Tel-Aviv University

Raymond and Beverly Sackler

Faculty of Exact Sciences

School of Computer Science

Feature selection methods for classification of gene expression profiles

Thesis submitted in partial fulfillment of the requirements for

M.Sc. degree in the School of Computer Science, Tel-Aviv University

By

Michael Gutkin

The research work for this thesis has been carried out at

Tel-Aviv University under the supervision of

Prof. Ron Shamir and **Prof. Gideon Dror**

MARCH 2008

Acknowledgments

I deeply thank Prof. Ron Shamir for introducing me to the wonderful world of Computational Biology and for supervising this research. His consistent support, advice, thoroughness and patience have made this work possible. I would also like to sincerely thank Prof. Gideon Dror for guiding me through the exciting world of Computational Learning and co-advising to this research. His help had made a crucial contribution to this work.

I want to thank my parents and my brother for giving me the best support a family can give and for cleverly putting things in perspective. I also wish to thank my girlfriend, Mor, for giving me the support I needed, and for accompanying me along the way.

I want to thank all my lab mates: Igor Ulitsky, Yonit Halperin, Ofir Davidovich, Daniela Raijman, Chaim Linhart, Adi Maron-Katz, Irit Gat-Viks, Michal Ozery-Flato, Michal Ziv-Ukelson, Rani Elkon, Seagull Shavit, Ofer Lavi, Guy Karlebach, Firas Swidan, Panos Giannopoulos and Falk Hueffner - for the fruitful conversations, advices, and especially for the laughs, arguments, and the great atmosphere in the lab.

Last, but not least, I would like to thank the GENEPARK project for showing me the missionary aspect of the field, and a difference that can be made.

This research was supported in part by the GENEPARK project which is funded by the European Commission within its FP6 Programme (contract number EU-LSHB-CT-2006-037544).

Abstract

A key challenge in biomedical studies in recent years is the classification of samples into categories such as cases and control (individuals who carry some illness and others who do not). This is done by first learning how to classify, based on a training set containing labeled samples from the two populations, and then predicting the label of new samples. Each sample consists of gene expression measurements.

An important sub-problem in such studies is that of feature selection. Microarrays can measure the levels of thousands of genes per sample. Using these data, the number of features (gene expression levels) far exceeds the number of samples. Standard classifiers do not work well in such situation. Selecting only the features that are most relevant for the discrimination between the two categories helps in constructing better classifiers, both in terms of accuracy and in terms of efficiency.

In this work we developed a novel family of methods for multivariate feature selection, based on the Partial Least Squares algorithm. We performed a systematic comparison of the family variants as well as common feature selection techniques. The comparisons were done across a large number of real case-control datasets and using several classifiers. We demonstrate the advantage of the new method and provide insights on the preferable combinations of classifier and feature selection technique.

Contents

1 Introduction and summary

Classification of samples, given as gene expression profiles, has become in the last few years an active topic in biomedical research. Such classification aims to distinguish between two types of samples. Usually, these two types are *positive*, or *case* samples (i.e., taken from individuals that carry some illness) and *negative*, or *control* samples (i.e., healthy individuals). One first obtains a collection of samples with known type labels and uses it to build a classifier, which can later be used to classify unlabeled samples.

The use of gene expression microarrays allows simultaneous measuring of tens of thousands of gene expression levels per sample. This high-throughput ability to measure gene expression generates data with number of features (genes) far exceeding the number of samples. The high dimension of the data poses a real problem for standard classifiers. By selection only a subset of the features (a process called dimension reduction) several goals are obtained:

- Improved performance of classification algorithms by removing irrelevant features (noise).
- Improved generalization ability of the classifier by avoiding over-fitting (learning a classifier that is too tailored to the training samples, but performs poorly on other samples).
- By using fewer features, classifiers can be more efficient in time and space.
- It allows us to better understand the domain.
- It is cheaper to collect and store data based on a reduced feature set.

Many feature selection techniques have been proposed. One of the most common techniques in use are *filters* [1], which are univariate methods for selecting the most relevant features one by one and filtering out the rest. Such techniques easily scale to very high-dimensional datasets, they are computationally simple and fast, and they are independent of the classification algorithm. As a result, feature selection needs to be performed only once, and then different classifiers can be evaluated [1]. However, when using filters, each feature is considered separately, thereby ignoring feature dependencies. Multivariate techniques may overcome this shortcoming.

In this study, we developed a novel family of feature selection techniques based on the Partial Least Squares (PLS) algorithm [2-4] , which we call SlimPLS. SlimPLS is a multivariate feature selection method, thus incorporating feature dependencies. In order to compare the performance of the SlimPLS based methods we used five classifiers – linear Support Vector Machine (SVM), radial SVM, Random Forest, K-nearest-neighbors (KNN), and Naïve Bayes. 19 different case-control expression profiles datasets were collected and used for training and testing. Our results show a significant gain in performance for some variants of the SlimPLS compared to filters techniques.

This thesis is organized as follows: in Chapter 2 we present the necessary background for this work, and review some of the relevant literature. In Chapter 3 we present the SlimPLS method and its variants. In Chapter 4 we present the datasets we collected and the different criteria we used for the comparisons, and the results of the different comparisons using several classifiers are presented. In Chapter 5 we discuss the results and their implications and present some possible future directions. Some more

evaluations and comparisons of the different feature selection techniques and classifiers are included in the appendix.

2 Background

2.1 What is Classification

2.1.1 Introduction

Supervised classification takes a set of data samples, each consisting of measurements on a set of variables, with associated labels called the class types, and uses them to learn a particular model. Using that model, the labels of new samples can be estimated. **Figure 2-1** gives an abstract illustration of the idea.

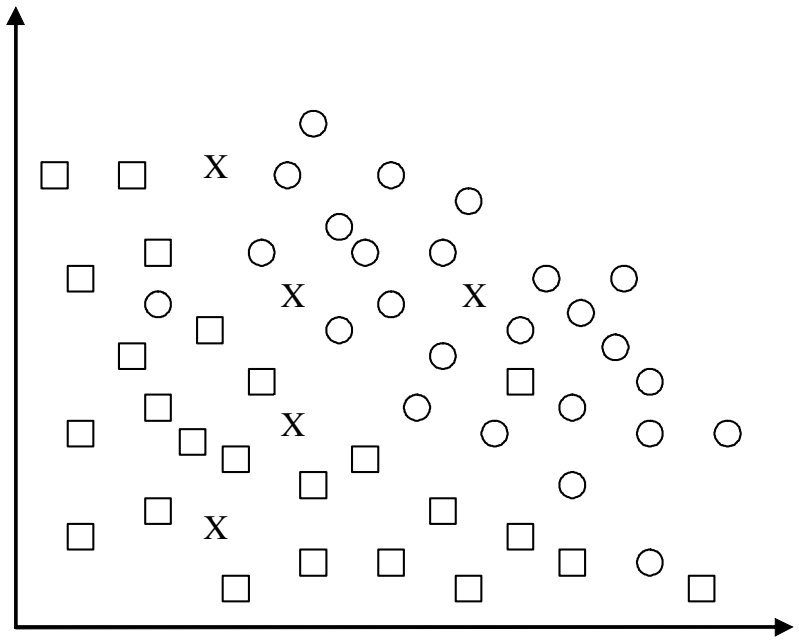

Figure 2-1. Two-dimensional points belonging to two different classes (circles and squares) are shown in the figure. A classifier will learn a model using these points and then use the model to accurately classify the new samples, marked by X.

Classification is used in various fields, e.g..

a) Biology. In recent years the study of gene expression microarrays became very popular. Each microarray is a *sample*, which gives the expression level of many genes

in an individual, and samples from different classes (e.g. sick and healthy individuals) are given. Gene expression data were successfully used to classify patients into different clinical groups, thus identifying new disease groups and the relevant genes for this clinical phenomenon [5].

b) Optical Character Recognition (OCR) uses classification to translate images of handwritten, typewritten or printed text (usually captured by a scanner) into machine-editable text [6].

c) Document classification. The task is to assign an electronic document to one or more categories, based on its contents. This is usually done using supervised classification techniques, e.g. [7].

The classification problem can be formally stated as follows: A *sample* is a pair (x_i, y_i), where x_i is a p-dimensional data vector of measurements. Usually, $x_i \in R^p$. y_i is the *label* of the sample, indicating the class it belongs to. Formally, it is a categorical variable taking values from a finite set of labels $\Omega=\{\omega_1,...,\omega_c\}$. The input to the classifier is a set of measurement vectors along with their known classes. This set, called the *training set*, is used to build the classifier. Once the classifier is built, given a new test example, x, its class can be predicted by the classifier.

The high dimensional nature of many classification tasks, i.e., the very large number of available features, may pose a real problem for classifiers, especially when we have relatively few samples (see Section 1). Therefore, in many cases we will need to select only a small subset of the available features that will contribute most to the classification. This task is called *feature selection*.

In the next sections we will review several classifiers and feature selection methods. In this work we concentrate on the two-class classification problem.

2.1.2 Difficulties

Our goal is to design a classifier that is as accurate as possible in classifying new test samples. This challenge is difficult for several reasons.

The first problem is that we are often given a relatively small training set. Thus classifiers have to infer a general behavior from relatively few samples. We assume that the training set faithfully represents the test set, or the 'real world'. However, when the sample is small, it is less likely to faithfully represent the real world, and more likely to be biased due to noise, population differences, etc.

Another problem is the complexity of the model and its generalizing capabilities. If the classifier is too simple it may fail to capture the underlying structure of the data. However, if the classifier is too complex and there are too many free parameters, it may incorporate noise in the model, leading to *over-fitting*, where the learned model highly fits the training set, but performs poorly on test samples. Thus, achieving optimal performance on the training set (in terms of minimizing some error criterion) is not a requirement. It may be possible for a classifier to achieve 100% classification accuracy on the training set but the *generalization* performance – the expected performance on a test data (or equivalently, the expected performance on the distribution from which the training set was sampled) – is poorer than could be achieved by different methods.

Another problem is the meaning of "optimal". There are several ways of measuring classifier performance. For binary classification problems the most common one is the error rate, but even this is not a simple task, as the error rate needs to be estimated and usually can not be directly calculated.

2.1.3 Error estimation

As was mentioned in the former section we seek to minimize the generalization error - the expected error (performance) on test data or 'real-world" data. In the next sections we will describe different classifier mechanisms and see how they 'infer', i.e., how they use the training data in order to classify new test samples as accurately as possible. But first we need a way to measure the quality of such an inference.

There are many methods for estimating generalization error, e.g. *test-set* method, *cross validation*, *bootstrap*, *jackknife* etc [11]. The focus in this work is on cross validation techniques, in particular the leave-one-out-cross-validation method.

2.1.3.1 Cross Validation

Cross-validation calculates the error by repeatedly partitioning the given training set into two disjoint subsets: the training subset and the test subset. When a sample belongs to the test subset, its label is hidden from the classifier built based on the training subset only, and the prediction of its class can be compared to its true class. The process is repeated with several partitions and gives an estimate of the performance of the classifier.

2.1.3.1.1 K-fold cross-validation

The *k*-fold cross-validation partitions the given training set into *k* subsets (preferably of equal size). Then, training is done on *k-1* subsets and testing is done on the remaining subset. This process is repeated as each subset is taken to be a test set in turn.

2.1.3.1.2 Leave-one-out cross-validation

In this method we use k-fold cross-validation with *k=n*, the number of samples in the training set. In each 'fold' we use *n-1* samples as training set and test the classifier on the remaining sample. This procedure is repeated for all samples. The estimated error is simply the fraction of wrongly classified samples.

This method is computationally expensive as it requires the construction of *n* different classifiers. However, it uses almost all the samples in each training subset, thus it is more suitable for smaller datasets. This method is used in our work.

2.2 Linear Discrimination

2.2.1 Introduction

Linear Discrimination algorithms are classifiers that assign to each sample a real value which is a linear combination of its feature values. A test sample is classified according to its real value. We first make the assumption that the decision boundaries are linear – i.e. samples from different classes can be separated using a linear function. Linear discrimination can be used in binary classification and in multi-class classification.

The problem of a binary linear discrimination can be formulated as follows. Suppose we have a set of training samples $\{(x_i, y_i), i = 1,...,n\}$ where $y_i \in \{-1,+1\}$. We seek a linear

function $g(x)$, consisting of weight vector w and a threshold w_0, such that its sign will predict the label y_i:

$$sign(g(x_i)) \geq 0 \rightarrow y_i = +1$$
$$sign(g(x_i)) < 0 \rightarrow y_i = -1$$

for each sample x_i.

A sample x_i will be classified correctly if $g(x_i) \cdot y_i > 0$. Ideally, we would like to find such $g(x)$ that makes $g(x) \cdot y$ positive for as many samples in the training set as possible. This criterion minimizes the misclassification error on the training set. If indeed $g(x) \cdot y > 0$ for all samples in the training set, the data are said to be *linearly separable*.

In non trivial problems it is not possible to find a perfect linear separation of the data. Moreover, insisting on a perfect linear separation when the data are noisy can lead to over-fitting. In some situations it is better to let some training samples be misclassified in order to handle noise better.

2.2.2 Support Vector Machines

2.2.2.1 Introduction

Support vector machines (SVMs) [8-10] are very popular linear discrimination methods that build on a simple yet powerful idea: Samples are mapped from the original input space into a high-dimensional feature space, in which a 'best' separating hyperplane can be found. A separating hyperplane *H* is best if its margin is largest. The *margin* is defined as the largest distance between two hyperplanes parallel to *H* on both sides that do not contain sample points between them (we will see later a refinement to this definition). It follows from the *risk minimization* principle (an assessment of the expected *loss function*,

i.e., the mis-classification of samples [11]) that the larger the margin, the better the generalization error of the classifier.

To demonstrate this idea let us consider **Figure 2-2**. We can see that for the same training set, different separating hyperplanes can be found. The separating hyperplane that leaves the closest points from different classes at maximum distance from it is preferred, as the two groups of samples are separated from each other by a largest margin, and thus least sensitive to minor errors in the hyperplane's direction.

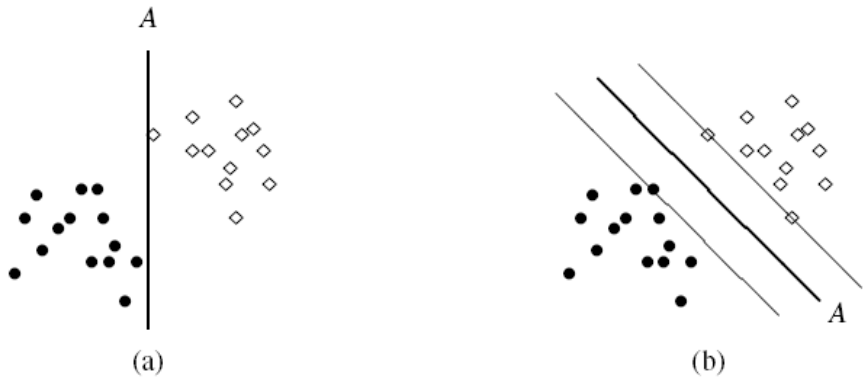

Figure 2-2. Separating hyperplanes and Margin. Two different possible separating hyperplanes are shown (thick lines). (a) A separating hyperplane parallel to the y-axis. (b) A separating hyperplane that leaves the closest points at maximum distance from it (the thin lines on the right identify the margin). This figure is taken from [11].

2.2.2.2 Linearly separable data

As mentioned earlier all training samples are correctly classified if

$$g(x_i) \cdot y_i = (w^T x_i + w_0) \cdot y_i > 0$$

for each training sample x_i. We would now like to take the margin into consideration in the above equation. Thus, changing it to

$$(w^T x_i + w_0) \cdot y_i \geq b$$

yields a solution for which all training samples x_i are at distance greater than $\frac{b}{|w|}$ from the separating hyperplane. We can scale b, w_0 and w while still having the distance unaltered. Therefore, without loss of generality $b = 1$ is taken. Setting b to the value of 1 defines the canonical hyperplanes as follows.

$$H_1 : w^T x + w_0 = +1$$
$$H_2 : w^T x + w_0 = -1$$

In addition, all training samples x_i satisfy:

$$w^T x_i + w_0 \geq 1 \text{ for } y_i = +1$$

$$w^T x_i + w_0 \leq -1 \text{ for } y_i = -1$$

The separating plane is defines by $g(x) = w^T x + w_0 = 0$, and the distance between each of the canonical hyperplanes and the separating hyperplane is $\frac{1}{|w|}$. This quantity is termed the *margin*. See **Figure 2-3**.

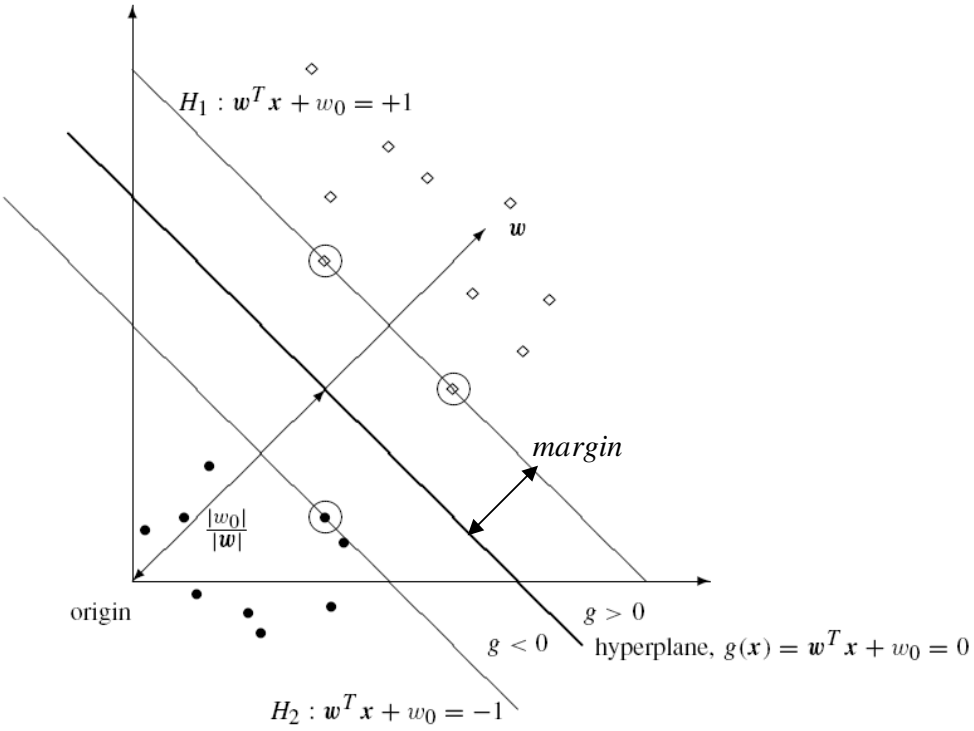

Figure 2-3. The geometry of the margin. H_1 and H_2 are the canonical hyperplanes. The margin is the distance between the separating hyperplane ($g(x) = 0$) and a hyperplane through the closest points (marked by a ring around the data points). These are termed the *support vectors*. This figure is taken from [11].

Now, we can formulate the learning problem of SVM as follows.

$$\max(\frac{1}{|w|}) \text{ s.t. } (w^T x_i + w_0) \cdot y_i \geq 1 \quad i = 1,..,n$$

where (x_i, y_i) is the set of training samples with their labels. It also can be written as follows.

$$\min(\frac{1}{2} w^T w) \text{ s.t. } (w^T x_i + w_0) \cdot y_i \geq 1 \quad i = 1,..,n$$

This formulation enables us to use the Lagrange formalism: The non-negativity constraints are multiplied by positive Lagrange multipliers α_i and subtracted from the

objective function. This leads us to the *primal form* of the objective function L_p, which is given as follows.

$$L_p = \frac{1}{2} w^T w - \sum_{i=1}^{n} \alpha_i ((w^T x_i + w_0) \cdot y_i - 1)$$

where $\{\alpha_i : i = 1,...,n; \alpha_i \geq 0\}$ are the Lagrange multipliers. Solving the minimization problem is equivalent to finding the values w, w_0, and $\alpha_i \geq 0$ that minimize L_p. To do so, we first differentiate L_p with respect to w and w_0. Then, by equating the derivates to zero we get

$$w = \sum_{i=1}^{n} \alpha_i x_i y_i \text{ (when differentiating with respect to } w)$$

$$\sum_{i=1}^{n} \alpha_i y_i = 0 \text{ (when differentiating with respect to } w_0)$$

Taking these two equalities and substituting into L_p yields the *dual form* of the Lagrangian. We want to maximize

$$L_d = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j x_i^T x_j$$

subject to

$$\alpha_i \geq 0 \quad \sum_{i=1}^{n} \alpha_i y_i = 0$$

This optimization formulation is expressed using inner product of the training samples x_i, and the number of parameters is n – the numbers of training samples. The solution for that problem is achieved by convex quadratic programming. Finding of the α-s that maximize L_d enables the computation of w and w_0.

After finding w and w_0, classification of a query pattern x_q simply requires finding the sign of $g(x_q) = w^T x_q + w_0$.

2.2.2.3 Linearly non-separable data

In the previous section the SVM learning process was introduced as an optimization problem, under the assumption that the data are linearly separable. However, in many practical problems there will be no linear boundary separating the classes. Hence, looking for a hyperplane in the former manner will yield no results: The optimization problem will be infeasible. Therefore, a relaxation of the constraints is needed. This is done by introducing new *slack variables* $\{\xi_i : i = 1,...,n; \xi_i \geq 0\}$ into the original constraints:

$$(w^T x_i + w_0) \cdot y_i \geq 1 - \xi_i$$

This way, for a training point to be misclassified by the hyperplane, we must have $\xi_i > 1$. Notice that this also allows a point to be inside the 'sterile' area of the margin, but to still be correctly classified (for $0 < \xi_i < 1$).

The next step will be incorporating the additional cost due to the non-separability into the objective function, using some kind of penalty:

$$\frac{1}{2} w^T w + C \sum_{i=1}^{n} \xi_i$$

and the minimization problem will be given as

$$\min(\frac{1}{2} w^T w + C \sum_{i=1}^{n} \xi_i) \text{ s.t. } (w^T x_i + w_0) \cdot y_i \geq 1 - \xi_i$$

The parameter C (called the *regularization parameter*) controls the penalty for 'outliers' and 'softer' margin. Optimal values of C are usually found by using the leave-one-out procedure on the training samples, and finding the value that yields the lowest error.

The primal and dual forms of the Lagrangian are built in a similar way as in the previous section. The dual form will be given as

$$L_d = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j x_i^T x_j$$

subject to

$$0 \le \alpha_i \le C \quad \sum_{i=1}^{n} \alpha_i y_i = 0$$

Therefore, the only change to the maximization problem is the upper bound of the α_i.

2.2.3 Kernels and SVM

Even when no separation is possible in the original space, samples can be mapped into high-dimensional feature space, where a separating hyperplane can be found. This is the principle behind many methods of classification: transform the input features nonlinearly to a high dimensional space in which linear methods may be applied. This space is called the *feature space*.

Suppose that we transform each sample x_i into a point $\phi(x_i)$ in the new feature space [12, 13]. In that space, we again look for the linear discriminating function

$$g(x) = w^T \phi(x) + w_0$$

and the dual form of the Lagrangian becomes

$$L_d = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j \phi(x)_i^T \phi(x_j)$$

where, as previously, $y_i = \pm 1$, *i=1,…,n*, are class labels values and $\{\alpha_i\}$ are the Lagrange multipliers satisfying

$$0 \le \alpha_i \le C \quad \sum_{i=1}^{n} \alpha_i y_i = 0$$

for a given regularization parameter *C*.

Notice that the only effect of the non-linear transformation on the problem is using the transformed vectors $\phi(x_i)$ instead of x_i, and more precisely, L_d relies only on calculating the dot product in the feature space instead of the input space.

Suppose there exists a function $k(x_i, x_j)$ (a *kernel* function) satisfying

$$k(x_i, x_j) = \phi(x)_i^T \phi(x_j)$$

then we can avoid computing the transformation $\phi(x)$ explicitly altogether and replace the dot product with $k(x_i, x_j)$. In other words, we use a function that calculates the dot product of two vectors in the feature space, where the two vectors are given in the input space. The advantage of using such a kernel function is obvious - we do not need to specify or compute ϕ explicitly.

There are many types of kernels that can be used in a SVM. Acceptable kernels must be expressible as an inner product in some feature space. The methods of finding such kernels are beyond of the scope of this work. Two common kernels are the Polynomial kernel, $(1 + x_i^T x_j)^d$, and the Gaussian kernel, $\exp(-\left|x_i - x_j\right|^2 / \sigma^2)$.

Notice that when using the linear kernel (i.e., the simple dot product in the input space) one must provide only one parameter to the SVM algorithm – the 'regularization' factor. However, when using non linear kernels, more parameters must be provided, which may result in over-fitting.

2.3 Random Forests

2.3.1 Introduction

A Random Forest [14] is a classifier that uses a collection of decision trees, each of which is a simple classifier learnt on a random sample from the data. The classification of a query example is done by majority voting of the decision trees. We first describe decision trees and then the Random Forest method.

2.3.2 Decision trees

Decision trees learning is a method for inferring a discrete-valued target function of the samples (in our case – the sample's class). The model is represented by a decision tree. Assume temporarily, for simplicity, that the feature values are discrete. Each inner node in the tree specifies a test of some features of the sample. Each branch from that node corresponds to a possible range of values for these features. Each leaf corresponds to a class label. Samples are classified by going down the tree from the root to some leaf node, according to the branch conditions. Each path from the root to a particular leaf corresponds to conjunction of feature values, thus the tree itself constitutes a disjunction of these conjunctions.

2.3.2.1 Basic decision tree construction algorithm

There are many algorithms for growing a decision tree. Most of them have a core mechanism that employs a top-down, greedy construction of the decision tree.

The ID3 algorithm [15] is a good example of decision tree construction using a top-down approach. It begins by determining which feature should be tested at the root of the tree. This is done by evaluating each feature using a statistical test to examine how well it alone classifies the training samples. The best feature is selected to be tested at the root node of the tree.

Then, a branch is made for each possible value of this feature (or, as we will see later – for some possible intervals for continuous values), and the training samples are sorted accordingly. The entire process is then repeated using the training samples associated with each child node. Only those training samples that have a value that matches the particular branch are taken into account when finding a candidate test-feature for the child node. When all features have been examined, a child leaf node is created with a class label equal to the majority label of all samples associated with the path to this leaf (or equal to one of the most common labels, randomly selected, in case of a tie). Note that this method performs a greedy search for the decision tree, and it never backtracks to reconsider earlier choices.

2.3.2.2 Choosing the best feature

As stated, the best feature under some criterion is chosen as the test at the root node, and later other features are chosen in the same way as roots of subtrees. Several optional criteria can be used. The ID3 algorithm uses the *information gain* measure, which

computes how well a given feature separates the training samples according to their class labels. To define it, we first introduce the *entropy* measure from information theory

$$Entropy(S) \equiv \sum_{i=1}^{c} -p_i \log_2 p_i$$

where c is the number of different classes (labels), and p_i is the proportion of S (the group of samples) belonging to class i.

Notice that the entropy is 0 if all members of S belong to the same class. If all the classes contain an equal number of samples ($p_i = \frac{1}{c}$ for all i) then the entropy of S is equal to $\log_2 c$, which equals the minimum number of bits needed to encode the classification of an arbitrary sample in S, when c is a power of 2. In the specific case where $c=2$, if both classes have the same number of samples then the entropy of S is equal to 1. This way, entropy gives us a measure of the impurity of the sample group.

Now we can define the *information gain* measure. It is simply the expected reduction in entropy caused by partitioning the samples according to a particular feature. The information gain *Gain(S, F)* of a feature F, given a collection of samples S, is defined as

$$Gain(S,F) \equiv Entropy(S) - \sum_{v \in Values(F)} \frac{|S_v|}{|S|} Entropy(S_v)$$

where *Values(F)* is the set of all possible values for feature F, and S_v is the subset of S consisting of samples for which feature F has the value v. Hence, *Gain(S, F)* is the information provided (the reduction in entropy) about the target function value (the class label), given the values of a particular feature F.

2.3.2.3 Over-fitting

The ID3 algorithm aims to construct a tree that perfectly classifies the training samples. This strategy can easily lead to over-fitting, especially if the training set is small, where the tree structure can be highly sensitive to small changes in the data, as can be seen in **Figure 2-4**.

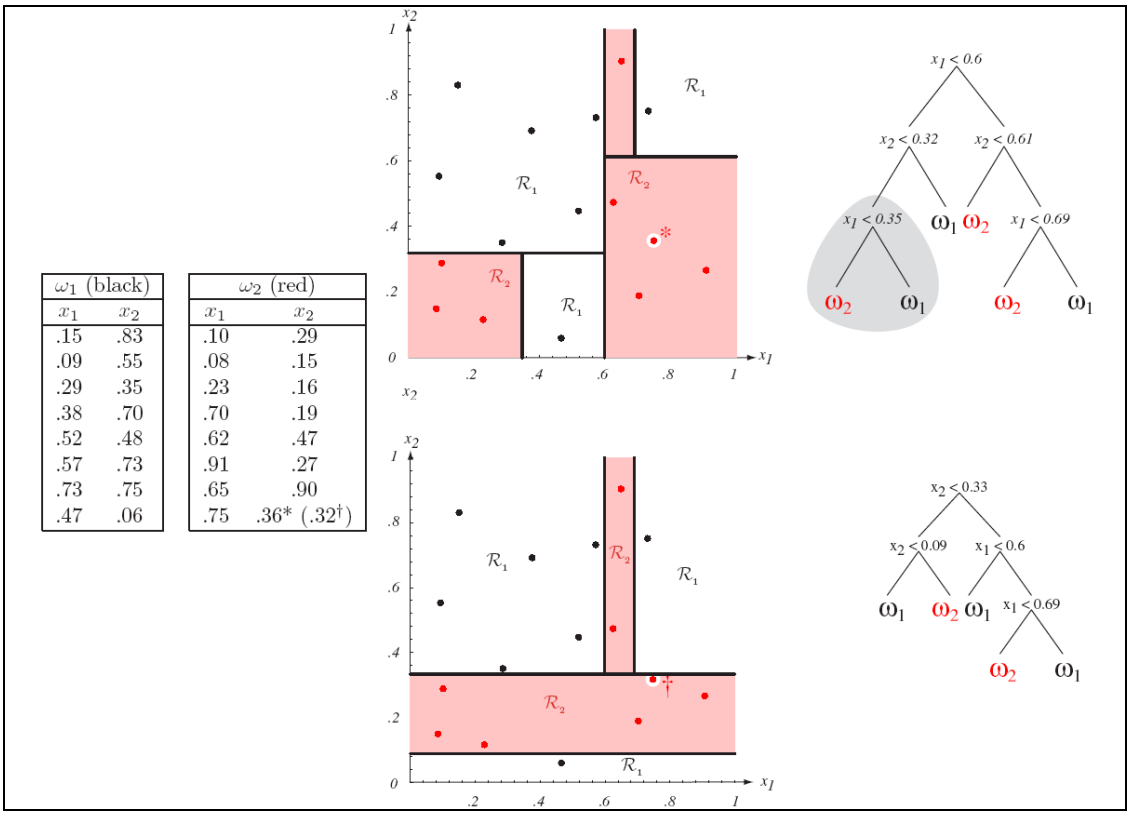

Figure 2-4. Training data and associated (unpruned) trees. Consider the following $n = 16$ points in two dimensions for training a binary tree. If the single training point marked * were instead slightly lower (marked †), the resulting tree and decision regions would differ significantly. This figure is taken from [68].

Over-fitting is a significant practical difficulty for decision tree learning. There are several approaches to avoid over-fitting, and they can be grouped into two classes. The first class of approaches stop growing the tree before it reaches its full potential size. The

second class of approaches fully grow the trees (perhaps causing over-fit of the data), and then prune it.

Although less direct, the pruning techniques have been found to be more useful [16], and a common implementation is to use a separate set of samples (*validation set*) to evaluate the benefit of pruning nodes from the tree.

For example, we can consider each of the decision nodes as a candidate for pruning. Pruning it means removing the sub-tree rooted at that node, thus making it a leaf node. The assigned class-label of this node is the majority class in the training samples associated with that node. In this approach, nodes are removed when the resulting pruned tree performs no worse than the original tree over the validation set. This approach is called *reduced-error pruning* [17].

2.3.2.4 Continuous-valued features

Until now, we assumed that features had only discrete values. In that case all tests in each node were of the form "does feature F equal to v?" When using continuous–valued features we need to re-define these tests. For example, the algorithm can dynamically create a new boolean feature F_c that is true if $F > c$ and false otherwise. The problem is to find this threshold. A possible way is to sort all samples according to feature *F*, and identify a threshold c that best partitions the samples according to their different class labels. The best partition can be chosen, e.g., by the information gain criterion.

2.3.3 Random Forest

A random forest is an ensemble of many decision trees that were grown using a random process. To classify a new sample, each of the trees assigns a class to it and the majority class is selected [14].

2.3.3.1 Growing a single tree

Given N training samples, each having M features, each tree is grown as follows: First, N instances are sampled at random (with replacement) from the training set. This sample is the training set of the tree. Then, at each node, $m << M$ of the features are selected at random. The best split on these m features is used to branch at the node. The same value of m is used for all trees in the forest. Each tree is grown to the largest extent possible, without pruning.

2.3.3.2 Error rate

Breiman has shown in [14] that the error rate in classification is related to m. The optimal value of m can be estimated as follows: Recall that in the process of growing a single tree N samples are selected with replacement at random. This means that on average about a third of the samples were not used for training that tree. Moreover, it means that any sample i was not used for training in about a third of the trees in the forest and therefore can be used as a test sample for them. We can classify sample i using only these trees and thus get an error value for that sample. The average error value across all samples is the called *out-of-bag error rate*.

2.3.3.3 Forest size

Although each individual tree grown by the Random Forest algorithm can severely over-fit the data, the whole Random Forest is very resistant to over-fitting, owing to the effect of averaging over many different trees. In this respect, the larger the number of trees - the better. Furthermore, the generalization error converges almost surely to a limit value [14]. Therefore, one can run as many trees as one desires.

2.4 Instance-based learning

2.4.1 Introduction

Most learning methods construct a general, explicit description or model of the target function as training samples are provided. Instance-based learning methods simply store the training samples. These samples might be pre-processed but no model is created. Generalizing beyond these samples is postponed until a new instance is to be classified. When a new query sample is introduced, its relationship to the stored training samples is examined in order to assign a target function value for the new instance. More precisely, a set of similar training instances is retrieved and used to classify the new query instance. Because of this delayed processing, instance-based methods are sometimes referred to as "lazy" learning methods.

The "laziness" has some advantages [16]. They can construct a different approximation to the target function for each distinct query instance. Moreover, many techniques construct only a local approximation to the target function, which applies in the neighborhood of the new query instance, in contrast to constructing a single

approximation designed to perform well over the entire instance space. This has significant advantages when the classifying function is very complex, but can still be described by a collection of less complex local approximations.

Instance-based approaches have two main disadvantages. First, nearly all computation takes place at classification time, and thus, the computational cost of classifying new instances can be high. Therefore, techniques for efficiently indexing the training examples are an important practical issue in reducing the computation required at prediction time. The second disadvantage is that instance-based methods typically consider all features of the samples when attempting to retrieve similar training examples from memory. If the target concept depends only on a few of them, then the instances that are truly most "similar" may appear as relatively distant in that high dimensional space.

2.4.2 K-Nearest Neighbor learning

The most basic instance-based algorithm is the *k*-Nearest Neighbor (KNN) algorithm [18, 19]. It assumes all instances correspond to points in the n-dimensional space R^n. The distance between instances is usually taken as the Euclidean distance, i.e., if an instance x_i is $x_i =< x_i^1, x_i^2, ..., x_i^n >$, where x_i^r denotes the value of the *r*-th feature of instance x_i, then the distance between two instances x_i and x_j is $d(x_i, x_j) = \sqrt{\sum_{r=1}^{n} (x_i^r - x_j^r)^2}$. Other distance metrics can be used as well.

In this thesis we only consider discrete-valued target functions (classes) of the form f: $R^n \rightarrow \Omega$, where Ω is the finite set $\{\omega_1,...,\omega_s\}$. As we will see, there is no difference when using KNN for two-class classification or for multi-class classification.

The *k*-Nearest Neighbor algorithm [16] assigns a query sample to the class that has a maximum number of representatives among the *k* training samples closest to it. Ties are usually broken at random. If $k = 1$ then the *k*NN algorithm assigns the query to the class of the nearest training sample. **Figure 2-5** illustrates the *k*NN algorithm. In this example the samples are points in the two-dimensional plane. The target function has a boolean value "– "or "+" (false and true, respectively). The query point x_q (sample) is shown in the center. If we are to use 1-Nearest Neighbor algorithm then x_q will be classified as negative. However, if we use 5-Nearest Neighbor algorithm then x_q will be classified as positive.

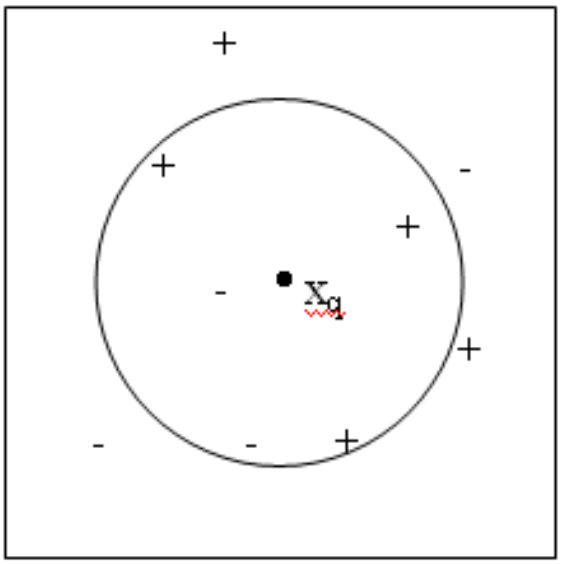

Figure 2-5. The kNN algorithm. Given the query point x_q, the k=5 closest points are determined, and the class having a majority among them (class '+' in this specific case) is assigned to the query. This figure is taken from [16], with some modifications.

This example introduces the problem of choosing k – the number of relevant neighbors. Although there is no rule for that, a common way (which was also used in this work) is to select k among several possible values using cross validation on the training samples. This way, each selection of k will lead to a different error estimation and the particular k that had the lowest error estimation on the training set is chosen.

2.4.2.1 Distance-Weighted Nearest Neighbor algorithm

One variant of the *k*NN algorithm is the Distance-Weighted Nearest Neighbor algorithm. The contribution of each of the k nearest neighbors is multiplied by a weight factor,

according to its distance from the query point x_q. A common weight factor of a neighbor is the inverse square of its distance from x_q. Thus, the classification rule will be

$$\hat{f}(x_q) = \arg\max_{v \in V} \sum_{i=1}^{k} w_i \delta(v, f(x_i))$$

where $f(x_i)$ is the known class label of x_i, $\delta(a,b) = 1$ if $a = b$ and $\delta(a,b) = 0$ otherwise, $x_1, ..., x_k$ are the *k* closest points to x_q and

$$w_i \equiv \frac{1}{d(x_q, x_i)^2}$$

2.5 Bayesian learning

2.5.1 Introduction

Bayesian learning [19] is a probabilistic approach to classification that provides a quantitative method for weighing the evidence supporting different hypotheses using probability distributions together with observed data. As a result, it has several advantages. First, it provides a flexible approach to learning, since each observed training sample can decrease or increase the estimated probability that a particular hypothesis is correct, but does not completely eliminate the hypothesis. A second advantage of Bayesian learning is that it can output probabilistic hypotheses, e.g., "the patient has 90% chance of not developing metastasis". This is in contrast with many classifiers that either just output a single most likely prediction, usually with some score that is not easily interpretable. Other advantages include the ability to combine prior knowledge (e.g., use

different prior probability for each candidate hypothesis), and to combine multiple hypotheses by weighing their probabilities. One practical difficulty is that these methods typically require initial knowledge of many probabilities. In case these probabilities are not known, they are usually estimated based on background knowledge and the given training data.

2.5.2 Bayes Theorem and maximum likelihood

A common problem in machine learning is determining the best hypothesis h from some space H, given the observed data D. The best hypothesis is defined as the most probable hypothesis, given the data D and any prior knowledge, or, more precisely, the hypothesis that would make the most probable classification given the data D and any prior information about the probabilities of the various hypotheses in H.

Bayes Theorem provides a direct way for calculating such probabilities. Bayes Theorem is

$$P(h \mid D) = \frac{P(D \mid h)P(h)}{P(D)}$$

where $P(h)$ is the initial probability that h holds, before we observed the data (the *prior probability* of h). It may reflect any background knowledge about h. $P(D)$ denotes the prior probability that data D will be observed. $P(D \mid h)$ denotes the probability to observe data D given that hypothesis h holds. $P(h \mid D)$ denotes the probability that h holds given the observed data D – and this is the quantity we are looking for. $P(h \mid D)$ also called the *posterior probability* of h.

In many learning applications the goal is to find the most probable hypothesis $h \in H$ given the observed data D. This hypothesis is called a *maximum a posteriori* (MAP) *hypothesis*, and it is defined as follows

$$h_{MAP} \equiv \arg\max_{h \in H} P(h \mid D) = \arg\max_{h \in H} \frac{P(D \mid h)P(h)}{P(D)} = \arg\max_{h \in H} P(D \mid h)P(h)$$

The term $P(D)$ is dropped in the final step as it is typically a constant independent of h. If we assume that every hypothesis in H is equally probable a priori, then we can further simplify the equation and only maximize the term $P(D \mid h)$. This term is called the *likelihood* of the data D given h. The hypothesis h that maximizes $P(D \mid h)$ is called *maximum likelihood* (ML) hypothesis h_{ML}. Therefore, when all h_i-s are equally probable a priori, h_{ML} is defined as

$$h_{ML} = \arg\max_{h \in H} P(D \mid h)$$

2.5.3 The Naïve Bayes Classifier

Denote the set of possible classes by Ω (e.g., $\Omega = \{\omega_1, \omega_2\}$ for the binary classification problem). Denote the query x described by the vector of feature values $< x^1, x^2, ..., x^n >$. The most probable hypothesis that we wish to find is actually the most probable class ω_i of the query instance. Therefore, we would like to find the most probable class (hypothesis), given the query instance (data):

$$v_{MAP} = \arg\max_{\omega_i \in \Omega} P(\omega_i \mid x^1, x^2, ..., x^n)$$

Using Bayes theorem we can now rewrite this expression

$$v_{MAP} = \arg\max_{\omega_i \in \Omega} \frac{P(x^1, x^2, ..., x^n \mid \omega_i) P(\omega_i)}{P(x^1, x^2, ..., x^n)} = \arg\max_{\omega_i \in \Omega} P(x^1, x^2, ..., x^n \mid \omega_i) P(\omega_i)$$

It is easy to estimate $P(\omega_i)$ by simply counting the frequency with which each target value ω_i occurs in the training set. However, estimating $P(x^1, x^2, ..., x^n \mid \omega_i)$ by counting is not feasible unless we have a huge training set, as we need to observe every possible feature values combination $< x^1, x^2, ..., x^n >$ many times to obtain reliable estimates.

The naïve Bayes classifier estimates this term by assuming that the feature values are conditionally independent given the target value, i.e., $P(x^1, x^2, ..., x^n \mid \omega_i) = \prod_j P(x^j \mid \omega_i)$. Thus, the naïve Bayes classification rule is simply

$$v_{NB} = \arg\max_{\omega_i \in \Omega} P(\omega_i) \prod_j P(x^j \mid \omega_i)$$

The training step is the estimation of the various $P(\omega_i)$ and $P(x^j \mid \omega_i)$, based on their frequencies over the training data [16].

2.5.4 Practical issues with Naïve Bayes classifiers

2.5.4.1 Continuous features

In the Naïve Bayes algorithm, the relevant probabilities for the classes are found based on their frequencies over the training data. While this is a simple task when dealing with discrete features, it is more complicated when the features attain continuous values. A simple but effective way of incorporating continuous features in Naïve Bayes classifier is

by discretizing them. Discretization can be unsupervised (i.e., a fixed partition into bins) or supervised (i.e., binning using information in training data).

A simple example of supervised discertization of the data is as follows. First, for each feature, its average value (or median) in the training set is computed. Then, every continuous feature value is replaced with zero if the value is lower than the average, otherwise it is replaced with one. Now the learning phase (i.e., extracting all relevant probabilities) can be done. In the prediction phase each feature value of the query sample is discretized in the same fashion and the Bayes classification rule can be applied.

2.5.4.2 Estimating probabilities

If a certain combination of a class and feature values never occurs in the training set, then its frequency-based probability estimate will be zero. This is problematic since it will reset to zero all information in the other probabilities when they are multiplied. Poor estimation occurs also when the number of observations of a particular value a_i of a feature is small. In other words, we estimate $P(x^j \mid \omega_i)$ by the fraction $\frac{n_c}{n}$, where n is the total number of training samples the belong to class ω_i, and n_c is the number of these for which feature j equals a_i. When n_c is very small, this fraction provides a poor estimation, and when n_c is zero it will reset to zero every prediction for a query sample having feature i equals to a_i.

To overcome these difficulties, and make sure that no probability is ever set to be exactly zero, a small-sample correction of all probability estimates is often used. One such correction is the *m*-estimate [20] defined as

$$\frac{n_c + mp}{n + m}$$

where p is the prior estimate of the probability we wish to calculate, and m is a constant that determines how to weight p relative to the observed data. A typical choice of p, when we have no other information, is to assume a uniform distribution, i.e., if a feature has k possible values, then $p = \frac{1}{k}$.

The m-estimate can be interpreted as expanding the n actual samples by additional m 'virtual' samples distributed according to p. Notice that in the limit, as the number of samples grows, this estimate converges to the simple estimate $\frac{n_c}{n}$.

2.6 Feature selection and extraction

2.6.1 Introduction

Often, samples have many features (i.e., they are represented as vectors in a high-dimensional space). The tasks of feature selection and feature extraction is to reduce the dimension of the data as much as possible while still retaining as much information relevant for the task at hand [11]. There are many reasons to perform such dimension reduction. It may remove redundant or irrelevant information and thus yield a better classification performance; subsequent analysis of the classification results is easier; low dimension results may be visualized, and thus enable better understanding.

There are two main ways to achieve dimension reduction for classification problems. The first way is to identify (by some criterion) those features that contribute most to the class

separability. For example, one may select d features out of all the given features, using some method of ranking (the univariate approach) or optimizing a criterion function (the multivariate approach), that will most contribute to the classification task. This strategy is termed *feature selection*. The other way is to find a transformation (linear or nonlinear) from the original high-dimensional input space to a lower dimensional feature space. This approach is termed *feature extraction*. This transformation may again be supervised or unsupervised. In the supervised case, the task is to find the transformation for which a particular criterion of class separability is maximized.

2.6.2 Feature Selection

The feature selection problem is defined as follows: "given a set of k measurements (features) on n labeled samples, what is the best subset of d features that contribute most to class discrimination?" The number of possible such subsets is $\frac{k!}{d!(k-d)!}$, which can be very large even for moderate values of k and d. Therefore, one resorts to various heuristics for searching through the space of possible features.

There are many strategies for feature selection. For example, one can define an objective function, e.g., one that measures accuracy on a fixed held out set, and use sequential forward or backward selection. A sequential forward selection (SFS) is a bottom-up search where new features are added to a feature set one at a time. At each stage, the chosen feature is one that, when added to the current set, maximizes the objective. The feature set is initially empty. The algorithm terminates when the best remaining feature worsens the objective, or when the desired number of features is reached. The main

disadvantage of this method is that it does not delete features from the feature set once they have been chosen. As new features are found in a sequential, greedy way, there is no guarantee that they should belong in the final set.

Sequential backward selection (SBS) is the top-down analog of SFS: Features are deleted one at a time until *d* features remain. This procedure has the disadvantage over SFS that it is computationally more demanding, since the objective function is evaluated over larger sets of variables.

2.6.2.1 Feature Selection classes

Feature selection techniques can be organized into three categories, depending on the way they combine the feature selection search with the construction of the classification model: filter methods, wrapper methods, and embedded methods [1].

Filter methods choose the *d* best individual features, by first ranking the features by some 'informativeness' criterion [1], for example, using their Pearson Correlation with the target. Then, the top *d* features are selected. Afterwards, this subset of features is presented as input to the classification algorithm.

Wrapper methods [1] use a search procedure in the space of possible feature subsets using some search strategy such as SFS or SBS, and various subsets of features are generated and evaluated. The evaluation of a specific subset of features is obtained by training and testing a specific classification model. In other words, the search for the desired feature subset is "wrapped" around a specific classifier and training algorithm.

In *embedded* methods [1] the search for an optimal subset of features is built into the classifier construction. Features are selected as a part of the building of the particular classifier, in contrast to the wrapper approach, where a classification model is used to evaluate a feature subset that is selected without using the classifier. The embedded and wrapper approaches are specific to a given classifier.

As the filter approach is the more common one [1], our study will focus on several filter methods.

2.6.2.2 Filters

In this section we will introduce four different common filter methods for feature selection.

2.6.2.2.1 Pearson Correlation Coefficient

The Pearson correlation coefficient is computed between each feature vector x (where each entry represents its value in a particular sample) and the class vector y (having only two values, e.g., "1" and "2", to identify the class label). Pearson correlation coefficient between two variables *x* and *y* sampled *n* times is defined as

$$r_{xy} = \frac{\sum_{i=1}^{n}(x_i - \overline{x})(y_i - \overline{y})}{(n-1)s_x s_y}$$

where $\overline{x}$ and $\overline{y}$ are the sample means of x and y, s_x and s_y are the sample standard deviations of x and y, and n is the number of samples.

The *d* features that yield the highest scores are selected. Pearson correlation is commonly used in the analysis of microarrays [21].

2.6.2.2.2 T-test

Those features whose measures are significantly different between the two classes of samples are candidates for selection [22]. A simple t-test statistic [23] can be applied to measure the statistical significance of a difference of a particular feature between the two classes. Then, those *d* genes with the largest *t*-statistic (or, equivalently, the lowest p-values) are selected. In this work we use a modified form of the *t*-statistic, known as the Welch test [23], as the feature values in the two classes may have different variance. Welch test is defined as

$$t = \frac{\mu_1(f) - \mu_2(f)}{\sqrt{\frac{s_1^2(f)}{n_1} + \frac{s_2^2(f)}{n_2}}}$$

where $\mu_i(f)$, $s_i(f)$ and n_i are the mean, standard deviation and sample size in class i=1,2 of feature f in the training set.

2.6.2.2.3 Golub criterion

This filter was introduced in [24]. Let $\mu_i(f)$ and $s_i(f)$ be defined as above. Then, PS is defined as

$$PS(f) = \frac{|\mu_1(f) - \mu_2(f)|}{s_1(f) + s_2(f)}$$

Features with larger PS are more informative. Hence, this filter selects those *k* features with the largest PS.

2.6.2.2.4 Mutual information

Mutual information I(X, Y) measures the mutual dependence between two random variables X and Y [25]. It compares the observed joint distribution and what the joint distribution would be if X and Y were independent. The mutual information of two discrete random variables X and Y is defined as

$$I(X,Y) = \sum_{y \in Y} \sum_{x \in X} p(x, y) \log(\frac{p(x, y)}{p(x)\, p(y)})$$

The needed probabilities are calculated by extracting relevant frequencies from the training set. Thus, I(X, Y) =0 if and only if X and Y are independent. Mutual information is calculated between each feature vector and the class vector. Of course, since the probability distributions p(x), p(y), and p(x,y) are usually not known, they must somehow be modeled or estimated. For example, when feature values are continuous one may resort to discretizing them by binning the values.

2.6.3 Linear feature extraction

The methods described in the previous section select those features that contain the most discriminatory information by some criterion. In feature extraction, all available features are used, and the original data are transformed into a low dimensional space. Thus, the original features are replaced by a smaller set of *extracted features* in the new space.

Both feature selection and feature extraction reduce the dimension of the data and aim to provide a more relevant set of features for a classifier. In many cases, feature extraction can reduce redundancy better, reveal meaningful behavior of data, and thus lead to greater understanding of processes. In this section the focus will be on linear feature extraction, and specifically Partial Least Squares methods.

2.6.3.1 Partial Least Squares

Partial Least Squares (PLS) is a broad class of methods for modeling relations between sets of observed features by means of latent variables called components [26]. It is an iterative method that finds the relationship between a two-dimensional sample × feature matrix X and the class vector y of the samples (in its most general form, PLS models relations between two matrices, but we shall present first the version where the second matrix is a vector, which is relevant to classification). PLS was developed by Herman Wold and coworkers [2-4].

2.6.3.1.1 Notation

For reference and consistency, we shall use the following notation in this section.

$\underline{v}$	vector
$\overline{\underline{v}}$	mean of vector $\underline{v}$ (a scalar)
M	matrix
~	estimated value of a parameter, or a predicted variable
$\square_i$	variable in the i-th iteration of PLS
a	number of desired components
n	number of samples
k	number of features

m number of target functions that we wish to predict (If we wish only to predict the class label then $m = 1$)

X $n \times k$ data matrix (specific matrix X)

$\underline{y}$ vector of n entries (specific vector $\underline{y}$)

$\underline{x}[j]$ column j of matrix X (vector of length n)

2.6.3.1.2 The basic algorithm

The basic goal of PLS is to obtain a low dimensional approximation of a $n \times k$ matrix X such that the approximation will be 'as close as possible' to an $n \times 1$ vector $\underline{y}$. The simplest approximation is one dimensional: One seeks a $k \times 1$ vector $\underline{w}$ such that $\|\underline{w}\| = 1$ and $\text{cov}(X\underline{w}, \underline{y})$ is maximal. $X\underline{w}$ is called the *component* of X with respect to $\underline{y}$, and denoted by $\underline{t}$. The *approximation error* is defined as $E = X - \underline{t}\underline{p}^T$ where $\underline{p}$ is a $k \times 1$ vector minimizing $\left\|X - \underline{t}\underline{p}^T\right\|$. Similarly, the approximation error of $\underline{y}$ is defined as $\underline{f} = \underline{y} - q\underline{t}$, where q is a scalar minimizing $\|y - q\underline{t}\|$. $\underline{p}$ and q are called the *loadings* of $\underline{t}$ with respect to X and $\underline{y}$, respectively.

The same process can be repeated iteratively by taking

$$\begin{matrix} X_0 = X \\ \underline{y}_0 = \underline{y} \end{matrix} \quad ; \quad \begin{matrix} X_1 = E \\ \underline{y}_1 = f \end{matrix}$$

Hence, in the second iteration, a second component of X with respect to $\underline{y}$, is computed, new approximation errors are obtained, which later can be used to compute the third component, etc.

The substitution of X and $\underline{y}$ by their approximation errors is called *deflation*. The desired number of components (hence, iterations) a is given to the algorithm as input.

This variant of PLS is called PLS1. The exact way of computing the approximations and the residuals defines the different variants of PLS [27].

2.6.3.1.3 PLS variants

PLS Mode A

This variant deals with the general case where both X and Y are matrices. Hence, X is defined as before, and Y is a $n \times m$ matrix, i.e., there are several target functions we wish to simultaneously infer. In each iteration of PLS Mode A it seeks two weight vectors: a $k \times 1$ vector $\underline{w}$, and an $m \times 1$ vector $\underline{c}$ that maximize $\text{cov}(X\underline{w}, Y\underline{c})$, such that $\|w\| = \|c\| = 1$. In this approach X and Y matrices are approximated using different components, thus the approximation errors are

$$E = X - \underline{t}\underline{p}^T; \qquad F = Y - \underline{u}\underline{l}^T$$

where $\underline{t}$ and $\underline{p}$ are as defined before, $\underline{u} = Y\underline{c}$, and $\underline{l}$ is an $m \times 1$ vector found in a similar way to $\underline{p}$, i.e., by minimizing $\left\| Y - \underline{u}\underline{l}^T \right\|$. Then, these approximation errors, also called *approximations residuals*, are passed to the next iteration as the new X and Y matrices.

This approach was originally designed by Herman Wold [28] to model the relations between different blocks of data. This process treats X and Y symmetrically and seems to be more appropriate for modeling existing relations between the blocks than for prediction purposes [27].

PLS2

PLS2 is the multidimensional version of PLS1, i.e., $\underline{y}$ is no longer a vector but rather a matrix Y. Both PLS1 and PLS2 are used as regression methods and are the most frequently used PLS approaches.

In contrast to PLS Mode A, the PLS1 and PLS2 approaches are asymmetric, i.e., they use only one type of components ($\{\underline{t}_i\}_{i=1}^{a}$) for the approximations. PLS1 and PLS2 find the $\{\underline{t}_i\}_{i=1}^{a}$ components of matrix X, and use them to approximate both matrices X and Y using the formulas $E = X - \underline{t}\underline{p}^T$ and $F = Y - \underline{t}\underline{q}^T$ (or $\underline{f} = \underline{y} - \underline{t}q$ for PLS1 mode). This iterative procedure guarantees mutual orthogonality of the extracted components $\{\underline{t}_i\}_{i=1}^{a}$ [29]

PLS-SB

In the above variants of PLS, the $\{\underline{t}_i\}_{i=1}^{a}$ components were calculated iteratively, by finding the relevant weight vector $\underline{w}$ on each iteration. It can be shown that the weight vector $\underline{w}$ can also be found by finding the first eigenvector of $X^T Y Y^T X$, i.e., by solving the system

$$X^T Y Y^T X w = \lambda w$$

PLS-SB variant deals with the problem of finding approximations to all the $\underline{w}$ vectors at once by solving eigenvectors equations of the form above [29-31]. In contrast to PLS1 and PLS2, the extracted components $\{\underline{t}_i\}_{i=1}^{a}$ are in general not mutually orthogonal.

SIMPLS

This method was introduced in [32] and basically is avoiding the deflation steps at each iteration of PLS1 and PLS2. It directly finds the weight vectors $\{\underline{w}_i\}_{i=1}^{a}$, which are then applied on the original, undeflated X matrix to obtain the components $\{\underline{t}_i\}_{i=1}^{a}$ (and therefore the $\{\underline{w}_i\}_{i=1}^{a}$ vectors are different from the previously found weight vectors, which were applied to the deflated X matrices). The mutual orthogonality of the extracted components $\{\underline{t}_i\}_{i=1}^{a}$ is kept in this form.

As we use the PLS1 form in this work, its more detailed mechanism is explained in the next section.

2.6.3.1.4 Classification with PLS1 Algorithm

The use of PLS1 in classification is done in two parts – learning and prediction. In the learning part PLS1 extracts the $\{\underline{t}_i\}_{i=1}^{a}$ components, by finding the weight vectors $\{\underline{w}_i\}_{i=1}^{a}$. These components are used to approximate the X matrix (expression matrix) and the $\underline{y}$ vector (class label vector).

In the prediction part the $\{\underline{t_i}\}_{i=1}^{a}$ components are extracted from the query sample $\underline{z}$ using the weight vectors $\{\underline{w_i}\}_{i=1}^{a}$ found in the learning phase. Together with the loadings $\{\underline{p_i}\}_{i=1}^{a}$ and $\{q_i\}_{i=1}^{a}$ found earlier, PLS1 can then estimate the value of $\tilde{y}_z$, i.e. the estimated value of the class label of the query sample.

It should be emphasized that PLS1 is designed for regression, and as such it does not predict the query sample's class. However, for binary classification problem one can represent the class variable by as a numeric variable with two possible values, typically 0 and 1. In such representation PLS1 can output, for example, "0.92" as the query sample's approximated class label.

The detailed algorithm is given as follows [33]:

Learning

1. From each column *j* of the matrix *X* and vector $\underline{y}$, subtract their mean ($\underline{\bar{x}}[j]$ and $\bar{y}$, respectively). Call the resulting arrays X_0 and $\underline{y}_0$, respectively.
2. For $i = 1,...,a$ do the following:
    a. Find a weight vector $\tilde{w}_i$ that maximizes the covariance between the linear combination $X_{i-1}\tilde{w}_i$ and $\underline{y}_{i-1}$ under the constraints that $\tilde{w}_i^T \tilde{w}_i = 1$. This corresponds to finding a unit vector $\tilde{w}_i$ that maximizes $\tilde{w}_i^T X_{i-1}^T \underline{y}_{i-1}$, the scaled covariance between X_{i-1} and $\underline{y}_{i-1}$. The solution is $\tilde{w}_a = cX_{i-1}^T \underline{y}_{i-1}$

where c is the scaling factor that makes the length of $\tilde{w}_a$ equal to one, i.e., $c = (\underline{y}_{i-1}^T X_{i-1} X_{i-1}^T \underline{y}_{i-1})^{-\frac{1}{2}}$.

b. Calculate the component $\underline{\tilde{t}}_i = X_{i-1}\underline{\tilde{w}}_i$.

c. Estimate the regression coefficients $\underline{p}_i$ by finding the Least Squares (LS) approximation of $X_{i-1} = \underline{\tilde{t}}_i \underline{p}_i^T + E$. Thus, $\underline{\tilde{p}}_i = \frac{X_{i-1}^T \underline{\tilde{t}}_i}{\underline{\tilde{t}}_i^T \underline{\tilde{t}}_i}$.

d. Estimate the regression coefficient q_i by finding the LS approximation of $\underline{y}_{i-1} = \underline{\tilde{t}}_i q_i + \underline{f}$. Thus, $\tilde{q}_i = \frac{\underline{y}_{i-1}^T \underline{\tilde{t}}_i}{\underline{\tilde{t}}_i^T \underline{\tilde{t}}_i}$.

e. Compute X and $\underline{y}$ approximation residuals by subtracting their estimations:

$$\tilde{E} = X_{i-1} - \underline{\tilde{t}}_i \underline{\tilde{p}}_i^T$$
$$\underline{\tilde{f}} = \underline{y}_{i-1} - \underline{\tilde{t}}_i \tilde{q}_i$$

f. Replace the former X_{i-1} and $\underline{y}_{i-1}$ with the new residuals $\tilde{E}$ and $\underline{\tilde{f}}$, and continue with the next iteration, i.e.

$$X_i = \tilde{E}$$
$$\underline{y}_i = \underline{\tilde{f}}$$
i=i+1

The complexity of each iteration is $O(n \times k)$, as this is the complexity of calculations of matrix products needed for the component construction. Therefore, The total complexity of the learning stage is $O(a \times n \times k)$.

Prediction

1. Given , a $k \times 1$ query instance $\underline{z}$, subtract from each feature the mean value of that particular feature found in the learning step. Denote the resulting vector by $\underline{z}_0$.
2. For $i = 1,...,a$, perform the following steps:
    a. Using $\underline{\tilde{w}}_i$, calculate new $\tilde{t}_i = \underline{z}_{i-1}^T \underline{\tilde{w}}_i$.
    b. Using $\underline{\tilde{p}}_i$, compute new residual $\underline{z}_i = \underline{z}_{i-1} - \tilde{t}_i \underline{\tilde{p}}_i$
3. Using $\underline{\overline{y}}$, and using $\{\tilde{q}_s\}_{s=1}^a$, predict the target function value of the query sample $\underline{z}$ by

$$\mathrm{tf}(\underline{z}) = \underline{\overline{y}} + \sum_{s=1}^{a} \tilde{t}_s \tilde{q}_s$$

4. Determine the inferred class by rounding $\mathrm{tf}(\underline{z})$.

The prediction stage is similar to the learning stage - $\{t_i\}_{i=1}^a$ components are calculated using the $\{\underline{w}_i\}_{i=1}^a$ weight vectors found earlier. However, now, we are only dealing with only one sample ($\underline{z}$), and not with a group of samples as in the learning stage. Therefore, each calculated component is actually a scalar (as a linear combination of features from

one sample yields a single number). Because of that, in each iteration we have $O(k)$ calculations, and the overall complexity of this step is $O(a \times k)$.

3 SlimPLS

Ranking-based filters utilize a univariate approach when selecting features. In some cases they can produce reasonable feature sets, especially if the features in the original set are uncorrelated. However, since the method ignores multivariate relationships, the chosen feature set will be suboptimal when the features of the original set are highly correlated: Some of the features will add little discriminatory power, although ranked relatively high [1, 11]. In these cases it is sometimes better to combine a more predictive feature (having a high rank according to some criterion) with some less predictive ones that correlate less with it. This way, the added features will be able to better 'explain' unexplained (or residual) 'behavior' of the samples than when using only top-scoring features. Moreover, in some cases the individual features are not highly predictive, but when combined together they gain predictive power. See **Figure 3-1** for example.

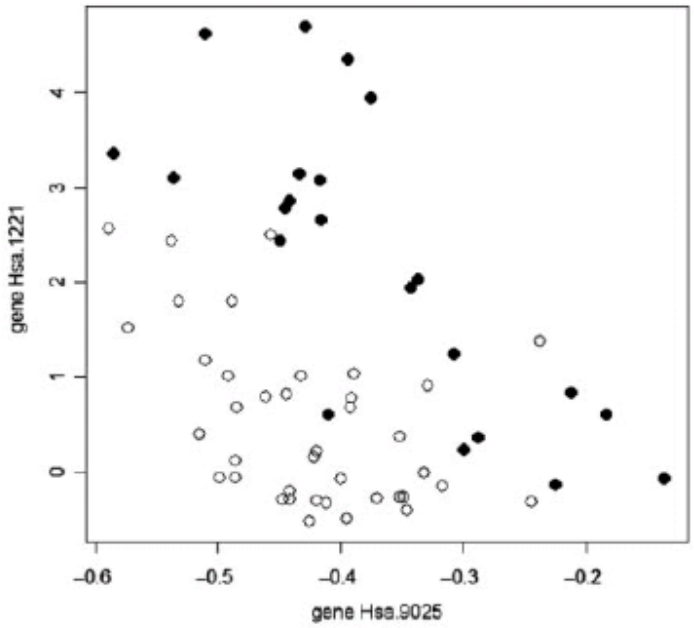

Figure 3-1. Example of synergy between two genes. The plot shows the expressions of genes Hsa.9025 and Hsa.1221 from a colon cancer dataset [34]. White dots represent sick patients and black dots normal controls. The combination of the two genes clearly distinguishes the two conditions, while the individual genes do not. This figure is taken from [35].

PLS is a good candidate for overcoming these problems, due to several reasons:

1. The PLS components are orthogonal and uncorrelated.
2. Each component tries to approximate the residual (or error) left after using all former components.

However, the method – in its original form – uses all the features without selection. Each component is constructed by a linear combination of all features using the weight vector *w*. By manipulating this vector, we can use PLS for feature selection or feature extraction, as will be described below. This way, we will choose only the most relevant features from each component before advancing to the next component. We call this technique SlimPLS.

3.1 Considerations in applying PLS for feature selection

The application of PLS for feature selection requires several decisions:

1. How many features should be selected? The performance of classification and feature selection methods depends, among other things, on the number of features that are selected. Too few features will not have enough classification power, while too many features may add noise and cause overfitting. Our analysis (see **Section 4.3.1**) showed clear improvement in performance when increasing the number of related features from 20 to 50, but no clear improvement when increasing the number of features beyond 50. Therefore, we used 20 and 50 feature configuration in our studies.
2. How many components of the PLS algorithm should be used? Typically, components computed at later iterations are much less predictive than former ones, as they approximate the residual of the residual etc., but one should determine the best number of components to use via some principled method.
3. How many features should be selected from each component? Exactly how should they be selected?
4. Should one use the selected features themselves as the output of the process, or perhaps use the extracted PLS component (a linear combination of the selected original features) as the output?

We considered several possible answers to each question, and tested systematically algorithm variants implementing combinations of such choices.

3.2 The number of components and the number of features per component

We studied two possible approaches to partitioning the number of features across the PLS components.

a) A constant partition approach (named **CONST**): Prespecify the number of total features x sought and the number of features from each component y. We denote such variant by x-y. For example, CONST-50-10 chooses a total of 50 features, 10 features from each component, thus iterating over five components; CONST-50-25 uses two components, selecting 25 features from each one; CONST-50-50 uses one component and chooses all features from it.

b) A dynamic partition approach based on computing p-values (named **PVAL**): This approach selects the number of components and the number of features from each component according to the properties of each component. A correlation coefficient is computed between each component and the original label vector (the $\underline{y}$ vector) and a p-value for that correlation is calculated [23]. Components participate in the feature selection only if they achieve p-values lower than a given threshold θ. Then, according to the distribution of the magnitudes of the p-values (-log (p-value)) of the relevant components, the numbers of the features taken from each component are determined.

   For example, suppose the threshold θ is set to 5×10^{-3}, and p-values for the correlation between the first ten components and the original label vector are calculated. The first

component has a p-value of 1.7×10^{-12}, the second has 5.2×10^{-5}, the third one 0.02, and all other components have p-values larger than 0.02. Since only the first two components have p-values smaller than the given threshold, features will be selected using only these components. Now, we have to decide how many features will be selected from each component. Beginning with the pair of p-values (1.7×10^{-12}, 5.2×10^{-5}) we calculate the –log (p-value): (11.77, 4.28). Then, we divide each score by the sum of all scores (11.77 + 4.28), to get the relevant proportions - (0.73, 0.27). The number of features is selected from each component according to these proportions. For example, if we wish to select 50 genes, then 37 will be chosen from the first component and 13 from the second.

After selecting the desired number of features from a particular component, we modify the original weight vector $\underline{w}$ by putting zeroes in all entries other than the selected features and then re-normalizing $\underline{w}$. This way a modified component is constructed (using the modified $\underline{w}$ vector) instead of the original component. Approximations to the *X* matrix and the $\underline{y}$ vector are computed using this new component, and then continuing to the next iteration, as in the original PLS algorithm.

3.3 Selecting features from a component

After finding the number of components and the number of features per component we need to find the features themselves. We studied two possible approaches.

a) Pick the top features in each component (variant **HIGH**): If we are to choose k features from a given component, we simply pick the k features that have the largest absolute weights in the weight vector $\underline{w}$ calculated for that component.

b) A hill-climbing improvement approach (variant **HC**): Use the group of features obtained in (a) as a base group, and begin a search using hill climbing [36] for a group of features of the same size that yield a lower approximation error ($E = X - \underline{t}\underline{p}^T$, where $\underline{t}$ is the constructed component using the selected group of features, and $\underline{p}$ is its loading) of the current $\underline{y}$ vector by this component. At each step of hill climbing, we randomly look for a better group of features, constructed by replacing one feature that currently belongs to the group by another feature that does not. The first switch that yields a lower approximation residual is chosen. This procedure ends when no improvement is found after given number of times (we used the number 50 in this study). The search is done separately for each component.

3.4 Feature selection and feature extraction

After finding the desired features in each component we can use them in two ways:

a) Use the selected features as the output. This approach is called **TOP**.

b) Use the components as extracted features: In each component use the selected features to modify the original weight vector $\underline{w}$ of that component, putting zeroes in all entries other than entries that belong to the selected features and then normalizing $\underline{w}$. The constructed modified components are the output. Hence, these components are the new extracted features, and each of them is a linear combination of some

original features. The total number of original features used is still as prescribed. In this approach the number of extracted features is the number of iterated PLS components. This approach is called **TCOMP**.

For examples, 50-HC-TCOMP-5×10^{-2} selects 50 features. The number of components and number of features in each component are selected using the PVAL approach with a threshold of 5×10^{-2}. Finally, this variant returns the modified components as the new extracted features.

Table 3-1 summarizes the different SlimPLS variants described in **Section 3.2** through **Section 3.4**.

Family	Feature Selector	Description
CONST	High-K-L-TOP	Select the L top features from each component
CONST	High-K-L-TCOMP	As above, but use the modified components as the extracted features
CONST	HC-K-L-TOP	Select L features from each component by hill climbing from the L top ones
CONST	HC-K-L-TCOMP	As above, but use the modified components as the extracted features
HIGH-PVAL	K-High-TOP-p	Select only components that show correlation p-value < p with the label vector; select the no. of features from each component according to their relative p-values
HIGH-PVAL	K-High-TCOMP-p	As above, but use the modified components as the extracted features
HC-PVAL	K- HC-TOP-p	Select only components that show correlation p-value < p with the label vector; select the no. of features from each component according to their relative p-values; Improve by hill climbing;
HC-PVAL	K-HC-TCOMP-p	As above, but use the modified components as the extracted features

Table 3-1. A summary of the SlimPLS variants and their properties. In all variants, the parameter K refers to the total number of features used. (In most tests below, K was set to 50 and the parameter is omitted from the feature selector's name).

3.5 Classification using PLS – prior studies

The concept of using PLS for classification is not new. There are several studies that constructed classifiers using PLS. In [37] the authors construct a classification procedure that involves dimension reduction using PLS and then classification using Logistic Discrimination (LD) and Quadratic Discrimination Analysis (QDA), which use the constructed components of PLS as the new extracted features. In addition, not all the genes are used for the construction of the components, but only a smaller sample is selected using t-test. In [67] the authors extend this two-step procedure to support multiclass classification. In [38] a two-class classification using PLS and penalized regression is described. First, q PLS components are constructed and a linear regression is built using the components. Then, using a penalizing procedure, only those genes that have coefficients larger than some threshold λ are kept. Both q and λ are determined in cross validation. The classification itself is made using the penalized linear regression. A similar procedure is done in [39] in order to combine information from two different datasets of gene expression (aiming to measure of the same phenotype) in order to achieve better classification.

The combination of PLS and linear regression techniques is further studied in [40]. In [41] a classification using PLS with penalized logistic regression is described. In this study different variants of PLS were studied resulting several variants of classifiers. This study, similarly to [37], usually used t-test filter before applying PLS. The discriminating abilities of PLS is studied in [42]. This study shows connection between PLS and Linear Discriminant Analysis in terms of classification. In addition, nonlinear extensions of PLS

were also published as kernel methods (e.g., [43, 44]), and their use together with SVM is described in [45].

All the above studies used PLS for classification, and when feature selection was involved, it was implicitly used. For example, in [38], where a penalizing process is applied to reduce the number of genes, the threshold parameter λ, which implicitly determines the number of features, is found using cross validation. Again, the goal in [38] is to construct a classifier rather than a feature selection technique that can be used with different classifiers.

For this reason, the SlimPLS method is novel in the sense it is dedicated to feature selection and does not propose a new classification procedure. As a result, it can be used with different classifiers, as a pre-process procedure. We shall use this fact in order to evaluate the performance of SlimPLS with different classifiers. For this reason, we shall compare the SlimPLS variants to other feature selection methods, and not to the PLS based classification methods mentioned above.

3.6 Implementation

Datasets were collected and stored as tab-delimited files of two-dimensional matrices of features (genes) and samples (a single file for each dataset). We used the R package [46] for the implementation of the SlimPLS methods, and used publicly available packages for the classifiers implementation (*e1701* [47] for SVM and Naïve Bayes, *class* [47, 48] for KNN, and *randomForest* [47, 49] for Random Forest). When running linear SVM, we used the grid $\{10^{-1},1,10,10^{2},10^{3},10^{4}\}$ of possible scores to find C . The Random Forest procedure was run with 1500 trees and $m=\sqrt{M}$. When running KNN, we used the grid

{1,3,5,7} of possible number of neighbors to find *k*. When using the *mutual information* filter we used ten equal-sized bins.

The implementation of the original PLS algorithm was done according Section 2.6.3.1.4, which is taken from [33]. The main function gets as an input the configuration file (tab-delimited) of the desired tests (which classifiers, feature selection techniques and datasets to use) and invokes the appropriate functions.

The main consideration of the implementation was the ability to monitor and continue long runs even if they are stopped in the middle. Therefore, files are written to the hard drive at several points in each iteration, which slows down the implementation.

The tests were done on three different platforms:

1. Windows XP, Intel Pentium 4 CPU, 3.00GHz, 2GB of RAM
2. Linux, Intel Xeon 5160 CPU, 3.00GHz, 4 GB of RAM
3. Linux, Intel Xeon 5150 CPU, 2.66GHz, 4 GB of RAM

To present some comparison of running times we used the HD caudate dataset [51] (containing 70 samples and 20223 features) and two classifiers – SVM-radial and KNN. We ran leave-one-out cross-validation on the dataset using the two classifiers, where, in each iteration, selected features were chosen and classification using both classifiers was made. Then, we repeated this procedure, only this time we did not use any classifiers, and only the feature selection phase was done. We used platform 2 above in these comparisons. **Table 3-2** summarizes the average running time of a single iteration when using different feature selection techniques.

Feature selection technique	Feature selection time	Classification time
Correlation filter	4.00	4.86
HC-50-50-TOP	9.71	4.71
HC-5×10^{-2}-TCOMP	27.00	3.00
HC-5×10^{-2}-TOP	26.29	5.43
HC-5×10^{-3}-TCOMP	27.57	2.43
HC-5×10^{-3}-TOP	27.86	4.43
HIGH-50-50-TOP	1.43	4.29
HIGH-5×10^{-2}-TCOMP	4.86	1.71
HIGH-5×10^{-2}-TOP	4.86	4.43
HIGH-5×10^{-3}-TCOMP	4.57	1.71
HIGH-5×10^{-3}-TOP	4.43	4.71

Table 3-2. Average running time of for different feature selection techniques. Times are seconds per iteration, performing leave-one-out cross validation. The SVM-radial and KNN classifiers were used after the feature selection phase.

Table 3-2 shows that HC variants are more time consuming. This is not surprising as the local search is added to these variants. In addition we can see that HC-50-50-TOP, which is HC-CONST variant, takes less time than other HC variants, as it does not use the dynamic evaluation of number of features per component.

When no hill climbing is used, i.e., the HIGH variants are used, a considerable improvement in the running time can be noticed. Some variants, such as the HIGH-CONST variants (only one variant of this family is shown – HIGH-50-50-TOP), achieve better running time than the correlation filter.

Another noticeable result is the shorter classification time when TCOMP variants are used. This is due to the fact that only few features (the extracted components) are given to the classifiers, thus making them faster.

4 Results

4.1 Datasets

We collected 19 datasets reported in the literature, of sample sizes 31-173 containing 2000-22283 features. The list of datasets appears in **Table 4-1**.

#	DataSets	PubMed ID	Paper	#Samples	#Class A	#Class B	#Probes
1	HD blood	16043692	[50]	31	14	17	22283
2	HD caudate	16467349	[51]	70	32	38	20223
3	Leukaemia	10521349	[24]	72	47	25	7129
4	HD cerebellum	16467349	[51]	66	27	39	20223
5	Prostate Cancer	12086878	[52]	102	50	52	12533
6	Breast Cancer	11823860	[21]	78	44	34	16783
7	Colon Cancer	10359783	[34]	62	40	22	2000
8	Crohn Disease blood	16436634	[53]	101	42	59	22215
9	Breast Cancer	17157792	[54]	118	43	75	22215
10	Liver Cancer	14675778	[55]	60	20	40	7070
11	Breast / Colon Cancer	16436632	[56]	104	62	42	22215
12	Lung Cancer	12118244	[57]	86	62	24	7129
13	Liver Cancer	12648972	[58]	60	40	20	7129
14	Prostate Cancer	11518967	[59]	53	19	34	4344
15	Breast Cancer	11507038	[60]	58	28	30	2166
16	Breast Cancer	11562467	[61]	49	25	24	2166
17	Ovarian Cancer	15897565	[62]	54	30	24	22283
18	Neural tissue (Mouse Muscle)	16002470	[63]	150	100	50	12488
19	Myeloma and Bone lesions	14695408	[64]	173	137	36	12625

Table 4-1. The Datasets that are used in this study. Datasets 12-19 were used in [65].

Our goal is to find the more informative features. However, different features have different scales, and in order to compare them a standardization of the data is needed. Therefore, we normalized each gene to have mean zero and a standard deviation equal to 1. This data standardization is a common pre-processing approach in microarray studies and was done previously when using PLS [38].

4.2 Performance evaluation criteria

Using the benchmark of 19 datasets, we tested five classifiers and 36 feature selection variants: four filters and 14 SlimPLS variants, and selecting a total of 20 and 50 features in each test. This gives a total of 180 combinations of classifiers and feature selection variants. To avoid confusion, we will call a feature selection algorithm simply a *feature selector* (FS), and reserve the term *"method"* for a combination of FS and classifier. Hence, we have to assess a total of 180 methods.

A key question is how to evaluate performance. As some datasets are harder to classify than others, evaluating performance by the number of errors in each would give these datasets higher weight. Relative ranking of performance gives equal weight to all datasets, but it ignores the absolute magnitude of the errors. For theses reasons we chose to use several criteria, each revealing a different aspect of the performance. Error rates were calculated using leave-one-out cross validation and performance was measured using five criteria:

a) Rank sum p-value. Define a three-dimensional array E where $E(i, j, k)$ is the error rate of classifier i and feature selector j on dataset k. Hence, the dimensions of E are $5 \times 36 \times 19$. Define an array R of the same dimensions where $R(i, j, k)$ is the rank of $E(i, j, k)$ among $E(i, *, k)$. Hence, $R(i, j, k)$ ranks feature selector j compared to all others for classifier i and dataset k. The score of a subset feature selectors $S = \{j_1, ..., j_n\}$ for classifier i is computed by comparing the distribution of the values $R(i, S, k)$ to the distribution of the values of $R(*, *, k)$, using the Wilcoxon rank-sum test [23]. This test determines to what extent a particular group of values (e.g., the error rates of one feature selector) tends to have low rank compared to the rest. The p-values calculated on each dataset were combined using Fisher's method [66]. This score compares the different combinations of classifier and feature selectors. This way, it also incorporates comparison between classifiers.

   Similarly, for each dataset, another comparison was made. This time the distribution of the values $R(i, S, k)$ is compared to the distribution of the values of $R(i, *, k)$ and a rank-sum score is computed as above. This score is used to compare the feature selectors using different individual classifiers, since it evaluates the performance of the different feature selectors using a particular classifier.

   We used the two scores defined here to compare combinations of a 'family' of feature selectors and classifier. In other words, we did not compare one feature selector to another, but compared groups of similar variants.

b) Average Rank. While the rank sum test determines the significance of the tendency of a method (or a feature selector) to be ranked higher or lower, we would also like to see the absolute differences between methods' ranks. For that reason we define another score that compares the average rank of a method. Formally, for classifier i and feature selector j, we define the score $\frac{1}{19}\sum_k R(i,j,k)$. Like (a), the values themselves do not matter, and only their relative ranking is considered. Unlike (a), this score is not assigned a probability. We use this score to compare between individual feature selectors using a particular classifier.

c) L2 distance. For a given classifier, 19 different error rates were calculated for a particular FS – one for each dataset. These values are the entries in a vector called *method-scores vector*. In addition, for the given classifier, another 19-dimensional vector is constructed, whose *i*-th entry is the minimal error rate achieved by any feature selector for dataset *i*. This is called the *minimum-scores vector*. The score of a method is the L2 distance between its method-scores vector and the minimum-scores vector. Formally, fix the classifier i. Let $\alpha^{ik} = \min_j E(i,j,k)$. Then the L2 score of feature selector j (using classifier i) is $(\sum_k (E(i,j,k) - \alpha^{ik})^2)^{1/2}$. This criterion was used in [65]. Unlike (a) and (b) it is not ranking-based, and it measures across all datasets how far a particular feature selector is from attaining the best score, given the classifier.

The next two criteria compare the methods in terms of exceptionally good scores and best scores over all datasets and classifiers (and not specifically for a particular classifier like the L2 distance criterion).

d) 95% confidence interval. Let $E(i,*,k)$ be the vector of error-rates of all feature selectors using classifier i on dataset k. Compute the average and 95% confidence interval of the average on each vector. Compute for each feature selector the fraction of dataset×classifier combinations on which it does better than the 95% confidence interval. Hence, this measure scores how often a feature selector obtains an exceptionally good score.

e) Best value rate. Calculate for each feature selector the proportion of tests on which it achieves the best score among all datasets and classifiers.

f) Binomial tail p-value. We used only 50 features configuration for the comparisons using this method. Let $E(i,j,k)$ be defined as before, using only the 50 features version of the feature selectors. Hence, the dimensions of E are $5\times18\times19$. $R(i,j,k)$ is defined as the rank of $E(i,j,k)$ among $E(*,*,k)$. To compare two methods m_1 and m_2, where the first one is combined of classifier i_1 and FS j_1, and the second one is combined of classifier i_2 and FS j_2, we compare the two vectors $R(i_1,j_1,*)$ and $R(i_2,j_2,*)$. Let $n_1 = \left|\{k \mid R(i_1,j_1,k) > R(i_2,j_2,k)\}\right|$ and let

$n_2 = \left|\{k \mid R(i_1, j_1, k) < R(i_2, j_2, k)\}\right|$. Then $n_d = n_1 + n_2$ is the number of datasets in which the ranks using method m_1 and method m_2 differ. Our null hypothesis is that the two methods show similar performance. In other words, after removing the entries that have identical values we assume that $R(i_1, j_1, k) > R(i_2, j_2, k)$ has a probability of 0.5. Therefore, n_1 has a binomial distribution $B(n_d, 0.5)$. The p-value for observing at least n_1 cases where method m_1 is ranked above method m_2 is: $P(n \geq n_1) = (0.5)^{n_d} \sum_{l=n_1}^{n_d} \binom{n_d}{l}$.

4.3 Results

In this section we will present and analyze the results of the different classifiers and feature selectors using the criteria described in the previous section.

4.3.1 The effect of the number of features

As was mentioned earlier, too few features will not have enough classification power, while too many features may add noise and cause overfitting. In order to compare the behavior of a particular feature selector j using a particular classifier i, we calculated the average error rate achieved by this feature selector using the particular classifier. Formally, we calculated $\frac{1}{19}\sum_k E(i, j, k)$. Notice that we use here the error rates, as we wish not to compare different feature selectors but the performance of a particular feature selector when using a different number of selected features.

For a given classifier, we calculated this average error rate for six different variants of SlimPLS: HC/HIGH-K-K-TOP, HC-5e-02-TOP/TCOMP and HC-5e-03-TOP/TCOMP when using nine different numbers of selected features – 20, 30, …, 100. Then, the average error rate over these feature selectors was calculated for each number of selected features. The results are summarized in **Figure 4-1** for two classifiers – KNN and SVM-radial.

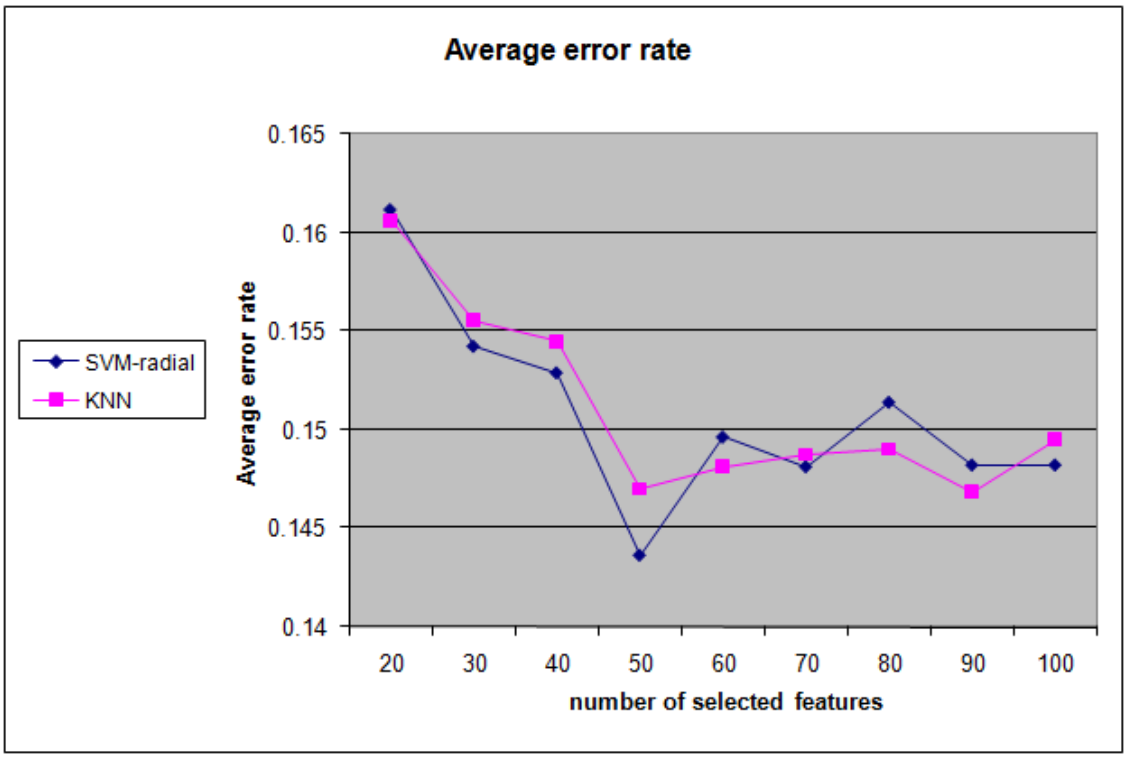

Figure 4-1. Average error of six different SlimPLS based feature selectors using the KNN and SVM-radial classifiers using different number of features.

As the number of selected features grew from 20 to 50, the improvement in classification was very clear. As the number of selected features grew even further, no additional improvement was noticeable and the average error rate usually got worse. Therefore, we will focus mainly on 50 feature configurations in the rest of the results, and will refer to the results with 50 features only, unless specified otherwise.

4.3.2 The effect of the classifier

The average rank-sum p-values of each classifier were calculated over three families of feature selectors:

a) Filters: The four filters used in this work.

b) CONST-50-50: The two variants that choose a constant number of features per component HIGH-50-50-TOP and HC-50-50-TOP.

c) HC-PVAL: The four variants that choose a variable number of features per component, depending on the p-values: HC-TCOMP-5e-03, HC-TOP-5e-03, HC-TCOMP-5e-02 and HC-TOP-5e-02.

The results are summarized in **Figure 4-2**. Among of the HC-PVAL variants, SVM (linear and radial) and KNN showed better performance than RF and NB. Moreover, these three classifiers together with the four HC-PVAL variants achieve the highest scores among all combinations. When using filters only, the RF classifier performs the best and SVM classifiers show second best performance. The worst performance of filters is obtained when using KNN classifier. In this classifier the difference in performance of HV-PVAL variants and filters is the most substantial (see discussion, **Section 5.1**).

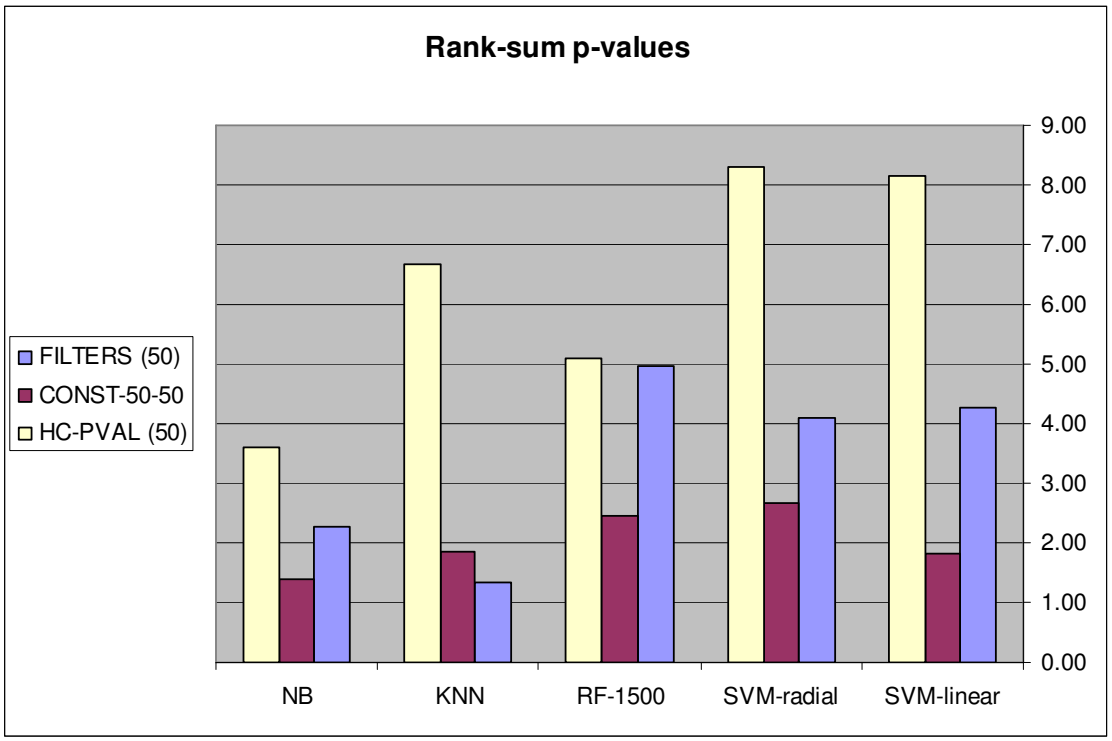

Figure 4-2. Rank-sum p-value of different classifiers using three families of feature selectors. -log(p values) of the combined Wilcoxon rank-sum tests for three families of methods using five different classifiers are shown. See test for the family definitions.

As SVM and KNN classifiers obtained the highest results we will show further focused analysis using these two methods in **Section 7.2** (appendix).

Figure 4-2 shows that SVM-linear obtained quite similar results to SVM-radial. When using the NB and RF-1500 classifiers, the HC-PVAL FS variants outperformed other feature selectors, but, in these cases the relative improvement is less dramatic.

To get a clearer understanding of the influence of the feature selectors on the different classifiers, we performed the second variant of the rank-sum test, as presented in **Section 4.2(a)**, i.e., this time we performed a comparison between the different feature selectors for each specific classifier separately. The results can be seen in **Figure 4-3**. The HC-

PVAL FS variants have a clear advantage over the other feature selectors. While the differences are mild when using RF-1500, they are stronger in the other classifiers.

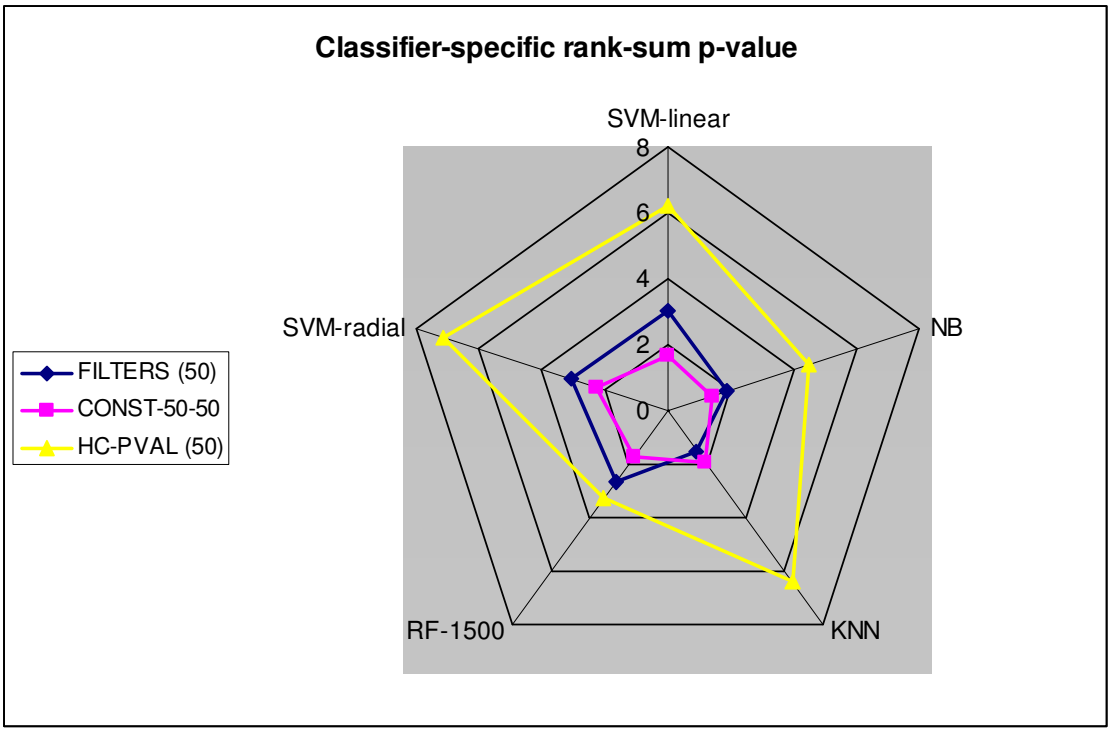

Figure 4-3. Rank-sum p-value of three families of feature selectors calculated separately for each classifier. -log(p values) of the combined Wilcoxon rank-sum tests for the three families using five different classifiers are shown. The concentric pentagons show the –log(p-value) scale. The results on separate classifiers are shown on the separate axes. This representation aims to emphasize the relative performance of each FS on each classifier separately, and not relative performance across classifiers. See text for the description of the families.

As in **Figure 4-2**, the greatest advantage of HC-PVAL feature selectors over the filters is attained when using the KNN classifier.

4.3.3 The effect of the feature selectors

To summarize the results we constructed *dominance maps*. These are graphs where each node is a method and a directed edge from method m_1 to method m_2 indicates that

method m_1 has significantly better performance (p-value ≤ 0.05) than method m_2. Performance is measured using the binomial tail for the relative accuracy of the two methods across the datasets. See **section 4.2 (f)** for more details.

We constructed five different maps, one for each classifier. Singletons, i.e., methods that were not significantly comparable to any other method (corresponding to isolated vertices in the map), are omitted. In addition, transitive edges were removed, i.e. if there three edges A→C, A→B and B→C exists, then edge A→C is removed. Out of the four variants of the HC-PVAL family of methods only two were taken – HC-PVAL-5e-03-TCOMP and HC-PVAL-5e-03-TOP. Finally, nodes (representing feature selectors) were categorized into five different groups of families and were colored accordingly. **Figure 4-4** summarizes the results.

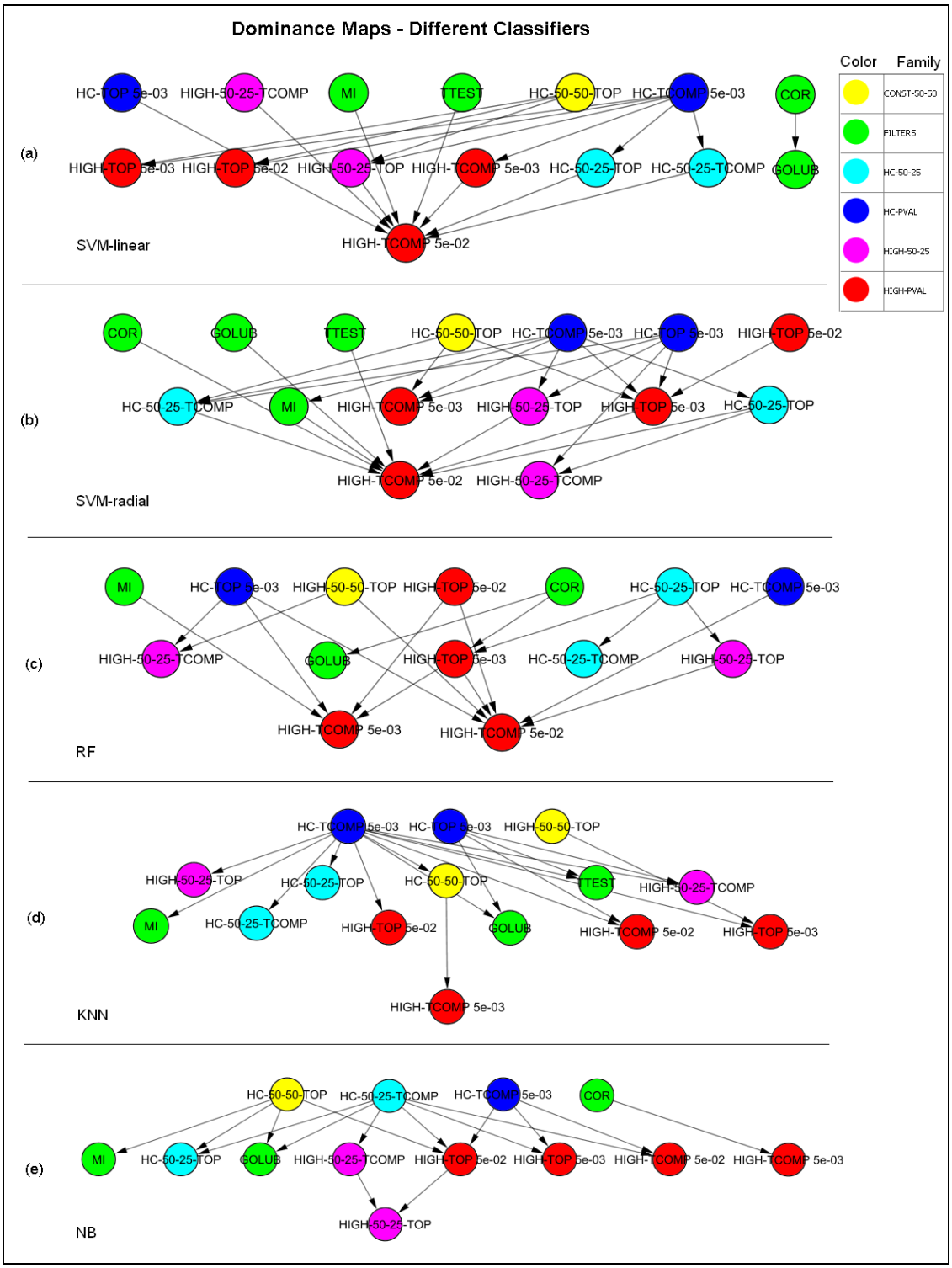

Figure 4-4. Dominance maps of feature selectors using different classifiers – (a) SVM-linear (b) SVM-radial (c) Random Forest (d) KNN (e) Naïve Bayes. An edge A→B indicates that A significantly outperforms B. In (d) the second and the third layer from the top were originally one layer that was divided into two rows for display purposes only. Methods in upper layers perform better than methods in lower ones.

One can notice a clear tendency of the HC-PVAL variants (the blue nodes) to appear in to the upper row, which consists of the better performing FS for the given classifier. The HC-PVAL nodes also tend to have more outgoing edges – showing dominance over a large set of other feature selectors.

In addition to the HC-PVAL variants, the FILTERS variants (green nodes) and CONST-50-50 variants (yellow nodes) also tend to perform well. A very strong dominance of the HC-PVAL variants is observed when the KNN classifier is used (**Figure 4-4 (d)**).

Four feature selectors were never dominated by others – HC-5e-03-TOP, HC-5e-03-TCOMP, HIGH-50-50-TOP and COR, the correlation filter (in some of the maps some of these methods are not visible as they are singletons).

4.3.4 Evaluation of the leading methods

In order to compare the different methods we used two-dimensional plots, where each point (x, y) in the graph represents a method and where x is the average error rate and y is the average rank of the method. The full evaluation is described in the appendix, **Section 7.1**. Here we show a different analysis focusing on the leading methods only.

We created another dominance map (**Figure 4-5**) containing only four feature selectors and all classifiers. We selected only these feature selectors that were not dominated by any others in the analysis in **Section 4.3.3**. These are: HC-5e-03-TOP, HC-5e-03-TCOMP, HIGH-50-50-TOP and COR.

All four methods using SVM-radial appear in the map in the upper layer. The combinations of SVM-linear and KNN with HC-5e-03-TCOMP dominate the largest number of others.

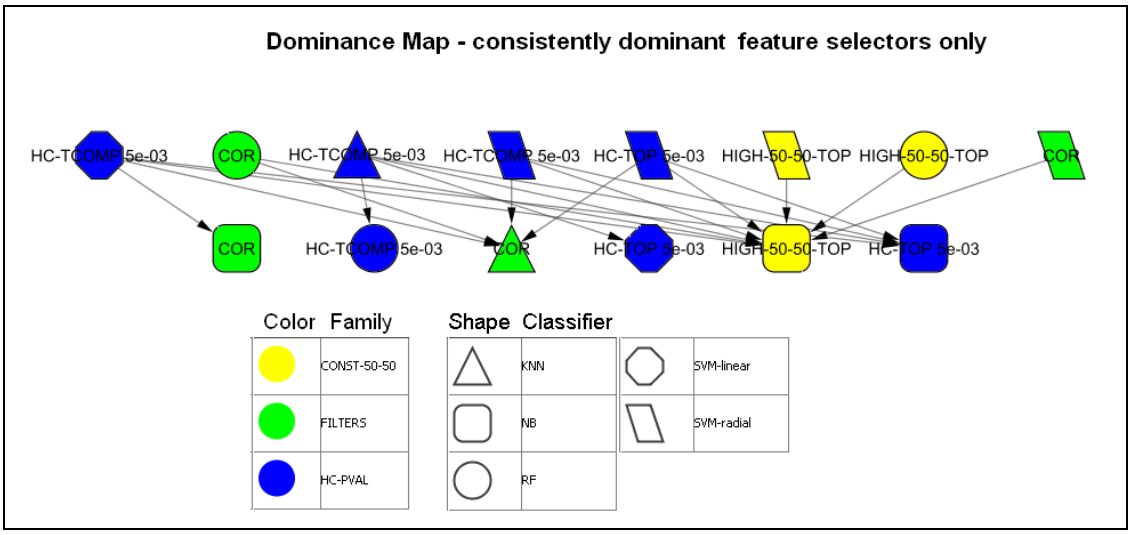

Figure 4-5. Dominance of the consistently dominant feature selectors using all classifiers. Singletons are omitted. Additional singletons not shown in the picture: KNN - HC-5e-03-TOP and HIGH-50-50-TOP, SVM-linear - HIGH-50-50-TOP and COR, RF - HC-5e-03-TOP, NB - HC-5e-03-TCOMP.

Most combinations involving the Naïve Bayes classifier (except for the combination with HC-5e-03-TCOMP, are dominated by others. This is consistent with **Figure 4-2**, where Naïve Bayes showed in general worse performance than other classifiers (the single exception is filters, which perform worse using the KNN classifier).

In views of these results, and consistently with the results in **Section 4.3.2**, further focused analysis on the KNN and SVM-radial classifiers was done , and it can be found in the **Appendix**.

4.3.5 Correlation between selected features

Features found by univariate approaches, like filters, tend to select correlated features. Multivariate approaches should have a lower average pairwise correlation between selected features, because features are selected as a group (or several groups). Less individually predictive features may be selected along with higher individually predictive ones and this will lead to lower average pairwise correlation between these features. Moreover, two features that are perfectly correlated will never be selected together by multivariate methods, since one will be redundant given the other. To measure these correlation values, for each dataset we found the features that were selected at least half of times in the leave-one-out cross-validation iterations, and measured their average pairwise correlation. We also recorded the number of such features. Then, we averaged both measures over all datasets. **Figure 4-6** summarizes these results.

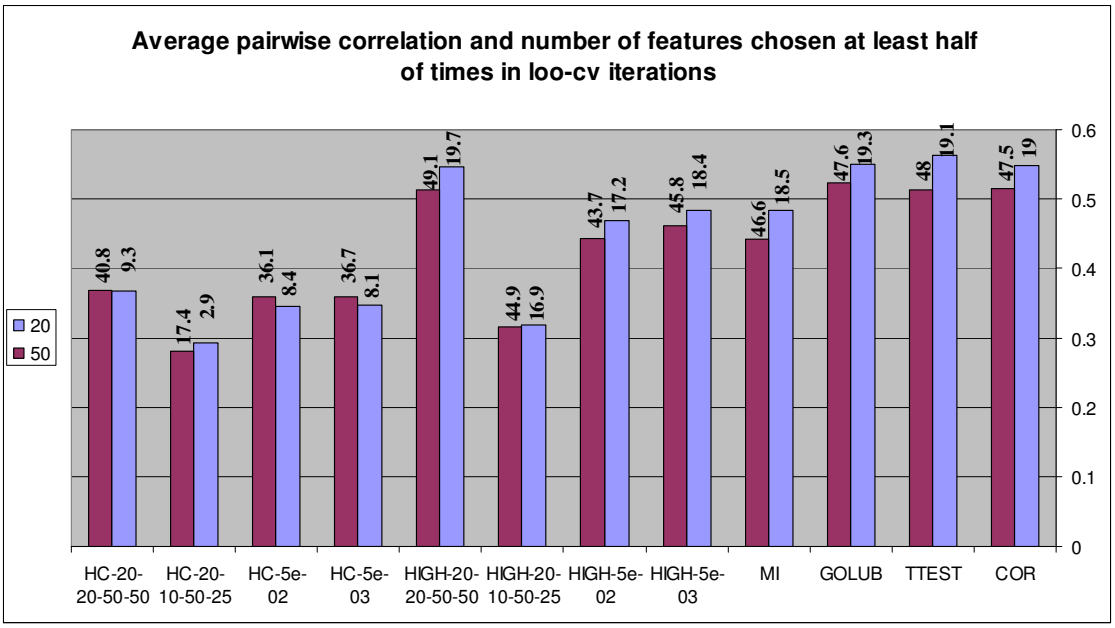

Figure 4-6. Average pairwise correlation and number of all features selected at least half of times in the leave-one-out cross-validation. Bars heights indicate the average pairwise Pearson correlation between the

expression patterns of the selected genes. The numbers written on top of the bars are the average numbers of selected features. Here, we do not distinguish between TOP and TCOMP variants, as the new extracted features returned by TCOMP variants are constructed using the same features selected in TOP variants, and it is their average pairwise correlation that we wanted to examine.

As expected, average correlation drops when using the slimPLS based methods, especially the HC variants, which actually have more potential for inter-feature variation because of the local search for a better subgroup of features. One exception is the HIGH-K-K variant, which presents similar results to the filters. Recall that this variant selects the K top absolute weighted features (where K is the total number of features to be selected) from the first component only. When no hill climbing is used, we found that features selected from the first component tend to be similar to features selected by the correlation filter (not shown). Therefore, this variant has a similar behavior to the correlation filter (as we also saw in **Figure 7-4** and **Figure 7-5**).

4.3.6 Main conclusions

Here we summarize the main conclusions of our analysis.

- Classifiers achieved a lower error rate when using 50 selected features compared to using 20 selected features. Increasing the number of features further did not show consistent improvement (**Figure 4-1**, see also **Figure 7-5**).
- Three families of feature selectors performed better than others: filters, CONST-K-K and HC-PVAL (**Figure 4-4**, see also **Figure 7-2**).
- Overall, the HC-PVAL variants showed the best performance among all the tested variants (**Figure 4-2**, **Figure 4-3**, **Figure 4-4**).

- The combination of the KNN classifier and the HC-TCOMP-5e-03 feature selector had the lowest average error-rate (see **Figure 7-2**). The combination of KNN and HC-TCOMP-5e-02 had the second lowest average error-rate (not shown). However, KNN tended to perform the worst when using filters (**Figure 4-2**).
- SVM-radial showed consistently high performance when using the better feature selectors (**Figure 4-5**).
- Although the HC-PVAL variants showed a slight advantage using the RF-1500 classifier (**Figure 4-2** and **Figure 4-3**), this classifier gave the most minor differentiation between feature selectors (**Figure 4-3**), and is therefore not recommended for SlimPLS variants.
- The filter methods tend to attain best performance when using the RF classifier (**Figure 4-2**).

5 Concluding remarks

5.1 Discussion

Our results show that the HC-PVAL variants of SlimPLS tended to outperform the other tested variants (**Figure 4-3** and **Figure 4-4**). This family of variants selects the number of features per component based on their significance and tries to improve the feature set by local search. Within this family, the TCOMP variants, which employ feature extraction, tend to achieve slightly better results than the TOP variants (e.g., **Figure 7-3** and **Figure 7-4**). This is not surprising as the components (that are actually the extracted features) are found in that way that maximizes the match to the class vector, i.e., the components are aimed to provide a good approximation of the class prediction. The TOP variants use the selected features for classification, but without the formulas that dictate how to re-build these components (i.e., the weight vectors). This way, the task of constructing the formulas, i.e., finding the relevant relationships between these features to get a good classification is left for the classifiers. When using the TCOMP variants we usually get one to three components, which already incorporate some 'collective' behavior of features found by SlimPLS. Moreover, each component tries to approximate the residual or the 'unexplained' behavior of the previous component. Therefore, these new extracted features show better contribution to the classification.

Another noticeable result is that the improvement achieved by the HC-PVAL variants compared to the filters is more dramatic when using the KNN classifier (**Figure 4-3**). This is an interesting result, as the KNN classifier is very sensitive to the selected features

(see **Section 2.4.1**). This fact may imply that SlimPLS based feature selection techniques manage to find good informative groups, especially when these groups are translated into new features, extracted in TCOMP variants.

As we indented, we can see that the average pairwise correlation between the features consistently selected in the leave-one-out cross-validation iterations is smaller when using the HC-PVAL variants (**Figure 4-6**). There are two reasons for that. The first is that when selecting features in a univariate fashion the top ranked features tend to highly correlate, as features are ranked individually. The second reason is the PLS mechanism. As the PLS components are orthogonal, the features taken from different components, which are most relevant to the construction of these components, tend to have a lower pairwise correlation.

5.2 Future work

We have shown that SlimPLS based feature selectors yielded improved performance compared to the very common used filters. Specifically, the HC-PVAL variants showed the best performance. Some future work directions in this area include:

a) Improving PVAL methodology, i.e., the dynamic methodology of choosing the number of components and the number of features per component. There are several options here:

    - The p-value threshold to PVAL variants can be calculated from the correlation significance of the first components. The faster the significance drops for

constructed components, the more significant a component will have to be to pass the threshold.

- Using a minimum number of features per component. In other words, the algorithm decides how many features will be taken from each component that passes the p-value threshold. Another parameter *m* can be used so that at least *m* features from each component will be taken, thus enabling the algorithm to 'capture' the component's behavior. If the algorithm chooses only $k<m$ features from a particular component, it then excludes this component and takes another *k* features from the previous one.

b) Improving the local search methodology, i.e., the search for a better subgroup of features. Currently we use hill climbing, but different approaches can be used.

- Using simulated annealing. Simulated annealing [36] lets us explore regions in the search space that may not be explored by hill-climbing, as it can occasionally choose a different subgroup, even if does not yield a better score for the objective function. Thus, it has some mechanism to escape local maxima. Simulated annealing needs a parameter for its run – the 'cooling' parameter, and a good way to determine its value should be found.
- Keep using hill climbing, but stop its run after a prescribed number of runs or after achieving a desired percentage of improvement of the objective function. This may be done to avoid over-fitting.
- Allowing a switch (i.e., moving from one subgroup of features to another by replacing one of the features currently selected with another one that is not)

only if the improvement (absolute or relative) of the target function is higher that some threshold.

c) Further research of the impact of the number of selected features on the overall scores of particular feature selectors and classifiers.

d) Inserting some biology-based logic to the hill climbing search. The greedy search tries to find a switch that improves the target function. A mechanism that does prevent some switches (even if they improve the target function) can be inserted. This way, e.g., one gene can be switched with another only if they belong to the same module in a given biological network. Alternatively, a switch can be allowed only if there are representative genes from at least (or at most) k different modules of the biological network in the resulting subgroup.

In this study we did not compare SlimPLS performance to previous methods using PLS, since those methods mix the feature selection and classification steps. Still, some of the PLS-based classification procedures discussed in **Section 3.5** can be adjusted to operate as feature selectors. For example, the λ parameter from [38] can be set so that only a desired number of features will be selected, and then return this list of features rather than continue with the linear regression classification, or alternatively, more powerful classifiers can be used in that step. Comparing such feature selector to SlimPLS can be interesting, as it does not consider explicitly which feature to select from each component

(as SlimPLS does), but filters out features after constructing a linear regression using all calculated components.

6 Bibliography

1. Saeys Y, Inza I, Larranaga P: A review of feature selection techniques in bioinformatics. *Bioinformatics* 2007, 23(19):2507-2517.
2. Wold H: Soft modeling: the basic design and some extensions. *Systems Under Indirect Observation* 1982, 2:1-53.
3. Wold H: Partial least squares. *"Encyclopedia of the Statistical Sciences"* 1985, 6:581–591.
4. Wold S, Ruhe H, Wold H, Dunn W, J, III: The collinearity problem in linear regression. The partial least squares (PLS) approach to generalized inverse. *SIAM Journal of Scientific and Statistical Computations* 1984, 5:735–743.
5. Tarca AL, Carey VJ, Chen XW, Romero R, Draghici S: Machine learning and its applications to biology. *PLoS Comput Biol* 2007, 3(6):e116.
6. LeCUN: Comparison of learning algorithms for handwritten digit recognition. In: *International Conference on Artificial Neural Networks: 1995; Paris*: EC2 & Cie; 1995: 53-60.
7. Joachims T: Text Categorization with Support Vector Machines: Learning with Many Relevant Features. In: *Machine Learning: ECML-98 10th European Conference on Machine Learning: 1998*; 1998: 137-142.
8. Vapnik V: Statistical Learning Theory. New York: John Wiley and Sons, Inc.; 1998.
9. Vapnik V: The Nature of Statistical Learning Theory. New York: Springer-Verlag; 1995.
10. Vapnik V: Estimation of Dependences Based on Empirical Data, Addendum 1. New York: Springer-Verlag; 1982.
11. Webb A: Statistical pattern recognition, 2 edn: Wiley; 2002.
12. Boser B, E., , Guyon I, M., Vapnik V: A training algorithm for optimal margin classifiers. In: *Fifth Annual Workshop on Computational Learning Theory: 1992; Pittsburgh*: ACM; 1992: 144-152.
13. Aizerman M, A, Braverman E, M, Rozoner L, I Theoretical foundations of the potential function method in pattern recognition learning. *Automation and Remote Control* 1964(25):821-837.
14. Breiman L: Random forest. *Machine Learning* 2001(45):5-32.
15. Quinlan J, R: Induction of decision trees. *Machine Learning* 1986, 1(1):81-106.
16. Mitchell TM: Machine Learning: McGraw-Hill International Editions; 1997.

17. Quinlan J, R. : Rule Induction with statistical data – a comparison with multiple regression. *Journal of the Operational Research Society* 1987(38):347-352.

18. Cover T, Hart P: neighbor pattern Classification. *IEEE Transactions on Information Theory* 1967, 13:21-27.

19. Duda T, Hart P: Pattern Classification and scene analysis. New York: John Wiley & Sons; 1973.

20. Cetnik B: Estimating Probabilities: A crucial task in machine learning. In: *Ninth European Conference on Artificial Intelligence: 1990; London*; 1990: 147-149.

21. van 't Veer LJ, Dai H, van de Vijver MJ, He YD, Hart AA, Mao M, Peterse HL, van der Kooy K, Marton MJ, Witteveen AT *et al*: Gene expression profiling predicts clinical outcome of breast cancer. *Nature* 2002, 415(6871):530-536.

22. Hastie T: The Elements of Statistical Learning: Springer; 2001.

23. Everitt BS, Hothorn T: A Handbook of Statistical Analyses Using R: Chapman & Hall/CRC Taylor & Francis Group; 2006.

24. Golub TR, Slonim DK, Tamayo P, Huard C, Gaasenbeek M, Mesirov JP, Coller H, Loh ML, Downing JR, Caligiuri MA *et al*: Molecular classification of cancer: class discovery and class prediction by gene expression monitoring. *Science* 1999, 286(5439):531-537.

25. Hamming RW: Coding and Information Theory: Prentice-Hall Inc.; 1980.

26. Rosipal R, Kramer N: Overview and recent advances in partial least squares. *Subspace, Latent Structure and Feature Selection* 2006, 3940:34-51.

27. Rosipal R, Kramer N: Overview and Recent Advances in Partial Least Squares. . *In Subspace, Latent Structure and Feature Selection Techniques, Series: Lecture Notes in Computer Science* 2006, 3940:34-52.

28. Wold H: Path models with latent variables: The NIPALS approach. *Quantitative Sociology: International perspectives on mathematical and statistical model building* 1975.

29. Hoskuldsson A: PLS Regression Methods. *Journal of Chemometrics* 1988, 2:211-228.

30. Sampson PD, Streissguth AP, Barr HM, Bookstein FL: Neurobehavioral effects of prenatal alcohol: Part II. Partial least squares analysis. *Neurotoxicol Teratol* 1989, 11(5):477-491.

31. Wegelin J, A: A survey of Partial Least Squares (PLS) methods, with emphasis on the two-block case. In. Seattle: Department of Statistics, University of Washington; 2000.

32. Dejong S: Simpls - an Alternative Approach to Partial Least-Squares Regression. *Chemometrics and Intelligent Laboratory Systems* 1993, 18(3):251-263.

33. Martens H, Naes T: Multivariate Calibration: John Wiley & Sons; 1989.

34. Alon U, Barkai N, Notterman DA, Gish K, Ybarra S, Mack D, Levine AJ: Broad patterns of gene expression revealed by clustering analysis of tumor and normal colon tissues probed by oligonucleotide arrays. *Proc Natl Acad Sci U S A* 1999, 96(12):6745-6750.

35. Hanczar B, Zucker JD, Henegar C, Saitta L: Feature construction from synergic pairs to improve microarray-based classification. *Bioinformatics* 2007, 23(21):2866-2872.

36. Russell SJ, Norvig P: Artificial Intelligence: a modern approach, 2 edn: Prentice Hall; 2003.

37. Nguyen DV, Rocke DM: Tumor classification by partial least squares using microarray gene expression data. *Bioinformatics* 2002, 18(1):39-50.

38. Huang X, Pan W: Linear regression and two-class classification with gene expression data. *Bioinformatics* 2003, 19(16):2072-2078.

39. Huang X, Pan W, Han X, Chen Y, Miller LW, Hall J: Borrowing information from relevant microarray studies for sample classification using weighted partial least squares. *Computational biology and chemistry* 2005, 29(3):204-211.

40. Ding B, Gentleman R: Classification Using Generalized Partial Least Squares. In: *Bioconductor Project.* 2004.

41. Fort G, Lambert-Lacroix S: Classification using partial least squares with penalized logistic regression. *Bioinformatics* 2005, 21(7):1104-1111.

42. Barker M, Rayens W: Partial least squares for discrimination. *journal of chemometrics* 2003, 17:166-173.

43. Rosipal R, Trejo L: Kernel Partial Least Squares Regression in Reproducing Kernel Hillbert Spaces. *journal of Machine Learning Research* 2001, 2:97-128.

44. Momma M, Kristin P, Bennet: Sparse Kernel Partial Least Squares Regression.

45. Rosipol R, Trejo LJ, Matthews B: Kernel PLS-SVC for Linear and Nonlinear Classification. In: *Twentieth International Conference on Machine Learning: 2003; Washington DC*; 2003.

46. The R Project for Statistical Computing [http://www.r-project.org/]

47. The Comprehensive R Archive Network / Packages [http://cran.r-project.org/]

48. Venables WN, Ripley BD: Modern Applied Statistics with S, 4 edn: Springer; 2002.

49. Random Forests [http://stat-www.berkeley.edu/users/breiman/RandomForests/]

50. Borovecki F, Lovrecic L, Zhou J, Jeong H, Then F, Rosas HD, Hersch SM, Hogarth P, Bouzou B, Jensen RV *et al*: Genome-wide expression profiling of human blood reveals biomarkers for Huntington's disease. *Proc Natl Acad Sci U S A* 2005, 102(31):11023-11028.

51. Hodges A, Strand AD, Aragaki AK, Kuhn A, Sengstag T, Hughes G, Elliston LA, Hartog C, Goldstein DR, Thu D *et al*: Regional and cellular gene expression changes in human Huntington's disease brain. *Hum Mol Genet* 2006, 15(6):965-977.

52. Singh D, Febbo PG, Ross K, Jackson DG, Manola J, Ladd C, Tamayo P, Renshaw AA, D'Amico AV, Richie JP *et al*: Gene expression correlates of clinical prostate cancer behavior. *Cancer Cell* 2002, 1(2):203-209.

53. Burczynski ME, Peterson RL, Twine NC, Zuberek KA, Brodeur BJ, Casciotti L, Maganti V, Reddy PS, Strahs A, Immermann F *et al*: Molecular classification of Crohn's disease and ulcerative colitis patients using transcriptional profiles in peripheral blood mononuclear cells. *J Mol Diagn* 2006, 8(1):51-61.

54. Chin K, DeVries S, Fridlyand J, Spellman PT, Roydasgupta R, Kuo WL, Lapuk A, Neve RM, Qian Z, Ryder T *et al*: Genomic and transcriptional aberrations linked to breast cancer pathophysiologies. *Cancer Cell* 2006, 10(6):529-541.

55. Okada T, Iizuka N, Yamada-Okabe H, Mori N, Tamesa T, Takemoto N, Tangoku A, Hamada K, Nakayama H, Miyamoto T *et al*: Gene expression profile linked to p53 status in hepatitis C virus-related hepatocellular carcinoma. *FEBS Lett* 2003, 555(3):583-590.

56. Chowdary D, Lathrop J, Skelton J, Curtin K, Briggs T, Zhang Y, Yu J, Wang Y, Mazumder A: Prognostic gene expression signatures can be measured in tissues collected in RNAlater preservative. *J Mol Diagn* 2006, 8(1):31-39.

57. Beer DG, Kardia SL, Huang CC, Giordano TJ, Levin AM, Misek DE, Lin L, Chen G, Gharib TG, Thomas DG *et al*: Gene-expression profiles predict survival of patients with lung adenocarcinoma. *Nat Med* 2002, 8(8):816-824.

58. Iizuka N, Oka M, Yamada-Okabe H, Nishida M, Maeda Y, Mori N, Takao T, Tamesa T, Tangoku A, Tabuchi H *et al*: Oligonucleotide microarray for prediction of early intrahepatic recurrence of hepatocellular carcinoma after curative resection. *Lancet* 2003, 361(9361):923-929.

59. Dhanasekaran SM, Barrette TR, Ghosh D, Shah R, Varambally S, Kurachi K, Pienta KJ, Rubin MA, Chinnaiyan AM: Delineation of prognostic biomarkers in prostate cancer. *Nature* 2001, 412(6849):822-826.

60. Gruvberger S, Ringner M, Chen Y, Panavally S, Saal LH, Borg A, Ferno M, Peterson C, Meltzer PS: Estrogen receptor status in breast cancer is associated with remarkably distinct gene expression patterns. *Cancer Res* 2001, 61(16):5979-5984.

61. West M, Blanchette C, Dressman H, Huang E, Ishida S, Spang R, Zuzan H, Olson JA, Jr., Marks JR, Nevins JR: Predicting the clinical status of human breast cancer by using gene expression profiles. *Proc Natl Acad Sci U S A* 2001, 98(20):11462-11467.

62. Berchuck A, Iversen ES, Lancaster JM, Pittman J, Luo J, Lee P, Murphy S, Dressman HK, Febbo PG, West M *et al*: Patterns of gene expression that characterize long-term survival in advanced stage serous ovarian cancers. *Clin Cancer Res* 2005, 11(10):3686-3696.

63. Zapala MA, Hovatta I, Ellison JA, Wodicka L, Del Rio JA, Tennant R, Tynan W, Broide RS, Helton R, Stoveken BS *et al*: Adult mouse brain gene expression patterns bear an embryologic imprint. *Proc Natl Acad Sci U S A* 2005, 102(29):10357-10362.

64. Tian E, Zhan F, Walker R, Rasmussen E, Ma Y, Barlogie B, Shaughnessy JD, Jr.: The role of the Wnt-signaling antagonist DKK1 in the development of osteolytic lesions in multiple myeloma. *N Engl J Med* 2003, 349(26):2483-2494.

65. Song L, Bedo J, Borgwardt KM, Gretton A, Smola A: Gene selection via the BAHSIC family of algorithms. *Bioinformatics* 2007, 23(13):i490-498.

66. Fisher RA: Combining independent tests of significance. *American Statistician* 1948, 2(5).

7 Appendix

7.1 Two-dimensional comparison of methods

In order to compare and evaluate all the different methods, we constructed a two-dimensional plot. Each method, combined of classifier i and feature selector j, is represented by a point (x, y), where x is the average error rate of the method and y is its average ranking. Formally, $x = \frac{1}{19}\sum_k E(i, j, k)$ and $y = \frac{1}{19}\sum_k R(i, j, k)$. E and R are hree dimensional matrices as defined in **Section 4.2(f)**. Again, we use only the 50 feature configurations for this comparison.

We present the plot in two ways. In the first way, each point (which represents a method) is colored according to the classifier. In the second way, each point is colored according to the feature selector. The results are summarized in **Figure 7-1** and **Figure 7-2**, respectively.

Most points are aligned along a straight line, indicating a high correlation between the rank and the error rate of the methods. The methods that do best in both criteria are on the 'north-west' corner of that line.. The best two methods in terms of both criteria are KNN based, using HC-PVAL based feature selectors (The corresponding feature selectors can be seen in **Figure 7-2**). **Figure 7-1** also shows that, overall, the Naïve Bayes classifier based methods tend to have poorer performance than others (consistent with **Figure 4-2**)

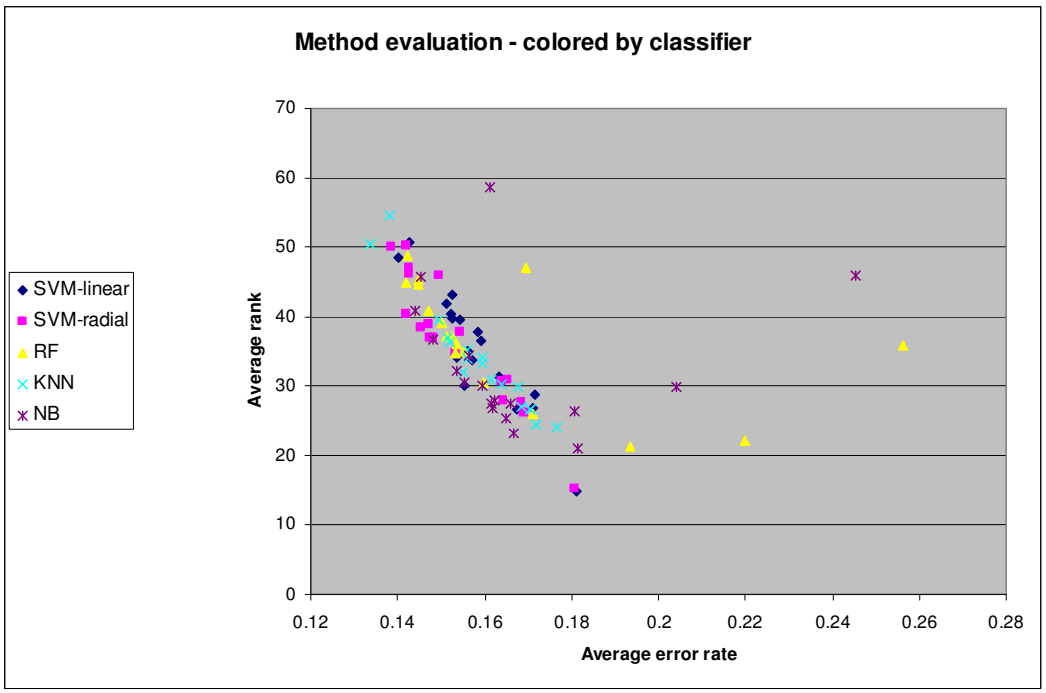

Figure 7-1. Two-dimensional evaluation of methods, where points colored by according to the classifier. Higher rank value means better performance.

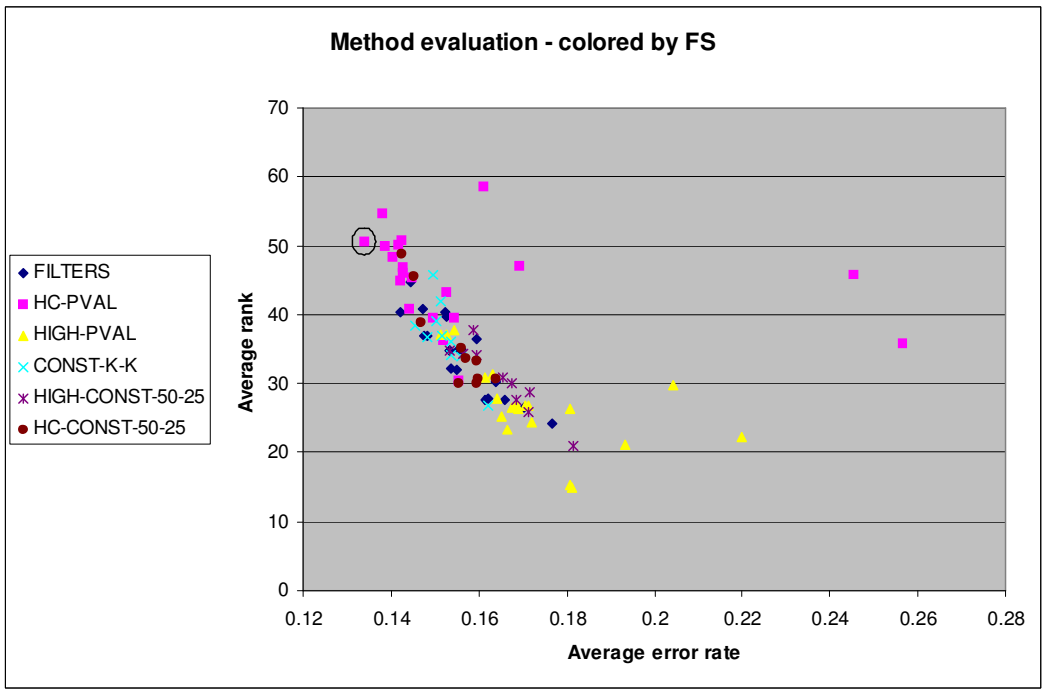

Figure 7-2. Two-dimensional evaluation of methods, where points colored by according to the feature selector. Higher rank value means better performance. The circled point corresponds to the KNN classifier and the HC-TCOMP-5e-03. This combination had the lowest average error rate.

Some points do not follow the straight line of most points. These points are methods that use Random Forest or Naïve Bayes classifier. This is due to the following reason. The versions of RF and NB that we used (from the R package) could not classify samples using only one feature. This is the case when using the PVAL-TCOMP variant, where only one component is chosen and returned as a new single feature. This case often occurs when the dataset is easy to classify. Therefore, the results from that datasets cannot be taken, and the average error rates for the Random Forest and the Naïve Bayes classifier are calculated using only the 'harder' datasets (for the PVAL-TCOMP variants). Hence, the average error rates in these cases compared to others are biased towards higher values. This, however, has little effect on the average rank.

Figure 7-2 shows a clear advantage to the HC-PVAL based methods, while the HIGH-PVAL based methods tend to have the poorest performance. While the HIGH-CONST variants select constant number of features from each component, the HIGH-PVAL variants dynamically calculate the number of features that are selected from each component. This has the effect of usually choosing more features from the first component and fewer features from the other components. Therefore, the 'structure' of the latter components is poorly expressed, and an improved set of features from these components ought to be found to have a better description of the components. This is done by the local search, which also improves the features set taken from the first component.

7.2 Further analysis of KNN and SVM-radial results

The noticeable difference in performance between the HC-PVAL variants and the filters using the KNN classifier (**Figure 4-2**) is shown also in **Figure 7-3** and **Figure 7-4**, where we compare the L2 distance and average rank, respectively, of the various feature selectors when using the KNN classifier.

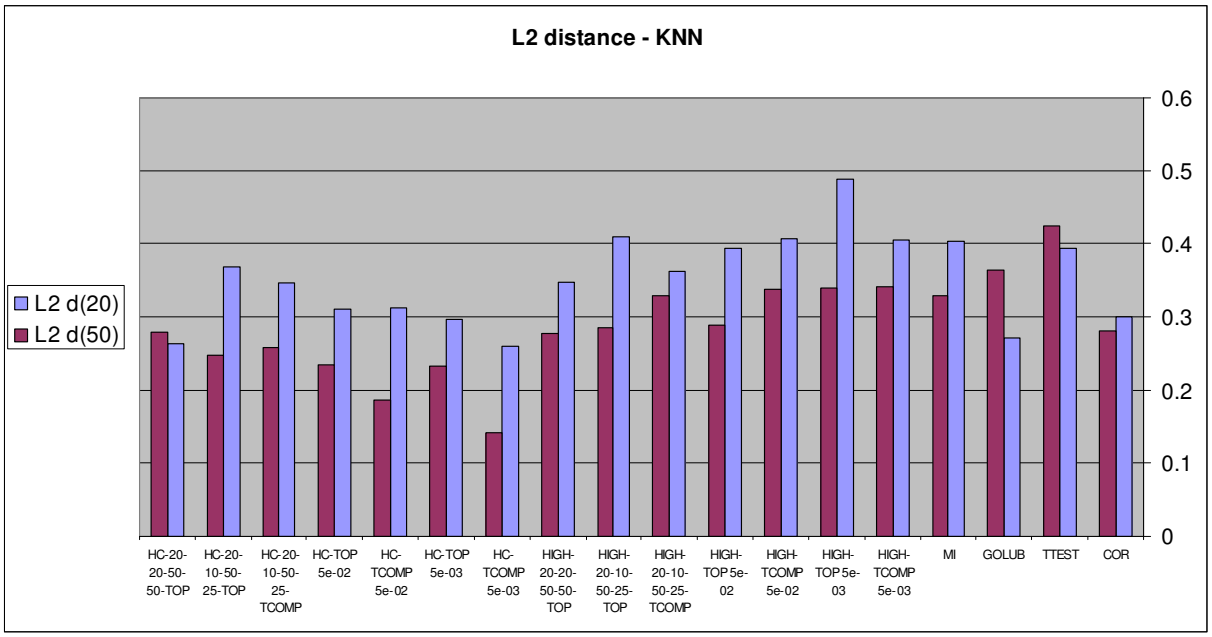

Figure 7-3. L2 distance score of the different feature selectors using KNN classifier.

We can see in **Figure 7-3** the relatively low L2 distance scores of HC-PVAL variants compared to other methods. Specifically, the TCOMP variants of HC-PVAL, i.e., HC-TCOMP-5e-03 and HC-TCOMP-5e-02 attained the best score and second best score, respectively. We can also see that CONST-50-50 variants (HC-50-50-TOP and HIGH-50-50-TOP) perform better than the filter variants, except for the correlation filter that shows comparable results.

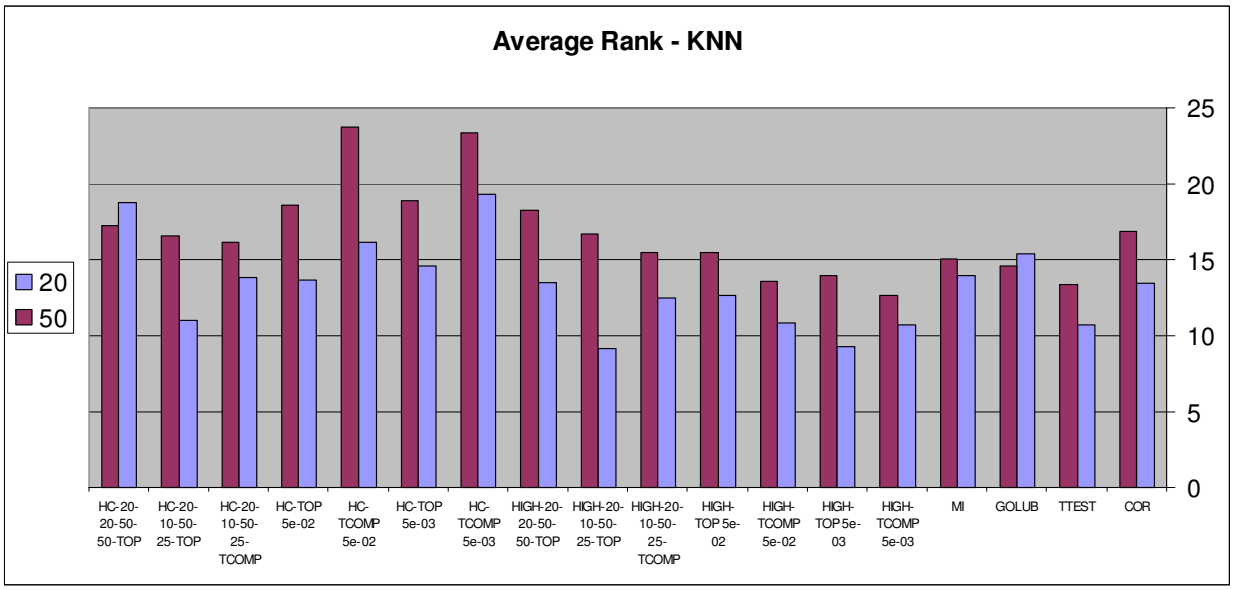

Figure 7-4. The average rank score of the different methods using KNN classifier.

Figure 7-4 shows the average rank results on the same combinations. Again, we see a noticeable advantage in favor of the two 50-HC-TCOMP variants. Moreover, these two variants, together with the KNN classifier, have the lowest average error-rates among all feature selectors and classifier combinations (not shown). High scores were also attained when using the HC-TOP and HC/HIGH-50-50-TOP variants.

Figure 7-3 and **Figure 7-4** also show again that the performance of the methods is in most cases better when selecting a total of 50 features rather than 20 features.

The SVM classifiers, SVM-radial and SVM-linear, showed high performance as well on the 19 datasets (see **Figure 4-2**). The summarized average rank scores of the different feature selectors using SVM-radial can be viewed in **Figure 7-5**. The HC-PVAL variants, as well as the HC-K-K variant, show best performance. Similarly to **Figure 7-4** one can notice a consistent drop in scores when selecting 20 features in total, instead of 50.

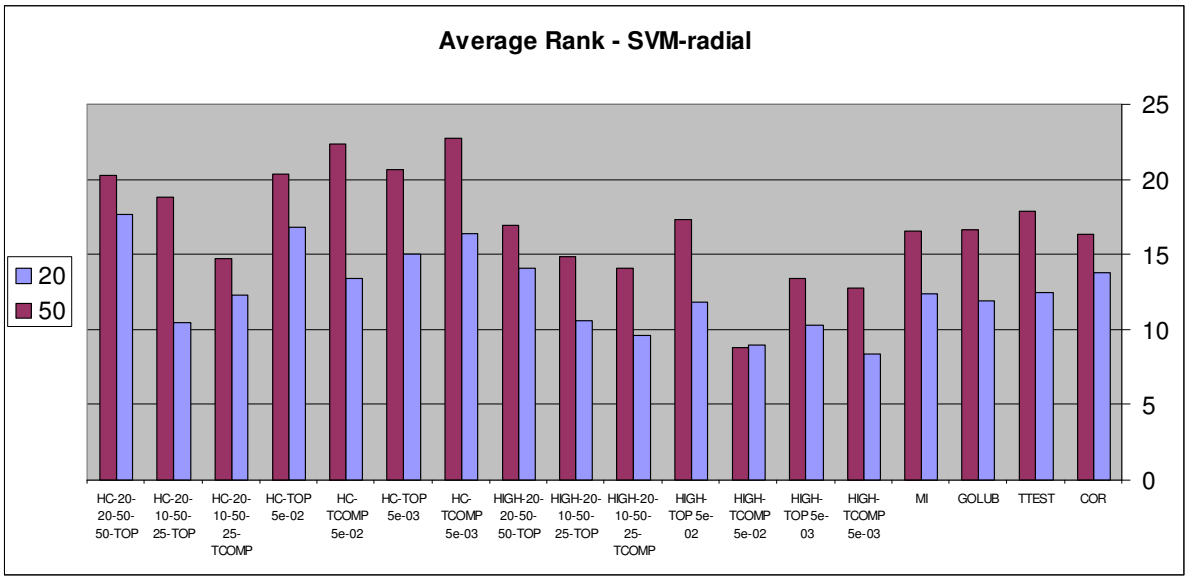

Figure 7-5. The average ranking score of the different methods using SVM-radial classifier

Notice in **Figure 7-4** and **Figure 7-5** the resemblance of HIGH-K-K variant scores to the correlation filter scores (as was seen in **Figure 4-6**).

7.3 Rates of exceptional results

Figure 7-6 and **Figure 7-7** provide an overall look on the performance of the different feature selectors using all datasets and all classifiers. **Figure 7-6** summarizes the 95% confidence rates (see **Section 4.2** for definitions). Similar to **Figure 4-2**, we can notice that HC-PVAL variants have a clear advantage over other methods. These variants obtain lower than average error rate more often than the other methods.

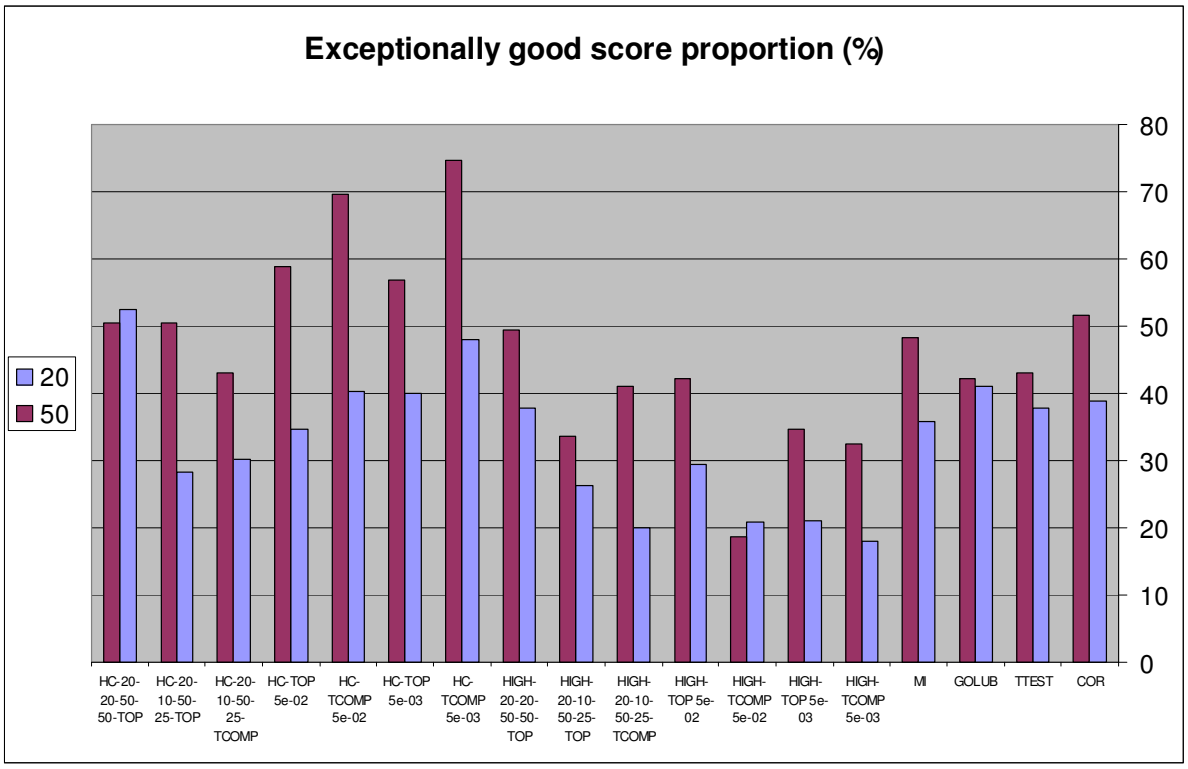

Figure 7-6. 95% confidence interval rates of each feature selectors. The rate-score of a method to perform better than the 95% confidence interval of the average using all classifiers and datasets.

The same behavior is observed in **Figure 7-7**, where the frequency of performing exceptionally bad is shown. The HC-PVAL methods have only about 10% or less error-rate scores that are significantly worse than the average score. We can also see in **Figure 7-6** and **Figure 7-7** that performance drops when selecting only 20 features.

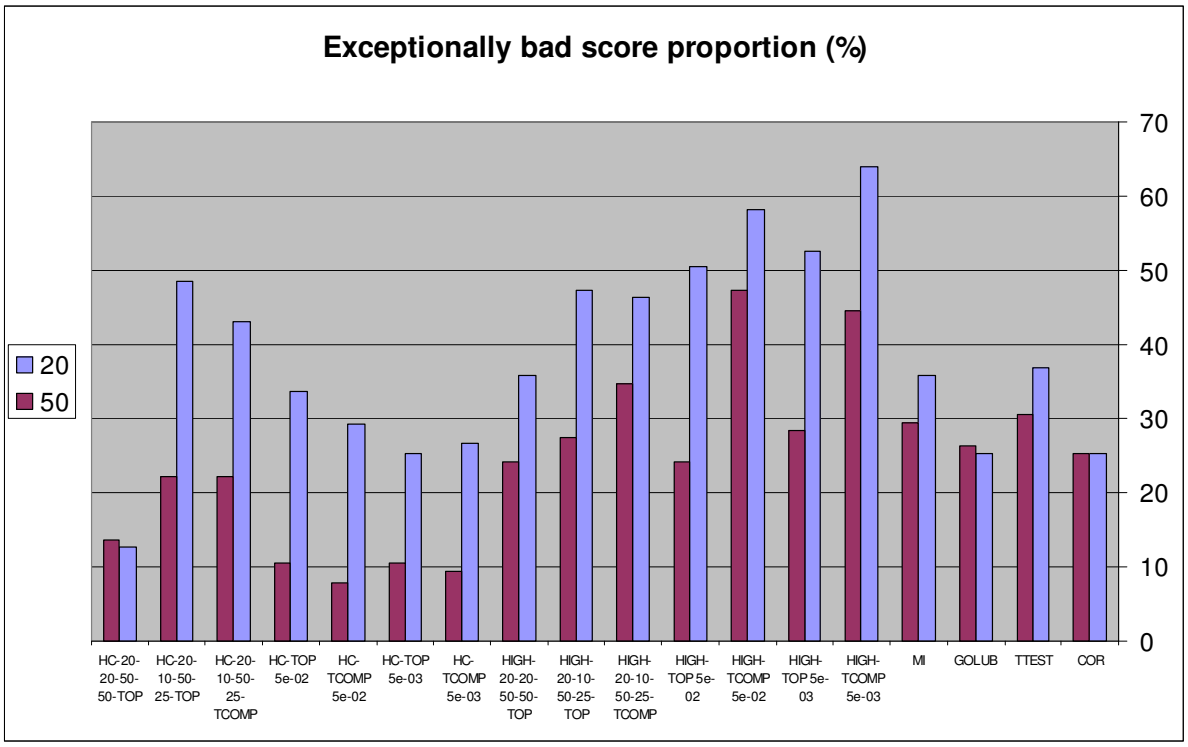

Figure 7-7. Exceptianlly bad score. The rate-score of a method to perform worse than the 95% confidence interval of the average using all classifiers and datasets.

The relatively better performance of the HC-PVAL variants is also reflected in their rate of attaining the best score. **Figure 7-8** compares these scores among the different feature selectors. A concentration of relatively high scores is shown in the HC-PVAL variants. Among them, HC-TCOMP-5e-02 has the highest rate, and HC-TOP-5e-02 has the second highest rate. Among the 20 feature configurations the mutual information (MI) filter has the highest rate. This filter and GOLUB filter have higher proportion of best score achieved when using 20 features.

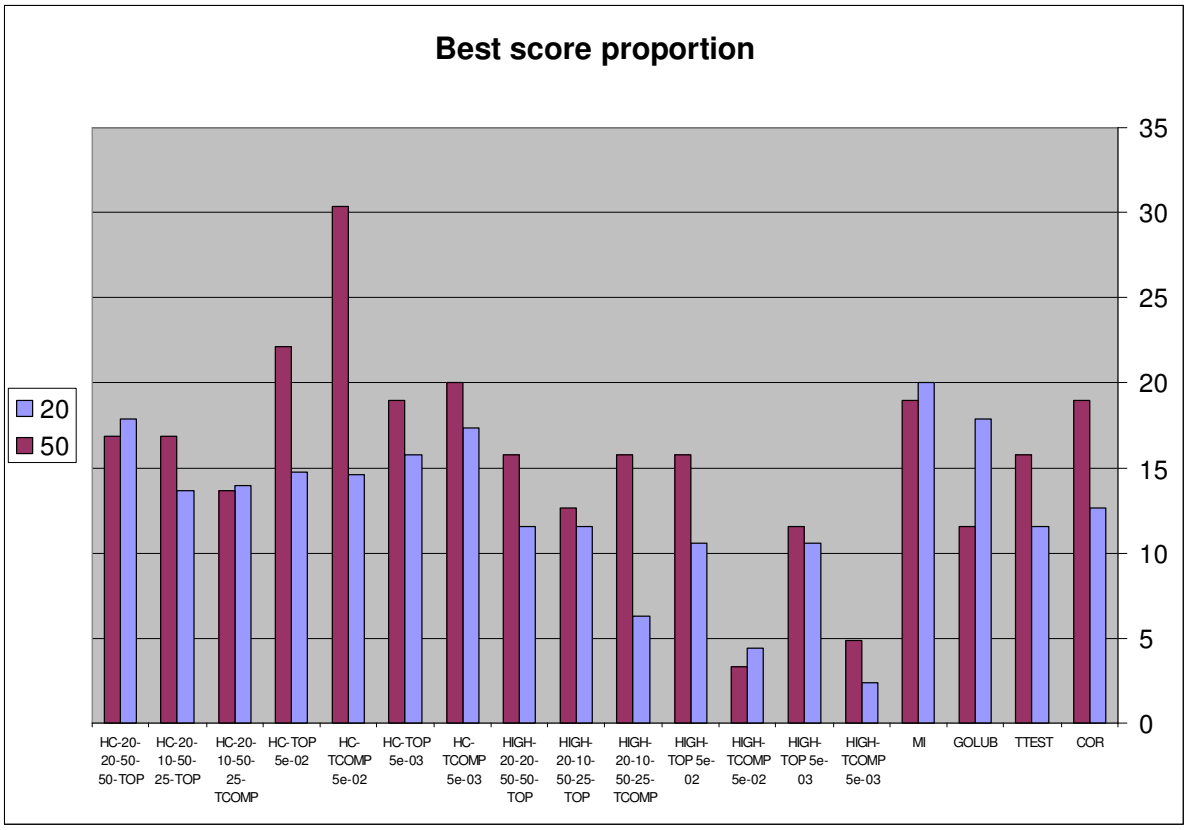

Figure 7-8. The percentage of best score achieved using all classifiers and datasets.