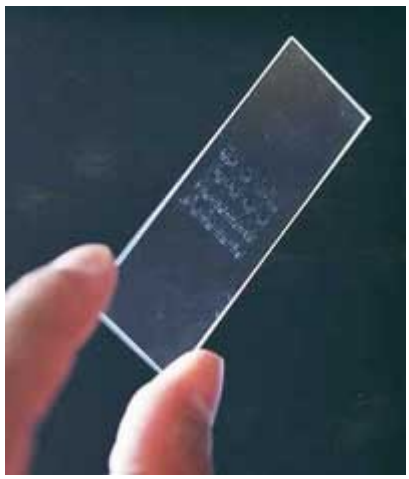
CHAPTER 1

Introduction to microarray technology



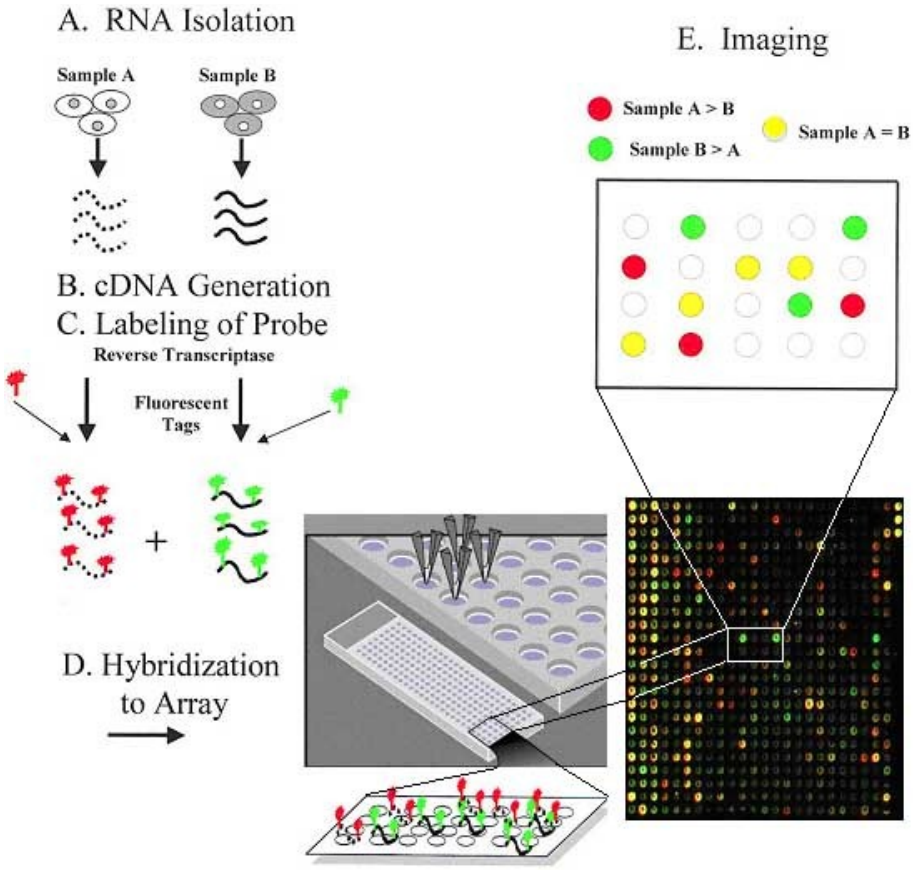
1. Introduction

Completion of the Human Genome Project has opened a new era in studies of functions of cells and organisms. Identification of the thousands of genes forming genomes brings us to the next frontier: elucidation of the functions of these genes and their interactions to "functional genomics" An experimental tool that allows surveying expression of the genetic information on a genome-wide scale at the level of single genes has been developed just a few years ago thanks to the microarray technology. Global gene expression profiling using microarrays is emerging as key technology for understanding fundamental biology of gene function development and for discovering new classes of diseases such as cancer and for understanding their molecular pharmacology. Numerous unsupervised and supervised learning methods have been applied to the task of discovering and learning to recognize classes of co-expressed genes. Multiple open source software is available for evaluating the significance of the information given by microarrays. Genes databases and microarrays datasets are free accessed by the scientists and universities who want to test ,to improve or to produce new algorithms and methods for better utilization of this big amount of raw information.

1.1Microarray technology

Microarray or microchip is a chip made of glass or other solid material, with an array of tiny DNA spots placed on it. Each spot contains fragments of DNA or RNA molecule whose sequence is predefined and corresponds to portions of a particular gene. The lengths of these fragments may vary from about 20 nucleotides in oligonucleotide microarrays to thousands of nucleotides in genome microarrays. Typical microarray contains several thousand spots on the surface of a quarter square inch, and a library of thousands of genes is placed on a single chip. To probe the global gene expression levels of many genes in biological samples such as cell lines, tissue extracts, or laser micro dissected cells, messenger RNAs are first extracted. mRNAs are then reverse-transcribed into cDNA. The amount of cDNA produced is then amplified by polymerase chain reactions (PCR), a standard molecular biology technique. Under proper experimental conditions *the concentrations of the cDNA for a specific gene reflect the amount of expressed mRNA of this gene in the probe*

sample (pic 1). The expression levels of individual genes present on the chip can then measured quantitatively by laser scanning (fluorescent probes) or by phosphoimaging (radio labelled probes) spots at different predefined locations on the chip. In another experiment scheme, the extracts from the sample tissue and a control tissue are marked with different dyes, and are hybridized simultaneously on the same chip. This approach provides information about the relative concentration of expressed RNA in sample and control tissues. As probe samples collected at different well-designed experimental conditions are applied, the relative expression levels of all genes on the chip can then be analyzed for changes in the expression patterns to obtain an integrated global picture about the underlying genetic networks [1].



Pic.1

1.1.1 DNA and oligonucleotide gene expression microarray platforms: principals and differences

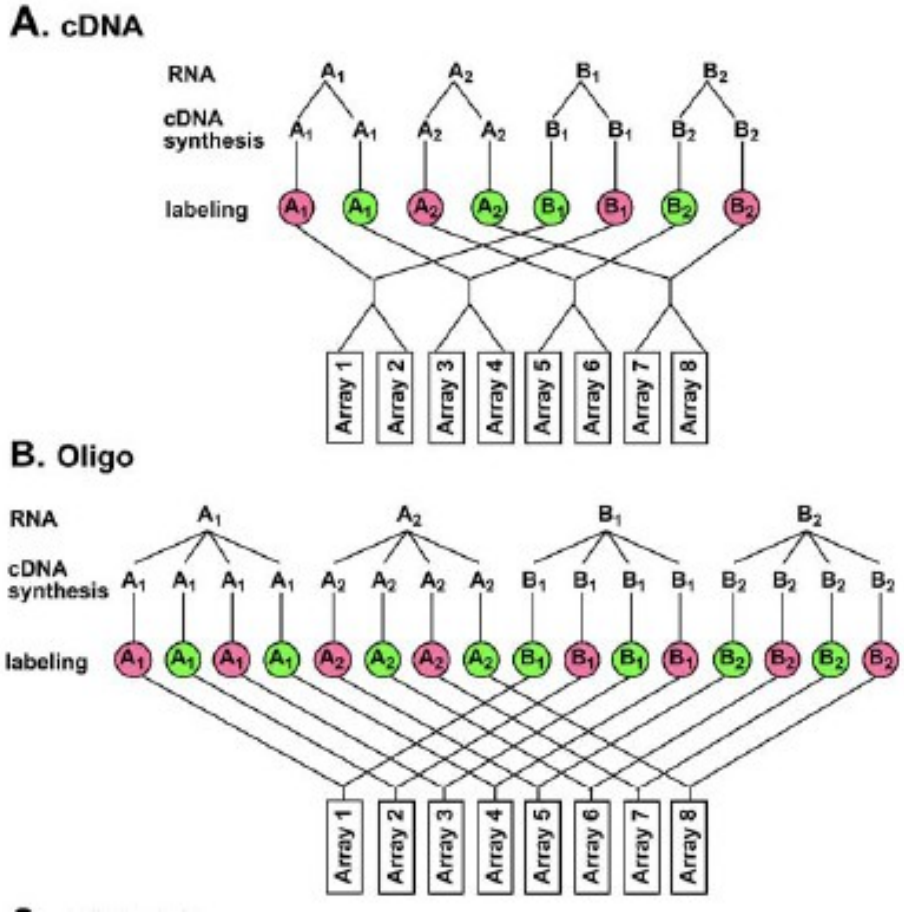
Since the inception of DNA microarrays, technological advances have lead to the development of two main types of arrays, namely clone-based and oligonucleotide based. Both expression systems, each with different experimental designs, are routinely used to compile comparative global mRNA expression profiles of tissues or cell lines.

cDNA microarrays (pic 2) are microscopic array which contain large sets of DNA sequences immobilized on a solid substrate. In a array experiment, many specific cDNAs are spotted on a single matrix. The matrix is then simultaneously probed with fluorescently tagged cDNA representations of total RNA pools from test and reference cells, allowing one to determine the relative amount of transcript present in the pool by the type of fluorescent signal generated. Current technology can generate arrays with over 10. 000 cDNA pre square centimetre. For cDNA arrays, each value is the logarithm of the ratios of the estimated abundances of mRNA in two tissue samples. [2].

Oligonucleotide microarrays (pic 2) each spot on the array contains a sort synthetic ilogonucleotide. The oligos are designed based on the knowledge of the DNA (or EST) target sequences to ensure high-affinity and specificity of each oligo to a particular target gene. The advantage of this type or microarrays is that they improve signal –to-noise ratio and accuracy of RNA quantization and reduce the rate of false positives and miscalls [2].

The above microarray platforms have two major differences. The cDNA microarray is characterized as fluorescent labelled platform and the oligonucloetide microarray as radioactively labelled platform. Comparisons and test taken on these two technologies have indicated no clear consensus as to the comparability of expression profiles from different platforms, using either ratio-transformed or single channel expression data. Additionally studies have led to the conclusion that oligonucleotide- based arrays, such as those produced by Affymetrix, and full-length clone-based arrays may be too different in experimental design to be expected to give global expression results that can be directly correlated. This suggests that microarray technologies should not be used as an absolute quantisation method and that pooling

of global expression profiles from different microarray platforms for the purposes of large-scale data mining should be undertaken with caution. [3].

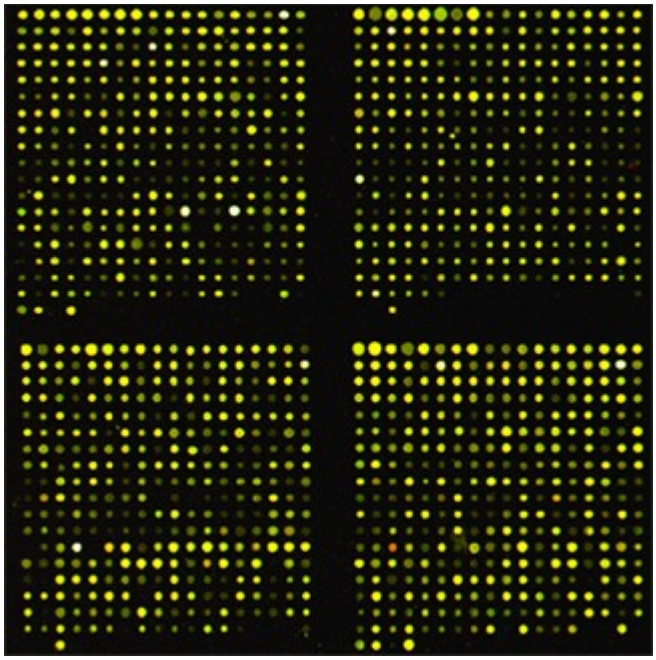


Pic. 2

1.2The importance of clinical data

A big question that arises from the usage of this technology is whether the laboratory samples and the methods involved to create the microarray are capable of giving the right picture of the pathological status of the patient and the expression profiles of the genes. In other words is there a systematic and analytic approach that gives trustworthy results? These questions arise from the tension between the relative ease of producing the results and the objective difficulty of dealing with the results. Additionally, the diversity of experimental designs and schemes adds to the confusion

[4]. The size of experimental groups and the design of the experiments also vary widely, from timecourse experiments to cross-sectional studies, from single observations with no repeats to analysis of hundreds of samples. Naturally, these diverse experimental schemes pose diverse computational requirements—the analysis of an experiment designed discover a new class of a disease is different from an experiment designed to test the immediate targets of a known transcriptional activator. The wealth and complexity of information that characterizes results of microarray experiments has led to the suggestion that there may not be a single “best” analytic approach and that indeed the application of several analytical and computational approaches to a dataset may aid in the exposure of different and complementary aspects of the data [3]. Additionally the collaboration between clinical and computer scientists and the usage of multiple tools are actions that will make possible the accurate analysis of the microarray data.



Pic. 3

1.3Methods

Microarray studies often generate massive amounts of data, which are difficult to be exhaustively examined by hand (arrays of high dimensionality with 24. 000 genes each)(pic 2). Bioinformatics analysis and interpretation to extract genetic patterns from these data are therefore essential for gaining biological insights from experiments. If we would like to group the methods that are being used for the exploration of the information that it is hidden beneath the huge amount of data of the microarrays, we would end up with 4 extensive categories:

- Genes selection
- Clustering (supervised or unsupervised)
- Classification
- Marker genes or feature selection

In first category we are trying to identify genes that experience significant changes in expression under different experimental conditions. For example specific genes are expressed to a healthy tissue and different genes are expressed in a tumour tissue. Permutation T-tests, neighbourhood analysis [5], p-values, correlation matrices, Sebestyen criterion [6] are methods used for identifying genes which are differentially expressed under different conditions and selecting them for further experimental validation [7].

With *clustering* we search for groups of genes that are likely to be co-regulated or participating in related metabolic and regulatory pathways. Algorithms like self organizing maps (SOMs) , hierarchical clustering , k-means clustering , bootstraps resampling methods , principal component analysis (PCA) [5] are widely used for the discovery of underlying structures in a dataset. A common example is the discovery of the two phenotypes of leukaemia AML and ALL with the combination of microarray's technology and the above clustering methods.

Classification involves predicting and classifying experimental samples whether they belong to a particular type of tissue, disease or phenotype classes. Particularly the supervised classifiers seek to function $f : \mathbb{R}^n \rightarrow \{1,....P\}$ that maps the unknown

sample i to one of the P classes(assuming that each sample belongs to strictly one of the P phenotypes) according to the global expression profile x_i [8].

Lastly the identification of candidate marker genes or marker genes clusters indicative of specific phenotypes is a task very important and significant for the development of new drugs and therapies. Furthermore the selection of a specific number of features is important because some methods of supervised learning perform inaccurately and/or slowly when asked to consider a large number of them. But even when using classification algorithms that are good at handling many features (such as the SVM method), there is a more practical concern. The method of typing cancers must be reliable, inexpensive, rapid, and easily performed for it to be employed by medical diagnostic laboratories. At the moment, these criteria exclude expression arrays with thousands of genes, and feature selection is required to reduce the feature set to a manageable number of genes [7].

1.4Problems dealing with microarrays datasets

Although the advantages and possibilities of this technique in the future are numerous, there are obviously associated drawbacks. The main disadvantages of this technique include the following:

The actual size of the array may be a problem, when the sample is being prepared. This is due to the DNA molecule itself. When working on the kind of scale necessary for microarrays - micrometers - DNA can actually be very difficult to handle. It has been described as acting 'almost like concrete coated with superglue' [9].To reduce these effects, a number of solutions are being developed: the electrical polarization of the array spots, developed by Nanogen, and the use of 'nonwettable' blank regions on arrays, created by microscopic surface tension barriers, by Protogene.

Another critical issue is the technical limitations The microarray technique is currently limited not by the number of probes on an array, but by the resolution of the scanner used. A solution to this problem is the **GeneArray Scanner**, developed by a partnership between Affymetrix and Hewlett-Packard. This scanner has a greater resolution of 3mm, and also has the capacity for a minimum of 400,000 probes. The

use of this scanner therefore has the potential to resequence 100,000 bases at one time.

Another problem associated with cDNA chips is the quantization of expression. Quantization of expression is affected by a non-linear relationship between the amount of probe present, and the strength of the signal that is generated. The problem that is being caused is an underestimation of the extent of any large changes that may have occurred in the gene expression. This particularly occurs for mRNA molecules that are present in large numbers in the cell.

One of the major problems involved with the technique of DNA micro arrays is concerned with the amount of data that is produced. In fact the development of methods that efficiently organize, distribute and of course interpret this data could be said to be the most formidable task for researchers. Large-scale, high-throughput experimental methods such as this require a great deal of information processing and data analysis. Currently in development is a software package known as LIMS - Laboratory Information Management Systems. This software and the databases associated with it involves: design microarrays track clones collect, analyze and interpret data from gene expression studies.

Another last issue someone should address when referring to microarrays methods is the problem of high dimensionality or else “the curse of dimensionality” [10]. If we consider the genes as the variables then we would have large n ($n>5000$) where n indicates genes and small m ($m<100$) where m indicates the cases. In order to overcome the *large n small m problem* a lot of methods have been proposed and implied. Principal component analysis[5],partial least squares(PLS),hybrid univariate/multivariate/conditionally univariate selections,Sebestyen criterion [6] are processes able to reduce the number of variables (genes) in order to decrease the computational cost and boost the accuracy of other methods such as classification.

1.5Breast cancer microarrays datasets

Our understanding of signal transduction components and their interactions regulating growth and arrest of breast cancer cells is still limited. Microarray technologies provide a powerful method to explore the complexities of transcriptional

profiles defined by selected pharmacological mitogens and inhibitors pivotal for breast cancer cell growth. This highlights the question of how genes contributing to the tumour subclassification are associated with a particular hormone or growth factor signal arrest. Human breast tumour cell lines have been used extensively as models of neoplastic disease, and accordingly, their expression profiles provide a frame of reference for assessing the biological significance of expression patterns in a specific tumour [11,*cunliffe et al* 18]. There have been several studies of gene expression analysis of breast cancer cells treated with a limited number of growth agonists and antagonists, producing catalogues of responsive genes. Among the distinctions made to date, the strongest separation is observed between ER(estrogens receptors)+ and ER- tumours but also between lymph node(-) and lymph node (+).

Additionally we should mention the existence of many breast cancer datasets and databases which can be accessed by anyone who wish to deal with this new promising area of biology and mathematics. Hedenfalk et al. , Perou et al. , Van Veer et al. are recent studies which are public-accessed and involve the implementation of many statistical and mathematical principals and the combination of biological and algorithmic knowledge in order to reveal important information about breast cancer genes. In the end of this work there is an index with the public accessed microarray databases. Most of the above works, which demand the co-operation of doctors and computer engineers, have extensively applied many classification and clustering methods on clinical data. Another important aspect, which has not been yet evaluated, is the combination of the above works, such as the search for common marker genes, which will help to establish a common way in exploring and treating this huge amount of information obtained from microarray technology.

1.5.1 Van Veer's breast cancer dataset

In Van Veer work [11], 98 primary breast cancers have been gathered: 34 from patients who developed distant metastases within 5 years, 44 from patients who continued to be disease free after a period of at least 5 years, 18 from patients with BRCA1 germline mutations and 2 from BRCA carriers. All patients were lymph node

negative. From each patient 5μg total RNA was isolated from snap-frozen tumour material and used to derive complementary RNA (cRNA). A reference pooling was made by pooling equal amounts of cRNA from each of the sporadic carcinomas. Two hybridizations were carried out for each tumor using a fluorescent dye reversal technique on microarrays containing approximately 25. 000 human genes synthesized by inkjet technology.

An unsupervised hierarchical clustering algorithm allowed to cluster the 98 tumors on the basis of their similarities measured all over 5. 000 significant genes. Similarly the 5. 000 genes were clustered on the basis of their similarities measured over these of 98 tumors (pic 4). In the dendrogram in figure 4 the length and the subdivision of the branches displays the relatedness of the breast tumors (left) and the expression of the genes (top). Two distinct groups of the tumors are the dominant feature in this two-dimensional display suggesting that the tumors can be divided into two types on the basis of this set of ~5000 genes. But still the dimension of the dataset is too high. It is necessary to be reduced in order to apply classification algorithms. So a powerful two-step supervised classification method applied on the dataset in order to reduce its dimensionality. The result was to reduce the number of genes to 231 from 5000 in the first step. In the second step these 231 genes were ranked ordered on the basis of the magnitude of the correlation coefficient. In the third step the number of genes in the prognosis classifier was optimized by sequentially adding subsets of 5 genes from the top of this rank-ordered list and evaluating its power for correct classification using the "leave-one out method" for cross validation. Classification was made on the basis of the correlations of the expression profile of the "leave-one out" sample with the mean expression levels of the remaining samples from the good and the poor prognosis patients. The accuracy improved until the optimal number of marker genes was reached (67 genes). Until now the dimensionality of the problem have been reduced from 25. 000 to 67. !!Our aim is to evaluate the accuracy of different algorithms and neural networks when they are applied as classifiers on this 67-dimensional dataset (pic 4).

1.5.2 West's breast cancer dataset

In this work [12] the collection of samples includes mostly Stage II cancers and above. All cancer samples have the same histology. The tumour samples were chosen to include roughly an equal representation of hormone receptor-positive versus hormone receptor-negative cancers. All tissues were screened for tumour content, and cases that contained less than 60% tumour cells were excluded. For the creation of the microarrays Affymetrix Human Gene GENECHIP DNA arrays were used.

The initial 49 tumours were classified as ER- or ER+ via immunohistochemistry (IHC) at time of diagnosis and then later via protein immunoblotting assay for ER to check the IHC results. In five cases, the IHC and blot test conflicted. These five cases and an additional four of the tumours selected randomly were separated from the rest to be treated as validation samples to be predicted on the basis of analysis of the remaining training cases. Of the latter, two were rejected due to failed array hybridization, leaving 18 ER- and 20 ER+. By using the ER outcomes of only the 38 training arrays, a simple screen was implemented to identify the 100 genes maximally correlated with outcome. This screen computed sample correlation coefficients between genes and ER-/ER+ binary outcomes and selected those genes giving the 100 largest absolute values of this correlation. With these 100 "best" genes (fig. 2) we will examine our classification algorithms in order to analyse their accuracy on predicting the class of unknown tumour tissues.

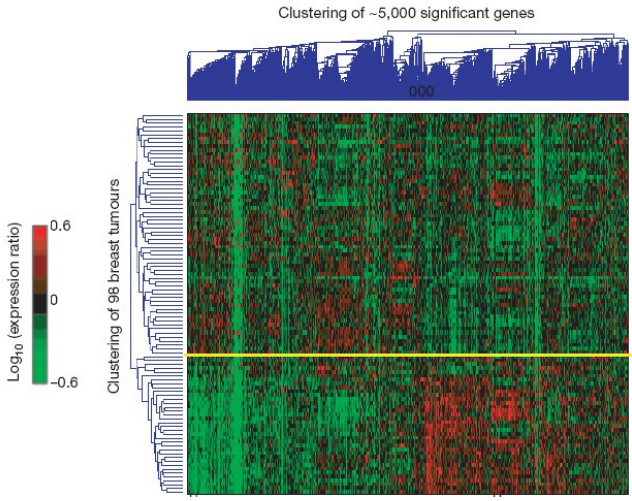


Figure 4 Two-dimensional presentation of transcript ratios for 98 breast tumors. Each row represents a tumour and each column a single gene. As shown in the colour bar red indicates upregulation, green downregulation black no change and grey no data available. The yellow line marks the subdivision into two dominant clusters.

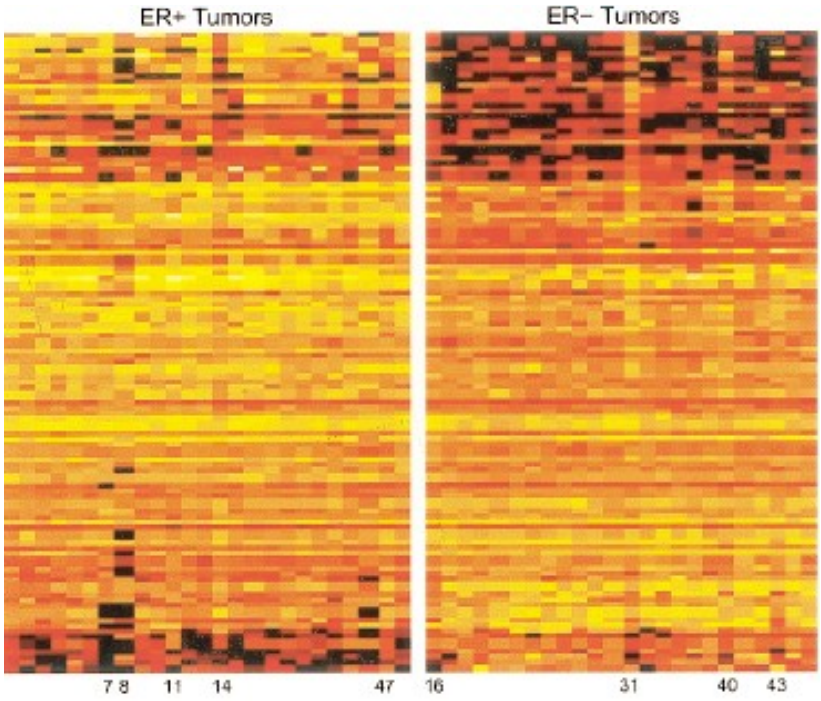
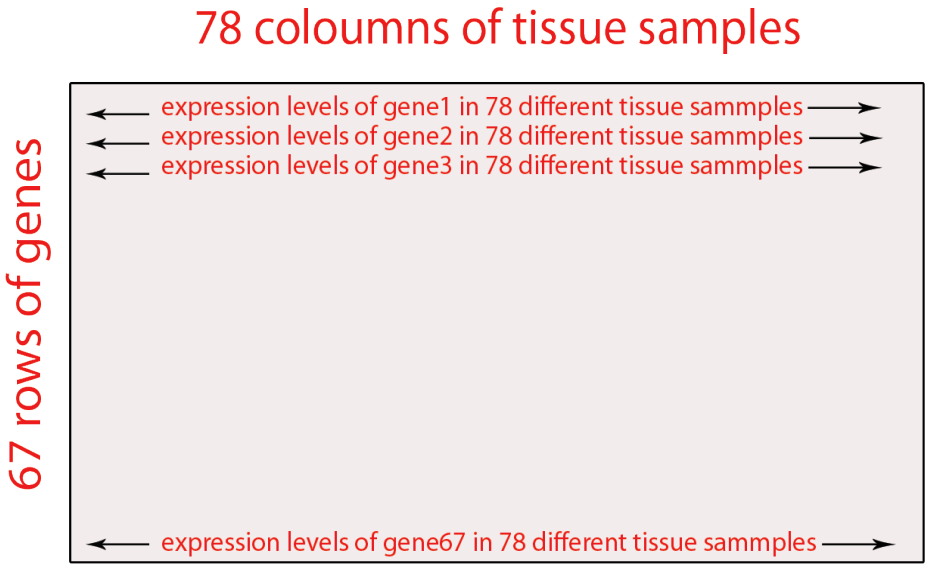


Figure 5. Expression levels of top 100 genes providing pure discrimination of ER status. Expression levels are depicted by colour coding, with black representing the lowest level, followed by red, orange, yellow, and then white as the highest level of expression. Each column in the figure represents all 100 genes from an individual tumor sample, which is grouped according to determined ER status. Each row represents an individual gene, ordered from top to bottom according to regression coefficients



Pic. 4

1.6Datasets obtained with RFE method

Except from the above Van Veer's method in which 67 marker genes had been selected ,we evaluated the RFE (SVM based) method in order to reduce the number of these genes and compare the results obtained from the classifiers with the results of Van Veer's genes.

RFE method uses the internal workings of a Support Vector Machine (SVM) to rank features. An SVM is trained on feature vectors derived from examples of two classes. The apparent importance of each feature is derived from the orientation of the class-separating hyper plane. The feature(s) with the least apparent importance are removed. The remaining features are used to retrain the SVM for the next iteration. Here the stopping criterion for the RFE method is different each time, that's why we have 3 different datasets of genes. In 64 genes ,we keep cutting genes in a way that the remaining number of genes is power of 2. We stopped the elimination when the accuracy of the SVM started falling under 97%. In dataset of 31 genes we started with 5000 genes and continued , cutting in each irritation 1000 genes, until we reached 1000 genes. Then we cut 500, we reach immediately the 500 and after that, we changed the cutting number to 50 each time. Having 50 genes, we changed again the cutting number to 1 each time until we found that reducing the number to less than 31 genes ,the accuracy of the SVM started falling.

1.7Software packages

For the accomplishment of the classification tasks ,we use mainly the Matlab software. Specifically we used to toolbox for neural networks and the prtoolbox for the rest of the algorithms and their fusion.For the creation of roc curves we used to Medcalc software which is a statistical toolbox which can be downloaded for evaluation purposes for 30-days.We wrote scripts where we first choose the evaluation method ,then we choose the classifier and last we get the accuracy results.

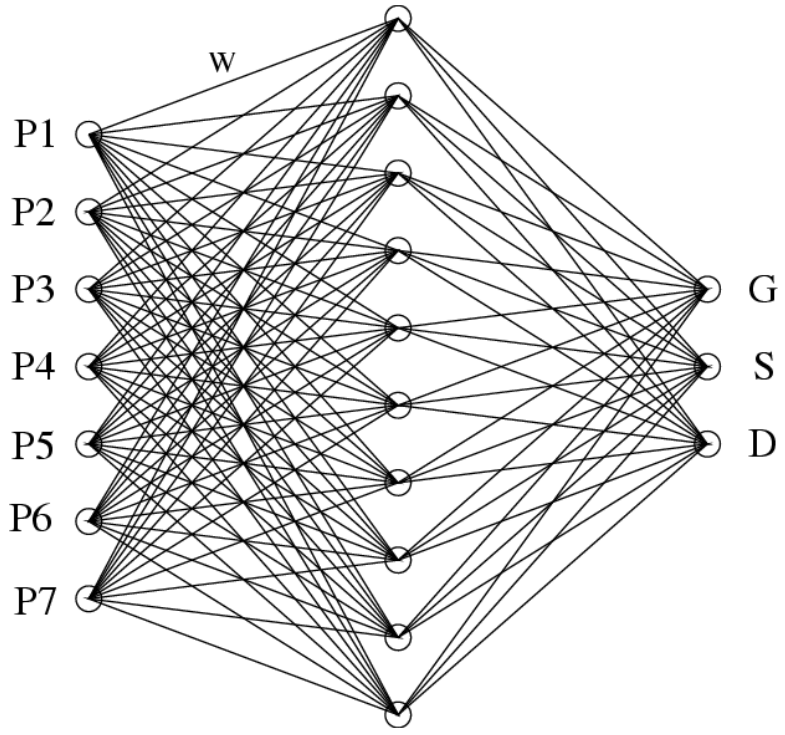
In the end of the work there are the internet addresses where these tools can be found and an example of the way the scripts have been written.

1.8Aims and goals of this work

The purpose of this assignment is the evaluation of 6 supervised classification algorithms on the above breast cancer datasets. We examine their ability on identifying unknown samples after their train procedure. The datasets with the reduced number of genes have been obtain, as we have mention above, with gene selection methods such as the RFE.As a result we have 5 datasets with marker genes (4 datasets of 19,31,64,67 marker genes each (Van Veer experiment) and 1 dataset of 100 marker genes (West experiment) . The accuracy of the classifiers is being tested with use of validation methods such as leave-one-out, v-fold cross validation, roc curves and combining classifiers(fusion).After getting the above results we examine their performance and try to find which of them performed better,

CHAPTER 2

Supervised classification algorithms



2. Methods for supervised classification

2.1Neural network classifier

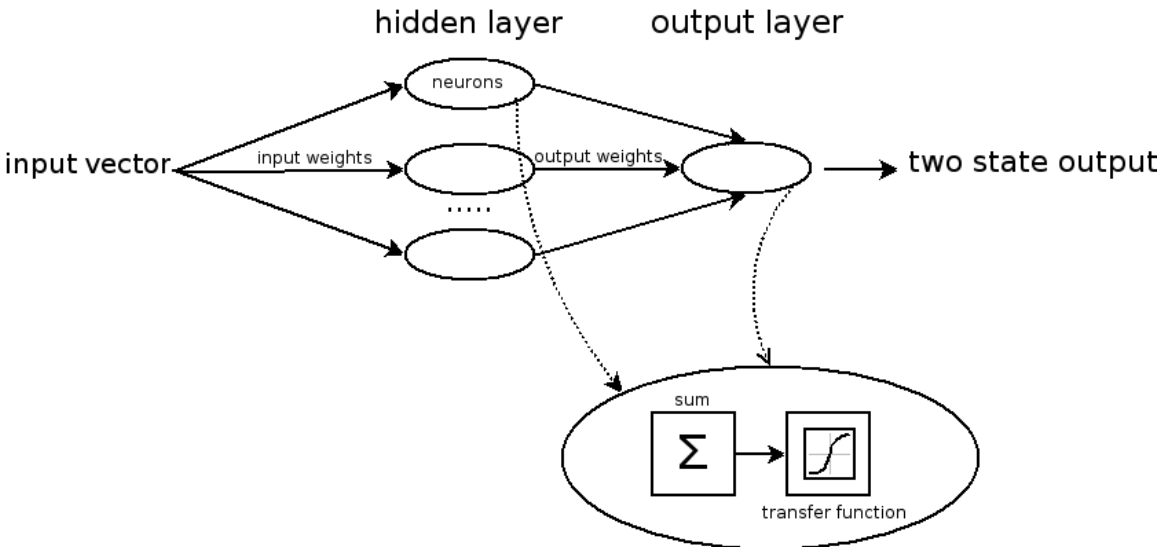
A neural network consists of units (neurons), arranged in layers, which convert an input vector into some output. Each unit takes an input, applies a (often nonlinear) function to it and then passes the output on to the next layer. Generally the networks are defined to be feed-forward: a unit feeds its output to all the units on the next layer, but there is no feedback to the previous layer. Weightings are applied to the signals passing from one unit to another, and it is these weightings which are tuned in the training phase to adapt a neural network to the particular problem at hand. This is the learning phase. In our neural network we utilize back propagation algorithm in order to train our binary model. The inputs were vectors whose columns are the values of the genes in a specific tissue. The output of the network is a two state signal : 1 if the tissue belongs to the 1st class or 0 if the tissue belongs to the 2nd class.

There are many ways to initialize a network ,to train it with the appropriate algorithm ,to update the weights. Each combination has different results so it's critical ,through setting up with multiple ways the network, to find the appropriate neural model that will give us the highest fidelity. For that reason we tested different transfer functions, back propagation algorithms and the way we stopped the training function e. g. maximum number of epochs, minimum square error, minimum gradient.

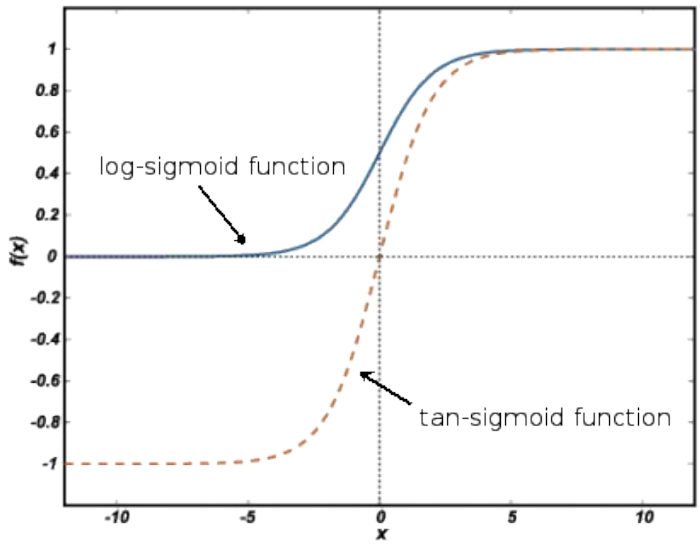
Another issue that we should address is the format of the data that we feed the network. In Van Veer dataset all the expression values of genes were log transformed and in West dataset all the values were normalized by dividing the columns by the mean column intensity.

2.2Setting up the neural network

We assembled a feed-forward back propagation network as we can see in the above schematic picture. We constructed one input layer ,one hidden layer and one output layer. The hidden layer had 5 neurons. As input we feed the layer with the expression values of all genes presented in a specific tissue. If the tissue belongs e. g to a healthy sample then the network would classify it as "1" or as "0" if we have a tumour sample. We let the network to "learn", to update the weight values in order to have the desired output for a specific number of epochs. We used 2 different transfer functions: for Van Veer dataset we applied the log-sigmoid function and the tan-sigmoid function for West dataset (pic. 6). Additionally, we trained the West's network with a gradient descent algorithm with adaptive learning rate and Van Veer's network with levenberg-marquardt algorithm and as a performance function we utilize the mean squared error function for both networks. In the next paragraphs we will explain in depth the above parameters of our neural network classifier and how we evaluated its classification performance.



pic 5



Pic.6

2.3Initializing and training the ANN

Before presenting the inputs to the neurons of the network we initialize the weights that connect each neuron with the input. A random valued weight matrix $n' \; i$,where n is the number of neurons and i the number of inputs, is created with each row normalized to 1 [13]. Once the network weights have been initialized, the network is ready for training. In our first network we utilize a backpropagation algorithm to train it. Like perceptron learning, back-propagation attempts to reduce the errors between the output of the network and the desired. The term backpropagation refers to the manner in which the gradient is computed for nonlinear multilayer networks.

2.3.1 Descent gradient back propagation algorithm

There are a number of variations on the basic algorithm that are based on other standard optimization techniques, such as conjugate gradient and Newton methods. More precisely we used a variation of a descent gradient backpropagation algorithm with adaptive learning rate. As a gradient descent algorithm the network weights are

moved along the negative of the gradient of the performance function which in our case is the mean square error .

Let x_{ji} be the input vector i for the j neuron, w_{ji} the weight matrix of j neuron , t the target vector . The output of each neuron is $y_j = \phi(u_j)$ where $u_j = \sum_{i=1}^{j} w_{ji}x_{ij}$ and ϕ is the transfer function. The error of the network when we compare the desire output with the real output is $e_j = t_j - y_j$ for the j neuron or $MSE = \frac{1}{m}\sum_{i=1}^{m}(y_i - t_i)^2$ if we are using the mean square error. We want to "propagate" this error backwards to all the neuron's output and input weights so as to fix them and minimize the error output. For this purpose we use the above delta rule for the change to the weight w_{ji} from node i to node j.

i. Output weight modification

Delta rule: $\Delta w_{ji} = \eta \times \delta_j \times x_{ji}$

Δw_{ji} : the weight change between hidden layer and output neuron

η : learning rate

δ_j : local gradient

x_{ji} : input signal to output neuron

The local gradient δ_j is the product of $\phi'(\sum w_{ji}x_{ij})$ and the error e_j. In our network $j = 1$ because we have 1 neuron. More precisely the δ_j comes from the above equation.

$$\delta_j = \frac{\partial E}{\partial (u_j)} = \frac{\partial}{\partial (u_j)} \times \frac{1}{2}(t_j - y_j)^2 = -\frac{\partial y_j}{\partial (u_j)}(t_j - y_j)$$

$$= -(t_j - y_j)\frac{\partial}{\partial (u_j)} y_j = -(t_j - y_j)\frac{\partial}{\partial (u_j)}\phi(u_j)$$

$$= -(t_j - y_j)(1 - y_j)y_j$$

$$\delta_j = \frac{\partial E}{\partial (\sum_{i=1}^{j} w_{ji}x_{ij})}$$

ii. Hidden layer weight modification

When we have to update the weights of the hidden layer meaning the weights that connect the input layer with the layer before the output layer we use the above type with a little modification. For the neuron j the gradient δ_j is the same for all the weights w_{ji} which are connected with the neuron. Also we use the output gradient δ_j which we have computed previously. For each hidden unit j ,we calculate:

$$\delta_j = y_j(1 - y_j) \sum_{k \in downstream(j)} w_{kj}\delta_j$$

Update each network weight w_{ji} as follows:

$$w_{ji} \leftarrow w_{ji} + \Delta w_{ji}$$

where $$\Delta w_{ji} = \eta \times \delta_j \times x_{ji}$$

Let's now see now what "adaptive" learning rate means. With standard steepest descent, the learning rate is held constant throughout training. The performance of the algorithm is very sensitive to the proper setting of the learning rate. If the learning rate is set too high, the algorithm may oscillate and become unstable. If the learning rate is too small, the algorithm will take too long to converge. For this reason we use three constants ,the *learning increase* and *learning decrease* and *max ratio*. First we let the network give an output and to compute the error with the initial learning rate. At the second epoch we calculate again the output error. We compare the new and the old error. If the ratio is more that the *max ratio* (~1. 04) then we decrease the learning rate, multiplying it with learn_decrease (0. 85). If the new error is less than the old error, the learning rate is multiplied by a learning increase factor (=1. 3).

Having decided to use the above methods for initializing the weights and training the network we started to configuring it. We tried different number of neurons, epochs and we changed a lot of times the values of learning parameters so as to find the combination that would give us the optimum results. For the West et al. dataset of 100 genes we used one hidden layer with 15 neurons, the tan-sigmoid function and the gradient descent training algorithm. We trained the network for 400 epochs. For Van Veer et al. dataset we set up the neural network with different parameters each time due to the fact we had three different datasets of significant genes. But first we will explain the training algorithm for this network.

2.3.2 Levenberg-Marquadt algorithm

As we have mentioned before our performance function is the mean square error:

$$MSE = \frac{1}{m}\sum_{i=1}^{m}(y_i - t_i)^2 = \frac{1}{m}\sum_{i=1}^{m}(v_i)^2$$

The Levenberg-Marquardt method which belongs to the batch training techniques tries to minimize this function. The method is based on the calculation of two Jacobian matrices: Jacobian error matrix Je and J^2 which in our situation obtain constants due to the fact that the data are linear. The weight change is calculated from the next equation

$$w_{i+1} = w_i - (J^T \times J + m \times diagJ)^{-1} Je$$

In our network ,where we have one output each time, the error vector is $e = [e_1, \ldots . e_m]$, m=number of samples. The weights are being updated in the same way with the previous learning algorithm. The scalar m controls the learning rate

and the learning method of the algorithm. Let's see how it succeeds (the scalar m) in combining gradient descent algorithm and Gauss-Newton method. If the errors decreases ,that means that the weight's update is good ,then the scalar m is decreased in order to reduce the influence of the gradient descent. On the other hand if the error increases we increase m so as to follow the gradient more with small learning rate and we keep the previous values of the weights [14]. The LM method is very fast and in a small number of epochs the network is trained. But due to the fact that we have to compute inverse matrices the cost of the update becomes prohibitive after the model

size increases to a few thousand parameters. For moderately sized models (of a few hundred parameters) however, this method is much faster than gradient descent.

2.4Accuracy of the neural classifier via leave-one-out method

We chose to validate the accuracy of the neural classifier with the leave-one-out method. The leave-one-out method is an important statistical estimator of the performance of a learning algorithm. It works as follows: We divide the input data into two sets. The one set will be used to train the classifier and we call it *training set* and the other set will be used to test the accuracy of the trained classifier and we call it *test set*. The formal expression of the leave one out method is:

$$R(f_d) = \frac{1}{m}\sum_{i=1}^{m} l(f^i, z_i)$$

where z_i is the training set , f^i is the test set and $R(f_d)$ is the leave-one-out error. Leave-one-out technique is supposed to be an "almost" unbiased estimate of the generalization error of f_d. A common belief is that the leave-one-out estimate has a large variance: when different training sets are sampled from the same distribution, the variance of the leave-one-out error computed over these samplings is generally larger than 10-fold cross validation. Also there are examples where leave-one-out error fails completely due to the instability of the learning algorithm. Despite all these handicaps, the leave-one-out estimate is used by many practitioners. It is generally believed as to be a fairly good estimator albeit not the best and it has been used successfully for model selection[15]. There are a lot of works ,theoretical and practical which point out that the performance of the above algorithm depends a lot on the kind of the classifier. In many cases where the classifier is stable, its leave-one-out error is close to its generalization[16]. In our neural network leave-one-out validation produces results where the performance varies each time we run the algorithm by a factor of 2%, which is a good result and it occures due to the fact that we use mean

square error, a continues error function which have been proved that helps the algorithm to perfome well. But still ,even for classifiers such as neural networks ,there are questions such as if the difference between leave-one-out error and network error give us a good “estimator” for the stability of the classifier [15].

At next pages we see the performance of the classifier using the aforementioned algorithm. In West et al. dataset we have 49 tissue samples of which 25 belong to ER+ (estrogens receptor) and 24 ER- and a neural network of 1 hidden layer with 15 nodes, an output layer of 1 node with two : `1` or `0` and an input vector of 100 genes. We trained the network for 400 epochs with the descent gradient algorithm. In Van Veer et al. dataset we have 78 tissue samples , 44 of them belong to D + (diseased) and 34 belong to D- (not diseased). The network is the same with the previous with the difference that we use 6 neurons and we train it with LM algorithm for 100 epochs. The first table presents the accuracy of the classifier at each dataset. If the equation gives a result more than 0. 5 then the tumor belongs to class “1”, else it belongs to class “0”. The other tables display the arithmetic results of the network when we applied the leave-one-out method. On the x-axis is the tissue samples and in the y-axis is the output of the neural each time we feed it with the test sample. The more close the graph to zero means that the sample belongs to class A and the more closer to one means that the sample belongs to class B.

Table of results: Each percentage represents the generalization ability of the network.

Classifier Marker genes	Feed forward Neural Network trained with backpropagation method
19 genes	97. 4% accuracy, 2 errors
31 genes	100% accuracy
64 genes	97. 4 % accuracy, 2 errors
67 genes (Van Veer)	98,7% accuracy, 1 error
100 genes (West)	83% accuracy, 8 errors

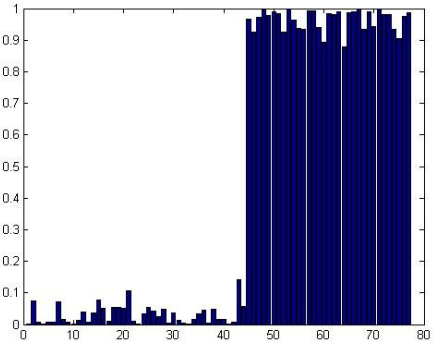


Table2

Leave-one-out results of the network using the test dataset of 19 genes. Each column represents the possibility of tumor belonging to a certain class.

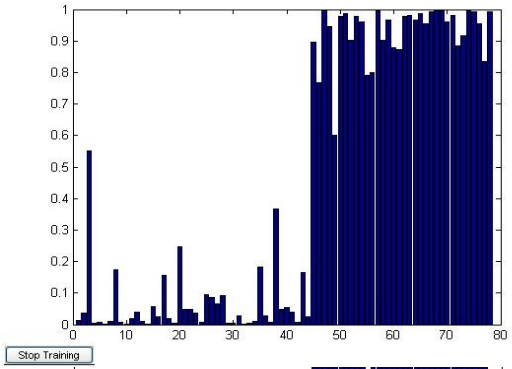


Table 3

Leave-one-out results of the network using the test dataset of 31 genes.

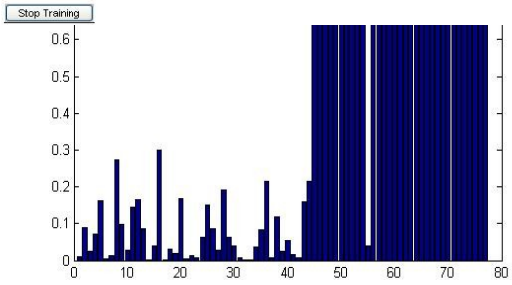


Table 4

Leave-one-out results of the network using the test dataset of 64 genes

Table 5

Leave-one-out results of the network using the test dataset of 67 (Van Veer) genes

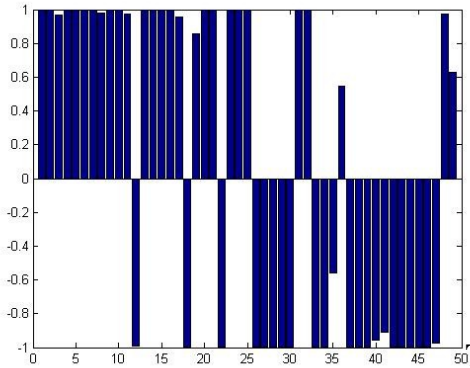


Table 6

Leave-one-out results of the network using the test dataset of 100 genes (West dataset).

Table 1

The x-axis represents the tissue samples and the y-axis the output of the neural each time we test it with the unclassified sample.

2.5Introduction to high order neural network (Honn)

High order neural networks are fully interconnected nets, containing high order connections of sigmoid functions in their neurons .They have been shown to have impressive computational, storage, and learning capabilities. This performance is because the order or structure of a high-order neural network can be tailored to the order or structure of a problem. For the above reason a neural network designed for a specific class of problems becomes specialized and very efficient in solving these problems.

Before starting explaining the above method we will give some information about the relationship between $w_i = w_i \pm ax_i$ n non linear data and TLU's. It is known that nonlinearly separable subsets of pattern space can be dichotomized by nonlinear discriminate functions. Attempts to adaptively generate useful discriminate functions led to the study of threshold logic units (TLUs). Most famous is the perceptron [16] which in its original form was constructed from randomly generated functions of arbitrarily high order. Minsky and Papert studied TLUs of all orders, and came to the conclusions that high-order TLUs were impractical due to the combinatorial explosion of high-order terms, and that first-order TLUs were too limited to be of much interest. They also notice that single feed-forward slabs of first-order TLUs can implement only linearly separable mappings. So which is the solution to the above problems? There are two suggestions: first we can cascade slabs of first order TLUs. But this will result in training problems because either there in no simple way in providing the hidden units with a training signal or multislab learning rules require thousand of iterations to converge which often don't give the correct results due to the local minimum problem . Second suggestion is the use of single slabs of high order TLUs. The high order terms are equivalent to hidden units and since there are no hidden units to be trained the *single slab learning rules* can be used. This methods has been used in this work and will be presented in the next chapter.

2.6Principals of the Honns

If we define x, y its input and output respectively, with $x \in \mathbb{R}^n$ and $y \in \mathbb{R}^q$ the input-output representation of a HONN is given by

$$y = W^T S(x)$$

where $\hat{W}$: L - dimensional vector of adjustable synaptic weights and

$S(x)$: L - dimensional vector with elements $S_i(x)$, $i = 1,2,...L$ of the form

$$S_i(x) = \prod_{j \in I_i} [s(x_j)]^{d_j(i)}$$

where $I_i, i = 1,2,....m$ is a collection of non-order subsets of {1,2…. . n} and $s(x_j)$ is a monotonically increasing, smooth function which is usually represented by sigmoid of the form:

$$s(x) = \frac{\mu}{1 + e^{-lx}} + \lambda$$

where the parameters μ , l represent the bound and maximum slope of sigmoid's curvature while λ is the vertical shift. [17]. In our experiment we created a second order neural network without hidden layers. That means that we feed the network with combinations of $x_i x_j$ where x_i, x_j are values random selected from the input vector x. So the input vector to the k neuron is

$$input = \begin{bmatrix} x_1 \\ x_2 \\ . \\ x_n \\ S_1(x) \\ S_2(x) \\ . \\ . \\ S_i(x) \end{bmatrix}$$

We trained the above network using the simple perceptron rule.

$$E = -\sum_{x \in M} d_x \vec{w} \vec{x}$$

where **M** is the set of misclassified input patterns, and **d** is the desired output of the TLU. This is the perceptron error criterion. If all input vectors are correctly classified the error **E=0**. The contribution of a misclassified input pattern to the error is its absolute distance from the decision boundary. The perceptron learning procedure minimizes this error function. The learning procedure is as follows:

1. train the perceptron with the test set until E=0
2. present the test set to the network (batch or single mode)
3. compare d and and output y and we update the weights as following

$$\Delta w_{iab} = \eta (t_i - y_i) x_a x_b$$

We used only 1 neuron in order the network to compute the function that minimizes the mean error and approximates the input data. Also we trained it for 500 epochs so the time spent on training was long enough. But as we know we must search for a combinations of $x_i x_j$ which give the optimum classifying result and give the parameters λ ,μ the As we see in the results below when the data have low dimensionality the classifier results does not fall below 90 percent. In Van Veer and West dataset the dimensionality is reduced not only because the significant genes are

more but also because we add the second order values to the input vector. For that reason we notice that the performance of the algorithm falls to 80.

Classifier / *Genes*	High order Neural Network with Perceptron rule
19 genes	98,7 % 1 error
31 genes	96,1% 3 errors
64 genes	97,4% 2 errors
67 genes	82% 11 errors
100 genes	79,5% 10 errors

Table 7

Results of the HONN classifier for five different breast cancer datasets

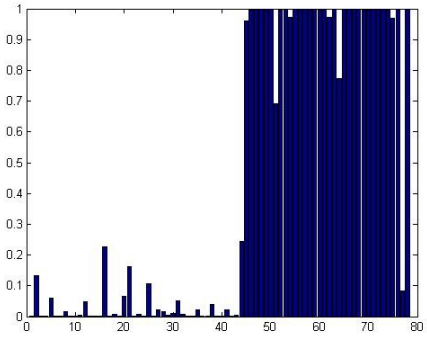


Table 8

Honn result on 19 genes dataset

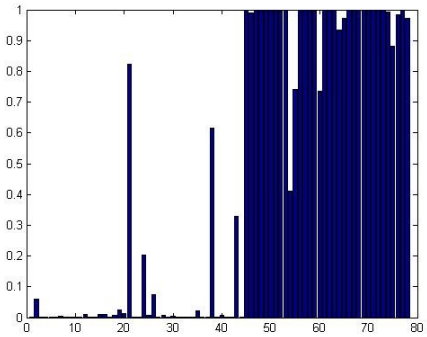


Table 9

Honn result on 31 genes dataset

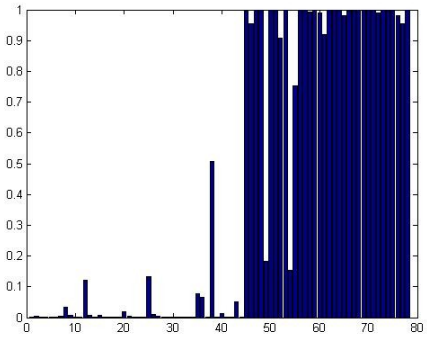


Table 10

Honn result on 64 genes dataset

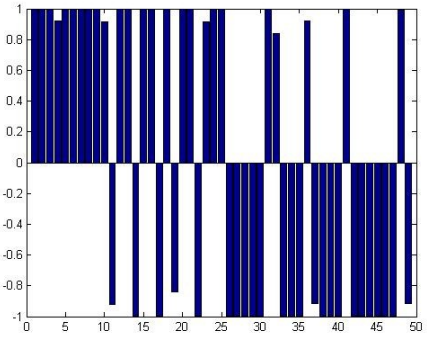


Table 12

Honn result on 100 genes dataset

Table 7

Graphical results of the HONN classifier for five different breast cancer datasets. The x-axis represent the tissues samples and the y-axis the output of the network at each of the previous samples.

2.7Generalization and regularization, two important issues

2.7.1 Generalization

Very often in literature about neural networks we meet the term generalization meaning the performance on unseen input patterns, i. e. input patterns which were not among the patterns on which the network was trained. If we train for too long, you can often get the total sum-squared error very low, by over-fitting the training data - we get a network which performs very well on the training data, but not as well as it could on unseen data. Also by stopping training earlier, we hope that the network will have learned the broad rules of the problem, but not bent itself into the shape of some of the more idiosyncratic (perhaps even noisy) training patterns. In our network we meet two situations. With LM algorithm the square error reached its minimum value too early so we stopped the training of the network after 20 epochs. With the gradient descent algorithm we let the network to be trained for more than 1000 epochs. How we can find the number of epochs where our network will maintain its ability to generalize to unseen data? We perform the above algorithm to find the right stopping point: we divide the dataset into two groups, the first will be the train set and the second the test set. We partitioned with ratio 80/20. We train the network we the train set and we test it with the other set for a specific number o epochs. We write down the mean error. Next we change the number of epochs and we train and simulate the network again. Repeating the previous steps we notice that while error on the training set falls monotonically with the number of epochs, error on the test set falls and then rises. We estimate the number of epochs where this happens and we use it as criterion as our network to have a good generalization ability.

2.7.2 Regularization

Regularization is another way of improving performance. This involves modifying the performance function, which is normally chosen to be the sum of squares of the network errors on the training set. How we do that? In the performance

function we add a term that consists of the mean of the sum of squares of the network weights and bias:

$$M = \alpha\, E_w(w \mid A) + \beta\, E_D(D \mid w, A)$$

where E_w is the familiar performance function , α and β are parameters which will be explained later and $E_D(D \mid w, A)$ is the term which is computed from the next equation:

$$E_w(w \mid A) = \frac{1}{n} \sum_{j=1}^{n} w_j^2$$

Using this performance function will cause the network to have smaller weights and biases, and this will force the network response to be smoother and less likely to overfit. Again there is a trouble maker. The parameters α ,β are very essential in order to improve the generalization ability of the network. If we choose big we might overfit the network. If we set it too small, the network will not adequately fit the training data. One solution, which we have adapted, is the Bayesian regularization.

2.7.3 Bayesian regularization

Although Bayesian analysis has been in use since Laplace,the Bayesian method of *model-comparison* has only recently been developed in depth[18]. What Bayesian regularization do, through the computation of several parameters and comparing several solutions, is objectively setting these settings in order to find the best architecture A which gives optimal results. The main idea is to force a probabilistic interpretation onto the neural network technique so as to be able to make objective statements. Lets suppose we have a network with a specified architecture A and connections w which is viewed as making predictions about the target outputs as a function x in accordance with the probability distribution :

$$P(t^m \mid x^m, w, \beta, A) = \frac{e^{-\beta E(t^m|x^m,w,A)}}{Z_m(\beta)}$$

where $Z_m(\beta) = \int e^{-\beta E} dt$ is the error for a single datum and β is a measure of the presumed noise included in t. Because in our situation the targets are binary outputs we will not use the parameter β but the matrix G that we will see later what it refers to. For the weights w, a prior probability is assigned in the form:

$$P(w \mid \alpha, A, R) = \frac{e^{-\alpha E_w(w|A)}}{Z_w(a)}$$

where $Z_w(a) = \int d^k w e^{-\alpha E_w}$ and α is a measure of the characteristic expected connection magnitude. Now for the posterior probability of the network connections w is then:

$$P(w \mid D, A, R, \alpha, \beta) = \frac{e^{-(\alpha E_w + \beta E_D)}}{Z_M(\alpha, \beta)}$$

where $Z_M(\alpha, \beta) = \int d^k w \chi e^{-(\alpha E_w + \beta E_D)}$

So under this framework, minimisation of $M = \alpha E_w + \beta E_D$ is identical to finding the maximum a posteriori parameters w_{MP};minimization of E_D by backpropagation is identical to finding the maximum likelihood parameters w_{ML}. Thus an interpretation has been given to back propagations functions E_D and

E_w and to the parameters α , β . But in our case there is a difference: our targets are binary and not probabilities estimates. This means that the parameter β will not be used as we will see in the next paragraph.

2.7.3.1 A framework for our classifier: determination of α and G

A classification model H consists of a specification of its architecture A and the regularizer R for its parameters w [19]. When a classification model's parameters are set to a particular value, the model produces an output $y(x : w, A)$ between 0 and 1, which is viewed as the probability $P(t = 1 \mid x, w, A)$. The likehood i. e the probability of the data as a function of w is then:

$$P(D \mid w, A) = \prod_m y^{t_m} (1 - y)^{1 - t_m}$$
$$= \exp G(D \mid w, A)$$

where

$$G(D \mid w, A) = \sum_m t_m \log y + (1 - t_m) \log(1 - y)$$

Now is we assign a prior over alternative parameter vector w,

$$P(w \mid a_c, A, R) = \frac{\exp(-\sum_c a_c E^{(c)}_w)}{Z_w(a)}$$

where $Z_w(a) = \int d^k w e^{-\alpha E_w}$ and α is a measure of the characteristic expected connection magnitude. Now for the posterior probability of the network connections w is then:

$$P(w \mid D, a_c, A, R) = \frac{e^{-(\sum_c \alpha_c E^{(c)}_w + G)}}{Z_M}$$

where $Z_M(\alpha) = \int d^k w \, e^{-(\alpha_c E_w + G)}$. Both Z_w and Z_M are normalizing constants. The calculation of the gradient and Hessian of G is as easy as for a quadratic E_D (is case we had the regression model) if the outputs units activation function is the traditional logistic $f(a) = 1/(1+e^{-a})$. In our case we compute the Hessian matrix through the levenberg-marquat Jacobian algorithm.

The gradient of G with respect to the parameters w and for a function $y(x^{(m)}) = f(a(x^{(m)}))$ is:

$$\nabla G = \sum_m (t_m - y) g(m)$$

where $g_{(m)} = \partial a / \partial w|_{x = x^{(m)}}$.

Now lets see how the training process gives the appropriate set of w_{MP} (*training*) and how the network responds to test sample (*classification*). Lets assume a locally Gaussian posterior probability distribution over $w = w_{MP} + \Delta w$, $P(w \mid D)$; $P(w_{MP}) \exp(-\frac{1}{2} \Delta w^T A \Delta w)$ and if we assume that the activation $\alpha(x; w)$ is a locally linear function of w with $\partial\alpha/\partial w = g$, then for any given x ,the activation α is approximately Gaussian distributed :

$$P(a(x) \mid D) = normal(a^{MP}, s^2) = \frac{1}{\sqrt{2\pi s^2}} \exp(-\frac{(a - a^{MP})^2}{2s^2}$$

where $a^{MP} = \frac{\gamma}{2E_W(w^{MP})}$ and $s^2 = g^T A^{-1} g$. The parameter γ is $\gamma = N - 2a^{MP} tr(H^{MP})^{-1}$ and it is called the effective number of parameters and it is a

measure of how many parameters in the neural network are effectively used in reducing the error function. It can range from zero to N.

So the moderate output is:

$$P(t=1|x,D) = \psi(a^{MP}, s^2) = \int f(a)\,normal(a^{MP}, s^2)$$

The above results give the moderate outputs which the same with the most probable networks outputs are since in our case the targets output are binary numbers and not possibilities.

Let's now see the steps made in order to compute the parameter α and the optimal w_{MP} [20].

1. Initialize the α and the weights giving them random values. After the first training step, the objective function parameters will recover from the initial state.

2. We take one step from the Levenberg-Marquard algorithm to minimize the objective function M

3. Compute the effective number of $\gamma = N - 2a^{MP} tr(H^{MP})^{-1}$ where H is found with the Gauss-Newton approximation using the Levenberg-Marquard training algorithm $H = \nabla^2 F(w) = J^T J + 2aI_N$ where J is the jacobian error matrix of the trainig set errors.

4. We compute the new estimate for $a^{MP} = \dfrac{\gamma}{2E_W(w^{MP})}$

5. We repeat steps from 1-4 until coverage

Each re-estimate of the objective function parameters, the objective function is changing; therefore, the minimum point is moving. If traversing the performance surface generally moves toward the next minimum point, then the new estimates for the objective function parameters will be more precise. Eventually, the precision will be good enough that the objective function will not significantly change in subsequent iterations. Thus, we will obtain convergence.

The Bayesian regularization method does not stop here. The next step that utilizes is the model comparison which is not evaluated in this work. In our neural network ,as we mention before ,we use the Bayesian regularization in order to

minimize a combination of squared errors and weights, and then to determine the correct combination so as to produce a network that generalizes well. In the next table we can see the result of the above method.

Classifier Marker genes	Feedforward Neural Network trained with backpropagation method and regularized by Bayesian rule	Feedforward Neural Network trained with backpropagation method
19 genes	100% accuracy 0 errors	97. 4% accuracy, 2 errors
31 genes	100% accuracy 0 errors	100% accuracy
64 genes	98,7% accuracy 1 error	97. 4 % accuracy, 2 errors
67 genes (Van Veer)	98,7% accuracy 1 error	98,7% accuracy, 1 error
100 genes (West)	92,3% accuracy 5 errors	83% accuracy, 8 errors

Table 8

Comparing bayesian regularized neural network with simple neural network

2.8Conclusions for neural networks

As we can clearly see the accuracy is very high. Even when the dimensionality is increased, the classifier has good performance although with HONN we notice a small decrease. One aspect that we should point out is this: if we tried to train the West's network with the LM algorithm and test its accuracy the results wouldn't be of the same quality. Additionally, if we would try to test the dataset of 67 genes in the neural classifier of the 32 gene's dataset the results would be totally different in a negative way. This occurrence is due to the fact that the two datasets have been obtained from different experiments and for different purposes. Moreover the clustering methods with which we have chosen the most significant genes each time result in datasets with different genes. With this example we want to specify the importance of the experiment and how it affects the results of the above algorithms when they appied on different datasets. So far there has not been established a unified way of treating a specific kind of dataset e. g. breast cancer or even to assent to a specific type of genes that should be examined when we set up microarrays to examine e. g brain cancer tissues.

Another issue is the results we got using regularization rule. As we easily notice the performance of the neural classifier is increased when we use this method to improve the generalization ability of the network. But we must agree that the results we get without the above method are very good and give small error rate and high generalization of the network.

LAURA FERGUSON

‘neural network 1’, painting

2.9Other Classification methods

We start this section by formally defining the structure of each classification algorithm. We will describe the mathematical and statistical principals they are based on,the correlation and distance measure they adapt and in the end how they perform with our datasets. The main aim of the algorithms is to find a hyperplane which separates best the two classes. Though our data is linear, meaning that they can be segregated by a simple line, we apply and non linear classifiers such as SVM,quadratic and k-nearest classifier in order to qualify their performance on linear data.

2.9.1 K-nearest neighbour classifier

One of the simplest classification procedures is the k- nearest neighbour classifier. To classify a query x, we find the most similar example in train set and the plurality class among the nearest neighbours is the class label of the new sample. To carry out this procedure we need to define a similarity measure on expression patterns. In our experiments, we use the *Pearson correlation* as a measure of similarity. Pearson's correlation reflects the degree of linear relationship between two variables. It ranges from +1 to -1. A correlation of +1 means that there is a perfect positive linear relationship between variables. A correlation of -1 means that there is a perfect negative linear relationship between variables. [21]. The computational form is:

$$r(x,y) = \frac{\sum xy - \frac{\sum x \sum y}{N}}{\sqrt{(\sum x^2 - \frac{\sum x^2}{N})(\sum y^2 - \frac{\sum y^2}{N})}}$$

where x is the input vector , y is a vector from the train set. The vector y_i with high $r(x, y_i)$ has high similarity with the test vector. At the end the algorithm returns the class of the x.

Lets see now how the algorithm works with our data. To determine the class of a new example *x* :

- calculate the distance $r(x, y)$ between *x* and all examples in the training set.
- select k-nearest examples to *x* in the training set
- assign *x* to the most common class among its k-nearest neighbors

The advantages of the classifier is that it is robust to noisy data by averaging k-nearest neighbors, easy to implement ,use and explain its prediction. In our tests we used a 7-neightborhood analysis and the accuracy were around 85% with both bootstrap methods.

2.9.2 Quadratic classifier

Before we start explaining the classifier we will introduce two main theories on which the classifier is based. We will begin presenting a simple parametric modelling for describing microarray data, the normal distribution. The normal distribution is a convenient model for studying a wide variety of physical processes. The probability density function of a multivariate normal distribution has the following form :

$$p(x)=N(x|\mu,\Sigma)=\frac{1}{(2\pi)^{d/2}|\Sigma|^{1/2}}\exp\left(-\frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu)\right)$$

χ Ξ ‴ d

where $\mu = \frac{1}{N}\sum_{n=1}^{N} x_n$ *mean value*

$$\varepsilon = \frac{1}{N-1}\sum_{n=1}^{N}(x-\mu)(x-\mu)^T \quad \textit{covariance matrix}$$

Another issue is the rule that we use to choose the right class for the unknown sample in a classification problem. The discriminant function $g_i(x)$,where *i* is the class, it is based on the Bayesian rule and has the next form:

$$g_i(x) = P(\omega_i \mid x) = \frac{P(x \mid \omega_i)P(\omega_i)}{P(x)} = \frac{1}{(2\pi)^{d/2}\,|\Sigma|^{1/2}}\exp\left(-\frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu)\right)P(\omega_i)\frac{1}{P(x)}$$

We eliminate the constant terms and we have:

$$g_i(x) = |\varepsilon_i|^{-1/2}\exp\left(-\frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu)\right)P(\omega_i)$$

Additionally we take the natural logs:

$$g_i(x) = -\frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu) - \frac{1}{2}\log|\varepsilon_i| + \log(P(\omega_i))$$

The above expression is called quadratic discriminant function. To see it in a more "quadratic" form we can reorganize the equation:

$$g_i(x) = x^T W_i x + w_i x + w_{io}$$

$$W_i = -\frac{1}{2}\Sigma_i^{-1}$$

Where $w_i = \Sigma_i^{-1}\mu$

$$w_{i0} = -\frac{1}{2}\mu^T\Sigma^{-1}\mu - \frac{1}{2}\log|\Sigma_i| + \log(P(\omega_i))$$

The covariance matrix Σ_i is the one that defines if the discriminant function is linear or quadratic and the decision boundaries are hyper-planes or parabolic.

There are five cases for Σ_i :

1. $\Sigma_i = \sigma^2 I$
2. $\Sigma_i = \Sigma$ *diagonal*
3. $\Sigma_i = \Sigma$ no diagonal
4. $\Sigma_i = \Sigma_j$ general case
5. $\Sigma_i = \sigma_i^2 I$

In our case we choose the 4th case which is more general case. We computed two different covariance matrices and when we presented the test sample to the trained classifier, it would choose the class according to the next rule (MAP):

decision rule $\rightarrow$ choose ω_i if $g_i(x) > g_j(x)$

where $g_i(x) = P(\omega_i \mid x)$

The construction of the quadratic classifier involves estimating mean vectors μ_i nd covariance matrices Σ_i altogether nΣ_i^{-1}(n + 3)/2 parameters. Estimating these parameters with high accuracy is necessary for constructing a good discriminant rule, because the calculation of the inverse matrices Σ_i^{-1} are often ill-conditioned. Estimating the high-dimensional covariance matrices requires a large amount of data. In our case the estimation did not take long due to the use of computer with lot of ram memory and processor with high speed.

2.9.3 Support vector classifier

Support vector machines (SVMs) have been successfully applied to a wide range of pattern recognition problems, including handwriting recognition, object recognition, speaker identification, face detection and text categorization. SVMs are attractive because they boast an extremely well developed theory. A support vector machine finds an optimal separating hyperplane between members and non-members of a given class in an abstract space. But many classifiers do the same thing. So what is the difference that makes the SVM so good classifiers? There are two major classification problems that SVM succeeds in bypassing them.

The two problems have to do with the non-linearity and the high dimensionality of the train data. As we know the real-world problems involve non-separable data for which there does not exist a hyperplane that successfully separates the class members from non-class members in the training set. One solution to the inseparability problem is to map the data into a higher-dimensional space and define a separating hyperplane there. This higher-dimensional space is called the feature space, as opposed to the input space occupied by the training examples. With an appropriately chosen feature space of sufficient dimensionality, any consistent training set can be made separable. However, translating the training set into a higher-dimensional space incurs both computational and learning-theoretic costs. Representing the feature vectors corresponding to the training set can be extremely expensive in terms of memory and time. Furthermore, artificially separating the data in this way exposes the learning system to the risk of finding trivial solutions that overfit the data.

Support vector machines elegantly sidestep both difficulties [22]. SVMs avoid overfitting by choosing a specific hyperplane among the many that can separate

the data in the featurespace. SVMs find the maximum margin hyperplane, the hyperplane that maximizes the minimum distance from the hyperplane to the closest training point (see figure 4). The maximum margin hyperplane can be represented as a linear combination of training points. Consequently, the decision function for classifying points with respect to the hyperplane only involves dot products between points. Furthermore, the algorithm that finds a separating hyperplane in the feature

space can be stated entirely in terms of vectors in the input space and dot products in the feature space. Thus, a support vector machine can locate a separating hyperplane in the feature space and classify points in that space without ever representing the space explicitly, simply by defining a function, called a kernel function that plays the role of the dot product in the feature space. This technique avoids the computational burden of explicitly representing the feature vectors.

The selection of an appropriate kernel function is important, since the kernel function defines the feature space in which the training set examples will be classified. The kernel function acts as a similarity metric between examples in the training set. As long as the kernel function is legitimate, an SVM will operate correctly even if the designer does not know exactly what features of the training data are being used in the kernel-induced feature space. Human experts often find it easier to specify a kernel function than to specify explicitly the training set features that should be used by the classifier. The kernel expresses prior knowledge about the phenomenon being modeled, encoded as a similarity measure
between two vectors.

Another appealing feature of SVM classification is the sparseness of its representation of the decision boundary. The location of the separating hyperplane in the feature space is specified via real-valued weights on the training set examples. Those training examples that lie far away from the hyperplane do not participate in its specification and therefore receive weights of zero. Only the training examples that lie close to the decision boundary between the two classes receive nonzero weights. These training examples are called the support vectors, since removing them would change the location of the separating hyperplane.

2.9.3.1 The theory of SVM

We have the labeled training data $\{x_i, y_i\}, i = 1,....l$, $y_i \in \{-1,1\}$ and $x_i \in \cdots^d$. Suppose we have some hyperplane which separates the positive from the negative results. The points x which lie on the hyperplane satisfy $w \cdot x + b = 0$ where w is normal to hyperplane , $|b|/\|w\|$ is the perpendicular distance from the hyperplane to the origin and $\|w\|$ is the Euclidean norm of w. Let d_+ (d_-) be the shortest distance from the separating hyperplane to the closest positive (negative) example. Define the "margin" of the separating hyperplane to be $d_- + d_+$. For the linearly separable case the support vector algorithm simply looks for the separating hyperplane with largest margin. This can be formulated as follows: suposse that all the training data satisfy the following constraints:

$$x_i \cdot w + b \geq +1 \text{ for } y_i = 1$$
$$x_i \cdot w + b \leq -1 \text{ for } y_i = -1$$

This can be combined into one set of inequalities:

$$y_i(x_i \cdot w + b) - 1 \geq 0$$

Also the distance between the two parallel lines that separate the two classes is $2/\|w\|$. So we can find the pair of the hyperplanes which gives the maximum margin by minimizing the $\|w\|^2$ with the above inequality as a constrain. Thus we have the minimization problem:

$$\text{Minimize } \|w\|^2$$
$$\text{subject to } y_i(x_i \cdot w + b) - 1 \geq 0$$

We will now switch to a Lagrangian formulation of the problem for two reasons. The constraints will be replaced by the lagrange multipliers and with the reformulation of the problem tha training data will only appear in the form of dot products between vectors. This is a crucial property that will allow us to generalize the procedure to the non linear case. The lagrangian of the above inequation is:

$$L_p = \frac{1}{2}\|w\|^2 - \sum_{i=1}^{l} a_i y_i (x_i \cdot w + b) + \sum_{i=1}^{l} a_i$$

We must now minimize L_p with respect to w , b and simultaneously require that the derivatives of L_p with respect to all the a_i vanish ,all subject to the constraints $a_i \geq 0$. This is convex quadratic programming problem

By setting the derivative of the Lagrangian to be zero the optimization problem can be written in terms of a_i

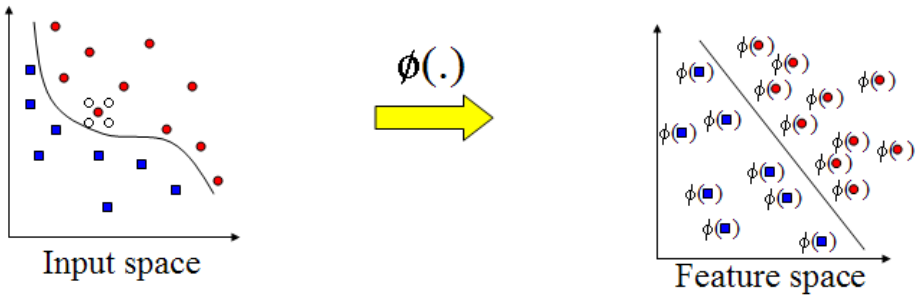
$$\max \quad W(a) = \sum_i a_i - \frac{1}{2}\sum_{i,j} a_i a_j y_i y_j x_i x_j$$

$$\text{subject to } a_i \geq 0 \text{ , } \sum a_i y_i = 0$$

Solving the above quadratic problem or else the “dual problem” we use the a_i in order to find the $w = \sum_{i=1}^{n} a_i y_i x_i$ which will give us the best parallel lines that separate the different classes. Many of the a_i are zeros. The x_i with the non zero a_i are called the support vectors and determine the decision boundary. So if t_j, $(j = 1.....s)$ is the indices of the s support vectors we can write $w = \sum_{j=1}^{s} a_{t_j} y_{t_j} x_{t_j}$ so as when we test unseen data z we compute

$w^T z + b = w = \sum_{j=1}^{s} a_{t_j} y_{t_j} (x_{t_j} z) + b$ and classify z as class 1 if the result is positive and class2 otherwise.

So far we have consider only large margin classifier with linear decision boundary. Lets see what happens when our data is non-linear. In this case we use the kernel trick. Before that, we will explain the idea of treating non-linear data. For this reason we introduce two terms, *input space* and *feature space.* The input space is where our data appear (which in our non-linear case are not separable by a straight line) and where the data in the training algorithm are in the form of dot products .



Now suppose we first mapped our data to some other (maybe infinite) Euclidean space H ,which we call it *feature* space, using a mapping $\Phi : R^d \to H$. In this space the data are linear separable. Then of course the algorithm would depend on the data through dot products $\Phi(x_i) \cdot \Phi(x_j)$ as usual and the computation would be easy. But still we don't know the function Φ . For this reason we use the kernel trick and we work in the feature space. We choose an appropriate function K such that $K(x_i, x_j) = \Phi(x_i) \cdot \Phi(x_j)$ so as to use K in the training algorithm even though the function Φ is unknown. So in one hand we succeed in working in a linear way by increasing the dimensionality of the data and in the other hand we can compute the inner product of $\Phi(x_i) \cdot \Phi(x_j)$ with the help of kernel function K. But which kernel functions are appropriate each time? First we will mention the types of kernels that are being used more.

- Polynomial kernel with degree *d*

$$w = \sum_{j=1}^{s} a_{t_j} y_{t_j} x_{t_j} \quad K(x, y) = (x \cdot y + 1)^p$$

- Radial basis function kernel with width s

$$K(x, y) = e^{-\|x-y\|^2 / 2\sigma^2}$$

- Sigmoid with parameter k and q

$$K(x, y) = \tanh(kx \cdot y - \delta)$$

So if we rewrite the dual problem using the kernel function K we will have:

$$\max \quad W(a) = \sum_i a_i - \frac{1}{2} \sum_{i,j} a_i a_j y_i y_j K(x_i).K(x_j)$$

$$\text{subject to} \quad a_i \geq 0 \quad \sum a_i y_i = 0$$

Accordingly if we want to test an unknown sample z we will use the modified test function:

$$f = \prec w, \phi(z) \succ + b = w = \sum_{j=1}^{s} a_{t_j} y_{t_j} K(x_{t_j}, z) + b$$

All above the kernels functions must obey the Mercer's condition which states that there is a mapping Φ and an expansion $K(x_i, x_j) = \Phi(x_i) \cdot \Phi(x_j)$ if and only if, for any $g(x)$ such :

$$\int g(x)^2 dx \text{ is finite}$$

$$\text{then } \int K(x, y) g(x) g(y) dx dy \geq 0$$

In other words the function must be positive definite in order to describe an inner product [23], [24]. Note that for specific cases, it may not be easy to check whether Mercer's condition is satisfied. However we can easily prove that the condition is satisfied for positive integral powers of dot product $K(x, y) = (x \cdot y)^p$.

Another issue that we should address is the uniqueness and the global solution that the classifier offers. It turns out that every local solution is also global. This is a property of every convex programming problem [Fletcher 1987]. Furthermore the solution is guaranteed to be unique if the objective function is strictly convex, which in our case means that the Hessian matrix is positive definite. However, even if the Hessian is semi positive definite the solution can still be unique.

By global we mean that there exists no other point in the feasible region at which the objective function takes a lower function. There are two cases where uniqueness may not hold: solutions for which $\{w, b\}$ are themselves unique but for which the expansion of w in $w = \sum_{j=1}^{s} a_{t_j} y_{t_j} x_{t_j}$ is not; and solutions whose $\{w, b\}$ differ. For this case there is a simple theorem that shows if a non unique solution occur,then the solution at one optimal point is continuously deformable into the solution at the other optimal point ,in such a way that all intermediate points are also solutions.

2.9.4 Bayes Classifier

The Naive Bayes Classifier technique is based on the so-called Bayesian theorem and is particularly suited when the dimensionality of the inputs is high. Despite its simplicity, Naive Bayes can often outperform more sophisticated classification methods. They can handle an arbitrary number of independent variables whether continuous or categorical. Bayesian classifiers assign the most likely class to a given example described by its future vector. Learning such classifiers can be

greatly simplified by assuming that features are independent given class, that is, $P(X \mid C) = \prod_{i=1}^{n} P(X_i \mid C)$ where $X = (X_1 \ldots\ldots X_n)$ a feature vector and C is the class. Despite this unrealistic assumption, the resulting classifier is remarkably successful in practice, often competing with much more sophisticated techniques [25]. Naive Bayes has proven effective in many practical applications, including text classification, medical diagnosis, and systems performance management [26].

The success of naive Bayes in the presence of feature dependencies can be explained as follows: optimality in terms of zero-one loss (classification error) is not necessarily related to the quality of the fit to a probability distribution. (i. e. , the appropriateness of the independence assumption). Rather, an optimal classifier is obtained as long as both the actual and estimated distributions agree on the most-probable class [25]. For example [25] prove naive Bayes optimality for some problems classes that have a high degree of feature dependencies, such as disjunctive and conjunctive concepts.

2.9.4.1 Definitions and background

Let $X = (X_1 \ldots\ldots X_n)$ be a vector of observed random variables called *features* where its feature takes value from its domain D_i. The set of all feature vectors (*example* or *states*) is denoted $\Omega = D_1 ' \ldots\ldots ' D_n$. Let C be an unobserved random variable denoting the *class* of an example where in our case takes two values 1 or 0. The Bayes classifier $h(x)$ uses as dicriminant functions the class posterior probabilities given a feature vector i. e $f_i(x) = P(C = i \mid X = x)$. Applying Byes rule gives :

$$P(C = i \mid X = x) = \frac{P(X = x \mid C = i)P(C = i)}{P(X = x)}$$

where $P(X = x)$ is identical for all classes and therefore it can be ignored. This yields Bayes discriminant functions:

$$f_i(x) = P(X = x \mid C = i)P(C = i)$$

where $P(X = x \mid C = i)$ is called the class conditional probability distribution (CPD).

Thus the Bayes Classifier

$$h(x) = \arg\max_i P(X = x \mid C = i)P(C = i)$$

finds the *maximum a posteriori probability* (MAP) hypothesis given example x. However direct estimation of the $P(X = x \mid C = i)$ from a given set of training examples is hard when feature space is high-dimensional. Therefore approximations are commonly used, such as using the simplifying assumption that features are independent given the class. This yields the naïve Bayes classifier $NB(x)$:

$$f_i(x)^{NB} = \prod_{j=1}^{n} P(X = x \mid C = i)P(C = i)$$

Therefore the rule that we use for the classification task is:

$$x : C_k \text{ if } P(C_k \mid x) = \max(\prod_{j=1}^{n} P(X = x \mid C = i)P(C = i))$$

The probability of a classification error or *risk* of a classifier h is defined as

$$R(h) = P(h(X) \neq g(X)) =$$
$$\varepsilon \quad P(h(X) \neq g(X))P(X = x) = E_x\{P(h(x) \neq g(x)\}$$

where E_x is the expectation error over x. We say that classifier *h* is optimal on a given problem if its risk coincides with the Bayes risk.

2.9.4.2 Limitations and optimality of the Bayes classifier

Some limitations of naive Bayes are well-known: in case of binary features (k_i=2 for all $i = 1,\ldots..,n$) it can only learn linear discriminant functions [27] and

thus is always suboptimal for non-linear separable concepts. When $k_i > 2$ for some features naïve Bayes is able to learn (some) polynomial discrimintant functions. Thus, polynomial separability is a necessary, although not sufficient, condition of naive Bayes optimality for concepts with finite-domain features.

Despite its limitations, naive Bayes was shown to be optimal for some important classes of concepts that have a high degree of feature dependencies, such as disjunctive and conjunctive concepts [27]. These results can be generalized to concepts with any nominal features:

Theorem: The naive Bayes classifier is optimal for any two-class concept with nominal features that assigns class 0 to exactly one example, and class 1 to the other examples, with probability 1.

Surprisingly, the accuracy of naive Bayes is not directly correlated with the degree of feature dependencies measured as the class-conditional mutual information between the features. Instead, a better predictor of accuracy is the loss of information that features contain about the class when assuming naive Bayes model. However, further empirical and theoretical study is required to better understand the relation between those information-theoretic metrics and the behaviour of naive Bayes. Further directions also include analysis of naïve Bayes on practical application that have almost-deterministic dependencies, characterizing other regions of naive Bayes optimality and studying the effect of various data parameters on the naive Bayes error.

Finally, a better understanding of the impact of independence assumption on classification can be used to devise better approximation techniques for learning efficient Bayesian net classifiers, and for probabilistic inference, e. g. , for finding maximum-likelihood assignments.

CHAPTER 3

Results of the supervised classifiers

3. Results

3.1 The leave-one-out and v-fold cross validation results

In this section we will analyze ,compare and try to evaluate the results we get after applying the four classifiers on the Van Veer's and West's datasets. For testing the accuracy of the classifiers we experimented with two methods, the leave-one-out and the v-fold cross validation. The first method has been analyzed previously so we will say a few words about the second. In v-fold cross validation we randomly create *test-subsets* of size *v* from the samples. We train the classifier with the *(v-1)* samples and the we test it with the *test-subset*. We repeat the process for *n* times(in our case we repeated it 200 times) ,each time creating new subsets. In each repetition we calculate the percentage of the correct classified samples, we sum up all the results after the end of the process in order to find the mean value which shows the accuracy of the classifier.

Before starting the evaluation of the classifiers we took the two datasets and log transformed all the numerical values. We set up each classifier ,trying different parameters each time, in order to succeed in getting the best results. At next table there are the results of the classifiers with both the leave one out and v-fold cross validation methods.

Classifiers / Datasets	**QDC** *Quadratic Classifier*	**SVM** *Support Vector Machine*	**NAIVEbc** *Naive Bayes classifier*	**K-MEANS** *K-nearest neighborhood classifier*
19 genes	87% 10 errors	100%	80.5% 15 errors	85.7% 11 errors
31 genes	88.3% 9 errors	100%	79. 2% 16 errors	81.8% 14 errors
64 genes	84.4% 12 errors	97.4% 2 errors	79.2% 16 errors	84.4% 12 errors
67 genes *Van Veer et al.*	77.9% 17 errors	80. 5% 15 errors	80. 5% 15 errors	83.1% 13 errors
100 genes *West et al.*	83% 8 errors	83% 8 errors	89. 7% 5 errors	85. 7% 7 errors

Table 9

Classification results via leave-one out method

Classifiers Datasets	**QDC** *Quadratic Classifier*	**SVM** *Support Vector Machine*	**NAIVEbc** *Naive Bayes classifier*	**K-MEANS** *K-nearest neighborhood classifier*
19 genes	74%	99. 6%	85,6%	88. 3%
31 genes	92%	100%	85. 3%	88. 6%
64 genes	91%	99%	91,2%	91%
100 genes *West et al*.	89%	91%	90%	89. 6%
67 genes *Van Veer et al.*	86. 6%	88%	80. 36%	84%

Table 10

Classification results via 3-fold cross validation out method

In the leave-one-out method ,we notice that the SVM classifier has the best performance ,specially with 19 and 30 gene's dataset where the accuracy is 100%. As the number of genes is increased ,the performance of classifier falls to 80%. One explanation would be the fact that the dimensionality of the input space is decreased so it's easier for the classifier to compute the correct discriminant function and classify unseen sample with more accuracy. But as we can notice all the classifiers ,except from the k-means, give better results with the 100 genes of West instead of the 67 genes of Van Veer, which are fewer. This phenomenon has to do with the quality of the selected marker genes. When the genes are informative, meaning that they help the classifier to correct find the hyperplane to separate the two classes, then the accuracy of the classifier is better than having less genes with poor 'quality'.

Continuing the analysis of the results we see that the *quadratic classifier* gives good results, which are around 80%-86% with all datasets. The *k-nearest neighborhood* has very good performance with 19 and 30 gene's datasets (81-85% accuracy) and it maintain its accuracy close to 83% with the rest. The *naïve Bayes* classifier gives results which are stable. The accuracy at West's dataset is 89.7%, it performs poor on 64 gene's dataset (79,2%) and performs almost the same on the dataset of 19 and 30 genes (~ 80%). As a conclusion we must admit that most of the classifiers gave very good results with a few classification mistakes. The *SVM classifier* performed well when dealing with a small number of genes, the *k-nearest classifier* has a mean accuracy value of 83% which is similar to the performance of the *quadratic classifier* and last, the naïve Bayes classifier performed at a stable manner (~80%).

Now we will evaluate and compare the results of the v-fold cross validation with the one we got through the leave-one-out process. A first notice we can make is that the performance of the classifiers is close to the previous results. Again, *SVM classifier* has a very good performance, giving accuracy results around 90%, *k-means* and *naïve Bayes* classifiers gave a high level of performance (their mean score is 82% and 84%) and *quadratic* classifier performed a little worst comparing with the leave-one-out results.

3.2 Evaluation through ROC curves

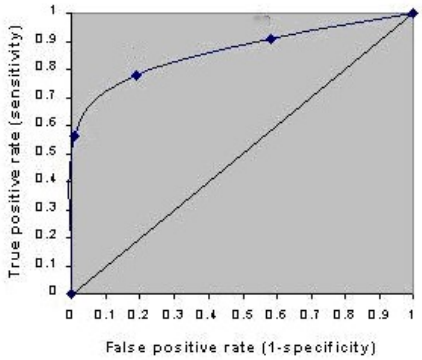
Roc curves is a very important method which can improve the accuracy of two class or multiclass classifiers. The usage only of the results of the classifiers cannot give as a right picture of their performance. In fact we can have two classifiers with the same numerical value of accuracy which act totally different! The Receiver Operating Characteristics (ROC) (pic. 7) of a classifier shows its performance as a trade off between selectivity and sensitivity. What these two terms mean? A more precise definition of the roc curves will help in understanding what they represent. A ROC curve is a provides a graphical representation of the relationship between the true-positive (specificity) and false-positive (sensitivity) prediction rate of a model. The y-axis corresponds to the sensitivity of the model, it shows the *True positive rate* and it is calculated as:

$$T\,\mathrm{Pr} = \frac{\text{number of positive instances correctly classified}}{\text{total number of positive instances}}$$

The x-axis corresponds to the specificity of the classifier ,it represents the *False positive rate* and it is calculated as:

$$F\,\mathrm{Pr} = \frac{\text{number of negative instances misclassified}}{\text{total number of negative instances}}$$

The greater the sensitivity at high specificity values (i. e. high y-axis values at low X-axis values) the better the model.



Pic. 7

Confusion matrix	Positive	Negative
True positive	TPF (true positive fraction)	FPF(false positive fraction)
True negative	FNF(false negative fraction)	TNF (true negative fraction)

Table 11

Confusion matrix

Before going on, we should mention the creation of a confusion matrix that help up to gather and have more accurate view of the data gathered for the creation of the Roc curve.

The most important information that we get from this matrix is the TPF and TNF which is identical with the TRr and FPr that we have explained above. With the help of Matlab we compute the TP and FP samples each time we run the classifier in order to find the pairs **(**FPr,TPr).

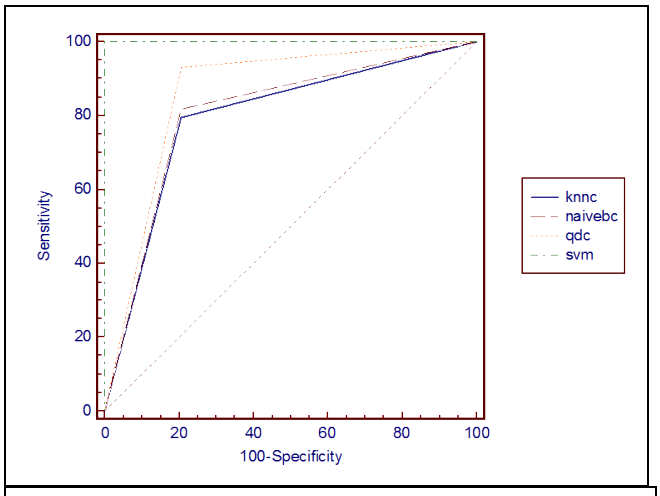
Another interesting aspect of the Roc curve *is the "area under the curve".* A numerical measure of the accuracy of the model can be obtained from this ,where an area of 1. 0 signifies near perfect accuracy, while an area of less than 0. 5 indicates that the model is worse than just random. In other words the quantitative-qualitative relationship between area and accuracy follows a fairly linear pattern. Also the area under the ROC is a convenient way of comparing classifiers which in our case is very useful. A random classifier has an area of 0. 5, while and ideal one has an area of 1. In practice to use a classifier one normally has to chose an operating point, a threshold. This fixes a point on the ROC. In some cases the area may be misleading. That is, when comparing classifiers, the one with the larger area may not be the one with the better performance at the chosen threshold (or limited range). For this reason we introduce the terms *classification cost* and *error rate* in order to obtain more information from the roc curve about the performance of the classifier.

There is a last and important point that we should address about the creation of a roc curve. That is the existence of a threshold parameter in the classifier that will help us to increase *TP* at the cost of an increased *FP* or decrease *FP* at the cost of a

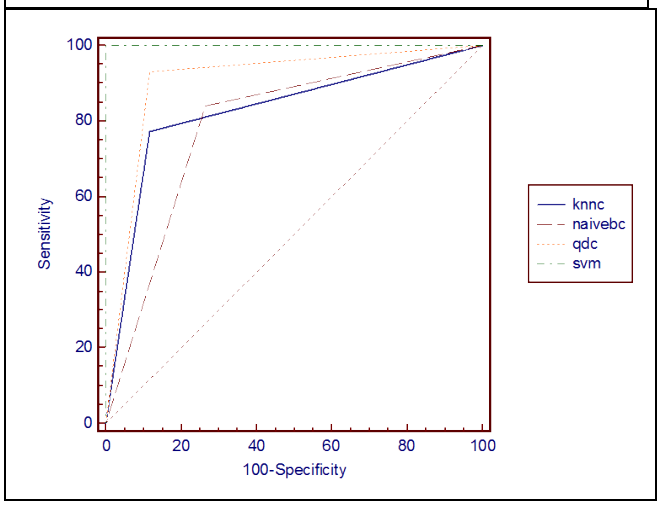
decrease in *TP*. Each parameter setting provides a (*FP*, *TP*) pair and a series of such pairs can be used to plot an ROC curve.

3.3 Results of Roc curves

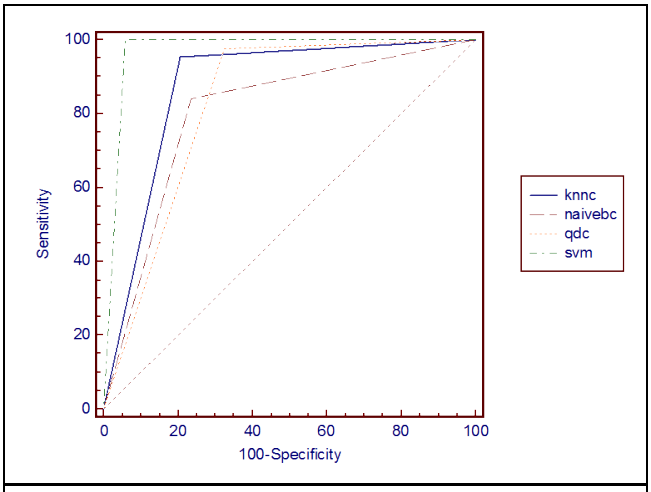
For the creation of the roc curves we took the above classifiers and thought the use of Matlab we created the next Roc curves. We took our datasets ,we split them into two groups, one for training and one for testing. In one graph we represent the 4 classifiers in order to compare them more easily:



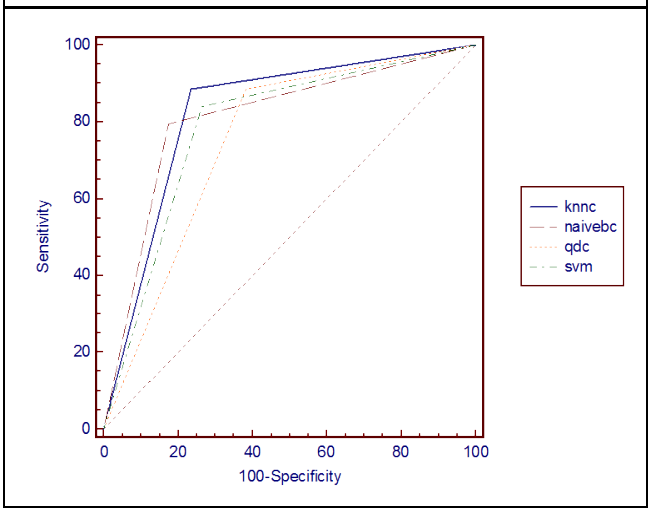
Roc curve for 19 genes



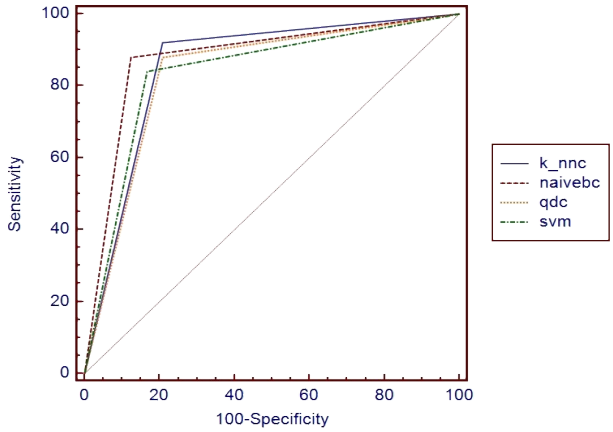
Roc curve for 31 genes



Roc curve for 64 genes



Roc curve for 67 genes



Roc curve for 100 genes

Table 12

Graphs of the roc curves

Observing the above Roc curves we notice two things. First we notice the differences that two classifiers can have even if they give the same results at leave-one-out method.In the dataset of 67 genes of Van Veer the svm and naïve-bayes

classifiers give the same classification results (80% accuracy).But if we examine the roc curve of each classifier we observe that naïve-bayes succeeds better in classifying the healthy samples (class 1) in contrast with svm who performs better on tumor tissues (class 2).Another issue is that comparing the roc curves results and the results of the previous validation techniques we see no differences. On the contrary the roc curves confirm the results we get with the others methods. Also in the above matrix we have the *area under the curve* which is a numerical value for the performance of the classifier:

Area under the curve	**SVM**	**QDC**	**K-NEAREST**	**NAÏVEBAYES**
19 genes	1,000	0,863	0,806	0,795
31 genes	1,000	0,907	0,788	0,828
64 genes	0,971	0,827	0,874	0,803
67 genes	0,788	0,752	0,826	0,809
100 genes	0,837	0,836	0,877	0,856

Area under the roc curve

As we can notice the performance of all classifiers decreases when the dimension of dataset increases. The area of most the classifiers is between 0.8 -0.9 which is a good performance value. Another aspect is that in some datasets, for example in 67 genes dataset, k-nearest and naïve bayes performed better than svm and qdc which give better results in the smaller datasets.

CHAPTER 4

Combining classifiers

4. Introduction to fussion classification

4.1 Beyond singularity: combining the above classifiers

The main objective of designing many machine learning systems is to achieve the best classification. This aim led to the development of different classification schemes for any machine learning problem. It has been observed that in such design studies, usually, none of the single classifiers, such as the classifiers we evaluated above, is enough to classify the unseen data optimally [28].

These observations motivated the interests in combining classifiers. Many single classifiers are used for decision making by combining their individual results to derive a more accurate decision. The aim is to determine an effective combination method that makes use of the benefits and avoids the weaknesses of each classifier.

There are different ways of how the single classifiers are combined [29]. In this work we will test and compare three combining methods: maximum combining, voting combining and product combining classifier. Let's now see the principals of the above combining rules.

4.2 Combining rules

Once the posterior probabilities $p_{mn}(x)$ for i classifiers and j classes have been computed for our test set, they must be combined into $s_m(x)$ which can be used for the final classification. We will use 3 different methods in computing the above term:

$$s'_m(x) = rulem(p_{mn}(x))$$

$$s_m(x) = \frac{s'_m(x)}{\sum_m p_m(x) = 1}$$

The final result is obtained from the equation:

$$f(x) = \arg\max m(s_m(x))$$

The 3 *rules* that we use is described next:

Maximum combining classifier: The maximum combining classifier selects the selection of the single classifier giving the highest normalized probability. So substituting the rule we then have:

$$s'_m(x) = p_{\arg\max(s_{mn}(x))}, n(x)$$

Voting combining classifier: The classification is done by counting the votes for each class over the input classifiers and selecting the majority class. Again if we replace in the first equation we have:

$$s'_m(x) = \sum_m (\text{argmaxm}(p_{mn}(x)) = m)$$

Product combining classifier: Each single classifier gives a normalized probability value for each class. Then all normalized probabilities are multiplied per class. The class with the highest probability product will be chosen. Again, we have:

$$s'_m(x) = \prod_m p_{mn}(x)$$

Having the above combining rules, we test them on our four classifiers and get the next results. We should mention that here the test set is small comparing to train set (a (1/4 ratio) :

	Quadratic	**Svm**	**Naïve-bayes**	**K-Nearest**	**Maximum Combining classifier**	**Product Combining classifier**	**Voting Combining classifier**
19 genes	87% 10 errors	100%	80.7% 15 errors	85.7% 11 errors	100%	84,4%	84,4%
31 genes	88.3%	100%	79. 2%	81.8%	98,7%	81,8%	80,5%
64 genes	84.4%	97.4%	79.2%	84.4%	96,1%	80,5%	90,9%
67 genes	77.9%	80. 5%	80. 5%	83.1%	79,2%	79,2%	79,2%
100 genes	83%	83%	89. 7%	85. 7%	80,3%	89,7%	85,7%

Table 13

Results of the combined methods

We can notice that the maximum combining classifier gives almost equal results with the individual classifier which performs best. The two other methods give results which are like the mean value of all the classifiers at each dataset. We should mention that having the roc curves, the two different cross validation techniques and the combined classifiers results we can have a better view of the performance of the each classifier individually so as to choose which one to use but also to decrease its error while we take into account the results of others.

CHAPTER 5

Conclusions

5. Conclusions

Microarray technology offers a huge amount of data that with the appropriate manipulation can reveal a lot of important information regarding the nature of many of the cancer diseases such a breast cancer. Many techniques have been applied on datasets in order to find specific genes or group of genes that are responsible for cancer symptoms. One aspect of this effort is the classification of clinical samples using algorithms which are widespread used to other areas of science and technology.

We must mention that all the algorithms have a common way of acting: they try to find a hyper plane which best separates the different classes that in our situation are two. The philosophy of computing this hyper plane different in each algorithm. In this work we applied the neural network method in order to predict the class of unseen samples from datasets obtained from two different microarrays experiments. We used different architectures of neural networks such as different backpropagation algorithms in order to test their performance. We got very good results ,most of them close to 90% of accuracy ,due to the fact of the quality of the genes and the power of neural networks. We continued our work using different proposed algorithms that have been used to same works. Again the results are very good with most the algorithms. To support the robustness of the results we used methods such as roc curves, two different cross validation techniques and combination of the classifiers so as to compare all of the results and get a more precise picture of the algorithm's performance each time. In the next tables we can examine the results of each classifier separately.

⇒ **SVM**

19 genes	***31 genes***	***64 genes***	***67 genes***	***100 genes***
100%	100%	97,4% 2 errors	80.5% 15 errors	83% 8 errors

⇒ **Quadratic**

19 genes	**31 genes**	**64 genes**	**67 genes**	**100 genes**
87% 10 errors	88.3% 9 errors	84.4% 12 errors	77.9% 17 errors	83% 8 errors

⇒ **Naïve Bayes**

\

19 genes	**31 genes**	**64 genes**	**67 genes**	**100 genes**
80% 15 errors	79.2% 16 errors	79.2% 16 errors	80.5% 15 errors	89.7% 5 errors

⇒ **K-means**

19 genes	*31 genes*	*64 genes*	*67 genes*	*100 genes*
85.7% **11 errors**	81.8% 14 errors	84.4% 12 errors	83. 1% 13 errors	85. 7% 7 errors

⇒ **Neural network**

19 genes	*31 genes*	*64 genes*	*67 genes*	*100 genes*
100%	100%	98.7% 1 error	98.7% 1 error	92.3% 5 errors

⇒ **High order neural network**

19 genes	*31 genes*	*64 genes*	*67 genes*	*100 genes*
98,7 % **1 error**	96,1% 3 errors	97.4% 2 errors	82% 11 errors	79,5% 10 errors

As we can observe the classifiers give good results, specially the more complicated neural (Bayesian regularized) and svm algorithms. But in all classifiers their performance starts to fall when the dimensionality of the data is increased but it does not fall under 80%.

If we compare the results of the two networks we observe the performance of the HoNN is lower than the performance of the simple neural network. This happens due to the fact that the input space in HoNN is increased when we introduce the second order terms. Increasing the dimensionality of the network we make more difficult for the algorithm to find the appropriate hyperplane to distinguish the two classes.

In general if we want to point out the results of this work we can discrete them as follows:

1. The 3-fold cross validation gave better results than the leave-one-out method.
2. The Bayesian regularization improved the performance of the neural network.
3. In small datasets the more complicated algorithms (NN's and SVM) performed much better than the simple classifiers.
4. Increase of dimensionality =decrease in the performance of the classifier
5. The combination of the classifiers resulted as a mean valued method from where the maximum combining classifier performed better than the others.

For the end we should indicate that the classification task performs well only when the previous clustering and gene selection methods select the most informative genes which help to the linearity of the data. So it is important the correct utilization of each task so as to contribute in the discover of genes and group of genes which need to be treated accordingly in order to fight the different types of cancer.

CHAPTER 6

Resources

6. Resources

6.1 Software for classification

- Prtoolbox (for the other algorithms) http://www.prtools.org/
- Matlab (for neural networks) http://www.mathworks.com/
- Medcalc (for roc curves) http://www.medcalc.be/

Here is an example of matlab script on how we implemented the methods for evaluating the classifier. It is an example of leave –one –out method for k-nearest classifier and for neural network classifier

k-means neighborhood classifier

```
clear
x=xlsread('G100.xls');
x=mameannorm(x);

 for i=1:1:49;

     X=x.';
     A=dataset(X,genlab([25 24],[1 -1]'));
     Test=A(i,:);
     A(i,:)=[];
     [classifier] = knnc(A,5);
     M=Test*classifier;
     e=M*testc;
     coutnting(i)=e;
 end

sum(coutnting)
accuracy=100-(sum(coutnting)*100)/49
```

neural network classifier

```
clear
y=xlsread('results.xls');

for i=1:1:49;
        x=xlsread('G100.xls');
        x=manorm(x);
        test=[x(:,i)];
        x(:,i)=[];
        net=newff(minmax(x),[15  1],{'tansig','tansig'},'traingda');
        net.trainParam.lr = 0.05;
        net.trainParam.lr_inc = 1.2;
        net.trainParam.lr_dec=0.9;
        net.trainParam.show=NaN;
        net.trainParam.mc = 0.5;
        net.trainParam.epochs = 400;
        net.trainParam.goal = 0;
        net.trainParam.max_perf_inc=1.1;
         [net,tr]=train(net,x,y);
        a_test(i)=sim(net,test);

end
```

6.2 Public breast cancer microarray datasets

1. Van't Veer et al.

http://www.rii.com/publications/2002/vantveer.html

25.000 genes reduced to 67 genes of two classes: patients who are free-disease after 5 years and patients who have developed metastases with in 5 years.

2. Perou et al.

Available at
http://genome-www.stanford.edu/breastcancer/molecularportraits/download.shtml

No description due to the fact that the paper needs to be bought

3. Huang et al.

Available at http://data.cgt.duke.edu/lancet.php

Gene expression patterns from the major metagenes that predict lymph node status from current and earlier Duke breast cancer study.(7.129 genes) 496 metagenes, 89 tumor samples in 4 clusters-

4. West et al.

Available at http://data.cgt.duke.edu/west.php

7129 genes per 49 breast tumors reduced to 100 informative genes and clustered to two pathologic groups: 25 tumors with Estrogen Receptor + (ER+) and 24 tumors with ER -

5. Martin et al.

Available at http://mbcf.dfci.harvard.edu/labs/pardee/expression patterns.html

No description due to the fact that the paper needs to be bought

6. Hedenfalk et al.

Available at http://research.nhgri.nih.gov/microarray/NEJM Supplement

This article highlights the overall impact at the gene expression level of diverse regulators of breast cancer growth and links the behavior of breast cancer cells in culture to important clinical. More precisely Expression profiles from three breast cancer cell lines, MCF7, T-47D (both ER), and MDA-MB-436 (ER), were compared at time points (2, 8, and 24 h) after treatment with growth agonists and antagonists known to affect breast cancer cell proliferation.
13.824 genes reduced to 1023 informative genes

6.3 Public microarray database (in alphabetical order)

- ArrayExpress - A public repository for microarray based gene expression data maintained by European Bioinformatics Institute.
- ChipDB - A searchable database of gene expression
- ExpressDB - A relational database containing yeast and E. coli RNA expression data. Reference[PubMed]
- Gene Expression Atlas - A database for gene expression profile from 91 normal human and mouse samples across a diverse array of tissues, organs, and cell lines. Reference[PubMed][pdf]
- Gene Expression Database (GXD) - A database of Mouse Genome Informatics at the Jackson laboratory.

- Gene Expression Omnibus - A database in NCBI for supporting the public use and disseminating of gene expression data. Reference[PubMed]
- GeneX - National Center for Genome Resources's initative to provide an Internet-available repository of gene expression data
- GermOnline - GermOnline provides information and microarray expression data for genes involved in mitosis and meiosis, gamete formation and germ line development across species. Reference[PubMed]
- Human Gene Expression Index (HuGE Index) - aims to provide a comprehensive database to understand the expression of human genes in normal human tissues. Reference[PubMed]
- List Of Lists Annotated (LOLA) - a web driven database allowing researchers to identify and correlate significant subsets of genes derived from microarray expression profiling.
- M-CHiPS (Multi-Conditional Hybridization Intensity Processing System) - M-CHIPS is a data warehousing concept and focuses on providing a structure suitable for statistical analysis of a microarray database's entire components including the experiment annotations. Reference [PubMed][web supplement]
- MUSC DNA Microarray Database - MUSC DNA Microarray Database is a web-accessible archive of DNA microarray data. Reference[PubMed]
- NASCArrays - a repository for Affymetrix data generated by NASC's transcriptomics service. Reference[PubMed]
- Oncomine - The goal of this project is to curate publicly available cancer microarray studies and provide data mining tools to efficiently query genes and datasets of interest as well as meta-analyze groups of studies. Links to various bioinformatics resources have been implemented including Unigene, Swissprot, Biocarta, HPRD, and KEGG, among others.
- Public Expression Profiling Resource (PEPR) - A web oracle data warehouse of quality control and standard operating procedure (QC/SOP) Affymetrix data. Reference[PubMed]
- READ (RIKEN cDNA Expression Array Database)- A database maintained by RIKEN (The institute of Physical and Chemical Research), Japan. Reference[PubMed]
- Rice Expression Database (RED) - RED holds raw and normalized data from expression profiles obtained by the Rice Microarray Project and other research groups. These data are open to the public less than one year after sending the data to each research group.
- RNA Abundance Database (RAD) - RNA Abundance Database (RAD) is a public gene expression database designed to hold data from array-based and nonarray-based (SAGE) experiments. The ultimate goal is to allow comparative analysis of experiments performed by different laboratories using different platforms and investigating different biological systems.
- Saccharomyces Genome Database (SGD): Expression Connection - A gene expression database of Saccharomyces genome database in Stanford University, provide simultaneous search of several microarray studies result for gene expression data for a given gene or ORF. Reference[PubMed]
- Soybean Genomics and Microarray Database (SGMD) - The SGMD attempts to provide an integrated view of the interaction of soybean with the

soybean cyst nematode and contains genomic, EST and microarray data with embedded analytical tools allowing correlation of soybean ESTs with their gene expression profiles. Reference[PubMed]

- Standford Microarray Database (SMD) - Standford Microarray Database (SMD) stores raw and normalized data from microarray experiments, as well as their corresponding image files. In addition, SMD provides interfaces for data retrieval, analysis and visualization. Data is released to the public at the researcher's discretion or upon publication. Reference[PubMed]
- Yale Microarray Database
- yeast Microarray Global Viewer (yMGV) - A database for yeast gene expression data maintained by Laboratoire d

7. References

[1] Jie Liang*, Seman Kachalo, Computational analysis of microarray gene expression profiles: clustering, classification, and beyond, Chemometrics and Intelligent Laboratory Systems 62 (2002) 199– 216

[2] Yong Woo,a Jason Affourtit,Sandra Daigle, Agnes Viale, Kevin Johnson, Jurgen Naggert, and Gary Churchilla A Comparison of cDNA, Oligonucleotide, and Affymetrix GeneChip Gene Expression Microarray Platforms, Journal of Biomolecular Techniques 15:276–284 © 2004 ABRF

[3] Nancy Mah, Anders Thelin, Tim Lu, Susanna Nikolaus, Tanja Kühbacher, Yesim Gurbuz, Holger Eickhoff, Günther Klöppel, Hans Lehrach, Björn Mellgård,A comparison of oligonucleotide and cDNA-based microarray systems, *Physiol. Genomics* 16:361-370, 2004. First published Nov 25, 2003, doi:10. 1152/physiolgenomics. 00080. 2003

[5] T. R. Golub, D. K. Slonim, P. Tamayo, C. Huard, M. Gaasenbeek, J. P. Mesirov, H. Coller, M. L. Loh, J. R. Downing, M. A. Calgiuri, C. D. Bloomfield, E. S. Lander, Molecular classification of cancer: class discovery and class prediction by gene expression monitoring, Science 286 (1999) 531–537.

[6] K. Fujarewicz, Z. Malgorzata ,Selecting diffentially expressed genes for colon tumor classification, Int. J. Appl. Math. Comput. Sci. , 2003, Vol. 13, No. 3, 327–335

[7] J. L. DeRisi, V. R. Iyer, P. O. Brown, Exploring the metabolic and genetic control of gene expression on a genomic scal, Science 278 (1997) 680–686.

[8] Jie Liang*, Seman Kachalo, Computational analysis of microarray gene expression profiles: clustering, classification, and beyond, Chemometrics and Intelligent Laboratory Systems 62 (2002) 199– 216

[9]Marshall, A., Hodgson, J., (1998). DNA chips: an array of possibilities. Nature Biotechnology 16: 27-31

[10] Griffin Weber, Staal Vinterbo, Lucila Ohno-Machado, Multivariate selection of genetic markers in diagnostic classification, Artificial Intelligence in Medicine (2004) 31, 155—167

[11]Laura J. van't Van Veer *Gene expression profiling predicts clinical outcome o breast cancer*

[12] Mike West*, Carrie Blanchette†, Holly Dressman‡, Erich Huang‡, Seiichi Ishida‡, Rainer Spang*, Harry Zuzan*,John A. Olson, Jr.†, Jeffrey R. Marks†, and Joseph R. Nevins‡§ *Predicting the clinical status of human breast cancer by using gene expression profiles.*Edited by Peter J. Bickel, University of California, Berkeley, CA, and approved August 3, 2001 (received for review April 3, 2001)

[13] *A. Pavelka and A. Proch´azka* ALGORITHMS FOR INITIALIZATION OF NEURAL NETWORK WEIGHTS,Institute of Chemical Technology, Department of Computing and Control Engineering. http://dsp. vscht. Cz

[14] Ananth Ranganathan, "The Levenberg-Marquardt Algorithm" 8th June 2004

[15] Andr´e Elisseeff , Leave-one-out error and stability of learning algorithms with applications, Max Planck Institute for Biological Cybernetics Spemannstrasse 38, 72076 Tuebingen Germany ,e-mail: andre. elisseeff@tuebingen. mpg. De

[16] F. Rosenblatt, *Principles of Neurodynamics* (Spartan, New York, 1962).

[17] A Hybrid Neural Network/Genetic Algorithm Approach to Optimizing Feature Extraction for Signal Classification

[18] David J. C MacKay ,A Practical Bayesian Frmework for BackProp Newtwork ,Computation and Neural Systems ,California Institute Of Technology 139-74

[19] David J. C MacKay . The Evidence Framework applied to Classification Networks,
,Computation and Neural Systems ,California Institute Of Technology 139-74

[20] F. Dan Foresee* and Martin T. Hagan**,GAUSS-NEWTON APPROXIMATION TO BAYESIAN LEARNING, email: fdf@lucent. com, mhagan@master. ceat. okstate. Edu

[21] http://davidmlane. com/hyperstat/A51911. html

[22] Christopher Burges , A tutorial on support vector machines for pattern recognition

[23] Vapnic, V. (1995) *The Nature of Statistical Learning Theory*. Springer.

[24] R. Courant and D. Hilbert. *Methods of mathematical physics*, volume 1. Interscience Publishers, New York, 1953

[25] P. Domingos and M. Pazzani. On the optimality of the simple Bayesian classifier under zero-one loss. Machine Learning, 29:103–130, 1997

[26] Tom M. Mitchell. Machine Learning. McGraw-Hill, 1997.

[27]R. O. Duda and P. E. Hart. Pattern classification and scene analysis. New York: John Wiley and Sons, 1973.

[30] Hüseyin Gökhan Akçay,Experiments on Combining Classifiers,Bilkent University, Department of Computer Engineering, 06800 Ankara, Turkey akcay@cs. bilkent. edu. tr

[31] M. van Erp, L. Vuurpijl, and L. Schomaker, An overview and comparison of voting methods for pattern recognition, in Proc. of the 8th IWFHR. 2002, pp. 195--200, IEEE