} \max_{\mathbf{w}\in\mathbb{R},\xi\in\mathbb{R}^n,\rho,b\in\mathbb{R}} \tau(\mathbf{w},\xi,\rho) = \frac{1}{2}\left\|\mathbf{w}\right\|^2 - \nu\rho + \frac{1}{n}\sum_{i=1}^{n}\xi_i \\ \text{subject to } y_i\left(\langle \mathbf{x}_i, \mathbf{w}\rangle + b\right) \geq \rho - \xi_i \text{ , and } \xi_i \geq 0 \text{ for all } i = 1,...,n,\ \rho \geq 0 \end{array} \tag{2.17}$$

where ρ is a parameter that controls the size of the margin. The dual problem in this case can be derived as

$$\begin{array}{l} \max_{\alpha} W(\alpha) = -\frac{1}{2}\sum_{i,j=1}^{n}\alpha_i\alpha_j y_i y_j \mathrm{k}(x_i,x_j) \\ \text{subject to } 0 \leq \alpha_i \leq \frac{1}{n},\ \sum_{i=1}^{n}\alpha_i y_i = 0,\ \sum_{i=1}^{n}\alpha_i \geq \nu \end{array} \tag{2.18}$$

The resulting decision function has the same form as in the C-SVM case. It can be shown [SchSmo02] that, if the ν-SVM leads to $\rho > 0$, the decision function is in fact identical to the one which can be obtained by a C-SVM with $C = 1/\rho$.

The last open question is, what happens, if we have more than just two classes to separate, lets say M. Common strategies in this case are the following [SchSmo02, pp. 211 - 213]

- Construct a set of binary classifiers $f^{(1)}, ..., f^{(M)}$, each trained to separate one class from the rest (*one-against-rest* approach). The ith classifier computes a decision boundary between class c and the other $M - 1$ classes. An unseen test pattern x is assigned to the class for which the distance of the image $\phi(x)$ from the margin in the positive direction (i.e. in the direction of class c) is maximal.

- Construct a set of binary classifiers $f^{(1)}, ..., f^{((M-1)M/2)}$, each trained to separate one pair of classes (*one-versus-one* approach). A classifier $f^{(i)}$ separating class c from class $\tilde{c}$ returns +1 if the image $\phi(x)$ of an unseen test pattern x has a positive distance from the hyperplane and -1 otherwise. A return of +1 can be interpreted as a vote for class c while a return of -1 can be interpreted as a vote for class $\tilde{c}$. The pattern x is assigned to the class which gets the maximum number of votes of all classifiers. At the first glance this seems to be less effective than the one-against-rest approach, since there are $(M-1)M/2$ binary classifiers to train. On the other hand, the advantage of this method is that the problems to be learned are usually easier (because only the patterns belonging to two classes have to be taken into account), and they are not as unbalanced as in the other approach. Especially with a high number of classes one usually would get a high number of negative versus a small number of positive examples, which can cause problems for the SVM to obtain a good generalization performance.

- Formulate a multi-class version of the SVM (e.g. [WesWat99]). This is more elegant, but it takes longer to train a multi-class SVM, since it has to deal with all support vectors at the same time. Binary classifiers, on the other hand, usually have much smaller support vector sets to take into account.

Especially when RBF kernels are used, one can compare SVMs with classical RBF networks (see e.g. [UltschVLNN99]). In both approaches the decision function is computed as a linear combination of kernel functions (in this case RBF kernels). What is different, is the choice of the RBF centers. In SVMs this is done automatically, and one doesn't have to take care about it. In RBF networks, the RBF centers are chosen usually as the centers of the clusters which are e.g. detected by the k-means clustering algorithm (e.g. [JaiMurFly99]). Thus in RBF networks the decision function depends on the identified cluster centers, while in SVMs the decision function depends on the identified support vectors. This can be interpreted as classification based on prototypes (RBF networks) versus classification based on extreme examples (SVMs). However, experimental results (e.g. [SchSunBurGirNiyPogVap97]) seem to show, that SVMs are superior to RBF networks.

At last it should be mentioned that SVMs are not only used for pattern recognition, but also for complete function estimation, that means what is commonly referred to as *regression* in classical statistics. I will not go into any detail here, since in this thesis I will only focus on classification, but a description on how to generalize the SVM algorithm for this case can be found e.g. in [SmoSch98].

2.2.3 Estimating the Generalization Capability

In 2.1 it was explained that the principle of structural risk minimization leads to a hypothesis that allows us to generalize well, and in 2.2.1 it was pointed out that the idea behind SVM is to minimize a bound on the risk. The question is, how one can actually "measure" the generalization performance of an already trained classifier. As it was already said in 2.1, an exact measure is not possible, since this would require the computation of (2.3) on page 8. Thus we can only use statistical tools to estimate the generalization capability. Again the problem is that we have a finite number n of training examples from which we want to predict the error made on any unseen set of data. Measuring the error obtained on the training data would systematically underestimate the true error, because the classifier has adapted the learned hypothesis to exactly this data set and is biased in favor of its elements (see 2.1). If we have a very large number of data, we can simply separate the data into one set which is used for training and one independent set, which should be as large as possible, and is used for testing. The classifier is constructed on the training data only and then asked to predict the output values for the data in the testing set.[3] The right answers are accumulated and averaged by the number of points in the test set. This is the so called *accuracy* [UltschVL00]. Similarly one can accumulate the proportion of errors on all testing data. This is the *classification error*. Additionally

[3]The following terms are commonly used to describe the properties of the classification of an example (x, y) with regard to a class c:

- (x, y) is true positive (T+): pattern x was correctly put into the considered class c.
- (x, y) is true negative (T-): pattern x doesn't belong to class c and the classifier correctly didn't put it there.
- (x, y) is false positive (F+): pattern x doesn't belong c, but the classifier put it there.
- (x, y) is false negative (F-): pattern x in fact belongs to c, but the classifier didn't put it there.

we can compute for each class the number of right answers with regard to a certain class in relationship to all data belonging in that class (*sensitivity*)[4] or, complementary, the number of data which was correctly not put in a certain class in relationship to all data which doesn't belong to that class (*specificity*)[5]. Sensitivity and specificity are a measure for how well a certain class is separated from the rest. At last for each class one can calculate the proportion of right answers with regard to a certain class on all data which was in fact put in that class (*positive predictive value*)[6].

The main problem with holding out an extra test set is that in order to get reliable results one needs a very high number (e.g. 10,000 points) of data in the test set, because otherwise the evaluation may depend heavily on which data points end up in the training set end which end up in the test set. Thus one improvement in the case where not so many data are available is to use *k-fold cross-validation* instead (e.g. [Rojas96]). The data set $\mathcal{E}$ is divided into k subsets, and the hold-out method is repeated k times. Each time, one of the k subsets is used for testing and the $k-1$ other subsets are put together and used for training. Then the accuracy/classification error is averaged across all k trials. The obvious advantage of this method is that it works also, if $\mathcal{E}$ is small, because it matters less how the data gets divided. Especially for very small data sets one can take the logical extreme and choose $k = n$. That means every data point is viewed as one subset. In every trial $n-1$ data points are used for training and 1 for testing. This is called *leave-one-out cross-validation*. This is the least biased estimate of the generalization capability one can get with cross-validation. On the other hand it is clear that this is very computationally expensive, because it requires n training runs of the classifier.

Another approach is *bootstrapping*. The idea behind bootstrapping is the following (e.g. [Rojas96]): Our n training points are drawn from an unknown probability distribution $\mathcal{P}$. The bootstrap assumption is that we can approximate this distribution by randomly drawing subsamples from

[4]i.e. sensitivity $= \frac{\#\{T+\}}{\#\{T+\}+\#\{F-\}}$

[5]i.e. specificity $= \frac{\#\{T-\}}{\#\{T-\}+\#\{F-\}}$

[6]i.e. positive predictive value $= \frac{\#\{T+\}}{\#\{T+\}+\#\{F+\}}$

the set $\mathcal{E}$ with replacement. In k trials n data points are selected from $\mathcal{E}$ with replacement. That means each training point can be selected more than once! Each time the classifier is trained on these *bootstrap samples* and tested on the training points that were not in the bootstrap sample. Then the accuracy/classification error is averaged across all k trials. The higher k the better is the approximation of the unknown distribution $\mathcal{P}$, which means the better is the estimation of the generalization performance. The advantage of bootstrapping is that one doesn't need much data to obtain quite reliable results. The disadvantage is that it is rather computationally expensive.

Apart from these general approaches there exist especially for SVMs theoretical bounds on the leave-one-out error ([Vapnik98, VapCha00, ChaVap00]). Three of them are cited here:

Let ρ be the size of the maximal margin and $\phi(x_1), ..., \phi(x_n)$ the images of the training patterns in features space which are lying within a sphere of radius R. Let α^* be the tuple of Langragian multipliers which are obtained by maximizing functional (2.12) on page 16. Then the following holds true:

Theorem 1 (Vapnik [Vapnik98]): *If the images of training data of size n belonging to a sphere of radius R are separable with the corresponding margin ρ, then the expectation of the test error probability has the bound*

$$EP_{err} \leq \frac{1}{n} E\left\{\frac{R^2}{\rho^2}\right\} = \frac{1}{n} E\left\{R^2 W^2(\alpha^*)\right\} \tag{2.19}$$

where the expectation is taken over sets of training data of size n.

That means that the generalization performance depends on the ratio $E\{R^2/\rho^2\}$ and not simply on the large margin ρ. Following [SchBurVap95] one can calculate R^2 by maximizing:

$$\begin{array}{l} R^2 = \max_\beta \sum_i \beta_i \mathrm{k}(x_i, x_i) - \sum_{i,j} \beta_i \beta_j \mathrm{k}(x_i, x_j) \\ \text{subject to } \sum_i \beta_i = 1,\ \beta_i \geq 0,\ i = 1, ...n \end{array} \tag{2.20}$$

where $\beta = (\beta_1, ..., \beta_n)$.

Another bound was derived by Vapnik and Chapelle. It uses the concept of the *span* of the support vectors ([VapCha00, ChaVap00]):

Theorem 2 (Vapnik, Chapelle [VapCha00, ChaVap00]): *Supposed we have a SVM without threshold (that means we have a decision function with $b = 0$). Under the assumption that the set of support vectors does not change when removing example p, we have*

$$EP_{err}^{n-1} \leq \frac{1}{n} E\left\{\sum_{p=1}^{n} \psi\left(\frac{\alpha_p^*}{\left(\mathbf{K}_{SV}^{-1}\right)_{pp}} - 1\right)\right\} \tag{2.21}$$

where $\psi : \mathbb{R} \to \{0,1\}$ is the step function $\psi(v) = \begin{cases} 0, & v \leq 0 \\ 1, & v > 0 \end{cases}$, $\mathbf{K}_{SV}$ is the matrix of dot products between support vectors in feature space, P_{err}^{n-1} is the probability of test error for the machine trained on a sample of size $n-1$ and the expectations are taken over the random choice of the sample.

As the computation of the inverse of the matrix $\mathbf{K}_{SV}$ is rather expensive, one might ask, if there was an appropriate approximation. In fact one can upper bound $\left(\mathbf{K}_{SV}^{-1}\right)_{pp}$ by $\frac{1}{\mathrm{k}(x_p, x_p)}$ [ChaVapBouMuk98]. Thus one recovers the Jaakkola-Haussler bound [JakHau99]:

$$EP_{err}^{n-1} \leq \frac{1}{n} E\left\{\sum_{p=1}^{n} \psi\left(\alpha_p^* \mathrm{k}(x_p, x_p) - 1\right)\right\} \tag{2.22}$$

Other bounds also exist, but they will not be taken into consideration here. In general all these bounds offer a less accurate estimate of the generalization error than k-fold cross-validation, but they are computationally attractive, because the SVM has to be trained just once. Additionally the whole training set can be used rather than leaving out n/k examples for each step of k-fold cross-validation.

2.3 The Problem of Feature Selection

So far the set $\mathcal{X}$ from which the training patterns are drawn was considered to be any nonempty set. In practical situations, however, $\mathcal{X}$ often will be a vector space (e.g. $\mathbb{R}^d$). Thus for the rest of the thesis there will be made the following assumptions:

Let $\mathcal{D} = \{\,(\mathbf{x}_i, y_i) |\, \mathbf{x}_i \in \mathcal{V}, y_i \in \mathcal{Y}, i = 1, ..., n\}$ be a set of labeled training data where the training patterns $\mathbf{x}_1, ..., \mathbf{x}_n$ are drawn from a vector space $\mathcal{V}$ of dimension d with a scalar product $\langle \cdot, \cdot \rangle$ and the labels $y_1, .., y_n$ are taken from a set of labels $\mathcal{Y}$ (in the two class case $\mathcal{Y} = \{-1, +1\}$). Each vector $\mathbf{x}_i^{\mathbf{T}} = (x_{i,1}, ..., x_{i,d})$ consists of d *feature* or *attribute* values. Thus a feature X_t is a vector $X_t = (x_{1,t}, , ..., x_{n,t})^{\mathbf{T}}$, $t = 1, ..., d$. This notation will be used during the rest of this thesis.

In the introduction there was already given a motivation for the problem of feature selection: We want to find out the most important features of our data to avoid overfitting. Besides there might be other interests. Especially in biology one might not be interested so much in maximizing the generalization performance of a machine, but in finding out the most **meaningful** features (e.g. gene expressions) which give us the most knowledge about the domain to study. This is a more problem oriented point of view, which requires a definition of what the term "meaningful" stands for in that context. Since this thesis doesn't deal with some special problem in biology or another domain, we will stick to the Machine Learning point of view here. Let's clarify what feature subset selection from this point of view actually means. Formally the problem of feature subset selection in Machine Learning can be addressed in the following two ways [WesMukChaPogVap01]:

1. Given a fixed $m \ll d$, find the m features that give the smallest expected generalization error; or

2. Given a maximum allowable generalization error γ, find the smallest number m of features.

This should not be confused with *feature extraction* as it is done e.g. in Principal Component Analysis (e.g. [Duda73]) or Independent Component Analysis (e.g. [KwaChoCho01]). In both methods new features are constructed from existing ones. The principal components obtained by PCA are a linear combination of the existing features and form an orthogonal system. In ICA, instead of the latter property, features are required to be statistically independent. In contrast to feature extraction, in feature selection one wants to find "a subset of the original features of a data set such that an induction algorithm that is run on data containing only these features generates a classifier with highest possible accuracy" [Kohavi97, p. 3].

Supposed we already knew the probability distribution $\mathcal{P}$ from which the data set $\mathcal{P}$ was drawn, then the problem of feature selection would not be of much interest: The Bayes classifier is a classifier that predicts the most probable class for a given training pattern by means of the classical Bayes rule[7] (e.g. [RusNor95, p. 426, p. 588]). This requires the probability distribution $\mathcal{P}$ to be known. The accuracy of the Bayes classifier is the theoretically highest possible accuracy. The optimal Bayes rule is monotonic, which means that adding features cannot decrease accuracy, and hence restricting the rule to a subset of features is never advised [Kohavi97]. In practical situations, however, the probability distribution $\mathcal{P}$ is not accessible, and thus the computation of the optimal Bayes rule not possible.

Feature selection can be seen as a combinatorial problem: Given a set of all features, find out the combination of features that maximizes the expected generalization capability (minimizes the expected generalization error). Especially if one wants to find the smallest number m of features which minimizes the generalization error at the same time (special case of problem 2) one can show that this is a NP-complete problem [DavRus94].

In the literature there are two main approaches to solve this problem: the so called wrapper approach and the so called filter approach [Kohavi97]. In the filter approach feature selection is done as a preprocessing step to the actual learning algorithm. That means first the feature selection algorithm tries to select the m most relevant features and after that the classifier is trained with the input data of dimensionality m. The term "relevant" here refers to some defined relevance measure. There exist a number of different definitions in the literature for what that actually means (see e.g. definitions in [Kohavi97, pp. 4 - 5], [BluLan97, pp. 2 - 4]). The reason for this variety is that it depends on the question "relevant to what?". E.g. Blum and

[7]The Bayes rule or Bayes theorem is the following simple rule: Let X, Y be random variables. Then $\Pr(Y|X) = \frac{\Pr(X|Y)\Pr(Y)}{\Pr(X)}$.

Langley [BluLan97, pp. 2 - 4] distinguish between "relevance to a target" and "relevance to a sample/distribution": A feature X is said to be relevant to a target classifier f, if there exists a pair of training examples r and s such that r and s differ only in their value of X and $f(r) \neq f(s)$. This means that X is relevant if there exists some example in the training data for which changing the value of X would affect the classification given by f. For the concept of "relevance to a sample" Blum and Langley make a difference between "strong relevance" and "weak relevance". A feature X is said to be strongly relevant to a sample $\mathcal{D}$ if there exist examples r and s that differ only in their value of X and have different labels. Similarly X is strongly relevant to target classifier f and the unknown distribution $\mathcal{P}$ from which $\mathcal{D}$ was drawn, if there exist examples r and s having non-zero probability over $\mathcal{P}$ that differ only in their values of X and satisfy $f(r) \neq f(s)$. This is just like the definition of "relevance to a target" except r and s are now required to be in $\mathcal{D}$ (or having non-zero probability). A feature X is said to be weakly relevant to sample $\mathcal{D}$ (or target f and distribution $\mathcal{P}$) if it is possible to remove a subset of the features so that X becomes strongly relevant.

The removal of features which are strongly relevant generally deteriorates the performance of the classifier, while features that are weakly relevant may or may not have an influence on the generalization performance.

A known problem of the filter approach [BluLan97, p. 8], [Kohavi97, p. 10] is that the selection of relevant features is done independently from the actual performance of the classifier system. There is no interaction between the feature selection algorithm and the learning algorithm, because the (estimated) generalization capability with regard to a selected subset of features is not used to determine the quality of the feature subset. This can lead to the problem shown in [Kohavi97, p. 8], that the filter algorithm selects features which are detected as important due to the used relevance measure in the algorithm, but in fact decrease the performance of the learning algorithm. This may especially happen, if there is a relatively high correlation between an irrelevant feature and a class label. The Corral dataset [JohKohPfl94] is an example for a possible scenario where this happens: Given is a Boolean domain with 6 features A0, A1, B0, B1, I and C. The target concept is simply

$$(A0 \wedge A1) \vee (B0 \wedge B1)$$

The feature I is uniformly random, and the feature C matches the class label 75% of the time. Thus a filter algorithm might select C as a relevant feature, although it doesn't contribute to the target concept and hurts performance.

In the wrapper approach on the other hand this probably won't happen, because each feature subset taken into account is evaluated by the estimated generalization performance of the classifier system with those selected features taken as an input. That means that for every potential solution there is a direct answer from the learning algorithm, but it also means that the classifier has to be retrained every time, which can be rather computationally expensive. What one wins, is a more reliable measure for the importance of a feature subset than with a separate relevance measure that may have an entirely different inductive bias. But that doesn't mean that a wrapper method is always superior to a filter approach. As for each considered feature subset there has to be one retraining of the classifier on the training data, minimizing a generalization error estimate, which is based on the training data as well, might lead to overfitting! This might especially happen, if the same feature subset is considered multiple times. Unfortunately this problem is unsolved so far.

A wrapper algorithm performs a search in the space of all possible feature subsets. Hence every allowed feature subset is a state in this search space. Several strategies are imaginable to sample the search space. Especially if problem 2 is given, a general strategy for a heuristic search would be either to begin the search with an empty feature subset and successively add features or with the full set of features and successively eliminate features. The first case is called *forward selection* and the latter case *backward elimination* [Kohavi97, p. 9].

A completely different method to sample large search spaces is given by Genetic Algorithms which will be described in the next section.

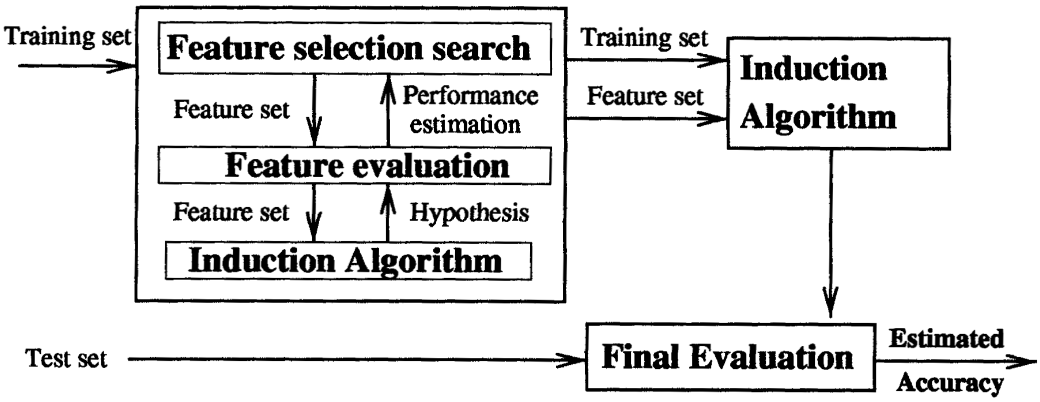

Figure 2.6: The wrapper approach to feature selection. The classifier is used to determine the performance of each feature subset (source: [Kohavi97, p. 2])

2.4 Genetic Algorithms

2.4.1 What are Genetic Algorithms?

Genetic Algorithms (e.g. [Whitley93, Goldberg98]) belong to a family of search algorithms which are inspired by natural evolution. These algorithms encode a potential solution to a specific problem on a *chromosome*-like structure and apply *recombination* (*crossover*) and *mutation* operators to these structures. An essential idea is the principle of "survival of the fittest", which means that potential solutions compete with each other in a way that those chromosomes which represent better solutions to the target problem are given more chances to "reproduce". Genetic Algorithms have been developed by John Holland [Holland75] and his students (e.g. De Jong [DeJong75]), and most theoretical work on Genetic Algorithms is referring to this original (*canonical*) Genetic Algorithm. Nevertheless there have been a plenty of variations by other researchers which adapted the original concept to their problems. Usually Genetic Algorithms are used in optimization. That means one wants to find the optimum of some function φ. Traditional optimization techniques like gradient descent are local in scope. The optima they seek are the best in a neighborhood of the current point. This causes problems, if the function has many local optima or, even worse, if the gradient is not computable. Enumerative search methods (e.g. the A*-algorithm [RusNor95]) on the other hand are only applicable, if the search space is discrete and not too large. The advantage of Genetic Algorithms is that on one hand one doesn't need to know much information about the function to optimize and on the other hand they are applicable even if the search space is very large. Genetic Algorithms are using random choice as a tool to guide a highly exploitative search through a discrete coding of the search space. That means the transition from one state in search space to another is probabilistic, not deterministic. Another characteristic of Genetic Algorithms is, that they don't search from a single point, but from a *population* of points at the same time. That means Genetic Algorithms are a global search procedure, because they parallelly explore the search space from many points. This way they can avoid local optima. However, it should be mentioned that Genetic Algorithms are not always the best choice. Depending on the problem at hand, a traditional gradient descent may actually be much faster than a rather computationally expensive Genetic Algorithm.

The first problem when using Genetic Algorithms is how to encode potential solutions of our optimization problem into a genome of an *individual.* The easiest and most common way to do this is by means of a binary code, but other codings are also possible. Let's e.g. consider one wants to find the maximum of the objective function $\varphi : \{0, ..., 255\} \to \mathbb{N},\ x \mapsto x^2$. Each point $x \in \{0, ..., 255\}$ can be coded as a 8 bit binary number. Putting p points together we get a population of individuals with chromosomes $(b_1, ..., b_8),\ b_i \in \{0, 1\}$. Each chromosome corresponds to a certain value of the *evaluation function* φ. As we want to maximize φ, an individual is so much better the higher the value of φ with regard to its genome is. This is expressed by means of the *fitness* value which is unique to each individual. The *fitness function* maps the outputs of the evaluation function for the different individuals in the population to the fitness value of one single individual. That means the fitness is defined with respect to other members of the population. In the canonical Genetic Algorithm the fitness is defined as $\varphi_i/\bar{\varphi}$, where φ_i is the evaluation associated with individual i and $\bar{\varphi}$ is the average evaluation of all individuals of the population [Whitley93, p. 4]. The fitness value can also be assigned based on a ranking of the individuals ([Goldberg98, p. 124]).

The execution of the Genetic Algorithm can be seen as a two stage process [Whitley93, pp. 5 and following]: It starts with the *current population.* Pairs of individuals are selected with regard to their fitness value to create an *intermediate population.* There exist different selection methods [Goldberg98, p. 121]. The easiest one is to map the entire population onto a roulette wheel, where each individual is represented on the wheel by a space that is proportional to its fitness. The roulette wheel is rotated and an individual is chosen until the intermediate population is filled up. Then the genomes of each pair of the intermediate population perform a *crossover* and mutation. That means they are exchanging parts of their genetic information, i.e. bits of the number which represents the genome, and afterwards each bit is flipped with a small probability (e.g. 1%). The simplest way to perform a crossover is to randomly choose a crossover point, cut both chromosomes at this point, and swap the fragments between the two parents. Thus by recombining e.g. (11110000) and (00001111) with crossover point 4 one would get (11111111) and (00000000). All the children which are obtained by crossover and mutation are inserted into the *next population.* Then the algorithm restarts with using the next population as the current population. The process of evaluation, selection, recombination and mutation forms one *generation* in the execution of the Genetic Algorithm. Goldberg [Goldberg98] refers to this basic implementation as the Simple Genetic Algorithm (SGA).

2.4.2 Why do they Work?

The question is, why an algorithm like the one described above should do any useful. The answer which is most widely given came out of Holland's [Holland75] work. Holland argues that a Genetic Algorithm implicitly samples hyperplane partitions of the search space (see e.g. [Whitley93, pp. 7 and following]): Suppose we have a problem which can be encoded in just 2 bits. Thus the search space consists of the bit strings (00), (01), (10) and (11). We can view each bit string as the coordinates of a rectangle in a 2 dimensional space. The left line of the rectangle contains all points that begin with 0.

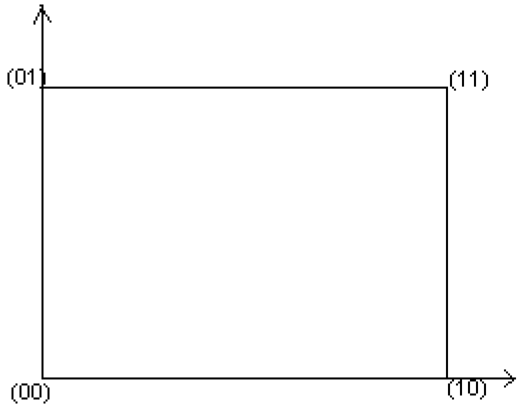

Figure 2.7: A 2-D search space

If “*” is used as a “don’t care” or wild card symbol, then this line (or hyperplane of dimension 1) can also be represented by the string (0*). Strings that contain “*” are referred to as *schemata*. Each schema corresponds to a hyperplane in search space. The *order* of a hyperplane refers to the number of actual bit values that appear in its schema. Thus (1*) is of order 1, while e.g. (1**1***0***) would be of order 3. The *defining length* of a schema is distance between the position of the first and the last actual bit value. E.g. (1**1***0***) has a defining length of 7, while (**0**1**) has a defining length of 3. All bit strings that match a particular schema are contained in the hyperplane partition represented by that particular schema. If l is the length of the binary genome, there are 3^l different hyperplane partitions, because each position can be either 0, 1 or *. Each single genome is a member of 2^l hyperplane partitions, because each position may take on its actual value or *. That means a particular bit string is contained in 2^l schemata[8]. If we

[8]Although this argumentation is made only for the case of a binary encoding, it should be mentioned again that other encodings are possible, too. However, since a non-binary alphabet can be translated into a binary one, without loss of universality the theory of Genetic Algorithms usually only considers binary encodings. Additionally several authors have argued in favor for binary alphabets [Whitley93, Goldberg98], because they maximize the number of hyperplane partitions: Suppose we want to represent just the numbers 0, 1, ..., 9. With a decimal encoding of length 1 we can represent 11, while with a binary encoding of length 4 we can represent $3^4 = 81$ schemata. That means each particular

have a population of size n then the whole population is contained in 2^l to $n \cdot 2^l$ schemata, depending upon the diversity of the population. I.e. usually there are many more schemata than individuals in the population. That means many different hyperplanes are sampled each time a single chromosome is evaluated. This fact is called *implicit parallelism* and verifies the idea of using a population of sample points rather than just examining one isolated point in search space at once. A population of strings provides information about numerous hyperplanes. Especially low order hyperplanes are sampled very often. In his fundamental *Schema Theorem* Holland [Holland75] proved that over time the number of low order schemata with short defining length and above average fitness will exponentially increase while the number of below average fitness schemata will exponentially decrease. That means Genetic Algorithms achieve their goal by successively sampling, recombining and resampling low order, highly fit schemata with a short defining length to form strings of potentially higher fitness. Due to their importance these low order, highly fit schemata with a short defining length are called *building blocks*. Just as a construction worker creates a complex house from simple pieces of stone, a Genetic Algorithm searches for the optimum of a function by compiling building blocks. Since the number of below average fitness schemata exponentially decreases in time, the average fitness of a population will increase in time and thus the algorithm will slowly converge against a local optimum.

decimal string contains just 10 schemata, while each particular bit string is contained in $2^4 = 16$ schemata. Nevertheless, depending on the problem at hand, a non-binary alphabet can be sometimes useful. As a rule of thumb Goldberg [Goldberg98, p. 80] suggests to "select the smallest alphabet that permits a natural expression of the problem".

Chapter 3

Some Existing Algorithms for Feature Selection

3.1 Algorithms for Feature Selection

In the last chapter we have seen that the problem of feature selection is an important issue for pattern recognition. It was said that there are two main approaches to solve this problem: the filter approach and the wrapper approach. Both approaches differ in the way they evaluate a given feature subset. The wrapper method for every feature subset taken into consideration trains the classifier system and evaluates the feature subset by estimating the generalization performance (i.e. the expected risk, if we want to use the vocabulary of Statistical Learning Theory) of the learning machine trained with this feature subset of the original data. The filter method on the other hand uses some relevance measure which is independent of the performance of the learning algorithm. In the following there will be described two popular filter methods and one wrapper method, whereas Recursive Feature Elimination was especially designed for SVMs.

3.2 Filter Approaches

3.2.1 Fisher Criterion Score

The Fisher Criterion Score (e.g. [Duda73, WesEliSchTip02]) is one of the simplest filter algorithms for feature selection. In the 2-class case for each feature t one computes the value

$$F_t = \frac{(\mu_1 - \mu_2)^2}{\sigma_1^2 + \sigma_2^2}$$

where μ_1 is the mean of the values of t belonging to class 1 and μ_2 the mean of the values of t belonging to class 2, and σ_1 and σ_2 are the standard deviations of class 1 and class 2. The idea behind this is, that the more a feature can separate the distributions of class 1 and class 2, the better it is.

If one has more than two classes, one can easily generalize this to[1]

$$F_t = \frac{\left(\mu_1 - \sum_{i=2}^{M} \mu_i\right)^2 + \ldots + \left(\mu_j - \sum_{\substack{i=1 \\ i \neq j}}^{M} \mu_i\right)^2 + \ldots + \left(\mu_M - \sum_{i=1}^{M-1} \mu_i\right)^2}{\sum_{i=1}^{M} \sigma_i^2}$$

where M is the number of classes. By means of the Fisher Criterion Score a ranking for all features is computed. The more relevant the feature the higher is the ranking. What characterizes the method is the implicit independence assumption between features that is made [GuyWesBarVap02]. Each coefficient F_t is computed with information about a single feature and does not take into account mutual information between features. The Fisher Criterion Score can be used for linear problems only.

3.2.2 Relief-F algorithm

The Relief-F algorithm as described e.g. in [KonHon97] is a more advanced filter approach than the Fisher Criterion Score. It is an extension of the

[1] This is of course not the only way to generalize Fisher Criterion Score for more than two classes. Similar to the multi-class approaches for SVMs one could also think about a one-versus-one or a one-against-rest strategy.

RELIEF algorithm introduced by Kira and Rendell [KirRen92] which originally couldn't handle multi class problems. The idea is, that a good feature should be able to discriminate close pairs (with regard to some given metric) of training patterns belonging to different classes and, furthermore, it is desirable to have the same value for close pairs of training patterns of the same class. To achieve this goal, for a given training pattern Relief-F searches for the k closest training instances of the same class and for the $k \cdot (M-1)$ nearest neighbors of the other classes (where M is the number of classes). The metric used to measure distances between training instances is usually the Manhattan distance. Then for a given feature $X_i = (x_{1,i}, ..., x_{n,i})^{\mathbf{T}}$ the score

$$\begin{aligned} q(X_i) \quad = \quad & \sum_{r \in \{1,...,n\}} \sum_{\text{class}(s) \neq \text{class}(r)} \left(\frac{\Pr(\text{class}(s)) \cdot d_{rs,i}}{1 - \Pr(\text{class}(r)) \cdot k} \right) - \\ & \sum_{r \in \{1,...,n\}} \sum_{\text{class}(s) = \text{class}(r)} \frac{d_{rs,i}}{k} \end{aligned}$$

is computed, where r is the index of the rth pattern which is chosen randomly from the set of all training patterns, s runs over the neighbors of r and $d_{rs,i}$ is defined as $d_{rs,i} = \min\left(\frac{|x_{r,i} - x_{s,i}|}{0.5 \cdot \text{range}(X_i)}, 1\right)^2$. As Kononenko and Hong show ([KonHon97]) the Relief-F measure approximates the difference of probabilities

$$\widetilde{q}(X_i) = \Pr(x_{r,i} \neq x_{s,i} | \text{ class}(s) \neq \text{class}(r)) - \Pr(x_{r,i} \neq x_{s,i} | \text{class}(s) = \text{class}(r)).$$

Like Fisher Criterion Score Relief-F computes a ranking for all features, but in contrast to this simple method it is able to deal with nonlinear problems as well.

[2] range(X_i) shall be the range of the values in feature X_i, i.e. $\max\{x_{ji} | j = 1, ..., n\} - \min\{x_{ji} | j = 1, ..., n\}$

3.3 Wrapper Approaches

3.3.1 Recursive Feature Elimination

Recursive Feature Elimination (RFE) [GuyWesBarVap02] is a wrapper method which performs a backward feature elimination (compare 2.3): The idea is to find the m features which lead to the largest margin of class separation. This combinatorial problem is solved in a greedy fashion. In the 2-class case the algorithm begins with the set of all features and successively eliminates the feature which induces the smallest change in the cost function $W^2(\alpha^*) = \sum_{i=1}^n \alpha_i^* - \frac{1}{2}\sum_{i,j=1}^n \alpha_i^*\alpha_j^* y_i y_j \mathrm{k}(\mathbf{x}_i, \mathbf{x}_j)$, where α^* is the tuple of the Lagrangian multipliers which are obtained by maximizing functional (2.12) on page 16. As for SVMs W^2 is a measure of predictive ability (and is inversely proportionate to the margin — compare (2.19) on page 16), the algorithm at each step eliminates the feature which keeps this quantity small. Assuming that the change of the set of support vectors (and hence of the tuple α^*) when removing only one feature is neglectable, this is done by performing the following iterative procedure [WesEliSchTip02]:

- Given a tuple α^* of Lagrangian multipliers as a solution of the SVM learning algorithm, calculate for each feature t:

  $$w^2_{(-t)}(\alpha^*) = \sum_{i,j=1}^{n} \alpha_i^*\alpha_j^* y_i y_j \mathrm{k}(\mathbf{x}_{(-t)_i}, \mathbf{x}_{(-t)_j})$$

  where $\mathbf{x}_{(-t)_i}$ means that the tth feature from training vector $\mathbf{x}_i$ has been removed.

- Remove the feature with the smallest value $Dw(t) = \left| w^2(\alpha^*) - w^2_{(-t)}(\alpha^*) \right|$ and retrain the SVM with the reduced set of features.

As one can see, this procedure requires d runs[3] of the SVM. If we are in the hard margin case and the kernel function is linear ($\mathrm{k}(\mathbf{x}_i, \mathbf{x}_j) = \langle \mathbf{x}_i, \mathbf{x}_j \rangle$)

[3] A possible speed up would be e.g. to remove half of the remaining features one at a time

the algorithm removes the feature with the smallest value ω_i at each iteration (where ω_i is the ith component of the weight vector $\mathbf{w} = \sum_i^n \alpha_i^* y_i \mathbf{x}_i$). RFE originally was designed to solve 2-class problems only, but a multi-class version can be derived easily, if the SVM is trained by discriminating one class against the others [WesEliSchTip02] or by building pair wise classifiers. For the latter case the idea is then to compute the value $Dw(t)$ for each pair of classes c and $\widetilde{c}$ and remove the feature which leads to the smallest value $\sum_{c=1}^{M(M-1)/2} Dw_c(t)$, where M is the number of classes and $Dw_c(t)$ is the change in the cost function for the SVM that computes decision boundary c. RFE like Fisher Criterion Score and Relief-F computes a feature ranking.

Chapter 4

Genetic Algorithms for Feature Selection

4.1 Feature Selection by Means of Genetic Algorithms

If one wants to use the wrapper approach explained in 2.3 to find an optimal subset of features with regard to criterion 2 on page 24, one has to deal with the problem that the search space is of size 2^d, because every feature can be either selected (“1”) or not (“0”) and thus every subset of $\{0,1\}^d$ is a potential solution. A simple combinatorial search in this case for large numbers of d (e.g. DNA micro array data with $d \geq 2,000$) is infeasible. Thus Genetic Algorithms have been applied to this problem a number of times before (e.g. [FerKadKit93, BriBroMar92, RicLan96, RayPunGooKuhJai00]). The standard approach is to view each subset of the space $\{0,1\}^d$ of all possible feature combinations as the binary genome of an individual and to evaluate its fitness by means of measuring the accuracy of the classifier, trained on the selected features of the input space, either on an independent validation set or via k-fold cross-validation. This can be rather slow, especially if k-fold cross-validation is used to obtain a less biased estimation. Additionally in the latter case we get a higher potential risk to overfit, because the same feature subset has to be evaluated more often on the training data .

4.2 Genetic Algorithms for Feature Selection using SVMs

If one uses SVMs as the classifier system then one can use one of the bounds (2.19), (2.22) on page on page 22 to estimate the generalization performance. Thus instead of using k-fold cross-validation to evaluate the fitness of an individual one can compute either the Jaakkola-Haussler bound or the R^2W^2 bound, which is much faster, because it requires to retrain the classifier not k times, but just once. This also reduces the potential danger of overfitting (see 2.3) which evolves from multiple evaluations of the same feature subset of the training data. Additionally the whole training set can be used rather than leaving out n/k examples for each step of k-fold cross-validation. In the multi class case SVM learning can be done by separating classes pairwise (see 2.2.2). Thus in this case one can compute the bound for each pair of classes c and $\widetilde{c}$ and sum it up. The term R^2 in bound (2.19) is estimated by the expression $\frac{1}{n}\sum_{i=1}^{n} \mathrm{k}(\mathbf{x}_i, \mathbf{x}_i) - \frac{1}{n^2}\sum_{i,j=1}^{n} \mathrm{k}(\mathbf{x}_i, \mathbf{x}_j)$, which means that every $\beta_i = \frac{1}{n}$ in (2.20). However, there is one problem with using the generalization bounds (2.19) respectively (2.22): They are only valid in the hard margin case, i.e. if the classes are perfectly separable. Nevertheless, using a quadratic penalty function[1] $C\sum_i \xi_i^2$ in (2.15) it can be shown [CorVap95], that the soft margin SVM can be just considered as a special case of the hard margin version with the modified kernel

$$\mathbf{K}' = \mathbf{K} + \frac{1}{C}\mathbf{I} \tag{4.1}$$

where $\mathbf{I}$ is the identity matrix and C a constant penalizing the training errors. Thus each feature subset represented by one individual is evaluated by training a C-SVM with quadratic penalty function and computing the bounds (2.19) respectively (2.22) using the modified kernel matrix (4.1).

This way for problem 2 on page 24 one can set up an evaluation function Ω of the following form: $\varphi(g) = \Omega(g) \cdot \tau(\#\{features\ selected\})$, where Ω

[1] This is the so called 2-norm C-SVM in contrast to the 1-norm C-SVM explained in section 2.2.2.

is a measure for the generalization capability of the SVM trained on the feature subset represented by genome g, and τ is a function which penalizes the number of selected features. Ω can be measured either using one of the bounds (2.19), (2.22) or in the traditional way by performing k-fold cross-validation.

The Genetic Algorithm which was used here, is the so called CHC algorithm [Eshelman91]. This is a non standard Genetic Algorithm which uses a much more aggressive and faster search strategy than the Simple Genetic Algorithm described earlier. The algorithm explicitly borrows from the $(\mu+\lambda)$ strategy of Evolutionary Strategies[2] [Rechenberg73, Schwefel81]: After recombination the best unique members of the offspring generation replace the worst members of the parent generation. This is called *population elitist strategy*. It implies that the mapping of the evaluation value to a fitness value is just the identity. In contrast to the SGA the fitness is thus not dependent on the fitness of the other individuals in the current population. The population elitist replacement method already guarantees considerable selective pressure, so that there is no need for traditional selection mechanisms. Thus CHC uses random selection, but restrictions are imposed on which individuals are allowed to mate. In binary encoding genomes have to have a certain Hamming distance, before they are allowed to reproduce. This form of incest prevention is used to assure diversity. Instead of the usual crossover operator Eshelman uses a special uniform crossover, called HUX (Half Uniform Crossover). Exactly half of the differing genes are exchanged between two mates. No mutation operator is applied during crossover. CHC is typically run on small populations. Eshelman proposes a population size of 50. With such small populations, however, the algorithm fastly begins to reproduce individuals of a similar form, which sooner or later leads to the fact that no individuals are allowed to mate any more. At this point a special mutation operator, called Cataclysmic Mutation, is used. The best individual is left untouched, but all others undergo heavy mutation (e.g. Eshelman proposes a mutation probability of 35% per gene) by using the best individual as a

[2] In the $(\mu+\lambda)$ strategy μ parents produce λ offspring. The population is then reduced again to μ individuals by selecting the best solutions from both parents and offspring.

template. After Cataclysmic Mutation the search is restarted using only HUX.

Sometimes one is not interested in finding the smallest number m of features which maximizes the generalization capability at the same time (special case of problem 2 on page 24), but one has a given number m of features and just wants to find the optimal combination of m features (problem 1 on page 24). In this case of course the search space is much smaller, because only the m-subsets of features have to be taken into account. These are only $\begin{pmatrix} d \\ m \end{pmatrix}$ which is much less than 2^d. Using a Genetic Algorithm like above in this case would be inefficient. Restricting the binary code to just m "1" would make trouble using the HUX-operator on the other hand. Therefore in this case it is reasonable to switch from binary to a decimal encoding $(c_1, ..., c_m)$ where $c_i \in \{1, ..., d\}$ $(i = 1, ..., m)$ indicates the number of the feature which is selected. Of course one has to make sure that each c_i just appears once in the code. Like in the binary case one can formulate the HUX for decimal codes as well. Because the genomes are much shorter with decimal index encoding than with binary encoding the probability to mutate was set from 35% to 50% for each gene.

Now it is interesting to see, how different fitness functions influence feature selection. Therefore 6 different Genetic Algorithms were formulated: three for the case where m is fixed and three for the case where m is not fixed:

1. m is **not** fixed, and the evaluation function is given by

$$\varphi(g) = \text{accuracy}(g) \cdot \left(1 - 0.001 \cdot \frac{\#\{features\ selected\}}{d}\right)$$

   Let's call this algorithm GAAcc. The accuracy can be measured either on an independent validation set or via k-fold cross-validation. This refers to the traditional approach presented in the last section. The goal of the term $0.001 \cdot \frac{\#\{features\ selected\}}{d}$ is to break a tie for smaller feature subsets. The constant 0.001 aims to be lower than the standard deviation of the estimate of the generalization performance for the feature subset represented by genome g.

2. m is **not** fixed, and the evaluation function is given by

$$\varphi(g) = -R^2W^2(g) \cdot \left(1 + 0.001 \cdot \frac{\#\{features\ selected\}}{d}\right)$$

   Let's call this algorithm GAR2W2.

3. m is **not** fixed, and the evaluation function is given by

$$\varphi(g) = \left(-\sum_{p=1}^{n} \alpha_p^* \cdot \mathrm{k}(\mathbf{x}_p^{(g)}, \mathbf{x}_p^{(g)})\right) \cdot \left(1 + 0.001 \cdot \frac{\#\{features\ selected\}}{d}\right)$$

   The notation $\mathbf{x}^{(g)}$ indicates, that the features encoded in genome g are selected from training vector $\mathbf{x}$. The step function in the bound (2.22) on page 23 has been removed to smoothen the fitness function, which helps the Genetic Algorithm to converge. Let's call this algorithm GAJH.

4. m is fixed, and the evaluation function is given by

$$\varphi(g) = \mathrm{accuracy}(g)$$

   Let's call this algorithm m-GAAcc. A penalty function of course in this case is due to the fixed m not necessary.

5. m is fixed, and the evaluation function is given by

$$\varphi(g) = -R^2W^2(g)$$

   Let's call this algorithm m-GAR2W2.

6. m is fixed, and the evaluation function is given by

$$\varphi(g) = -\sum_{p=1}^{n} \alpha_p^* \cdot \mathrm{k}(\mathbf{x}_p^{(g)}, \mathbf{x}_p^{(g)})$$

Let's call this algorithm m-GAJH.

GAAcc, GAR2W2 and GAJH were stopped, if the best solution didn't change for the last 200 generations and at least 10,000 generations had been computed. For the m-GAAcc, m-GAR2W2 and m-GAJH at least 1,000 generations were necessary.

4.3 Optimizing Kernel Parameters with Genetic Algorithms

The Genetic Algorithms described above can also be used for the optimization of the regularization parameter C in equation (4.1). If we have a binary genome $(b_1, ..., b_d),\ b_i \in \{0, 1\}$, we can simply concatenate a binary representation of the parameter C (see also [Goldberg98, pp. 82 - 84]) and run the algorithm GAAcc, GAR2W2 respectively GAJH on the new genome. That means we are trying to select an optimal feature subset and an optimal C at the same time. This is reasonable, because the choice of the parameter C is influenced by the feature subset taken in to account and vice versa. Usually it isn't necessary to consider any arbitrary value of C, but only certain discrete values, e.g. $0.001, 0.01, 0.1, 1, 10, 100, 1000$ respectively $10^{-3}, 10^{-2}, 10^{-1}, 10^{0}, 10^{1}, 10^{2}, 10^{3}$. With just 3 bits we can code the numbers $0, ..., 7$, or, if we shift this representation by -3, the numbers $-3, ..., 4$. If we interpret these numbers as powers of 10, then we get a coding for the possible C values $0.001, 0.01, 0.1, 1, 10, 100, 1000, 10000$. In a similar manner one could use Genetic Algorithms to optimize e.g. the parameter σ for the RBF kernel.

Chapter 5

Experiments

5.1 What and how to Compare

If we want to compare the algorithms described in this thesis, there are arising the following interesting questions:

1. How do Genetic Algorithms in general behave in comparison with Fisher Criterion Score, Relief-F and RFE?
2. Does the choice of the fitness function for the Genetic Algorithm have any influence on the generalization performance?
3. Is the performance of the Genetic Algorithm improved by optimizing the regularization parameter C within the algorihm?
4. Do wrapper algorithms win over filter methods?

Of course it should be clear beforehand that the answers to these questions depend on the data sets which are investigated. Here two artificial data sets as well as two real life data sets (DNA micro array data) are taken into account.

Before we go on to the experiments, we have to think about the first and the last question a little more: Fisher Criterion Score, Relief-F and RFE return a ranking of all features. Thus one always has to say how many features one actually wants to select. This corresponds to problem 1 on page 24 and hence is easily comparable to the m-GA algorithms. However, if we don't know the appropriate number of features beforehand, what are we doing then? With Genetic Algorithms this is no problem, but with the other methods there is no other way than trying out different values of m and choosing the one which induces the least estimated generalization error. This error can then be compared to the one achieved by the GAAcc, GAR2W2 respectively GAJH algorithm with binary encoding. Additionally we can look at the number of features selected by these Genetic Algorithm in comparison with each other and to the best m of the other methods.

Another important question is, what observed differences between test errors are really *significant*, i.e. are that high that they cannot be interpreted as just statistical artifacts. To decide this question one needs a statistical test. The corrected resampled t-test by C. Nadeau and Y. Bengio [NadBen00] is such a statistical test, which has been recently proposed to compare the performance of different learning algorithms. Let n_1 be the size of the training set, n_2 be the size of the test set and J the number of (dependent) trials over which the obtained test errors ε_j, $j = 1, ..., J$ are averaged. Let $\mu^1_{\varepsilon_J}$ be this average for algorithm 1 and $\mu^2_{\varepsilon_J}$ be this average for algorithm 2. Usually J is chosen greater than 1 to obtain more reliable results (e.g. in the following experiments $J = 8, 30, 50$). Now we want to test the hypothesis $H_0 : \mu^1_{\varepsilon_J} - \mu^2_{\varepsilon_J} = 0$ at significance level $1 - \alpha$. According to the corrected resampled t-test, H_0 is rejected at significance level $1 - \alpha$, if

$$\left|\mu^1_{\varepsilon_J} - \mu^2_{\varepsilon_J}\right| > t_{1-\alpha/2,J-1} \cdot \sqrt{\left(\frac{1}{J} + \frac{n_2}{n_1}\right) S^2_J} \tag{5.1}$$

where S^2_J is the variance of $\left|\varepsilon^1_j - \varepsilon^2_j\right|$, $j = 1, ..., J$ for the error rates ε^1_j for algorithm 1 respectively ε^2_j for algorithm 2, and $t_{1-\alpha/2,J-1}$ is the $1 - \alpha/2$ quantile of the Student t_{J-1} distribution (i.e. the Student t distribution with $J - 1$ degrees of freedom). The corrected resampled t-test assumes the differences $\left|\varepsilon^1_j - \varepsilon^2_j\right|$ to be approximately normally distributed. (This can be verified by using e.g. the χ^2-test). The test behaves very conservatively, which means the probability of rejecting H_0 when H_0 is actually true is smaller than the prescribed α. Thus the test might state that a certain observed difference of results is not significant although it actually is, more often than one might expect by the significance level of α. For the following experiments a significance level of 95% was used.

5.2 Data Sets

5.2.1 Toy Data

5.2.1.1 Linear problem

The artificial data set is created as described in [WesMukChaPogVap01]: Six dimensions of 202 were relevant. The probability of the two classes $y = 1$ and $y = -1$ was equal. The first three features $\{X_1, X_2, X_3\}$ were drawn as $X_i = y\mathcal{N}(0,1)$ and the second three features $\{X_4, X_5, X_6\}$ were drawn as $X_i = \mathcal{N}(0,1)$ with a probability of 0.7, otherwise the first three were drawn as $X_i = \mathcal{N}(0,1)$ and the second three as $X_i = y\mathcal{N}(i-3,1)$. The remaining features are noise $X_i = \mathcal{N}(0,20)$, $i = 7, ..., 202$, and the first six features still have redundancy. Figures 5.1 - 5.3 show how well the 2 classes can be separated by just 2 features on a training set of 100 points. How one can see, the quality of the separation depends heavily on which features are selected. Features 3 and 6 induce a very good separation, while features 2 and 3 are much worse and e.g. features 4 and 5 fail completely.

5.2.1.2 Nonlinear problem

2 dimensions of 50 were relevant. Each feature X_i $(i = 1, ..., 50)$ was drawn uniform randomly from the interval (0, 1). Two classes were given, and y was determined by the formula $y = 2\cdot(\max(0, \mathrm{sgn}(X_1 - 0.5)) \text{ XOR } \max(0, \mathrm{sgn}(X_2 - 0.5))) - 1$. Figure 5.4 shows the classification induced by features 1 and 2 on a training set of 100 points.

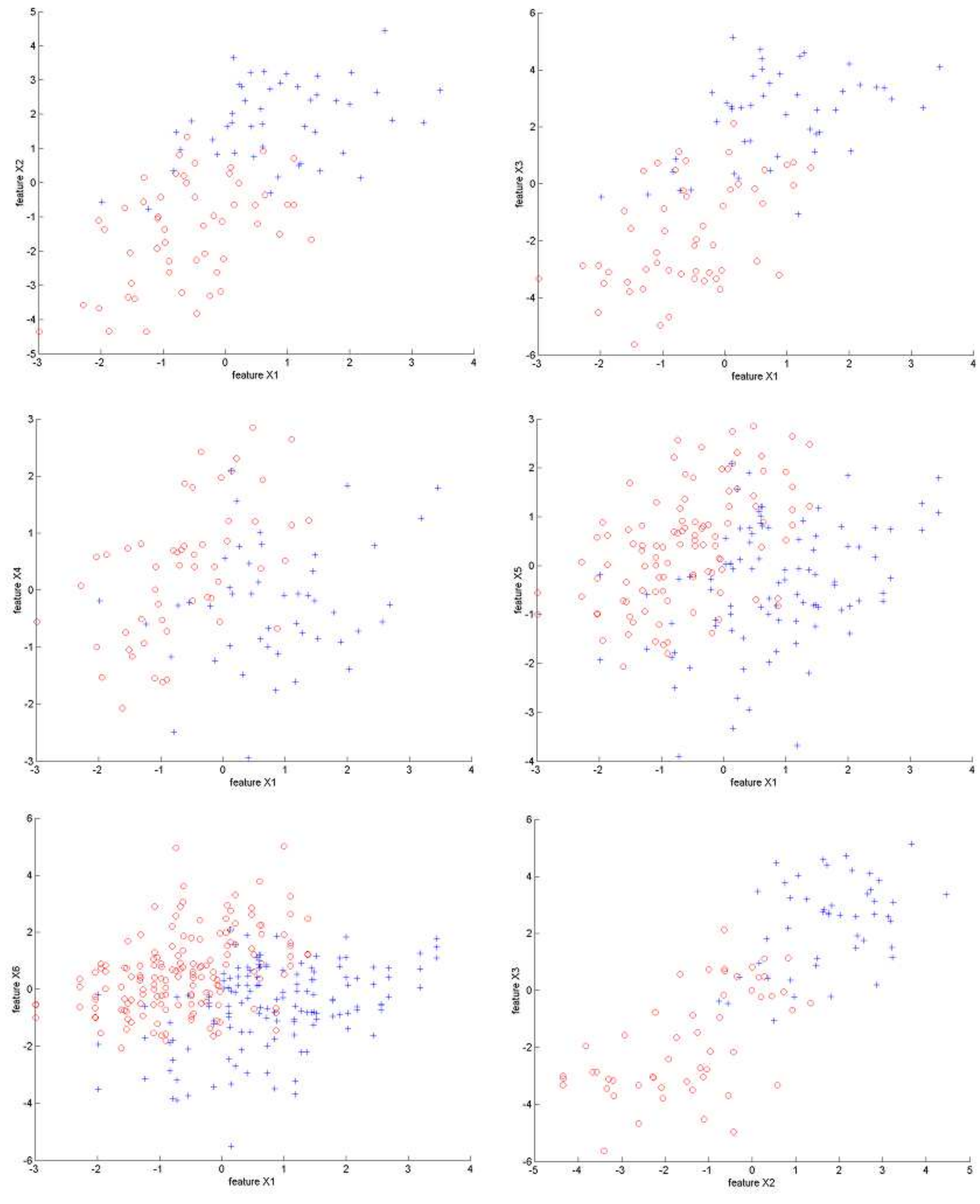

Figure 5.1: Linear toy problem: separation of classes induced by combinations of 2 features which are taken from the first six ones (blue = class +1, red = class -1).

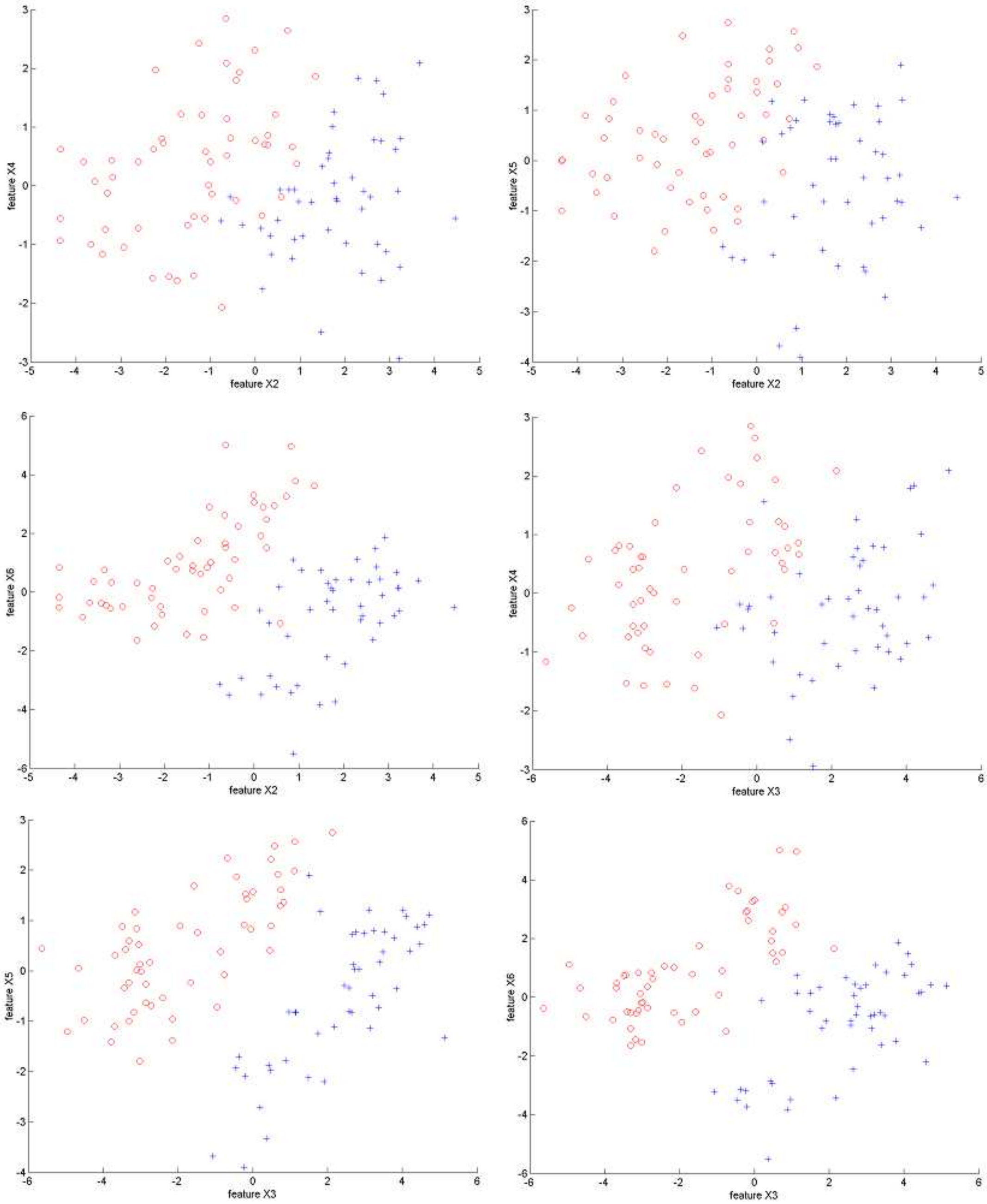

Figure 5.2: Linear toy problem: separation of classes induced by combinations of 2 features which are taken from the first six ones (blue = class +1, red = class -1).

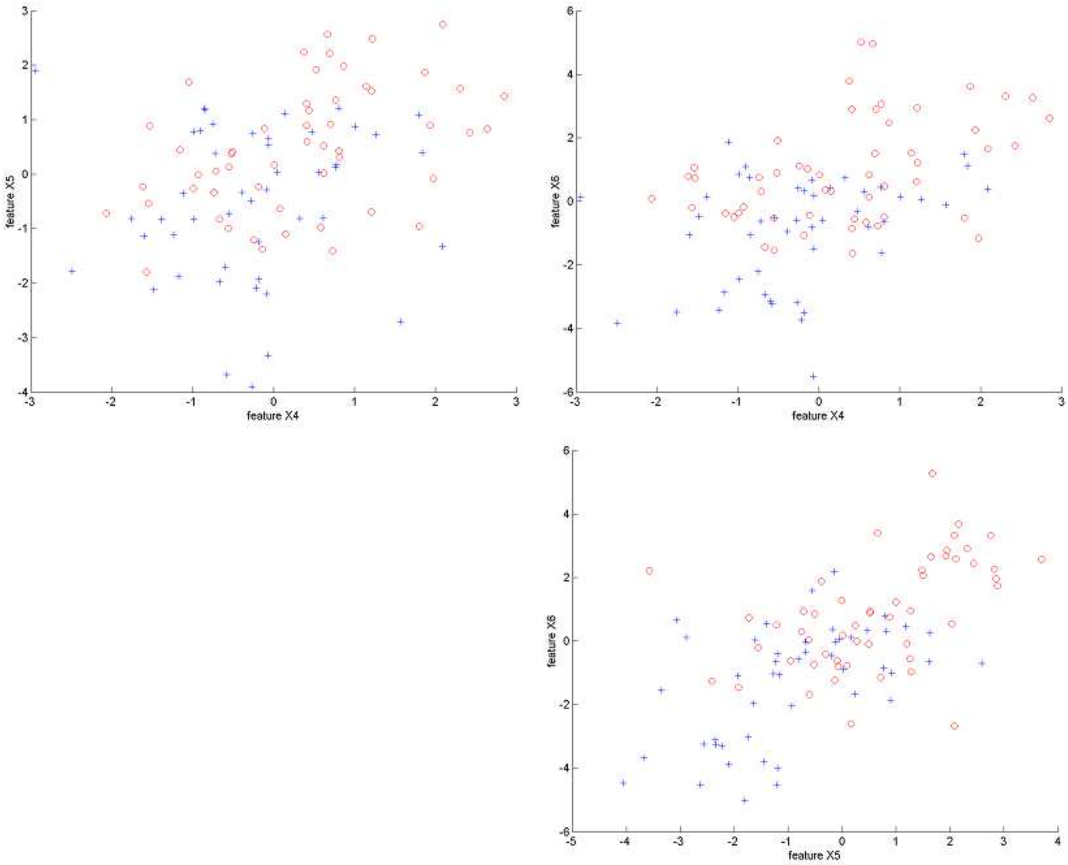

Figure 5.3: Linear toy problem: separation of classes induced by combinations of 2 features which are taken from the first six ones (blue = class +1, red = class -1).

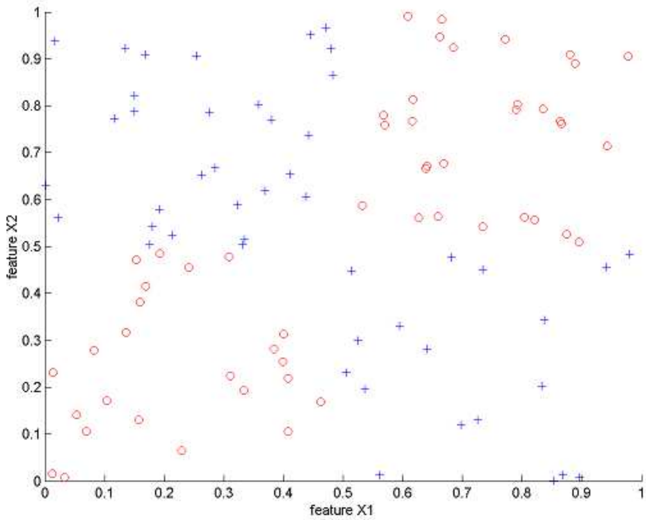

Figure 5.4: Nonlinear toy problem: separation of classes induced by features 1 and 2 (blue = class +1, red = class -1).

5.2.2 Real life data

5.2.2.1 Colon cancer

The Colon cancer problem is described e.g. in [AloBarNotGisYbaMacLev99]: 62 tissue samples probed by DNA micro arrays contain 22 normal and 40 colon cancer examples. These two classes have to be discriminated by the expression of 2,000 genes.

5.2.2.2 Yeast data set

The Yeast micro array data set (Brown Yeast data set, see e.g. [PavWes01]) consists of 208 genes that have to be discriminated into 5 classes based on 79 gene expression values corresponding to different experimental conditions.

5.3 Results

5.3.1 Toy Data

5.3.1.1 Linear problem

30 instances of 100 training points were created. The solution returned by each algorithm was tested on 10,000 independent test points. All data was scaled to the interval [-1, 1]. For the GAAcc algorithm the fitness of each individual was determined by means of 4-fold cross-validation. In the experiment where the number of features to be selected is fixed, m was set to $2, 4$. A 2-norm C-SVM with a linear kernel was used. The regularization parameter C was determined by trying out the values $0.001, 0.01, 0.1, 1, 10, 100, 1000$ and running a complete experiment with each value. Afterwards the C which induces the best average test error was picked. The average results over 30 trials are shown in table 5.1.

For $m = 2$ there is a significant difference between the filter and the wrapper methods. While Fisher Criterion Score and Relief-F have about 15% test error, the other methods obtain around 3%. All wrapper methods find the important features 3 and 6 in most cases (2-GAR2W2, 2-GAJH, RFE 29/30, 2-GAAcc 27/30), but Fisher Criterion Score and Relief-F select features 2 and 3 in all respectively in 29/30 cases. 2-GAAcc performs worse than 2-GAR2W2 and 2-GAJH, but not significantly.

For $m = 4$ the situation is different. Now RFE performs best by finding the relevant features 2, 3, 5, 6 in 27/30 cases. 4-GAR2W2 and 4-GAJH select these features in 25/30 cases, and 4-GAAcc only in 5/30 cases! The filter algorithms use them in only 4/30 cases and select features 1, 2, 3, 6 in 20 (Fisher Criterion Score) respectively 24 cases (Relief-F) instead. Although directly from the test errors there can't be quoted a significant difference between any of these algorithms according to the corrected resampled t-test, this suggests — as it was already suspected in 4.2 — that the problem of overfitting is more serious for 4-GAAcc than for 4-GAR2W2 respectively 4-GAJH, because the classifier has to be retrained on the same data more often.

If m is not fixed, GAAcc performs better than GAR2W2 and GAJH. This is a somewhat astonishing, since with a fixed m it wasn't. Obviously the overfitting problem of cross-validation in a much larger search space is less serious here. Nevertheless the difference to GAR2W2/GAJH is not significant as well as the differences to the best Fisher Criterion Score, best Relief-F and best RFE algorithm in the case where $C = 1$. Table 5.2 shows the number of features which are selected by the Genetic Algorithms. GAAcc selects 4 features on average for both $C = 1$ and C not fixed, which is the same as for the best Fisher Criterion Score/Relief-F/RFE algorithm. In contrast to this GAR2W2 selects 13 respectively 16, and GAJH 12 respectively 18 features. The features selected by GAAcc where among the first 6 in 29/30 cases. Obviously the less accurate estimate of the generalization error by GAR2W2 respectively GAJH leads to a substantially higher number of selected features.

5.3.1.2 Nonlinear problem

Like in the linear problem, 30 instances of 100 training points were created, and the solution returned by each algorithm was tested on 10,000 independent test points. All data was scaled to the interval [-1, 1]. For the GAAcc algorithm the fitness of each individual was determined by means of 4-fold cross-validation. As the Fisher Criterion Score cannot handle nonlinear data this method was not taken into consideration. In the experiment where the number of features to be selected is fixed, m was set to 2. A polynomial kernel of the form $\mathrm{k}(\mathbf{x}, \mathbf{y}) = (0.5 \cdot \langle \mathbf{x}, \mathbf{y} \rangle)^2$ was used. The parameter C in the cost function for the 2-norm C-SVM was set to 1000 following the same procedure as described above. The average results over 30 trials are shown in table 5.3.

In contrast to the linear problem above there is no advantage for the wrapper methods, if $m = 2$. All algorithms find out the relevant features 1 and 2.

For m not fixed the situation is completely different: While GAAcc selects on average 3 features and obtains a test error around 3% for both experiments, GAR2W2 and GAJH fail completely with $C = 1000$ and perform still significantly worse than GAAcc, if C is not fixed beforehand. The difference between GAR2W2 and GAJH is not significant in both cases. As one can see in table 5.4, GAR2W2 doesn't select essentially more features than GAAcc and also finds the important features 1 and 2. The first reason why it performs much worse than GAAcc nevertheless with regard to the test error is that in this problem each additionally selected unimportant feature extremely decreases the generalization capability. The second reason is, that instead of selecting a high C value GAR2W2 and GAJH always choose a small value of 1 or 0.1. Obviously for this problem the R^2W^2 respectively the Jaakkola-Haussler bound are not appropriate, because they don't measure the real generalization error precisely enough. Nevertheless one should not infer from this result, that GAR2W2 respectively GAJH are not applicable to nonlinear problems in general.

5.3.2 Real-life data

5.3.2.1 Colon cancer

The data was split into a training set of 50 and a test set of 12 for 50 times, and results were averaged over these 50 trials. The data was normalized to mean 0 and standard deviation 1 for each feature. For the GAAcc algorithm the fitness of each individual was determined by means of 10-fold cross-validation. In the experiment where the number of features to be selected is fixed, m was set to $20, 50, 100, 250, 500, 1000$. A linear kernel with a 2-norm C-SVM was used. The parameter C in the cost function for the C-SVM was set by the same procedure as described above. The results are shown in table 5.5.

Form m fixed m-GAAcc is the overall worst performing algorithm, especially if m is small. For $m = 20$ the difference to the other methods is significant (figure 5.3.2.1). This can be interpreted as an overfitting problem in the same way as mentioned above. As the number of subsets to be evaluated is higher the smaller m is, the overfitting problem becomes less serious with increasing m. Other differences are not significant for m fixed according to the corrected resampled t-test.

m-GAJH gives the best results with just 20 features, Fisher Criterion Score with 50, Relief-F with 100, RFE with 250, and m-GAAcc and m-GAR2W2 need 500 features. The result obtained by 500-GAR2W2 is comparable to the RFE with 250 features and slightly worse than the Relief-F with 100 features, while the result of 500-GAAcc is almost the same as 20-GAJH.

If m is not fixed, GAR2W2 and GAJH are both better than GAAcc again. For $C = 0.01$ GAAcc performs significantly worse than the best Relief-F algorithm. In the case where C is not fixed, the difference between GAAcc and GAR2W2/GAJH is significant (figure 5.3.2.1). Both GAR2W2 and GAJH seem to win something if C is optimized by the Genetic Algorithm, whereas GAAcc doesn't. GAR2W2 and GAJH reach approximately the same test error as the best RFE in this case. They are only slightly worse than the best Relief-F algorithm.

GAR2W2 and GAJH select substantially more (around 200 respectively 400) features than GAAcc (around 40 respectively 50 — see table 5.6). This essential difference corresponds to the results of the other data sets. It should be compared to the 250 features selected be the best RFE, 100 features selected by the best Relief-F and 50 features selected by the best Fisher Criterion Score algorithm (figure 5.3.2.1).

5.3.2.2 Yeast data set

8-fold cross-validation was performed on this data for all methods. That means that the data was split into 8 sets of 26 examples, and each time one of these 8 sets was used for testing, and the remaining 7 sets (consisting 182 examples) for training. Afterwards results were averaged. The data was normalized to mean 0 and standard deviation 1 for each feature. For the GAAcc algorithm the fitness of each individual was determined by means of 7-fold cross-validation. In the experiment where the number of features to be selected is fixed, m was set to $10, 20, 40$. A linear kernel with a 2-norm C-SVM was used which discriminates classes pairwise (one-versus-one method).

The parameter C in the cost function for the C-SVM was set by the same procedure as described above. The results are shown in table 5.7.

For $m = 10$ Relief-F is significantly worse than 10-GAR2W2. This again shows the potential problems of a filter method. Like in prior experiments m-GAR2W2 and m-GAJH are always better than m-GAAcc, but this is not significantly here.

Interestingly m-GAAcc gives the best results with only 10 features, while Relief-F needs 20 and the other algorithms 40 features (figure 5.3.2.2).

If m is not fixed and $C = 100$, GAAcc performs the same as GAR2W2 and a bit worse than the best RFE and the best Fisher Criterion Score (figure 5.3.2.2). But if C is determined by the Genetic Algorithm, GAR2W2 is better than GAAcc and GAJH. Like in prior experiments, the Genetic Algorithms using bounds seem to win something, if C is optimized during the search process, while this is not the case for the GAAcc algorithm. However, according to the corrected resampled t-test the differences between all the algorithms are not significant. Table 5.3.2.2 shows the number of features, which are selected by the Genetic Algorithms. Like in prior experiments, the GAAcc algorithm selects the fewest features (9 ± 1 for $C = 100$, 8 ± 1 for C not fixed). GAR2W2 and GAJH need about three times as many features (GAR2W2 $23 \pm 3/23 \pm 2$, GAJH $26 \pm 3/23 \pm 2$) which should be compared to 40 features taken by the best Fisher Criterion Score and RFE respectively 20 taken by the best Relief-F (figure 5.3.2.2).

Experiment type		Method (average % test error $\pm$ standard deviation)						
m	C	**GAAcc**	**GAR2W2**	**GAJH**	**Fisher**	**Relief-F**	**RFE**	**no feat. sel.**
2	1	3.4 ± 3	2.8 ± 1.6	2.8 ± 1.6	15 ± 0.4	15 ± 0.4	2.8 ± 1.6	9.3 ± 5
4	1	2.6 ± 1.3	1.3 ± 1.3	1.2 ± 0.8	$\mathbf{1.8 \pm 2}$	$\mathbf{1.4 \pm 0.6}$	$\mathbf{1.05 \pm 0.7}$	9.3 ± 5
not fixed	1	$\mathbf{1.6 \pm 1.2}$	2.5 ± 1.5	3.4 ± 3.8				9.3 ± 5
	not fixed	$\mathbf{1.5 \pm 1}$	3.2 ± 2	3.5 ± 2.1				

Table 5.1: Linear toy problem

C	**number of selected features** (average number ± standard deviation)		
	GAAcc	**GAR2W2**	**GAJH**
1	4 ± 1	13 ± 3	12 ± 4
not fixed	4 ± 1	16 ± 4	18 ± 4

Table 5.2: Linear toy problem: number of selected features

Experiment type		**Method (average % test error ± standard deviation)**					
m	C	**GAAcc**	**GAR2W2**	**GAJH**	**Relief-F**	**RFE**	**no feat. sel.**
2	1000	1.6 ± 0.9	1.6 ± 0.9	1.6 ± 0.9	1.6 ± 0.9	1.6 ± 0.9	41.3 ± 0.9
not fixed	1000	3 ± 1.6	36.9 ± 1.3	35.2 ± 2			41.3 ± 0.9
	not fixed	2.5 ± 1.3	13.5 ± 4.3	16.7 ± 4.3			

Table 5.3: Nonlinear toy problem

C	**number of selected features** (average number $\pm$ standard deviation)		
	GAAcc	**GAR2W2**	**GAJH**
100	3 ± 1	29 ± 3	25 ± 3
not fixed	3 ± 1	4 ± 1	6 ± 2

Table 5.4: Nonlinear toy problem: number of selected features

Experiment type		**Method (average % test error $\pm$ standard deviation)**						
m	C	**GAAcc**	**GAR2W2**	**GAJH**	**Fisher**	**Relief-F**	**RFE**	**no feat. sel.**
20	0.01	22.8 ± 10.5	15.8 ± 9.3	$\mathbf{16.3 \pm 10.4}$	15 ± 9.4	14.7 ± 8.8	16.3 ± 9.5	16.2 ± 9.4
50	0.01	20.3 ± 10.3	16 ± 9	16.7 ± 9.4	$\mathbf{14.8 \pm 9.1}$	14.3 ± 9.8	16.2 ± 9.1	16.2 ± 9.4
100	0.01	19.7 ± 10.6	15.7 ± 9.5	17.2 ± 9.1	15.2 ± 9.7	$\mathbf{14.3 \pm 9.5}$	16.3 ± 8.7	16.2 ± 9.4
250	0.01	17.2 ± 11	16.2 ± 8.8	16.7 ± 8.9	15.7 ± 9	16.5 ± 10	$\mathbf{15.8 \pm 8.3}$	16.2 ± 9.4
500	0.01	$\mathbf{16.3 \pm 8.7}$	$\mathbf{15.3 \pm 9.1}$	16.5 ± 9	17.2 ± 9	15.5 ± 9.4	16 ± 9.4	16.2 ± 9.4
1000	0.01	17.2 ± 8.5	16.7 ± 9.7	16.7 ± 9.4	17.8 ± 8.1	16 ± 9.3	16.3 ± 9.4	16.2 ± 9.4
not fixed	0.01	21.3 ± 12	$\mathbf{16.2 \pm 9}$	17.5 ± 8.5				16.2 ± 9.4
	not fixed	21.7 ± 10.8	$\mathbf{15.5 \pm 8.7}$	15.8 ± 9.3				

Table 5.5: Colon data set

C	**number of selected features** (average number $\pm$ standard deviation)		
	GAAcc	**GAR2W2**	**GAJH**
100	42 ± 20	221 ± 14	236 ± 17
not fixed	49 ± 30	382 ± 35	388 ± 35

Table 5.6: Colon data set: number of selected features

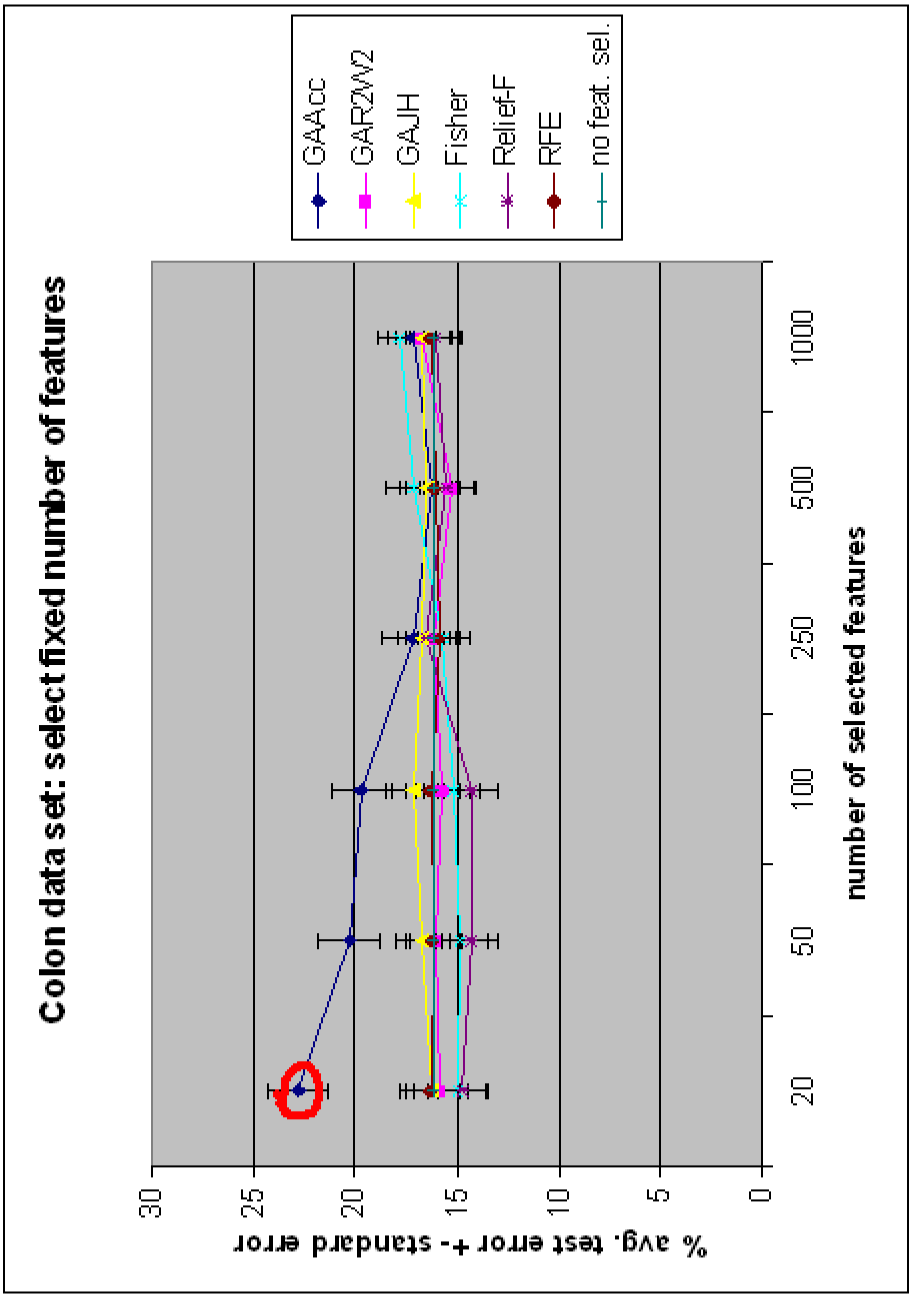

Figure 5.5: Colon data set: select fixed number of features

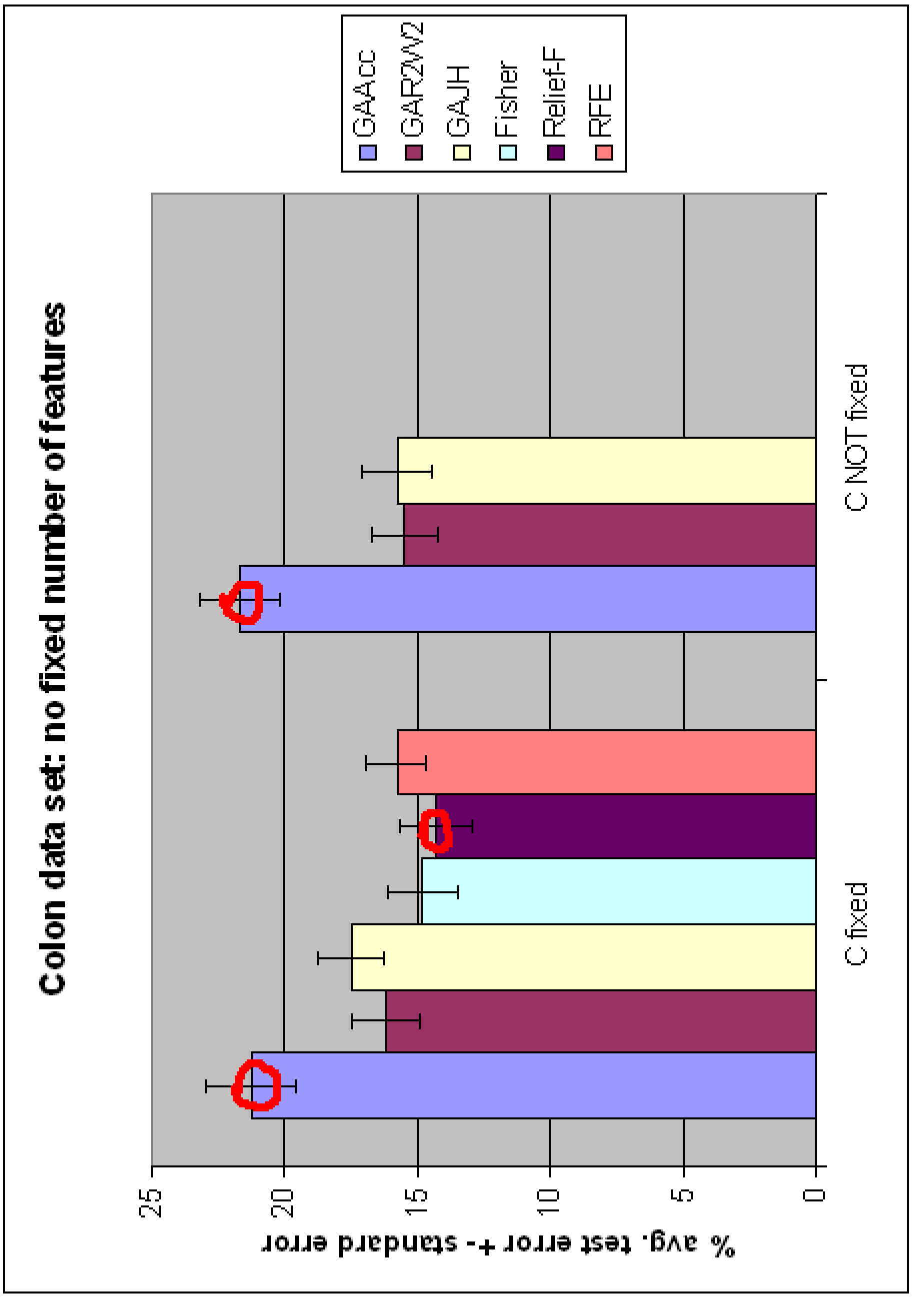

Figure 5.6: Colon data set: no fixed number of features

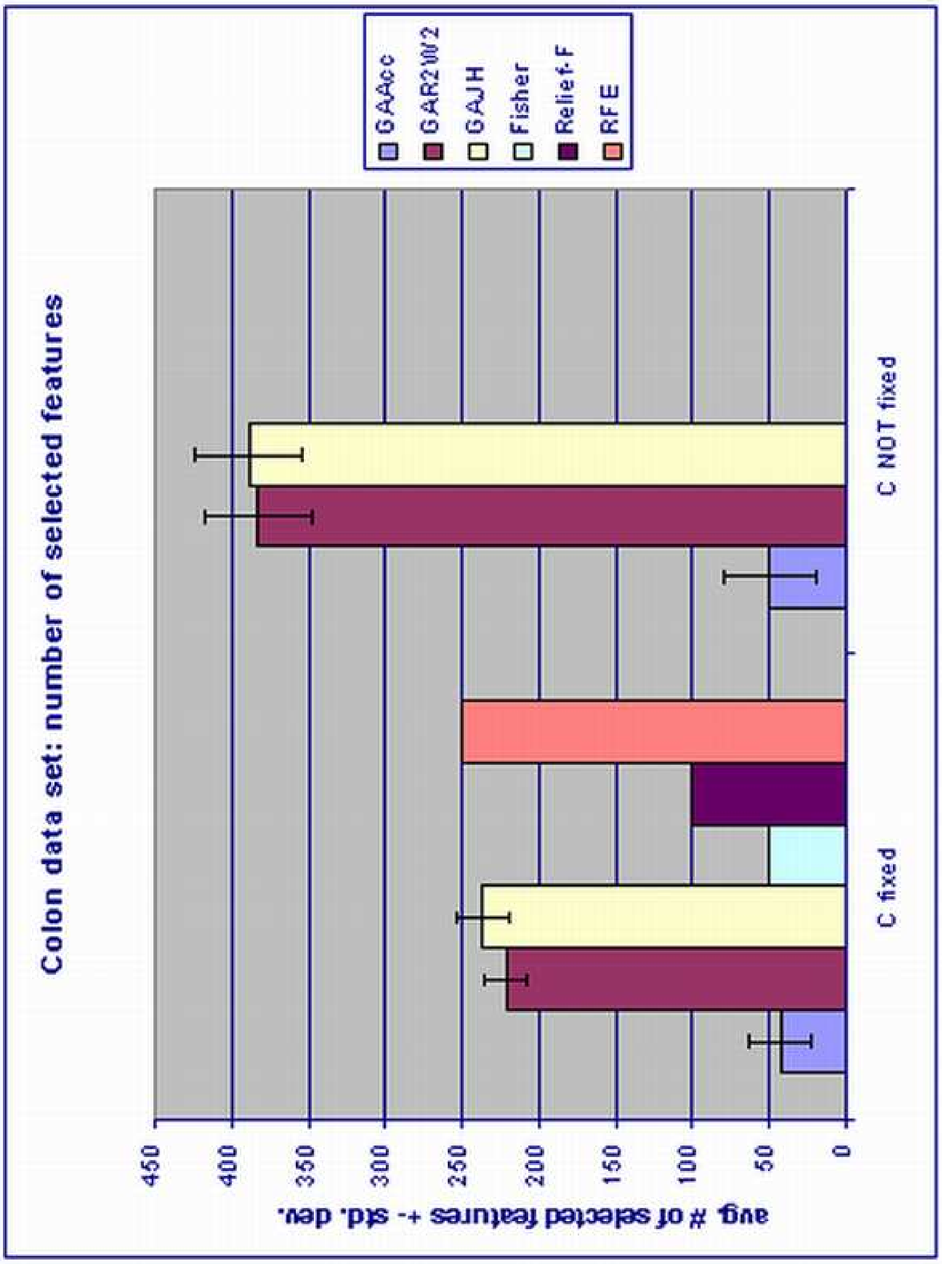

Figure 5.7: Colon data set: number of selected features

Experiment type		Method (average % test error ± standard deviation)						
m	C	**GAAcc**	**GAR2W2**	**GAJH**	**Fisher**	**Relief-F**	**RFE**	**no feat. sel.**
10	100	$\mathbf{5.8 \pm 5}$	4.3 ± 4.8	5.3 ± 4.1	8.7 ± 8.4	11.5 ± 6.2	5.8 ± 4.1	3.8 ± 3.6
20	100	6.7 ± 5.7	5.3 ± 5	6.7 ± 5.7	5.8 ± 5.8	$\mathbf{5.3 \pm 3.5}$	5.8 ± 5.8	3.8 ± 3.6
40	100	6.3 ± 4.6	$\mathbf{3.8 \pm 4.1}$	$\mathbf{3.8 \pm 4.1}$	$\mathbf{3.9 \pm 4.1}$	5.8 ± 4.6	$\mathbf{3.4 \pm 3.8}$	3.8 ± 3.6
not fixed	100	$\mathbf{4.8 \pm 4}$	$\mathbf{4.8 \pm 4.8}$	5.8 ± 5				3.8 ± 3.6
	not fixed	5.3 ± 5	$\mathbf{3.4 \pm 3.2}$	4.3 ± 4.3				

Table 5.7: Yeast data set

C	**number of selected features** (average number $\pm$ standard deviation) **GAAcc**	**GAR2W2**	**GAJH**
100	9 ± 1	23 ± 3	26 ± 3
not fixed	8 ± 1	23 ± 2	23 ± 2

Table 5.8: Yeast data set: number of selected features

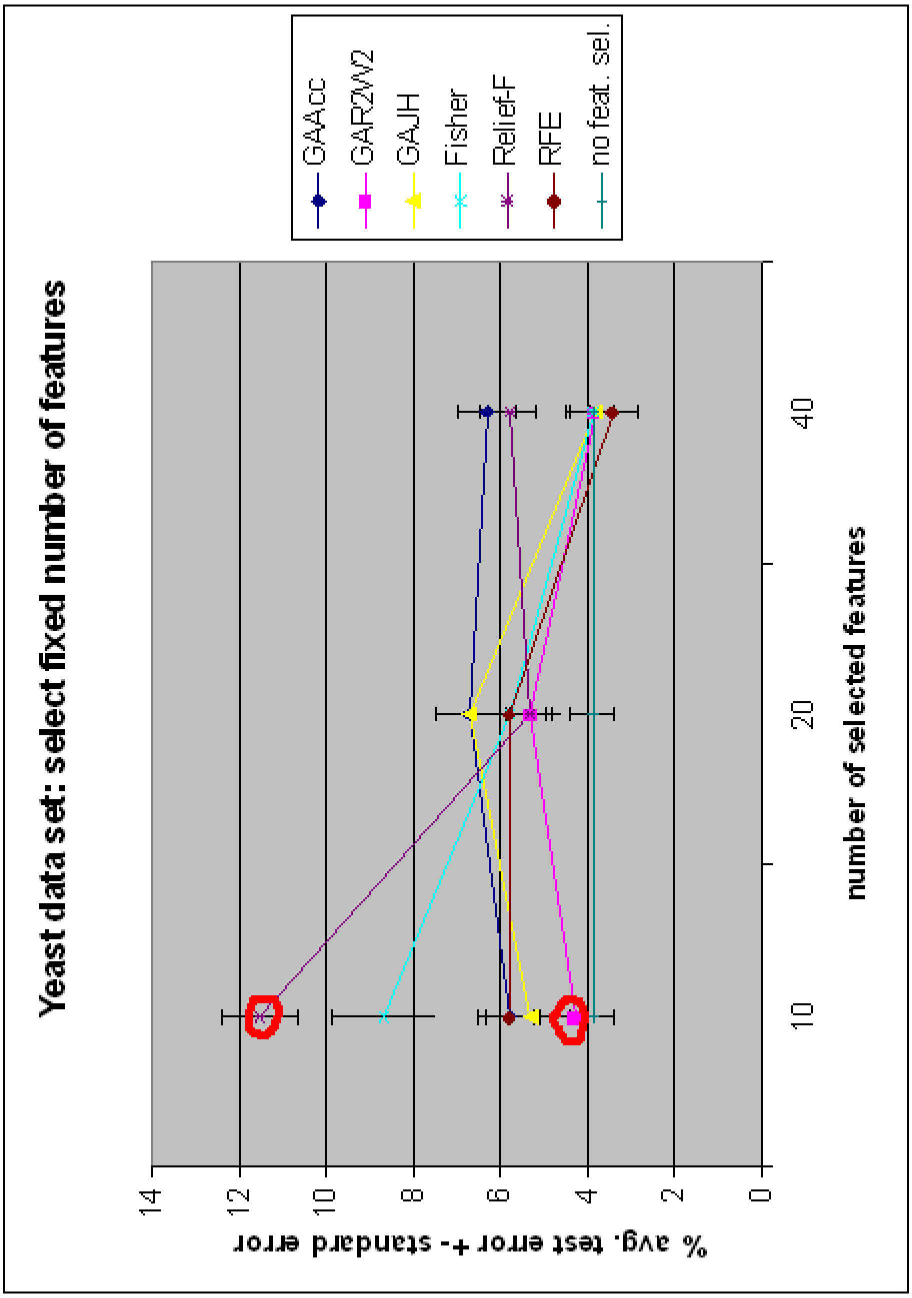

Figure 5.8: Yeast data set: select fixed number of features

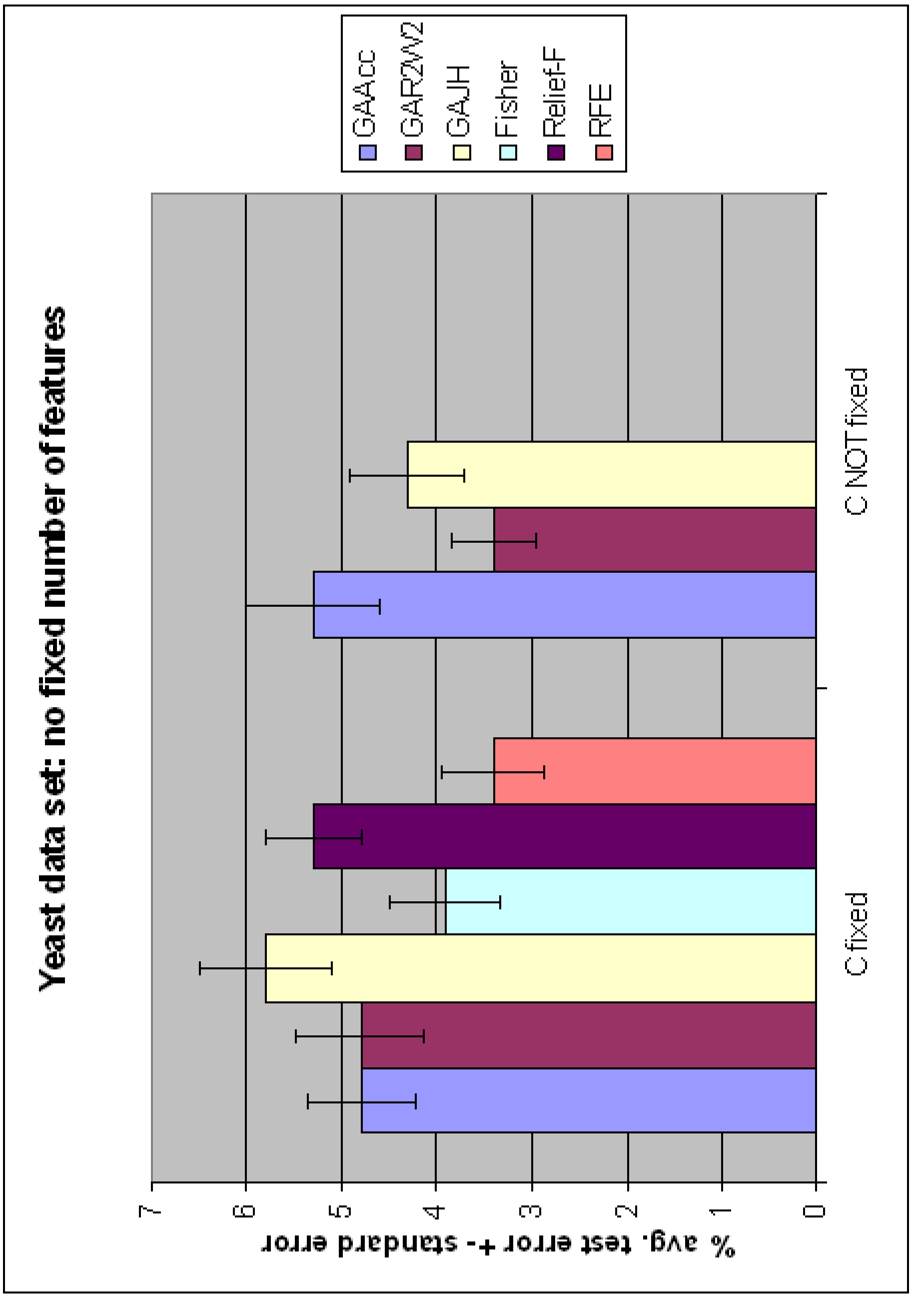

Figure 5.9: Yeast data set: no fixed number of features

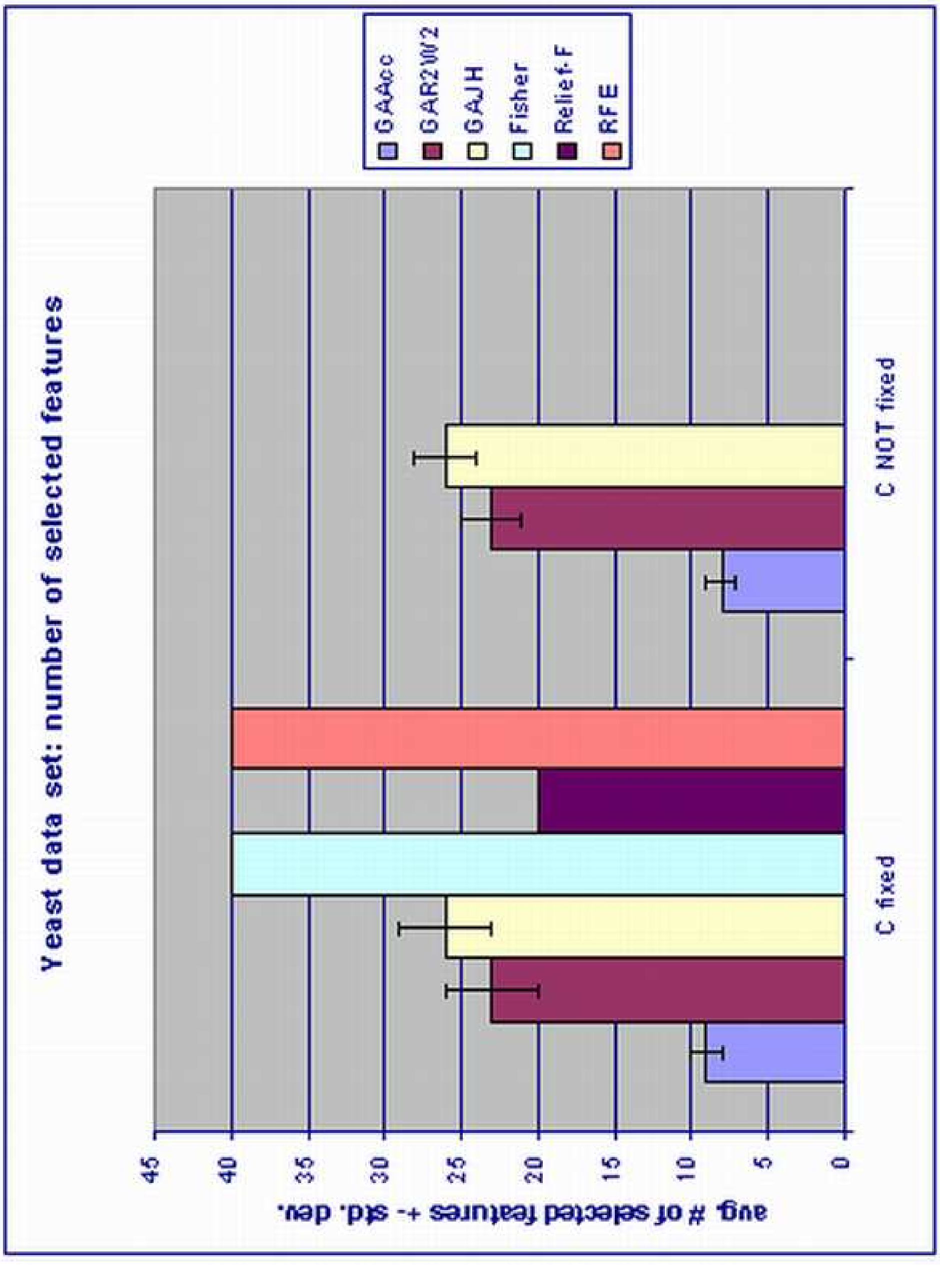

Figure 5.10: Yeast data set: number of selected features

Chapter 6

Discussion of Results

In the last chapter different Genetic Algorithms were compared with each other and with Fisher Criterion Score, Relief-F and RFE on two toy data problems as well as on two real life data sets. On real world data sets the GAAcc/m-GAAcc algorithm shows in some cases a significant overfitting problem. Here is an advantage of the GAR2W2/m-GAR2W2 respectively GAJH/m-GAJH algorithm. Both algorithms have a comparable performance, GAR2W2/m-GAR2W2 being slightly better in most experiments. If m is not fixed beforehand they select about the same number of features, which is substantially higher than the number obtained by GAAcc. Especially GAR2W2 and GAJH benefit from the optimization of the parameter C within the Genetic Algorithm. The average test error of GAR2W2 with parameter optimization is on real life data comparable to a RFE for which kernel parameters and the appropriate number of features is determined by multiple trials. However, the number of selected features is, except on the Yeast data set, higher. Comparing computation times, one needs for one run (that means for one of J trials) of GAR2W2 with parameter optimization on the Colon data set about 30 minutes on a Pentium IV with 2 GHz. One run of a RFE takes about 2 minutes, but to determine the regularization parameter $(0.001, 0.01, 0.1, 1, 10, 100, 1000)$ and the appropriate number of features $(20, 50, 100, 250, 500, 1000)$ one would need all in all $7 \cdot 6 = 42$ runs. Thus in fact it takes $42 \cdot 2 = 84$ minutes, which is almost 3 times more than with GAR2W2! Thus in fact one can save time using GAR2W2 instead of multiple runs of RFE.

Fisher Criterion Score and Relief-F are in some cases significantly worse than e.g. m-GAR2W2 (see Yeast data set, linear toy data set). Especially the linear toy data shows the drawback of filter methods. While for $m = 2$ none of the wrapper methods had problems finding the important feature subset, the filter algorithms significantly failed. That shows that wrapper algorithms offer a more reliable measure of importance for feature subsets than filter approaches which induce a separate inductive bias by means of the importance measure taken (compare 2.3).

As a general conclusion from the experiments shown above one can state the following:

- Genetic Algorithms using cross-validation to evaluate a given feature subset show in some cases a significant overfitting problem.

- Using leave-one-out error bounds instead is an alternative. It leads to a better generalization performance in most cases, but if the number of features to select is not fixed beforehand, a higher number of features is selected than with cross-validation.

- Optimizing kernel parameters within the Genetic Algorithm is useful, especially if leave-one-out error bounds are used.

- Genetic Algorithms using the R^2W^2 bound show a comparable generalization performance to RFE. The number of selected features is in most cases a bit higher. If the number of features is not fixed beforehand and kernel parameters are optimized within the Genetic Algorithm, one can in fact save time by using such an algorithm instead of running RFE multiple times to determine kernel parameters and an appropriate number of features. If the number of features to select is known beforehand, however, Genetic Algorithms don't offer these advantages any more. Thus in this case Genetic Algorithms are not recommendable.

- Filter methods are partially significantly worse than wrapper algorithms.

It has to be mentioned that the significances, which were found here, depend on the used statistical test. Hence a more liberal statistical test might have found more significant differences. It is also important to understand, that the termination conditions of Genetic Algorithms have an influence on the obtained results. Finally the experimental results depend on the data sets used.

Chapter 7

Conclusion

This Diploma Thesis dealt with the problem of feature selection for Support Vector Machines by means of Genetic Algorithms. Feature selection is an important issue in Machine Learning and of great practical importance, e.g. in bioinformatics. Especially in this domain we have data with very high dimensionality (> 1000), but only a few features are relevant to obtain the best possible generalization performance of our learning machine, e.g. Support Vector Machine. The two central problems of feature selection are

- What does the term “relevant” mean, i.e. what is an appropriate relevance measure for features?
- How can relevant features be found?

Formally the problem of feature subset selection in Machine Learning can be addressed in the following two ways:

1. Given a fixed $m \ll d$, find the m features that give the smallest expected generalization error; or
2. Given a maximum allowable generalization error γ, find the smallest number m of features.

This should not be confused with feature extraction as it is done in PCA or ICA. Two general approaches are known to solve the feature selection problem: the filter approach and the wrapper approach. In a filter method feature selection is performed as a preprocessing step to the actual learning algorithm, i.e. before applying the classifier to the selected feature subset. Features are selected with regard to some predefined relevance measure which is independent of the actual generalization performance of the learning algorithm. This can mislead the feature selection algorithm. Wrapper methods, on the other hand, train the classifier system with a given feature subset as an input and return the estimated generalization performance of the learning machine as an evaluation of the feature subset. This step is repeated for each feature subset taken into consideration. Thus the wrapper approach requires multiple training runs of the learning machine on the training data and hence induces a potential danger of overfitting, especially, if k-fold cross-validation is used as an estimator of the generalization error, because the same feature subset has to be evaluated k times.

The problem of feature selection can be viewed as a combinatorial task: Given the set of all features, find out the combination of features that minimizes the expected generalization error. Especially if we have very high dimensional data, e.g. DNA micro array data, the search space is too big to perform an exhaustive search. Thus Genetic Algorithms as non-exhaustive, stochastic search methods offer a natural way to explore the search space. This has been done before in combination with cross-validation and a binary encoding of the search space.

In this thesis Support Vector Machines are used as the learning machine and thus one can take existing theoretical bounds for the generalization error instead if cross-validation to evaluate a given feature subset. On hand hand these bounds give a less accurate estimate of the generalization error. But on the other hand they reduce the potential danger of overfitting, if they are used within a wrapper approach. Additionally they are computationally attractive. If the number of features to be selected is fixed and hence the search space is much smaller than with a variable number of selected features, we can use a decimal encoding, which is much more efficiently than a binary encoding that representes all possible feature subsets of any size. If the number of features to be selected is not fixed beforehand, the usual binary encoding was taken. Additionally to the selection of a feature subset, one can optimize kernel parameters such as the regularization parameter C of the SVM by means of Genetic Algorithms. This is reasonable, because the choice of the feature subset has an influence on the appropriate kernel parameters and vice versa.

Existing algorithms such as Fisher Criterion Score, Relief-F and Recursive Feature Elimination were compared to Genetic Algorithms using cross-validation and to Genetic Algorithms using two different error bounds on two toy problems and two DNA micro array data sets. Hereby Recursive Feature Elimination is a heuristic wrapper algorithm which was especially designed for SVMs, and Fisher Criterion Score and Relief-F are two filter algorithms. As a conclusion of the experiments one can state that Genetic Algorithms using the R^2W^2 bound and optimizing various kernel parameters are a recommendable alternative, if the number of features to select is not known beforehand. It reduces overfitting in comparison with k-fold cross-validation in most cases. Additionally, in comparison with running RFE multiple times to determine the kernel parameters and an appropriate feature subset, one in fact saves time.

A further application of Genetic Algorithms in Machine Learning could be e.g. to construct features as products from existing ones rather than searching for feature subsets. The Genetic Algorithm could be used to find the products which give the smallest expected generalization error. This way maybe one could detect nonlinearities in the data. The search space in this case is much bigger than for the problem of feature selection. Especially if Support Vector Machines are used, one could try a similar setting as described in this thesis and evaluate a given set of products by means of the bounds for the generalization error which were described here. This should be subject to further investigation.

Appendix A

Implementation

A.1 Program Description

All feature selection algorithms which are presented in this work are implemented in a program “Featuresel”. The program has two different user interfaces: a graphical interface and a command line version. The graphical interface was developed especially for Microsoft Windows and is mainly for demonstration purposes, while the command line version is completely platform independent and quicker to use. The command line version can be either run as a binary or as a Java class under Windows:

```
featuresel [#features to select] [options] <training_file>
<test_file> <output_file> [<validation_file]
```

respectively

```
java featuresel.Featuresel [[#features to select] [options]
<training_file> <test_file> <output_file> [<validation_file]]
```

is under Windows the same, if arguments are passed. If no arguments are passed the command

```
java featuresel.Featuresel
```

will invoke the graphical interface. The meaning of `options` is the following:

```
-t kernel_type: set type of kernel function (default 2):

    0 -- linear (u*v')
    1 -- polynomial ((gamma * u*v'+ coef0)^degree)
    2 -- radial basis function (exp{-gamma * (u*v')^2})
    3 -- sigmoid (tanh{gamma * u*v' + coef0})
-d degree: degree in kernel function (default 3)
-g gamma: gamma in kernel function (default 1/#{input features}
-r coef0: coef0 in kernel function (default 0)
-c cost: parameter C of C-SVM (default 1)
-v n: n-fold cross-validation for testing
-x n: n-fold cross-validation for training
-o: optimize regularization parameter C (only usable with
genetic algorithms and binary encoding)
-M n: show messages yes/no (default 1)
    0 -- no
    1 -- yes
-F n: set feature selection algorithm (default 0)
    0 -- no feature selection
    1 -- Fisher Criterion Score
    2 -- Relief-F
    3 -- Recursive Feature Elimination
    4 -- Genetic Algorithm (accuracy on validation set as
         criterion or - if no validation set is given -
         cross validation on
         training set)
    5 -- Genetic Algorithm (R^2W^2 criterion)
    6 -- Genetic Algorithm (Jaakkola-Haussler criterion)
```

The format of the training and testing data file is:

```
<label> <index>:<value> {<index>:<value>}
.
.
.
```

`<label>` is an integer which represents the class label of the training respectively testing example. `<index>` is an integer starting from 1 for each example, and `<value>` is a real number. If no number of features to select is passed and the algorithm chosen is **not** a Genetic Algorithm, no feature selection is performed. If a Genetic Algorithm is chosen, the algorithm tries to find out the appropriate number of features by itself. In this case it is also possible to optimize the regularization parameter C simultaneously.

The following picture shows the main dialog of the graphical interface: The user can select the algorithm, the necessary files, whether to perform cross-validation or not and the kernel parameters interactively. If the check box "show messages" is activated each algorithm prints out some information during the run into an output window. Otherwise nothing is printed during the run, but in the end "ready" is written to the standard I/O output stream. If the message output window is open, the user can stop the computation of an algorithm by clicking the "Finish" button. If the user chose a Genetic Algorithm as the feature selection algorithm, common statistic evaluations can be seen in a plot: The plot below shows the number of features of the best individual of a certain generation. The plot above shows the fitness value of the best individual, the average fitness per generation, the online performance (i.e. the running average over all fitness values up to the current generation) and the offline performance (i.e. the running average of the fitness values of the best individuals up to the current generation). The latter definitions follow De Jong [DeJong75]. The user can use the mouse to zoom into the plot by dragging a rectangle beginning with the upper left corner, or zoom out of the plot by dragging a rectangle beginning with the lower right corner. If the plot window is closed the computation continues until it is ready or stopped by clicking the "Finish" button in the message output window.

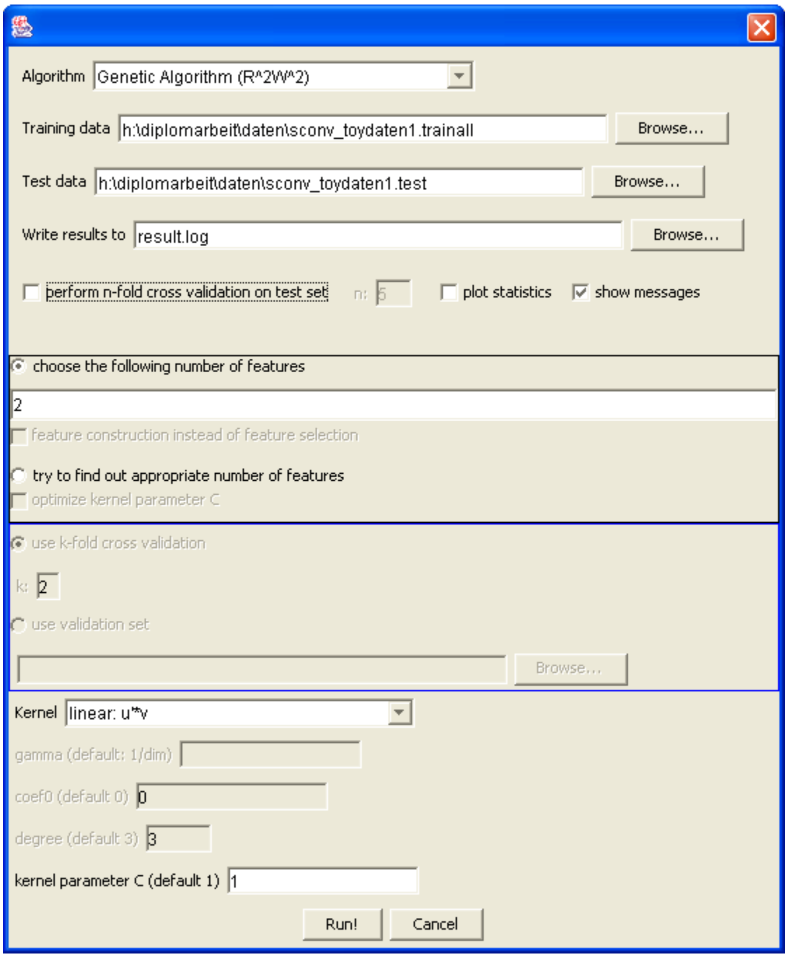

Figure A.1: Main dialog of the graphical interface

A.2 Program Architecture

Basically it would be right not to talk about one but two programs: one which offers a graphical interface and one which doesn't. The program which doesn't is completely written in C++ and therefore creates an executable binary. The one with the graphical interface is a combination of Java and C++. The graphical interface itself is written in Java, while the core algorithms are encapsulated in a Dynamic Link Library (DLL). The DLL is loaded at the initialization of the program. Whenever a feature selection algorithm shall be started, a new thread is created in which the DLL is entered and the algorithm is invoked. The communication between Java and C++

is done by means of the Java Native Interface (see e.g. `http://java.sun.com/docs/books/tutorial/native1.1/index.html`). From the C++ side the only difference between the DLL and the executable binary version of the program is the code for the invocation of the feature selection algorithms. For the executable binary program this is done in the file `main.cpp` which includes the main method of the program. The command line is parsed and afterwards the correct algorithm with the right parameters is started. The DLL has a file `invoke.cpp` instead which offers two DLL methods, one to start an arbitrary algorithm and one to stop it (which happens, if the "Finish" button is clicked). In this case a boolean variable is switched which is regularly checked and which tells the algorithm to terminate. The invocation of an algorithm is done in a similar manner like in the `main.cpp` file.

A.3 Class Architectures

Figure A.3 shows a class diagram of the basic classes:

The class `GeneticAlgorithm` contains the Genetic Algorithms, the class `RecursiveFeatureElimination` contains the RFE algorithm, and the filter methods are implemented in the class `FilterRanking`. The Wrapper algorithms require the training of a Support Vector Machine repeatedly. The SVM algorithm is a public available implementation, called `libsvm`, which can be downloaded from `http://www.csie.ntu.edu.tw/~cjlin/libsvm`. It defines several interface methods in the file `svm.h`. The most important are `svm_train` and `svm_predict` which lead to a training of a SVM with a certain input respectively a prediction of a ready trained SVM on some pattern. The static method `k_function` of the class `Kernel` inside `libsvm` are used to compute the value of the kernel function on some training vectors. The `libsvm` defines three important structures: `svm_model`, `svm_problem` and `svm_parameter`. The structure `svm_model` represents the hypothesis which a ready trained SVM has learned. The structure `svm_problem` stores a dataset in memory, and the structure `svm_parameter` describes all paramters, e.g. kernel parameters, of a SVM. For more information about `libsvm` see the documentation which comes along with this package.

The purpose of the classes `TextWrapper` and `PlotWrapper` is to make the implementation of the classes `GeneticAlgorithm`, `RecursiveFeatureElimination` and `FilterRanking` independent of whether the graphical interface is used, i.e. the output is printed into some window, or not. In the latter case outputs are printed to some output stream. That means the overloaded `<<` operators in class `TextWrapper` simply passes the argument to the output stream which was given to the constructor before. On the other hand, if the graphical interface is used, an other constructor is used which receives a pointer to a Java object. The Java object has a method `printString` which prints a certain string to the output window of the graphical user interface. Whenever the second constructor is called the private variable `java` is set to false which indicates the overloaded `<<` operators to invoke the java method `printString` and pass the argument to this method. Similar the class `PlotWrapper` has a constructor which receives the pointer to a Java object that has a method `addPoint`. The purpose of this method is to add a given point to a certain dataset within some plot. Whenever the `PlotWrapper` method `addPoint` is called it simply invokes the Java method `addPoint` and passes the given arguments. This way the class `GeneticAlgorithm` doesn't need to know whether the graphical interface is used or not. It simply calls the methods from `PlotWrapper` and `TextWrapper` and they pass the arguments in the right way. The `PlotWrapper` methods are called only, if the `PlotWrapper` object wasn't `NULL`. That means if one doesn't want to use the graphical interface, the constructor of `GeneticAlgorithm` can be simply called with `NULL` instead of an existing `PlotWrapper` object.

The class `Population` stores a population of individuals for the Genetic Algorithm in an array. Instead of the objects only pointers to objects of class `Individual` are stored. The class `Population` offers a method which prints out the actual population to a `TextWrapper` object. For this purpose it makes use of the overloaded `<<` operator of class `Individual`. `Population` also offers a `equals` method which returns true if to populations contain the same individuals. For this it makes use of the `equals` method of class `Individual`.

The class `Individual` represents an individual for the Genetic Algorithm. To make it more general in use, the constructor receives an object of class `GenomeInd` which is an abstract superclass for all genomes of an individual. Additionally it receives a pointer to an object of class `RandGen`. `RandGen` offers a general purpose random generator. The implementation is taken from [Goldberg98] and more information can be found there as well. `Individual` offers methods `computeFitnessVal`, `computeFitnessCV` and `computeFitnessBound` which compute the fitness of an individual with a genome representing some feature subset by means of validation on an independent validation set, k-fold cross-validation on the training set or by one of the bounds (2.19), (2.22). For the necessary SVM training the functionality of the `libsvm` is used. By means of the method `getSelectedFeatures` one returns a feature subset of a given input set with respect to the genome of the individual.

As mentioned before the class `GenomeInd` is an abstract superclass for all genomes of an individual. Among others it offers the methods `HUX` and `mutate` which perform a HUX between two genomes respectively a Cataclysmic Mutation on the actual genome. Two concrete subclasses are `Bitvec` and `Decvec`All. `Bitvec` stores a binary genome as an array of bits, and `DecvecAll` stores a decimal genome in an integer array. A subclass of `DecvecAll` is the class `Decvec` which differs from `DecvecAll` in the way that each decimal gene has to be unique. This is used for the m-GA algorithm family, which aims at finding the best m best features.

`FilterRanking` offers the implementation of the Fisher Criterion Score and the Relief-F algorithm. Both algorithms are put together in this class because they both compute a ranking of features according to some criterion. The ranking and the value of the criterion are store in the structure `Score`. Thus the methods `getAllReliefF` and `getAllFisherScore` return an array of `Score` structures.

The RFE algorithm is implemented in the `RecursiveFeatureElimination` class. Here the method `getAll` returns an integer list of the features in the order they were eliminated.

Finally the class `Algo` offers some general purpose things like the cross-validation algorithm or a method to read in files. All methods are made static.

Figure A.4 shows a class diagram of the graphical interface classes which are encapsulated in a package `featuresel`:

The class `Featuresel` contains the main method of the graphical interface. It first checks whether arguments are passed. If this is the case they are parsed and the native method `invoke_algorithm` is called. Otherwise an instance of class `GUI` is created. If the user clicks "File | New" an object of class `GUI_New` is created. If the user clicks "Run" the method `prep_algorithm` passes the settings the user has made in the dialog to the actual instance of `Featuresel`. Depending on whether the user has selected "show messages" or "plot statistics" a reference to an object of class `GUI_TextOutput` respectively `Plotter` or `null` is handed over. Then a new thread of class `Featuresel` is started. The `run` method of this thread simply calls the native method `invoke_algorithm` with the arguments formerly given by `prep_algorithm`. If the user clicks "Finish" in the output window, the native method `exit_algo` is called and the execution of the thread is finished.

The class `GUI_TextOutput` is simply an extension of class `java.awt.Frame`. Each instance contains a reference to a `TextArea` object. Whenever `printString` is invoked the passed string is added to this text area.

The class `Plotter` is also an extension of class `java.awt.Frame`. It contains two attributes of class `Plot`. This class is part of the public available package `ptolemy` which can be downloaded from `http://ptolemy.eecs.berkeley.edu/java/ptplot5.2/ptolemy/plot/doc/index.htm`. The `Plot` class handles plotting, rescaling, zooming and adding legends automatically.

A more detailed documentation of all classes can be found in the API on the program CD.

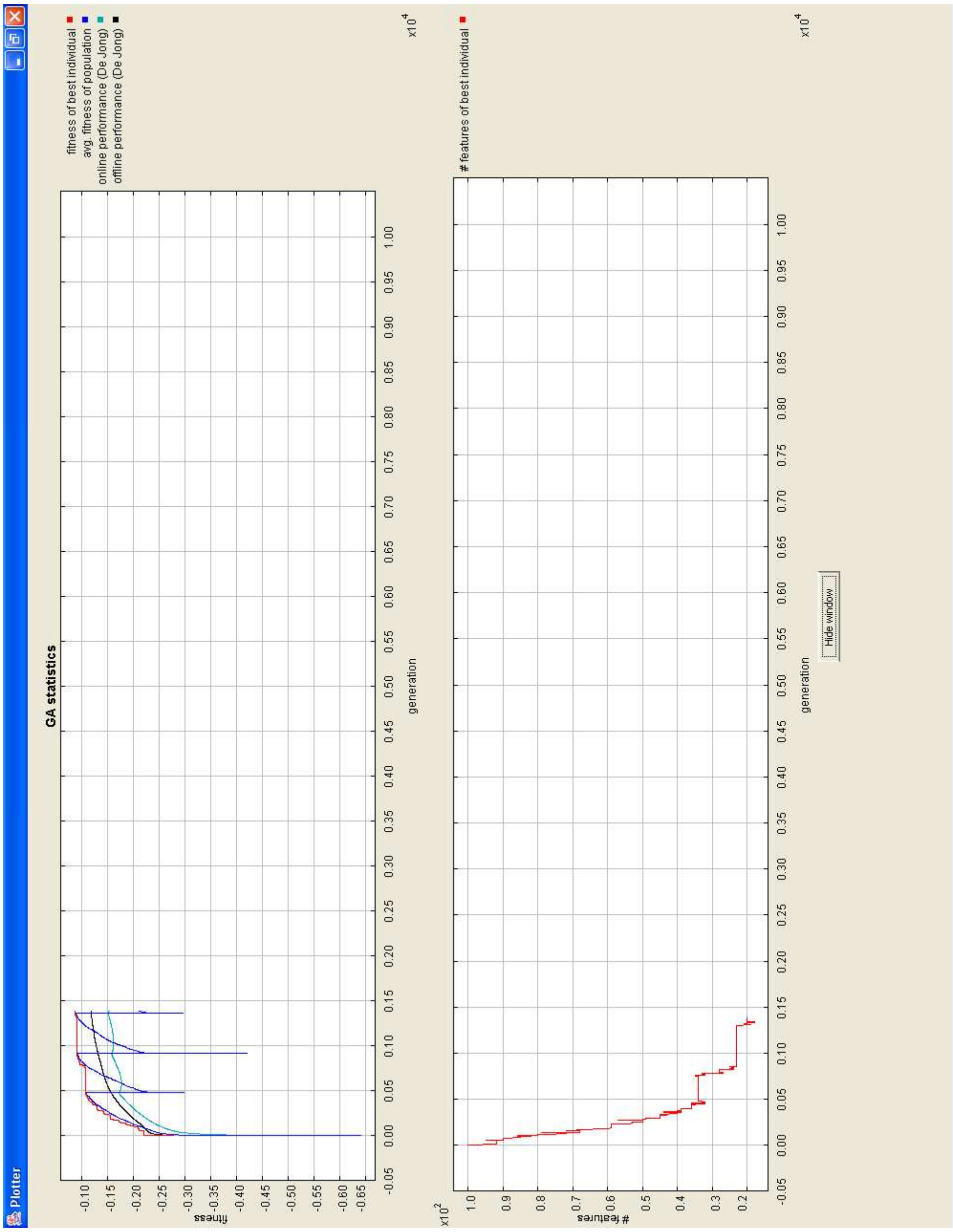

Figure A.2: Plotter window

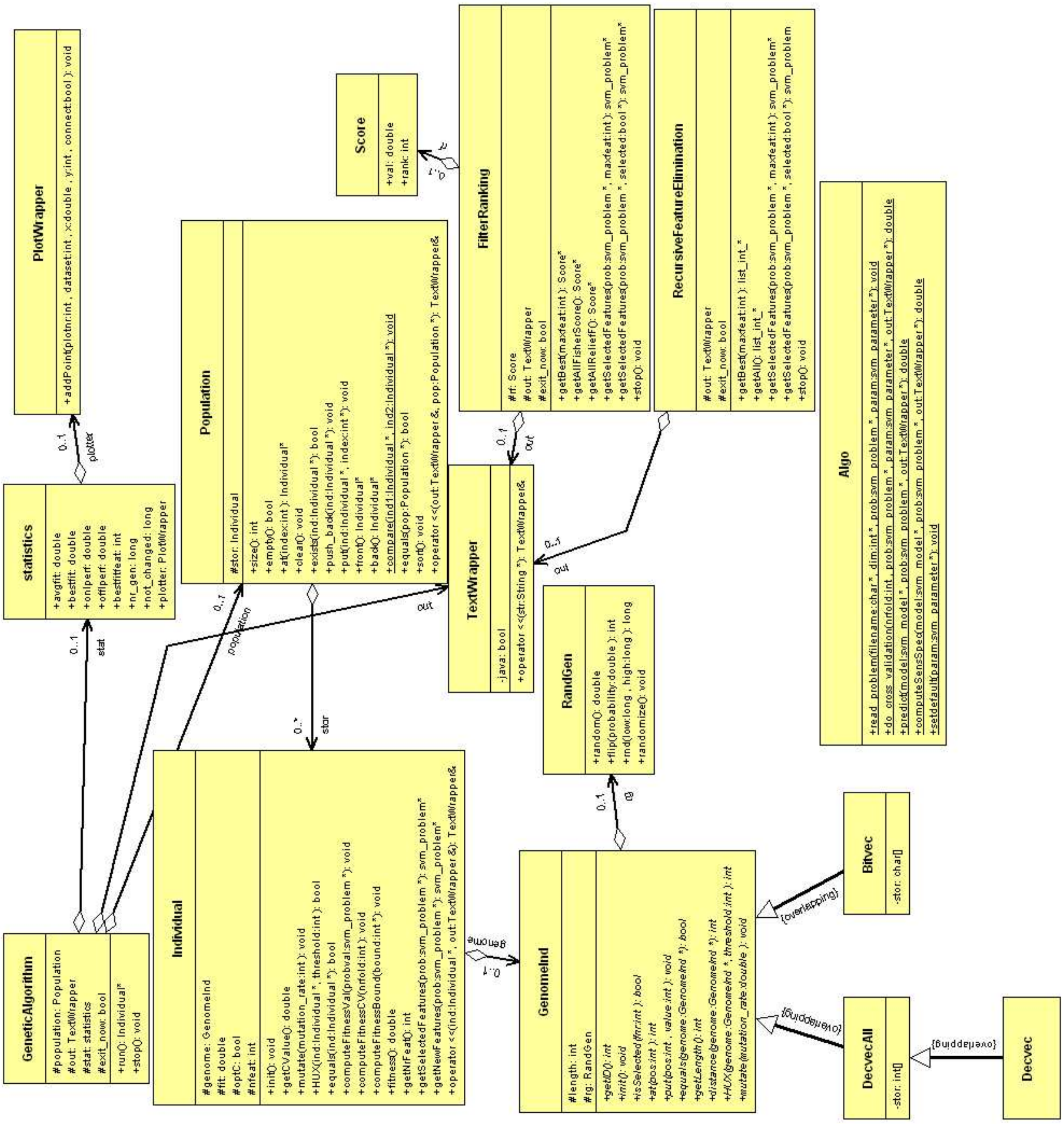

Figure A.3: Class diagram of main classes

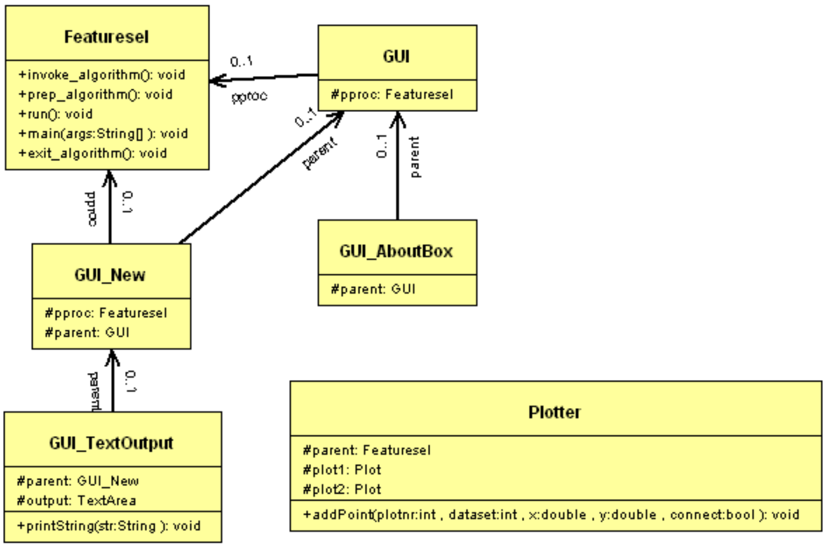

Figure A.4: Class diagram of graphical interface classes

List of Figures

List of Tables

Bibliography

[BosGuyVap92] Bernhard E. Boser, Isabelle M. Guyon, Vladimir N. Vapnik, A Training Algorithm for Optimal Margin Classifiers, In: D. Haussler, *Proceedings of the 5th Annual Workshop on Computational Learning Theory*, ACM Press, Pittsburgh, PA, 1992, pp. 144 - 152

[BriBroMar92] F. Brill, D. Brown, W. Martin, Fast genetic selection of features for neural network classifiers, In: *IEEE Transactions on Neural Networks*, 3(2): 324 - 328, 1992

[Duda73] Richard O. Duda, Peter E. Hart, Pattern classification and scene analysis, Wiley, 1973

[Eshelman91] L. Eshelman, The CHC Adaptive Search Algorithm, How to Have Safe Search When Engaging in Non-traditional Genetic Recombination, Morgan Kaufman, S. 265 - 283

[FerKadKit93] F. J. Ferri, V. Kadirkamanathan, J. Kittler, Feature Subset Search using Genetic Algorithms, In: *IEE/IEEE Workshop on Natural Algorithms in Signal Processing*, Essex, 1993

[Goldberg98] D. Goldberg, Genetic Algorithms in Search, Optimization in Machine Learning, Addison Wesley, Reading, 1998

[GuyWesBarVap02] I. Guyon, J. Weston, S. Barnhill, V. Vapnik, Gene Selection for Cancer Classification using Support Vector Machines, In: *Machine Learning*, Vol. 46, 2002, S. 389 - 422

[Holland75] J. H. Holland, Adaption in Natural and Artificial Systems, University of Michigan Press, Ann Arbor, 1975

[JaiMurFly99] Jain, A.K, M.N. Murty, P.J. Flynn, Data Clustering: A Review, In: *ACM Computing Surveys*, Vol. 31, Nr. 3, 1999, pp. 264-323

[Kohavi97] R. Kohavi, G. John, Wrappers for Feature Subset Selection, In: *Artificial Intelligence*, Vol. 97:12, 1997, pp. 273 - 324

[KwaChoCho01] N. Kwak, C. - H. Choi, J. Y. Choi, Feature Extraction using ICA, In: Dorffner et al. (editor), *Proc. Of ICANN 2001*, pp. 568 - 573, Berlin, 2001, LNCS 2130, Springer

[RicLan96] M. Richeldi, P. Lanzi, A Tool for Performing effective feature selection by investigating the deep structure of the data, In: *Proceedings of the International Conference on Tools with Artificial Intelligence*, pp. 102 - 105, 1996

[Rojas96] R. Rojas, Neural Networks — A Systematic Introduction, Springer, Berlin, 1996

[RusNor95]	S.J. Russel, P. Norvig, Artificial Intelligence: A Modern Approach, Prentice Hall, Englewood Cliffs, New Jersey 07632, 1995
[SchBurSmo99]	B. Schölkopf, C. J. C. Burges, A. J. Smola, Advances in Kernel Methods: Support Vector Learning, MIT Press, Cambridge, MA, 1999.
[Schoelkopf97]	B. Schölkopf: Support vector learning, PhD thesis, Universität Berlin, Oldenburg Verlag, 1997
[SchSmo02]	B. Schölkopf, A. J. Smola, Learning with Kernels, MIT Press, Cambridge, MA, 2002
[SmoSch98]	A. J. Smola, B. Schölkopf: A Tutorial on Support Vector Regression, NeuroCOLT2 NC2-TR-1998-030, 1998
[Ultsch02]	A. Ultsch, Eine Begründung der Pareto 80/20 Regel und Grenzwerte für die ABC-Analyse, In: *Bericht Nr. 30, Reihe Informatik*, Philipps-Universität Marburg
[UltschVL00]	A. Ultsch, Knowledge Discovery, Vorlesung an der Universität Marburg, Sommersemester 2000
[Vapnik95]	V. Vapnik, The Nature of Statistical Learning Theory, Springer, New York, NY, 1995
[WesMukChaPogVap01]	J. Weston, S. Mukherjee, O. Chapelle, M. Pontil, T. Poggio, V. Vapnik, Feature selection for SVMs, In: Sara A. Solla, Todd K. Leen, Klaus-Robert Müller (editors), *Advances in Neural Information Processing Systems 13*, MIT Press, Cambride, MA, 2001

[KonHon97] I. Kononenko, S. J. Hong, Attribute Selection for Modeling, In: *Future Generation Computer Systems*, ISSN 0167 - 739X, 13 (2 - 3), pp. 181 - 195, 1997

[Whitley93] D. Whitley, A Genetic Algorithm Tutorial, Department of Computer Science, Colorado State University, 1993

[KirRen92] K. Kira, L. Rendell, A practical approach to feature selection, In: *Proc. Int. Conf. on Machine Learning*, ICML 92, pp. 249 - 256, 1992

[Kononenko94] I. Kononenko, Estimating attributes: Analysis and extensions of RELIEF, In: *Proc. European Conf. on Machine Learning*, pp. 171 ¯ 182, 1994

[WesEliSchTip02] J. Weston, A. Elisseeff, B. Schölkopf, M. Tipping, Use of the zero-norm with linear models and kernel methods, In: *JMLR special Issue on Variable and Feature Selection*, 2002

[RayPunGooKuhJai00] M. Raymer, W. Punch, E. Goodman, L. Kuhn, A. Jain, Dimensionality Reduction Using Genetic Algorithms, In: *IEEE Transactions on Evolutionary computing*, 2000

[ChaVapBouMuk98] O. Chapelle, V. Vapnik, O. Bousqet, S. Mukherjee, Choosing Multiple Parameters for Support Vector Machines, In: *Proc. 13th International Conference on Machine Learning*, pp. 82 - 90, San Francisco, CA, 1998

[Vapnik98] V. Vapnik, Statistical Learning Theory, John Wiley and Sons, New York, 1998

[AloBarNotGisYbaMacLev99] U. Alon, N. Barkai, D. Notterman, K. Gish, S. Ybarra, D. Mack, A. Levine, Broad patterns of gene expression revealed by clustering analysis of tumor and normal colon cancer tissues probed by oligonucleotide arrays, In: *Cell Biology*, 96:6745-6750, 1999

[PavWes01] P. Pavlidis, J. Weston, J. Cai, W. Grundy, Gene functional classification from heteregoneous data, In: *Proceedings of the fifth International Conference on Computational Molecular Biology*, pp. 242 - 248, 2001

[DavRus94] S. Davies, S. Russel, NP-Completeness of Searches for Smallest Possible Feature Sets, In: *Proceedings of the 1994 AAAI Fall Symposion on Relevance*, pp. 37 - 39, AAAI Press, 1994

[Rechenberg73] I. Rechenberg, Evolutionsstrategie, Frommann-Holzboog, 1973

[Schwefel81] H. Schwefel, Numerical optimization of computer models (M. Finnis, trans.), Chichester: John Wiley, 1981 (Original work published 1977)

[JakHau99] T. S. Jaakkola, D. Haussler, Probalistic kernel regression models, In: *Proceedings of the 1999 Conference on AI and Statistics*, 1999

[BluLan97] A. L. Blum, P. Langley, Selection of Relevant Features and Examples in Machine Learning, In: *Artificial Intelligence*, Vol. 97:12, S. 245-271, 1997

[HyaRiv76] L. Hyafil, R. L. Rivest, Constructing optimal binary decision trees is NP-complete, In: *Information Processing Letters 5 (1)*, pp. 15 - 17, 1976

[Vapnik79] V. Vapnik, Estimation of Dependencies Based on Empirical Data [in Russian], Nauka, Moscow, 1979 (English translation: Springer Verlag, New York, 1982)

[VapChe74] V. Vapnik, A. Chervonenkis, Ordered risk minimization, In: *Automation and Remote Control*, 35:1226-1235, 1403-1412, 1974

[VapLer63] V. Vapnik, A. Lerner, Pattern recognition using generalized portrait method, In: *Automation and Remote Control*, 24:774-780, 1963

[VapChe64] V. Vapnik, A. Chervonenkis, A note on one class of perceptrons, In: *Automation and Remote Control*, 25, 1964

[VapChe79] V. Vapnik, A. Chervonenkis, Theory of Pattern Recognition [in Russian], Nauka, Moscow, 1974 (German translation: W. Wapnik, A. Tscherwonenkis, Theorie der Zeichenerkennung, Akademie-Verlag, Berlin, 1979)

[SchSunBurGirNiyPogVap97] B. Schölkopf, K. Sung, C. Burges, F. Girosi, P. Niyogi, T. Poggio, V. Vapnik, Comparing support vector machines with Gaussian kernels to radial basis function classifiers, In: *IEEE Transactions on Signal Processing*, 45:2758-2765, 1997

[UltschVLNN99] A. Ultsch, Neuronale Netze, Vorlesung an der Universität Marburg, Wintersemester 1999/2000

[CorVap95] C. Cortes, V. Vapnik, Support vector networks, In: *Machine Learning*, 20:273-297, 1995

[PogGir90] T. Poggio, F. Girosi, Networks for approximation and learning, In: *Proc. IEEE*, 78:1481-1497, 1990

[WesWat99] J. Weston, C. Watkins, Multi-class support vector machines, In: M. Verleysen (editor), *Proceedings ESANN*, Brussles, 1999

[DeJong75] K. DeJong, An Analysis of the Behavious of a Class of Genetic Adaptive Systems, PhD Dissertion, Department of Computer and Communication Sciences, University of Michigan, Ann Arbor, 1975

[CrisSha00] N. Cristianini, J. Shawe-Taylor, An Introduction to Support Vector Machines, Cambridge University Press, 2000

[Burges98] C. Burges, A Tutorial on Support Vector Machines for Pattern Recognition, In: *Data Mining and Knowledge Discovery,* 2:121-167, 1998

[JohKohPfl94] G. John, R. Kohavi, K. Pfleger, Irrelevant features and the subset selection problem, In: *Machine Learning: Proceedings of the Eleventh International Conference*, Morgan Kaufmann, pp. 121 - 129, 1994

[CriShaEliKan02] N. Cristianini, J. Shawe-Taylor, A. Elisseff, J. Kandola, On Kernel-Target Alignment, In: T. G. Dietterich, S. Becker, Z. Ghahramani (editors), *Advances in Neural Information Processing Systems 14,* MIT Press, Cambridge, MA, 2002

[LesEskNob02] C. Leslie, E. Eskin, W. Noble, The spectrum kernel: A string kernel for SVM protein classification, In: *Proc. Pacific Symposium on Biocomputing*, pp. 564 - 575, 2002

[NadBen00] C. Nadeau, Y. Bengio, Inference for the Generalization Error, In: Sara A. Solla, Todd K. Leen, Klaus-Robert Müller (editors), *Advances in Neural Information Processing Systems 12*, MIT Press, Cambridge, MA, 2000

[SchBurVap95] B. Schölkopf, C. Burges, V. Vapnik, Extracting support data for a given task, In: *First International Conference for Knowledge Discovery and Data Mining*, U. N. Fayyad, R. Uthurusamy (editors), AAAI Press, Menlo Park, 1995

[ChaVap00] O. Chapelle, V. Vapnik, Model selection for support vector machines, In: Sara A. Solla, Todd K. Leen, Klaus-Robert Müller (editors), *Advances in Neural Information Processing Systems 12*, MIT Press, Cambridge, MA, 2000

[VapCha00] V. Vapnik, O. Chapelle, Bounds on error expectation for support vector machines, In: *Neural Computation*, 12 (9), 2000