MACHINE LEARNING ALGORITHMS IN CLASSIFICATION AND DIAGNOSTIC PREDICTION OF CANCERS USING GENE EXPRESSION PROFILING

by

Tuba Altındal

B.S., Physics, Boğaziçi University, 2003

Submitted to the Institute for Graduate Studies in
Science and Engineering in partial fulfillment of
the requirements for the degree of
Master of Science

Graduate Program in Physics

Boğaziçi University

2006

ACKNOWLEDGEMENTS

First, I would like to say that I am grateful to my thesis supervisor, Prof. Dr. M. Levent Kurnaz for providing me with an unconditional freedom in my study, and, also for introducing me to the academicians who helped me a lot in the course of this study.

I would like to thank Prof. Dr. Ethem Alpaydın, who is one of the mentioned academicians, for sharing his broad knowledge on machine learning with me.

I thank Asst. Prof. Dr. Işıl Aksan Kurnaz for helping me to get insight of microarray experiments.

Özgür Delice also deserves thanks due to his kind help with LaTeX 2_ε.

Last but not least, I would like to thank my family for supporting and encouraging me through all these years, and my partner for helping me grow mentally stronger.

ABSTRACT

MACHINE LEARNING ALGORITHMS IN CLASSIFICATION AND DIAGNOSTIC PREDICTION OF CANCERS USING GENE EXPRESSION PROFILING

The performance of certain machine learning algorithms in classification and diagnostic prediction of small round blue cell tumors (SRBCTs) of childhood is investigated. Before classifying samples, including both tumor biopsy material and cell lines, based on their gene expression profiles, dimensionality of the problem is reduced. Dimensionality reduction is achieved in a two-step procedure that includes correlation-based feature selection (CFS) followed by principal components analysis (PCA). To classify the samples into four distinct diagnostic categories, logistic model trees (LMT) and multilayer perceptrons (MLP) are trained. The posterior probabilities provided by LMT and MLP algorithms for each sample are then used to construct a measure, by means of which one might decide whether to classify a sample into one of the diagnostic categories or to reject classifying.

ÖZET

ÖĞRENME ALGORİTMALARININ GEN PROFİLLERİNDEN YOLA ÇIKARAK KANSER TÜRLERİNİN SINIFLANDIRILMASI VE TEŞHİSİNDE KULLANIMI

Bu tezde bir takım öğrenme algoritmalarının çocukluk döneminde görülen küçük, yuvarlak, mavi hücreli tümörleri, gen profillerine dayanarak, sınıflandırma ve teşhis koymadaki başarıları incelenmektedir. Tümör biyopsi materyali ve kültür hücrelerinden oluşan numuneleri gen profillerine dayanarak sınıflandırmadan önce problemin boyutu küçültülmüştür. Boyut küçültme korelasyon tabanlı parametre seçimini takip eden temel bileşenler analizinden oluşan iki basamaklı bir işlem aracılığıyla gerçekleştirilmiştir. Numuneleri sınıflandırmak üzere mantıksal model ağaçları ve de çok katmanlı yapay sinir ağları eğitilmiştir. Her bir numune için yapay sinir ağlarından ve mantıksal model ağaçlarından elde edilen sınıf olasılıkları kullanılarak modelin yaptığı sınıflandırmaya paralel bir teşhis konulup konulmayacağını belirleyen bir kriter oluşturulmuştur.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF SYMBOLS/ABBREVIATIONS

A	Adenine
c	Number of distinct classes in an instance subset S
C	Cytosine
C_i	Class i
Cov($\mathbf{X}$)	Covariance of a matrix $\mathbf{X}$
d	Number of inputs: Input dimensionality
d_c	Distance from the actual output of the sample to the ideal output for class c
d_j^p	j^{th} element of the desired output of the network when input pattern vector p was input to the network
$E(A, T; S)$	Class information entropy of the partition induced by cut point T, for a set of instances S, and a feature A
$\boldsymbol{e_i}$	Eigenvector of $\mathbf{\Sigma}$ corresponding to λ_i
Ent(S)	Class entropy of a subset S
E^p	Error in the output of the network when input pattern vector p was input to the network
F_j	Activation function associated with unit j
G	Guanine
h	Hidden unit
$H(Y)$	Entropy of Y
$H(Y\|X)$	Entropy of Y after observing X
i	Input unit
j, k	Unit j, k
k	Number of features in a feature subset S
M_S	Merit of a feature subset S
$N_{h,n}$	Number of hidden units in the n^{th} hidden layer
N_i	Number of input units
N_o	Number of output units
N_S	Number of instances in an instance subset S
o	Output unit

$p(C_i\|S)$	Fraction of instances belonging to class C_i in an instance subset S
$p(y)$	Probability of observing y
$p(y\|x)$	Probability of observing y after observing x
$\mathbf{R}$	Sample correlation matrix
$\overline{r_{cf}}$	Mean feature-class correlation
$\overline{r_{ff}}$	Mean feature-feature correlation
S	Set of instances
$\mathbf{S}$	Sample covariance matrix
s_j^p	Total input to unit j when input pattern vector p was input to the network
T	Thymine
U	Uracil
$\mathrm{Var}(z)$	Variance of the observed values z of a variable
w_{jk}	Weight of the connection from unit j to unit k
x, y	Observed values of X and Y respectively
X, Y	Random variables
$\boldsymbol{x}$	Input (pattern) vector
$\mathbf{X}$	Data matrix
x_j^p	j^{th} element of the p^{th} input pattern vector
y_j^p	Activation value of unit j when input pattern vector p was input to the network
α	Momentum coefficient
α	Lagrange multiplier
δ_{ij}	Kronecker delta function
η	Learning rate
θ_j	Bias input to unit j
λ_i	i^{th} highest eigenvalue of $\mathbf{\Sigma}$
$\mathbf{\Sigma}$	Covariance matrix of $\mathbf{X}$

ANN	Artificial neural network
BL	Burkitt lymphoma

cDNA	Complementary DNA
CFS	Correlation-based feature selection
Cy3	Cyanine 3
Cy5	Cyanine 5
DNA	Deoxyribonucleic acid
EWS	Ewing family of tumors
LMT	Logistic model trees
MDL	Minimum description length
MLP	Multilayer perceptrons
mRNA	Messenger RNA
NB	Neuroblastoma
NHL	Non-Hodgkin lymphoma
PCA	Principal components analysis
PSP	Postsynaptic potential
RMS	Rhabdomyosarcoma
RNA	Ribonucleic acid
SRBCT	Small round blue cell tumor
tRNA	Transfer RNA

1. INTRODUCTION

Machine learning methods share in common the purpose of extracting knowledge from observational data. It is assumed that there is an underlying structure in data. That's why it is important that the data is generated by the very same process from which the structure stems. The techniques used in machine learning applications such as learning associations, classification, regression, clustering, and reinforcement learning might be implemented solely or in combinations to deal with problems from many fields. Medical diagnosis, image and speech recognition, problems involved in robotics and bioinformatics are the most familiar ones.

Physics as a discipline sharing the same purpose with machine learning provides many problems to which the machine learning algorithms could easily be applied.

In high energy physics, Maggipinto et al. [1] studied the role of artificial neural networks in the search of Higgs boson, the only particle predicted by the Standard Model of the electroweak interactions that had not been discovered yet. They used neural network classifiers to discriminate events characterized by Higgs boson production from background events due to multi-hadron production induced by strong interactions of quark and gluons.

On the other hand, Garrido and Juste [2] used artificial neural networks instead of standard binned procedures to obtain W-pair production and background probability density functions in terms of the reconstructed di-jet invariant masses.

In nuclear physics, Bass et al. [3] showed that the neural network approach to impact parameter determination in a heavy ion collision improves the performance by a factor of two as compared to classical techniques.

In astrophysics, Rohde et al. [4] applied machine learning algorithms to the problem of matching catalogues with significant positional uncertainties. Catalogue

matching would allow efficient access to the vast amounts of data being collected by all sky surveys in many wavelengths.

In this thesis, certain machine learning algorithms are going to be utilized to enable the classification and diagnostic prediction of cancers belonging to several diagnostic categories based on their gene expression signatures. These algorithms comprise correlation-based feature selection, principal components analysis, multilayer perceptrons, and logistic model trees.

Gene expression profiles of the samples are determined through microarray experiments. Microarray technology is a powerful tool for genetic research that utilizes nucleic acid hybridization techniques and recent advancements in computing technology to evaluate mRNA expression profile of thousands of genes within a single experiment [5].

1.1. Thesis Overview

Chapter 2 introduces the main actors of gene expression process and briefly discusses the process itself. It also provides an introduction to microarray experiments.

Chapter 3 clarifies how data are organized in a typical machine learning application in the first section. It is mainly concerned with dimensionality reduction and classification algorithms. Section 3.2.1 describes the correlation-based feature selection algorithm which is based on the hypothesis that a good feature subset is one that contains features highly correlated with the class, yet uncorrelated with each other. It also elucidates the Fayyad and Irani's discretization method which is used to convert continuous-valued features into nominal before subset selection. Section 3.2.2 deals with principal components analysis by means of which one finds a mapping from the inputs in the original d-dimensional space to a lower dimensional space, with minimum loss of information. Section 3.3.1 reviews artificial neural networks and backpropagation algorithm used in training ANNs. Section 3.3.2 discusses logistic model trees which are hybrid structures based upon the tree induction and logistic regression approaches

to learning. In the next section, performance evaluation technique used in this thesis is explained.

Chapter 4 gives the optimal values of predefined model parameters. And, it contains the results obtained from two models we trained: one MLP classifier and one LMT classifier.

Chapter 5 discusses the accuracy of our models in the light of the results of Chapter 4.

2. GENE EXPRESSION PROFILING

2.1. Fundamentals of Gene Expression

Deoxyribonucleic acid (DNA) and *ribonucleic acid* (RNA) are the key elements of the *gene expression* process. DNA is a double-stranded molecule twisted into a helix. Each spiraling strand, comprised of a sugar-phosphate backbone and attached bases, is connected to a complementary strand by non-covalent hydrogen bonding between paired bases, adenine (A) with thymine (T) and guanine (G) with cytosine (C) [6, 7]. Contrary to DNA, RNA is a single-stranded molecule in which thymine (T) is replaced by uracil (U).

Gene expression process eventuates in two steps [6, 7]. In the first step, named *transcription*, one strand of the DNA is used as a template by the *RNA polymerase* to synthesize a *messenger RNA* (mRNA). In most organisms, only a small fraction of DNA is capable of being transcribed to mRNA or expressed as proteins [7]. Then mRNA migrates from the nucleus to the cytoplasm. During this period, mRNA goes through a mechanism called *splicing* where the non-coding sequences (*introns*) are eliminated. The coding mRNA sequence (*exons*) can be described as a unit of three nucleotides called a *codon*. The ribosome binds to the mRNA at the start codon (AUG) that is recognized only by the *initiator tRNA*. The ribosome proceeds to the second stage of protein synthesis which is named *translation*. During this stage, complexes, composed of an amino acid linked to tRNA, sequentially bind to the appropriate codon in mRNA by forming complementary base pairs with the tRNA anticodon. The ribosome moves from codon to codon along the mRNA. Amino acids are added one by one, translated into polypeptidic sequences dictated by DNA and represented by mRNA. At the end, a release factor binds to the stop codon, terminating translation and releasing the complete polypeptide from the ribosome.

2.2. Hybridization

The two strands of a DNA molecule are denatured by heating to about 100 °C = 212F. At this temperature, the complementary base pairs that hold the double helix strands together are disrupted and the helix rapidly dissociates into two single strands. The DNA *denaturation* is reversible by keeping the two single strands of DNA for a prolonged period at 65 °C = 149F. This process is called DNA *renaturation* or *hybridization*. Similar hybridization reactions can occur between any single stranded nucleic acid chain: DNA/DNA, RNA/RNA, DNA/RNA. If an RNA transcript is introduced during the renaturation process, the RNA competes with the coding DNA strand and forms double-stranded DNA/RNA hybrid molecule. These hybridization reactions can be used to detect and characterize nucleotide sequences using a particular nucleotide sequence as a probe [7].

2.3. Microarray Experiments

In many biomolecular studies, the main concern is to measure real gene expression, that is the abundance of proteins. However, DNA microarray experiments measure the abundance of mRNA, assuming there is a direct one to one mapping from DNA to mRNA to protein. One of the main reasons why researchers are conducting DNA microarray studies, despite all their caveats, is the fact that protein expression studies are very expensive, and often involve highly specialized and delicate techniques [7].

DNA microarrays exploit a potent feature of the DNA duplex - the sequence complementarity of the two strands. This feature makes hybridization possible.

To start with, let us look at color-coded genes in competitive and comparative hybridization illustrated in Figure 2.1 [8]. In order to detect and measure the amount of mRNA that is contained in an investigated sample, the mRNA must be labelled with reporter molecules. The reporters currently used in microarray experiments include fluorescent dyes, for example, cyanine 3 (Cy3) and cyanine 5 (Cy5).

Let us assume we have two samples of transcribed mRNA from two different sources, sample 1 and sample 2. Both samples may consist of multiple copies of many genes. We have also a probe, which is a specific nucleic acid sequence, perhaps a gene, or a characteristic subsequence of a gene, or a short, artificially composed nucleotide sequence. Like the two samples, the probe will contain many copies of the sequence in question. This is important because sufficient amounts are needed to get the hybridization reaction going, and to be able to detect and measure the various concentrations. What we want to find out is the relative abundance of the mRNA complementary to the probe sequence within sample 1 and sample 2. To find out the exact answer, we proceed as follows:

- Prepare a mixture containing identical probe sequences.
- Label sample 1 and sample 2 with green and red-dyed reporters respectively.
- Simultaneously give both sample mixtures the chance to hybridize with the probe mixture. Here, sample 1 and sample 2 are said to compete with each other in an attempt to hybridize with the probe.
- Gently stir for five minutes.
- Filter the mixture to retain only those probe sequences that have hybridized, that is, formed a double-stranded polymer.
- Measure the amount or intensity of green and red in the filtered mixture, and compare the amounts to determine the relative abundance of the probe sequence.

Because of RNA's inherent chemical instability, it is often useful to work with a more stable *complementary DNA* (cDNA) made by reverse transcription, rather than with mRNA. However, before the array is made, the cDNA is denatured to allow the hybridization reaction.

Spotted microarrays consist of a solid surface (e.g., a microscope glass slide) onto which small sized spots containing nucleotide sequences are placed in a grid-like arrangement. Each spot represents a specific gene, an *expressed sequence tag* (partial gene sequence providing a tag for a gene of which the full sequence or function may not be known), a *clone* (population of identical DNA sequences) derived from cDNA

libraries, or an *oligonucleotide* (short sequence specifically synthesized for experiment). The spots serve as probes against which target and reference mRNA is hybridized.

In spotted arrays, the probes are placed on the array by an automated process called *contact spotting* or *printing* (similar to ink-jet printer technology). The spotting machinery prints nucleotide spots with a diameter of approximately 100 micrometers in close proximity on the array. In this way, 10000 to 30000 probes can be arranged on a single array. However, the number of probes does not necessarily match the number of genes. For reasons of reproducibility, a gene may be represented by more than one probe on the array [7].

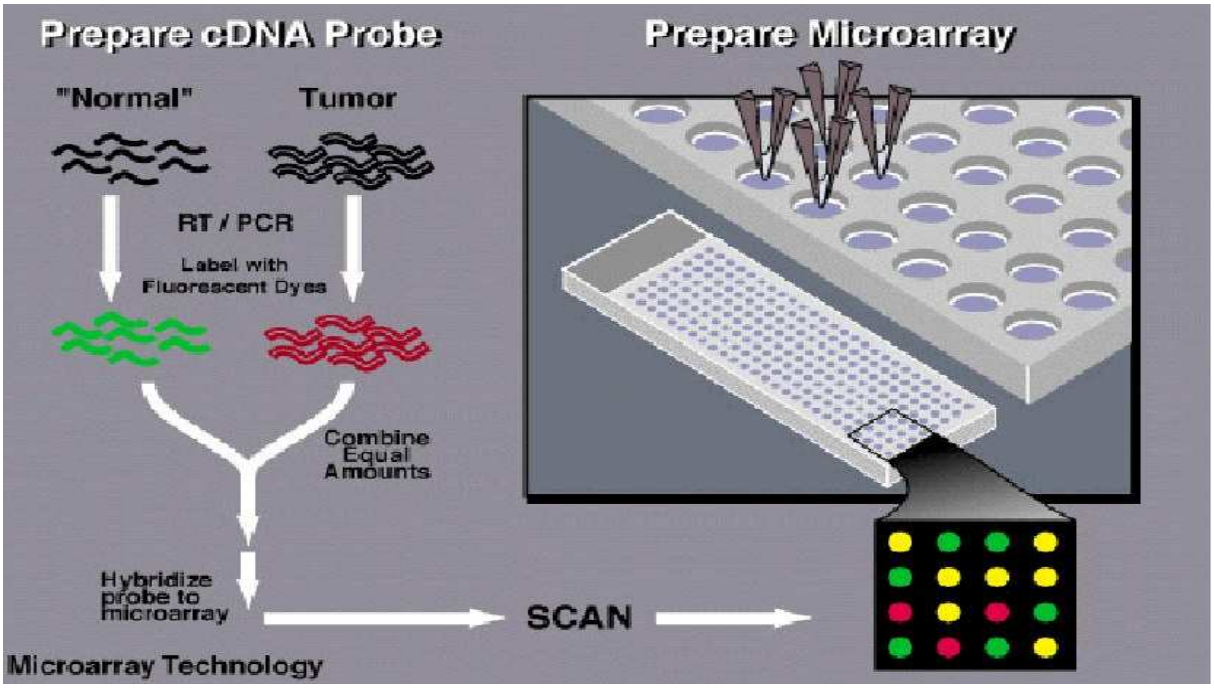

Figure 2.1. Schematic illustration of a two-channel microarray experiment

3. CLASSIFICATION AND DIAGNOSTIC PREDICTION

3.1. Data Representation

Machine learning applications require several measurements and observations to be made on a group of *instances*, or *samples*. These measurements and observations are called *features*, or *attributes*. The instances are the things that are to be classified, or associated, or clustered. Each instance is characterized by its values on a fixed, predefined set of features.

A typical machine learning application requires two sets of samples: training samples and test samples. The set of training samples is used to induce the underlying structure in data and a separate set of test samples is needed to evaluate the accuracy. So, the machine learning data may be represented by two tables of instances. In a classification task, each instance is described by its values on a fixed, predefined set of features, along with a label that denotes its class. When testing, the machine learning algorithm takes, as input, a test sample and produces, as output, a class label (predicted class for that sample).

In this thesis, the terms feature and attribute denote gene expression levels and tumor samples are referred by the terms instance and sample.

In microarray experiments, the term *gene expression profile* is commonly used to describe the expression values for a single gene across many samples or experimental conditions, and for many genes under a single condition or sample. To distinguish between these two types of gene expression profiles: A *gene profile* is used to denote gene expression profile that describes the expression values for a single gene across many samples or conditions. An *array profile* refers to a gene expression profile that describes the expression values for many genes under a single condition or sample. This is also called *expression signature* [7].

3.2. Dimensionality Reduction

Although most machine learning algorithms are designed to learn which are the most appropriate features to use for making their decisions, it is common to precede learning with a feature selection stage to avoid the negative effect of irrelevant features on machine learning algorithms. Reducing the dimensionality of the data also speeds up the learning process, and, to be of greater importance, provides a more compact representation of the underlying structure.

There are two main methods for reducing dimensionality: *feature selection* and *feature extraction*. In feature selection, one is interested in finding k of the d dimensions that give us the most information and the other $(d - k)$ dimensions are discarded. In feature extraction, one seeks for a new set of k dimensions that are the combination of the original d dimensions.

On the other hand, the feature subset selection algorithms perform a search through the space of feature subsets, and, as a consequence, must address four basic issues affecting the nature of the search:

- *Starting point.* Selecting a point in the feature subset space from which to begin the search can affect the direction of the search. One option is to begin with no features and successively add features. In this case, the search is said to proceed forward in the search space. Conversely, the search can begin with all features and successively remove them. In this case, the search proceeds backward through the search space. Another alternative is to begin somewhere in the middle and move outwards from this point.
- *Search organization.* An exhaustive search of the feature subspace is prohibitive for all but a small initial number of features. With N initial features there exist 2^N possible subsets. Heuristic search strategies are more feasible than exhaustive ones and can give good results, although they do not guarantee finding the optimal subset.
- *Evaluation strategy.* How feature subsets are evaluated is the single biggest differ-

entiating factor among feature selection algorithms for machine learning. Some algorithms use heuristics based on general characteristics of the data to evaluate the merit of feature subsets. Another school of thought argues that the bias of a particular induction algorithm should be taken into account when selecting features. This method, called the *wrapper*, uses an induction algorithm along with a statistical resampling technique such as crossvalidation to estimate the final accuracy of feature subsets.

- *Stopping criterion.* A feature selector must decide when to stop searching through the space of feature subsets. Depending on the evaluation strategy, a feature selector might stop adding or removing features when none of the alternatives improves upon the merit of a current feature subset. Alternatively, the algorithm might continue to revise the feature subset as long as the merit does not degrade. A further option could be to continue generating feature subsets until reaching the opposite end of the search space and then select the best.

One simple search strategy, called *greedy hill climbing*, considers local changes to the current feature subset. Often, a local change is simply the addition or deletion of a single feature from the subset. When the algorithm considers only additions to the feature subset it is known as *forward selection*; considering only deletions is known as *backward elimination*. An alternative approach, called *stepwise bi-directional search*, uses both addition and deletion. Within each of these variations, the search algorithm may consider all possible local changes to the current subset and then select the best, or may simply choose the first change that improves the merit of the current feature subset. In either case, once a change is accepted, it is never reconsidered.

Best first search is a search strategy that allows backtracking along the search path. Like greedy hill climbing, best first moves through the search space by making local changes to the current feature subset. However, unlike hill climbing, if the path being explored begins to look less promising, the best first search can backtrack to a more promising previous subset and continue the search from there. Given enough time, a best first search will explore the entire search space, so it is common to use a stopping criterion. Normally this involves limiting the number of fully expanded

subsets - those in which all possible local changes have been considered that result in no improvement.

3.2.1. Correlation-based Feature Selection

The concept of correlation-based feature selection is based on the following hypothesis stated by Hall [9]:

> "A good feature subset is one that contains features highly correlated with (predictive of) the class, yet uncorrelated with (not predictive of) each other."

The correlation-based feature selection (CFS) algorithm used in this thesis evaluates the merit of each feature subset on the basis of the following evaluation function:

$$M_S = \frac{k\overline{r_{cf}}}{\sqrt{k + k(k-1)\overline{r_{ff}}}}, \tag{3.1}$$

where M_S is the merit of a feature subset S containing k features, $\overline{r_{cf}}$ is the mean feature-class correlation, and $\overline{r_{ff}}$ is the mean feature-feature correlation. Decision tree induction provides a number of methods for estimating how predictive one feature is of another. Regarding the outcomes of his study, Hall [9] states that of the three correlation measures; *symmetrical uncertainty*, *relief* and *minimum description length* (MDL) tested with CFS, symmetrical uncertainty and MDL are superior to relief in both artificial and natural domains, and also symmetrical uncertainty outperforms MDL in natural domains due to its ability to cope with datasets of fewer training instances. Since microarray experiments suffer from the limited number of training instances, symmetrical uncertainty will be an appropriate choice as a correlation measure to be used in Equation 3.1.

To be able to treat feature and class values in a uniform fashion, it is required to transform continuous valued features to nominal. Hall [9] proposes to discretize continuous features using the method of Fayyad and Irani. Fayyad and Irani [10] use a *minimum entropy heuristic* to discretize numeric features. The algorithm uses the class

entropy of candidate partitions to select a cut point for discretization. The method can then be applied recursively to the two intervals of the previous split until some stopping conditions are satisfied, thus creating multiple intervals for the feature. For a set of instances S, a feature A, and a cut point T, the *class information entropy* of the partition induced by T is given by

$$E(A,T;S) = \frac{N_{S_1}}{N_S}\text{Ent}(S_1) + \frac{N_{S_2}}{N_S}\text{Ent}(S_2), \tag{3.2}$$

where S_1 and S_2 are two intervals of S bounded by cut point T, and $\text{Ent}(S)$ is the *class entropy* of a subset S given by

$$\text{Ent}(S) = -\sum_{i=1}^{C} p(C_i,S)\log_2(p(C_i,S)). \tag{3.3}$$

For feature A, the cut point T which minimizes Equation 3.2 is selected as a *binary discretization boundary*. Fayyad and Irani [10] employ a stopping criterion based on the *minimum description length principle*. The stopping criterion prescribes accepting a partition induced by T if and only if the cost of encoding the partition and the classes of the instances in the intervals induced by T is less than the cost of encoding the classes of instances before splitting. The partition induced by cut point T is accepted iff

$$\text{Gain}(A,T;S) > \frac{\log_2(N-1)}{N} + \frac{\Delta(A,T;S)}{N}, \tag{3.4}$$

where N is the number of instances in the set S,

$$\text{Gain}(A,T;S) = \text{Ent}(S) - E(A,T;S), \tag{3.5}$$

and

$$\Delta(A,T;S) = \log_2(3^c-2) - [c\text{Ent}(S) - c_1\text{Ent}(S_1) - c_2\text{Ent}(S_2)]. \tag{3.6}$$

In Equation 3.6, c, c_1, and c_2 are the number of distinct classes present in S, S_1,and S_2 respectively.

A probabilistic model of a nominal valued feature Y can be formed by estimating the individual probabilities of the values $y \in Y$ from the training data. Entropy is a measure of the uncertainty or unpredictability in a system. The entropy of Y is given by

$$H(Y) = -\sum_{y \in Y} p(y) \log_2(p(y)). \tag{3.7}$$

If the observed values of Y in the training data are partitioned according to the values of a second feature X, and the entropy of Y with respect to the partitions induced by X is less than the entropy of Y prior to partitioning, then there is a relationship between features Y and X. The next equation gives the entropy of Y after observing X.

$$H(Y \mid X) = -\sum_{x \in X} p(x) \sum_{y \in Y} p(y \mid x) \log_2(p(y \mid x)). \tag{3.8}$$

The amount by which the entropy of Y decreases reflects additional information about Y provided by X and is called the *information gain*, or, alternatively, *mutual information*. Information gain is given by

$$\begin{aligned} \text{gain} &= H(Y) - H(Y \mid X) && (3.9)\\ &= H(X) - H(X \mid Y) && (3.10)\\ &= H(Y) + H(X) - H(X, Y). && (3.11) \end{aligned}$$

Unfortunately, information gain is biased in favor of features with more values. Furthermore, the correlations should be normalized to ensure that they are comparable and have the same effect. Symmetrical uncertainty compensates for information gain's

bias toward attributes with more values and normalizes its value to the range [0,1]:

$$\text{symmetrical uncertainty coefficient} = 2.0 \times \left[\frac{\text{gain}}{H(Y) + H(X)}\right]. \tag{3.12}$$

3.2.2. Principal Components Analysis

PCA is a projection method. It aims to find a mapping from the inputs in the original d-dimensional space to a new ($k < d$)-dimensional space, with minimum loss of information [11]. The input in the original d-dimensional space may be represented by a *data matrix*,

$$\mathbf{X} = \begin{pmatrix} x_1^1 & x_2^1 & \dots & x_d^1 \\ x_1^2 & x_2^2 & \dots & x_d^2 \\ \vdots & \vdots & \ddots & \vdots \\ x_1^N & x_2^N & \dots & x_d^N \end{pmatrix}, \tag{3.13}$$

where the d columns correspond to d features and the N rows correspond to N samples.

PCA uses no class information. It accomplishes dimensionality reduction by rotating the axes of the original feature space such that they line up with the directions of highest variance, then ignoring the axes that contribute to the variance less.

The first principal component is the direction of $\boldsymbol{e_1}$ such that the samples, after projection onto $\boldsymbol{e_1}$, are most spread out so that the distinction between the samples becomes most apparent. For a unique solution and to make the direction the important factor, $\boldsymbol{e_1}$ should be normalized to one. It is known that if $z_1 = \boldsymbol{e_1}^T \boldsymbol{x}$ with $\text{Cov}(\mathbf{X}) = \boldsymbol{\Sigma}$, then

$$\text{Var}(z_1) = \boldsymbol{e_1}^T \boldsymbol{\Sigma} \boldsymbol{e_1}. \tag{3.14}$$

The problem of finding $\boldsymbol{e_1}$ such that $\text{Var}(z_1)$ is maximized subject to the constraint that

$\boldsymbol{e_1}^T\boldsymbol{e_1} = 1$ can be solved by the *method of Lagrange multipliers*. Taking the derivative of

$$\boldsymbol{e_1}^T\boldsymbol{\Sigma}\boldsymbol{e_1} - \alpha(\boldsymbol{e_1}^T\boldsymbol{e_1} - 1)$$

with respect to $\boldsymbol{e_1}$ and setting it equal to zero, results in

$$2\boldsymbol{\Sigma}\boldsymbol{e_1} - 2\alpha\boldsymbol{e_1} = 0, \tag{3.15}$$

and therefore

$$\boldsymbol{\Sigma}\boldsymbol{e_1} = \alpha\boldsymbol{e_1} \tag{3.16}$$

which holds if $\boldsymbol{e_1}$ is an eigenvector of $\boldsymbol{\Sigma}$ and α the corresponding eigenvalue. Since

$$\boldsymbol{e_1}^T\boldsymbol{\Sigma}\boldsymbol{e_1} = \alpha\boldsymbol{e_1}^T\boldsymbol{e_1} = \alpha, \tag{3.17}$$

the first principal component is the eigenvector of $\boldsymbol{\Sigma}$ with the largest eigenvalue, $\lambda_1 = \alpha$. Repeating the method of Lagrange multipliers but this time including the constraint that $\boldsymbol{e_2}$ should be orthogonal to $\boldsymbol{e_1}$, it is found that the second principal component, $\boldsymbol{e_2}$, is the eigenvector of $\boldsymbol{\Sigma}$ with the second largest eigenvalue. Similarly, the other principal components are given by the eigenvector with decreasing eigenvalues.

If the variances of the original x_i dimensions vary considerably, they affect the direction of the principal components more than the correlations, so a common procedure is to preprocess the data so that each dimension has zero mean and unit variance, before using PCA, or, one may use the eigenvectors of the *sample correlation matrix*, $\mathbf{R}$, instead of the *sample covariance matrix*, $\mathbf{S}$, which are the maximum likelihood estimates of the correlation and covariance matrices respectively, for the correlations to be effective rather than the individual variances. The sample correlation matrix is given

by

$$\mathbf{R} = \begin{pmatrix} r_{11} & r_{12} & \dots & r_{1d} \\ r_{21} & r_{22} & \dots & r_{2d} \\ \vdots & \vdots & \ddots & \vdots \\ r_{d1} & r_{d2} & \dots & r_{dd} \end{pmatrix}, \tag{3.18}$$

with

$$r_{ij} = \frac{s_{ij}}{s_i s_j}, \tag{3.19}$$

where

$$s_i^2 = \frac{\sum_{t=1}^{N} \left(x_i^t - m_i\right)^2}{N}, \tag{3.20}$$

$$s_{ij} = \frac{\sum_{t=1}^{N} (x_i^t - m_i)(x_j^t - m_j)}{N}, \tag{3.21}$$

and

$$m_i = \frac{\sum_{t=1}^{N} x}{N}, \qquad i, j = 1, 2, \dots, d. \tag{3.22}$$

To reduce the dimensionality of the problem, only the leading k components that explain more than a predefined percentage of the variance are taken into account. When λ_i are sorted in descending order, the proportion of the variance explained by the k principal components is

$$\frac{\lambda_1 + \lambda_2 + \ldots + \lambda_k}{\lambda_1 + \lambda_2 + \ldots + \lambda_k + \ldots + \lambda_d}.$$

Scree graph is the plot of variance explained by each eigenvector. By visually analyzing it, one can decide on k. At the elbow, adding another eigenvector does not significantly increase the variance explained.

3.3. Classifiers

3.3.1. Artificial Neural Networks

Artificial neural networks emulate signal transmission between biological neurons. This process may be summarized as follows [12]: Electrical pulses impinge on the afferent neuron at *synapses*. This results in the release of *neurotransmitters* that cause potential change in the *dendritic membrane*. The contribution of each pulse to the *postsynaptic potential* (PSP) depends on factors such as the geometry of the synapse and the type of neurotransmitter. The afferent neuron integrates these contributions over synapses and over time. If the integrated potential, *activation potential*, exceeds a threshold, it generates a pulse.

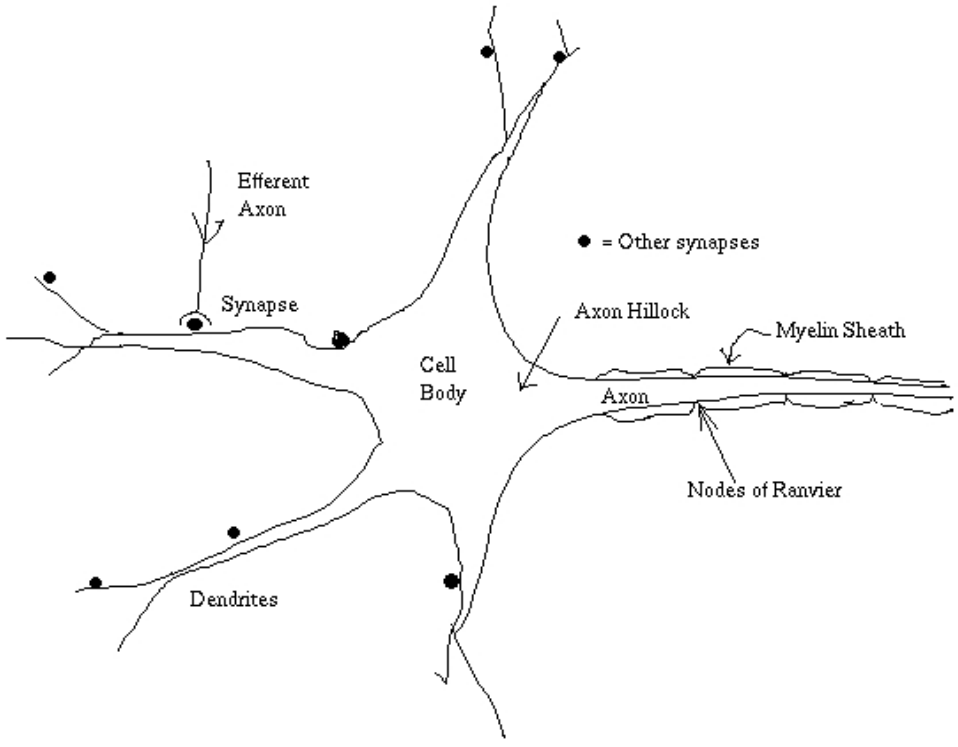

Figure 3.1. Schematic of a biological neuron

An artificial neural network is an interconnected assembly of simple processing elements, *units* or *nodes*, which communicate by transmitting signals to each other over a large number of weighted connections. Each unit receives *input* from other units or

external sources and processes this input to produce an *output* which is propagated to other units. The processing ability of the network is stored in the inter-unit connection strengths, *connection weights* or *synaptic weights*, obtained by a process of learning from a set of training samples.

Within artificial neural networks it is useful to distinguish three types of units: *input units* which receive data from outside the network, *output units* which send data out of the network and *hidden units* whose input and output signals remain within the network.

The total input to unit k which is the weighted sum of the outputs from the connected units plus a *bias* or offset term θ_k is given by

$$s_k = \sum_j w_{jk} y_j + \theta_k. \tag{3.23}$$

The units with a propagation rule Equation 3.23 are called *sigma units*.

The effect of the total input on the output, or *activation*, of the unit depends on the *activation function* F_k. It takes the total input s_k and produces a new value of the activation of the unit k:

$$y_k = F_k(s_k). \tag{3.24}$$

When the pattern of connections between the units and the direction of data flow in these connections are considered, artificial neural networks can be classified in two groups: *feedforward networks* and *recurrent networks*. A feedforward network is a layered structure where the data flow from input to output is strictly feedforward.

A single layer feedforward network consists of output units o, each of which is connected with a synaptic weight w_{io} to all of the input units i. A single layer feedforward network with a single sigma output unit whose activation is determined

by the *threshold function*,

$$F(s) = \begin{cases} 1 & \text{if } s > 0, \\ -1 & \text{otherwise,} \end{cases} , \tag{3.25}$$

can decide whether an input sample belongs to one of the two classes, assuming they are linearly separable. In other words, as

$$\sum_i w_{io} x_i + \theta = 0 \tag{3.26}$$

defines a hyperplane, such a network can only implement a *linear discriminant function*.

A single layer network has representational limitations. Minsky and Papert [13] showed in 1969 that for binary inputs, a two layer feedforward network can overcome these limitations. In 1986, Rumelhart et al. [14] presented a solution to the problem of how to adjust the weights from input to hidden units. The solution suggests to determine the errors for the units of the hidden layer by backpropagating the errors of the units of the output layer. For this reason the method is often called the *backpropagation algorithm*.

A feed-forward network has a layered structure. Each layer consists of units which receive their input from units from a layer directly below and send their output to units in a layer directly above the unit. There are no connections within a layer. The N_i inputs are fed into the first layer of $N_{h,1}$ hidden units. The input units are merely fan-out units; no processing takes place in these units. The activation of a hidden unit is a function F_i of the weighted inputs plus a bias. The output of the hidden units is distributed over the next layer of $N_{h,2}$ hidden units, until the last layer of hidden units, of which the outputs are fed into a layer of N_o output units.

Although back-propagation can be applied to networks with any number of layers, just as for networks with binary units it has been shown [15] that only one layer of hidden units suffices to approximate any function with finitely many discontinuities to

arbitrary precision, provided the activation function of the hidden units are nonlinear.

The activation is a differentiable function of the total input, given by

$$y_k^p = F(s_k^p) \tag{3.27}$$

in which

$$s_k^p = \sum_j w_{jk} y_j^p. \tag{3.28}$$

To get the correct generalization of the delta rule, we must set

$$\Delta_p w_{jk} = -\eta \frac{\partial E^p}{\partial w_{jk}}. \tag{3.29}$$

The error measure E^p is defined as the total quadratic error for pattern p at the output units:

$$E^p = \frac{1}{2} \sum_{o=1}^{N_o} (d_o^p - y_o^p)^2, \tag{3.30}$$

where d_o^p is the desired output for unit o when pattern p is clamped. We can write

$$\frac{\partial E^p}{\partial w_{jk}} = \frac{\partial E^p}{\partial s_k^p} \frac{\partial s_k^p}{\partial w_{jk}}. \tag{3.31}$$

By Equation 3.28 we see that the second factor is

$$\frac{\partial s_k^p}{\partial w_{jk}} = y_j^p. \tag{3.32}$$

When we define

$$\delta_k^p = -\frac{\partial E^p}{\partial s_k^p}, \tag{3.33}$$

we will get an update rule which is equivalent to the *delta rule*, resulting in a *gradient descent* on the error surface if we make the weight changes according to:

$$\Delta_p w_{jk} = \eta \delta_k^p y_j^p. \tag{3.34}$$

The trick is to figure out what δ_k^p should be for each unit k in the network. The interesting result is that there is a simple recursive computation of these δ's which can be implemented by propagating error signals backward through the network.

To compute δ_k^p we apply the chain rule to write this partial derivative as the product of two factors, one factor reflecting the change in error as a function of the output of the unit and one reflecting the change in the output as a function of changes in the input. Thus, we have

$$\delta_k^p = -\frac{\partial E^p}{\partial s_k^p} = -\frac{\partial E^p}{\partial y_k^p}\frac{\partial y_k^p}{\partial s_k^p}. \tag{3.35}$$

By Equation 3.27 we see that

$$\frac{\partial y_k^p}{\partial s_k^p} = F'(s_k^p), \tag{3.36}$$

which is simply the derivative of the activation function F for the k^{th} unit, evaluated at the net input s_k^p to that unit. To compute the first factor of Equation 3.35, we consider two cases. First, assume that unit k is an output unit $k = o$ of the network. In this case, it follows from the definition of E^p that

$$\frac{\partial E^p}{\partial y_o^p} = -(d_o^p - y_o^p). \tag{3.37}$$

Substituting this and Equation 3.27 in Equation 3.35, we get

$$\delta_o^p = (d_o^p - y_o^p) F_o'(s_o^p) \tag{3.38}$$

for any output unit o. Secondly, if k is not an output unit but a hidden unit $k = h$, we do not readily know the contribution of the unit to the output error of the network. However, the error measure can be written as a function of the net inputs from hidden to output layer; $E^p = E^p(s_1^p, s_2^p, ..., s_j^p, ...)$ and we use the chain rule to write

$$\frac{\partial E^p}{\partial y_h^p} = \sum_{o=1}^{N_o} \frac{\partial E^p}{\partial s_o^p} \frac{\partial s_o^p}{\partial y_h^p} = \sum_{o=1}^{N_o} \frac{\partial E^p}{\partial s_o^p} \frac{\partial}{\partial y_h^p} \sum_{j=1}^{N_h} w_{jo} y_j^p = \sum_{o=1}^{N_o} \frac{\partial E^p}{\partial s_o^p} w_{ho} = -\sum_{o=1}^{N_o} \delta_o^p w_{ho}. \quad (3.39)$$

Substituting this in Equation 3.35 yields

$$\delta_h^p = F'(s_h^p) \sum_{o=1}^{N_o} \delta_o^p w_{ho}. \quad (3.40)$$

Equations 3.38 and 3.40 give a recursive procedure for computing the δ's for all units in the network, which are then used to compute the weight changes according to Equation 3.34. This procedure constitutes the generalized delta rule or a feed-forward network of non-linear units.

Take as the activation function F the sigmoid function:

$$y^p = F(s^p) = \frac{1}{1 + e^{-s^p}}. \quad (3.41)$$

In this case the derivative is equal to

$$F'(s^p) = \frac{\partial}{\partial s^p}\left(\frac{1}{1 + e^{-s^p}}\right) = \frac{1}{(1 + e^{-s^p})^2} = \frac{1}{(1 + e^{-s^p})} \frac{-e^{-s^p}}{(1 + e^{-s^p})} = y^p(1 - y^p). \quad (3.42)$$

Then, the error signal for an output unit can be written as:

$$\delta_o^p = (d_o^p - y_o^p) y_o^p (1 - y_o^p). \quad (3.43)$$

The error signal for a hidden unit is:

$$\delta_h^p = F'(s_h^p) \sum_{o=1}^{N_o} \delta_o^p w_{ho} = y_h^p (1 - y_h^p) \sum_{o=1}^{N_o} \delta_o^p w_{ho}. \tag{3.44}$$

The learning procedure requires that the change in weight is proportional to $\frac{\partial E^p}{\partial w}$. True gradient descent requires that infinitesimal steps are taken. The constant of proportionality is the learning rate η. For practical purposes we choose a learning rate that is as large as possible without leading to oscillation. One way to avoid oscillation at large η, is to make the change in weight dependent on the past weight change by adding a momentum term:

$$\Delta w_{jk}(t+1) = \eta \delta_k^p y_j^p + \alpha \Delta w_{jk}(t) \tag{3.45}$$

where t indexes the presentation number and α is a constant which determines the effect of the previous weight change.

3.3.2. Logistic Model Trees

A logistic model tree is basically a standard *decision tree* with *logistic regression functions* at the leaves. The LMT algorithm [16] is based upon the tree induction and logistic regression approaches to learning.

A decision tree is a hierarchical model which is composed of internal *decision nodes* and terminal *leaf nodes*. A test on one of the features is associated with every decision node. For a nominal feature with m values, the node has m *child nodes*, and samples take one of the m branches depending on their value of the feature. For numeric features, the node implements a threshold function and generates two nodes. The samples with feature values greater than the threshold are assigned to one of the child nodes, and the others to the other node. This recursive splitting starts at the *root* and stops when the leaf nodes are reasonably pure in the sense that they contain

samples with mostly identical class labels.

Regression refers to the process of estimating a numeric target variable as opposed to a discrete one. Classification problems can also be solved via regression.

Assume we have a class variable G that takes on values $1, \ldots, J$. The idea is to transform this class variable into J numeric indicator variables $G_1, \ldots, G_J$ to which the regression learner can be fit. The indicator variable G_j for class j takes on value 1 whenever class j is present and value 0 everywhere else. A separate model is then fit to every indicator variable G_j using the regression learner. When classifying an unseen instance, predictions $u_1, \ldots, u_J$ are obtained from the numeric estimators fit to the class indicator variables, and the predicted class is

$$j^* = \text{argmax}_j u_j. \tag{3.46}$$

One way to use regression for classification tasks is to use a logistic regression model that models the posterior class probabilities $\Pr(G = j|X = x)$ for the J classes. Given estimates for the class probabilities, we can classify unseen instances by

$$j^* = \text{argmax}_j \Pr(G = j|X = x). \tag{3.47}$$

Logistic regression models these probabilities using linear functions in x while at the same time ensuring they sum to one and remain in $[0, 1]$. The model is specified in terms of $J - 1$ log-odds that separate each class from the base class J:

$$\log \frac{\Pr(G = j|X = x)}{\Pr(G = J|X = x)} = \beta_j^T x, \qquad j = 1, \ldots, J - 1 \tag{3.48}$$

or, equivalently,

$$\Pr(G = j|X = x) \;=\; \frac{e_j^{\beta_j^T x}}{1 + \sum_{l=1}^{J-1} e^{\beta_l^T x}}, \qquad j = 1, \ldots, J - 1 \tag{3.49}$$

$$\Pr(G = J|X = x) \quad = \quad \frac{1}{1 + \sum_{l=1}^{J-1} e^{\beta_l^T x}}. \tag{3.50}$$

Note that this model produce linear boundaries between the regions in the instance space corresponding to the different classes. For example, the x lying on the boundary between a class j and the class J are those for which $\Pr(G = j|X = x) = \Pr(G = J|X = x)$, which is equivalent to the log-odds being zero. Since the equation for the log-odds is linear in x, this class boundary is effectively a hyperplane. The formulation of the logistic model given here uses the last class as the base class in the odd-ratios; however, the choice of the base class is arbitrary in that the estimates are equivariant under this choice.

Fitting a logistic regression model means estimating the parameter vectors β_j. The standard procedure in statistics is to look for the maximum likelihood estimate: choose the parameters that maximize the probability of the observed data points. For the logistic regression model, there are no closed-form solutions for these estimates. Instead, we have to use numeric optimization algorithms that approach the maximum likelihood solution iteratively and reach it in the limit.

In a paper that links boosting algorithms to additive modelling in statistics, Friedman et al. [17] propose the LogitBoost algorithm for fitting additive logistic regression models by maximum likelihood. These models are a generalization of the linear logistic regression models described above. Generally, they have the form

$$\Pr(G = j|X = x) = \frac{e^{F_j(x)}}{\sum_{k=1}^{J} e^{F_k(x)}}, \qquad \sum_{k=1}^{J} F_k(x) = 0, \tag{3.51}$$

where $F_j(x) = \sum_{m=1}^{M} f_{mj}(x)$ and the f_{mj} are not necessarily linear functions of the input variables. Indeed, the authors show that if regression trees are used as the f_{mj}, the resulting algorithm has strong connections to boosting decision trees.

Appendix A gives the pseudocode for the algorithm. LogitBoost performs forward stagewise fitting: in every iteration, it computes response variables z_{ij} that encode the

error of the currently fit model on the training examples (in terms of probability estimates), and then tries to improve the model by adding a function f_{mj} to the committee F_j, fit to the response by least-squared error.

The logistic regression functions are constructed by incrementally refining logistic models already fit at higher levels in the tree. Assume we have split a node and want to build the logistic regression function at one of the child nodes. Since we have already fit a logistic regression at the parent node, it is reasonable to use it as a basis for fitting the logistic regression at the child. It is expected that the parameters of the model at the parent node already encode global influences of some attributes on the class variable; at the child node, the model can be further refined by taking into account influences of attributes that are only valid locally, i.e. within the set of training examples associated with the child node.

The LogitBoost algorithm provides a natural way to do just that. It iteratively changes the linear class functions $F_j(x)$ to improve the fit to the data by adding a simple linear regression function f_{mj} to F_j, fit to the response variable. This means changing one of the coefficients in the linear function F_j or introducing a new variable/coefficient pair. After splitting a node we can continue running LogitBoost iterations, fitting the f_{mj} to the response variables of the training examples at the child node only.

Tree growing starts by building a logistic model at the root using the LogitBoost algorithm to iteratively fit simple linear regression functions, using five fold cross-validation to determine the appropriate number of iterations.

A split for the data at the root is constructed using a splitting criterion that tries to improve the purity of the class variable. Tree growing continues by sorting the appropriate subsets of data to the child nodes and building the logistic models at the child nodes in the following way: the LogitBoost algorithm is run on the subset associated with the child node, but starting with the committee $F_j(x)$, weights w_{ij} and probability estimates p_{ij} of the last iteration performed at the parent node. Again, the optimum number of iterations to perform is determined by a five fold cross-validation.

3.4. Performance Evaluation

The performance of a classifier on the training set is not a good indicator of performance on an independent test set. As the classifier has been learned from the very same training data, any estimate of performance based on that data will be optimistic. The error on the training data is called the *resubstitution error* [18], because it is calculated by resubstituting the training instances into a classifier that was constructed from them.

To predict the performance of a classifier on new data, one needs to assess its error on a dataset that played no part in the formation of the classifier. This independent dataset is called *test set*. It is assumed that both the training data and the test data are representative samples of the underlying structure.

In machine learning, there are cases where some of the parameters of a model to be trained should be defined before training takes place. For instance, to train a multilayer perceptrons its structure (i.e. the number of hidden layers and the number of hidden units in each layer) along with the learning rate, momentum coefficient and the number of training epochs should be predefined. In such cases, we talk about three datasets: the *training set*, the *validation set*, and the *test set*. The training set is used to train models. The validation set is used to optimize parameters that cannot be optimized during training. Then the test set is used to calculate error of the optimized model. Each of the three sets must be chosen independently: the validation set must be different from the training set to get good performance in the optimization, and the test set must be different from both to get a reliable estimate of the true error.

When there is not a vast supply of data, the holdout method is used that reserves a certain amount for testing and uses the remainder for training. The standard way of doing this is to use *stratified tenfold cross-validation*. In tenfold cross-validation, the data is split randomly into ten parts, in each of which the class is represented in approximately the same proportions as in the full dataset. Each part is held out in turn and the model is trained on the remaining nine-tenths; then its error is calculated

on the holdout set. Thus the learning procedure is executed ten times, on different training sets. Finally, the ten error estimates are averaged to yield an overall error estimate.

A single tenfold cross-validation might not be enough to get a reliable error estimate. In this case, tenfold cross-validation procedure is repeated several times each time shuffling the dataset before splitting into ten folds. In this way, different combinations of training and validation data can be obtained in each fold.

In this thesis, the root mean squared error is used as a performance measure. The root mean squared error is computed by summing the squared difference between a binary vector that contains a 1 for the actual class value of an instance and 0's for all other classes and the predicted class probability distribution for each instance and then taking the square root of the average. The root mean squared error is more sensitive to outliers in the data than the mean absolute error.

3.5. Diagnostic Prediction

In clinical settings , it is important to be able to reject a diagnostic classification including samples not belonging to any of the diagnoses. Therefore, to be able to reject classifications for each class an empirical probability distribution for distances is constructed. An *Euclidean distance* is computed for each class, between the average class probability distribution for a sample and the ideal output for that class; normalized such that it is unity between classes. A distance d_c from a sample to the ideal output for each class is defined as:

$$d_c = \left[\sum_{i=1}^{4} o_i - {\delta_{ic}}^2 \right]^{\frac{1}{2}}. \tag{3.52}$$

where c is a class, o_i is the average probability distribution for class i, and $\delta_{i,c}$ is unity if i corresponds to class c and zero otherwise.

4. RESULTS

All algorithms used in this study are part of the Weka machine learning workbench [19] release 3.4.5.

The publicly available gene expression data of small, round blue cell tumors (SRBCTs) of childhood, which include neuroblastoma (NB), rhabdomyosarcoma (RMS), non-Hodgkin lymphoma (NHL), and the Ewing family of tumors (EWS) will be discussed throughout the analysis [20]. As mentioned in Section 3.1, any machine learning application requires at least two distinct sets of data: the training set and the test set. The set that we used in training models to classify samples into one of the SRBCT categories consists of 23 EWS, 8 Burkitt lymphoma (BL; a subset of NHL), 12 NB and 20 RMS samples. On the other hand, the test set contains 6 EWS, 3 BL, 6 NB, 5 RMS and 5 non-SRBCTs (consisting of 2 normal muscle tissues and 3 cell lines including an undifferentiated sarcoma, osteosarcoma and a prostate carcinoma). In the analysis, gene expression data from cDNA microarrays containing 2308 genes was used.

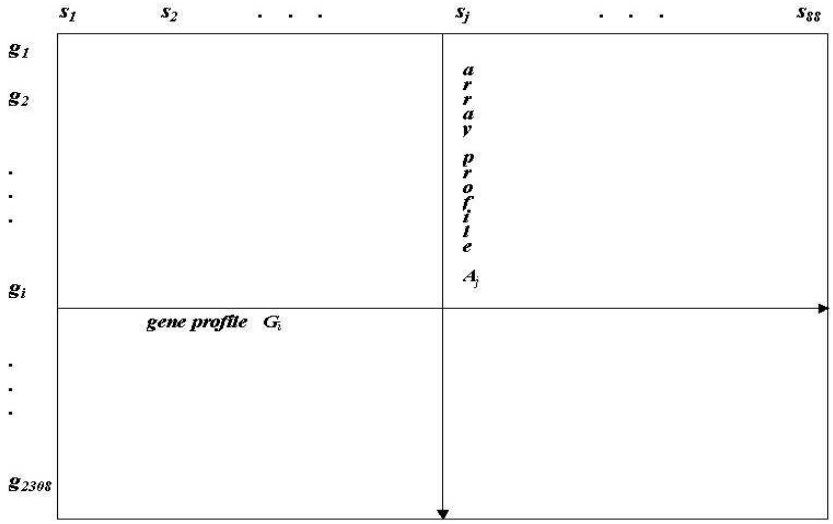

Figure 4.1. Relative gene expression data belonging to 88 samples and containing 2308 genes

To diminish the effects of errors involved in array preparation, sample labelling, hybridization and scanning , the digital image data from different arrays should be normalized and standardized [7]. This aspect of the problem is out of the scope of this thesis. However, it should be mentioned that, in the original experimental study, the reproducibility of the experiments was tested by performing two independent microarray experiments for two samples, and each array was normalized across all experiments to obtain relative gene expression values [21].

To prepare the gene expression data for further analysis, the first step to be taken is to log-transform each relative gene expression value. By performing this transformation, the positive skewness of the ratio data can be eliminated. Hereby, a more normally distributed data is achieved.

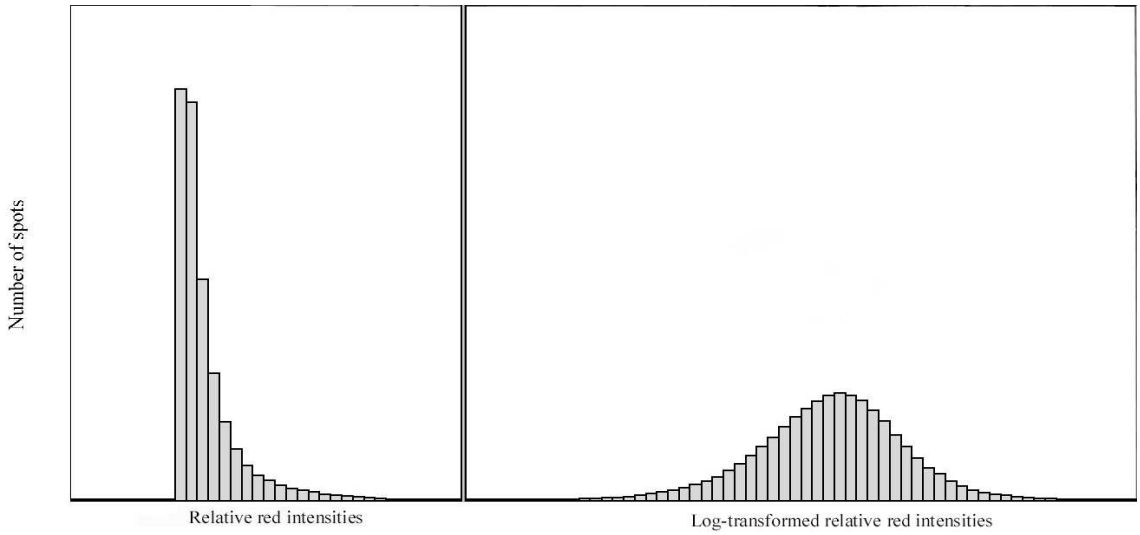

Figure 4.2. Relative gene expression data before and after log-transformation

Owing to the facts discussed in Section 3.2, dimensionality of the gene expression data of SRBCTs should be reduced before training a model. In this analysis, dimensionality reduction was carried out in two steps.

In the first step, irrelevant and redundant genes were eliminated using the CFS algorithm whose detailed description is given in Section 3.2.1. First, the log-transformed relative gene expression values were discretized using the method of Fayyad and Irani.

Once they became compatible with the nominal class values, the correlations between genes and classes and the inter-correlations between genes were calculated using symmetrical uncertainty coefficient formula given by Equation 3.12. Then these correlation values were plugged into Equation 3.1. The gene with the highest merit was added to the current gene subset. The gene subset space was searched by best first algorithm in the forward direction. Forward selection begins with no features and greedily adds one feature at a time. To prevent the best first search from exploring the entire gene subset space, a stopping criterion was imposed. The search terminated if five consecutive fully expanded subsets show no improvement over the current best subset. After all, CFS algorithm with these settings concluded that the 84 genes are highly correlated with the class, yet uncorrelated with each other. As a consequence, the dimensionality of the array profile vectors was reduced from 2308 to 84.

In the next step, PCA further reduces the dimensionality of the problem. The samples located in the space spanned by the genes emerging from CFS were projected onto a new lower dimensional space. As discussed in Section 3.2.2, PCA finds the directions in which data is most spread out, i.e. has the largest variance. The new lower dimensional space is spanned by these direction vectors that are nothing but the eigenvectors of the correlation matrix of the training samples which explain the certain amount of the variance. For this particular dataset, the first 30 principal components explain 95% of the variance.

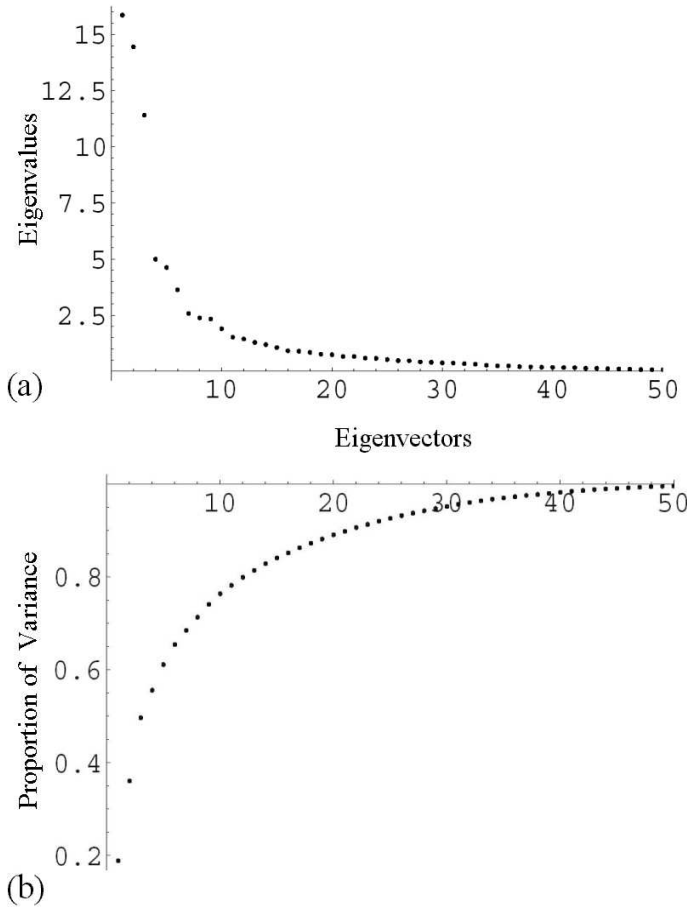

Figure 4.3. Scree graph (a) and the proportion of variance explained as a function of the number of principal components that are kept (b)

As expected, the top three principal components turned out to be highly discriminative when plotted in groups of two.

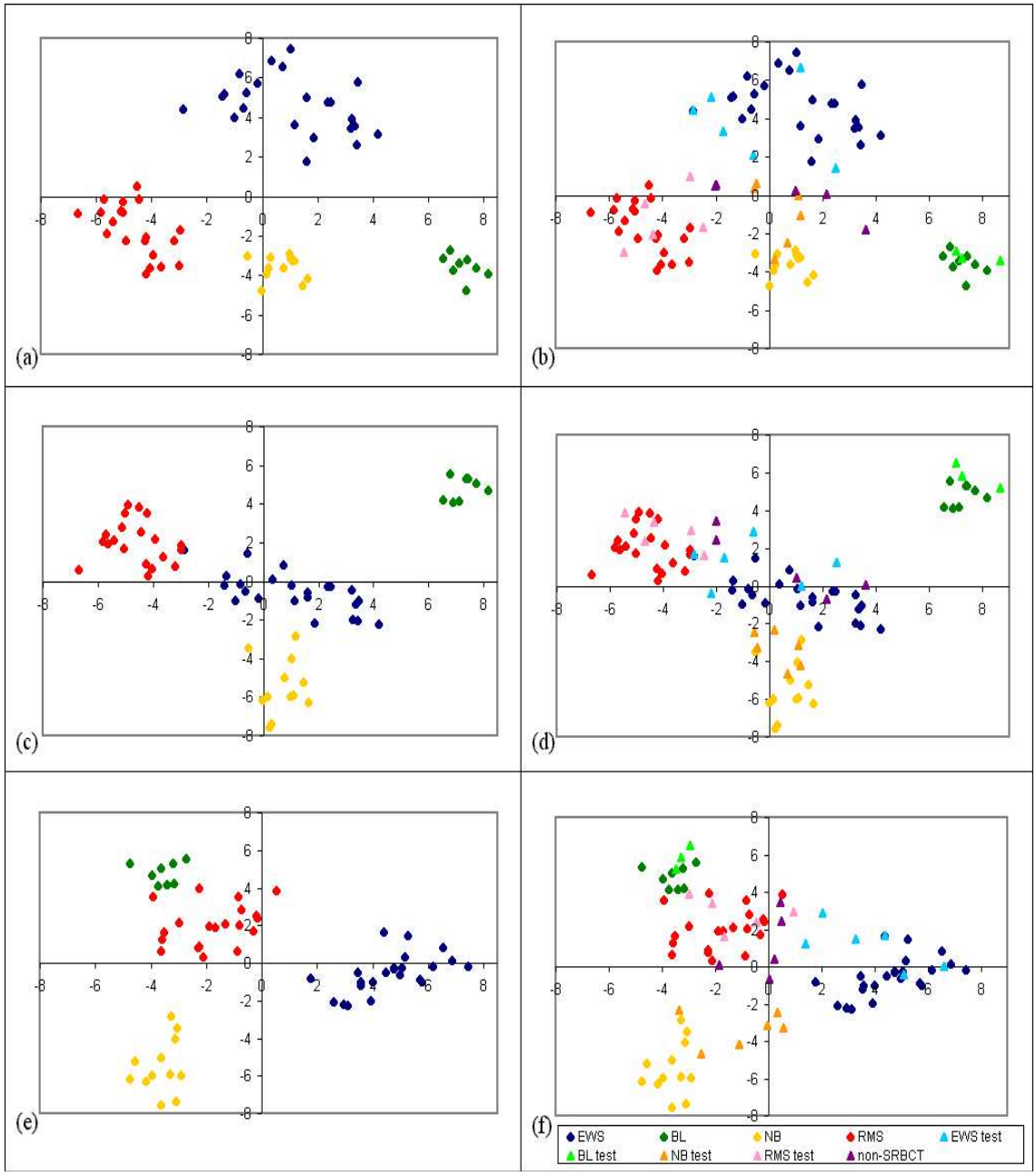

Figure 4.4. Distribution of the samples in the principal components space: The training samples after projected onto the first and the second principal components (a) and both training and test samples after the projection (b); the training samples after projected onto the first and the third principal components (c) and both training and test samples after the projection (d); the training samples after projected onto the second and the third principal components (e) and both training and test samples after the projection (f)

4.1. ANN Results

A single layer network could only represent linear discriminant functions that separate each class from the others. By introducing hidden units with non-linear activation functions, non-linear discriminant functions might also be learned. Hornik et al. has shown that only one layer of hidden units suffices to approximate any function with finitely many discontinuities, provided the activation functions of the hidden units are non-linear [15]. Then, the question to be answered turns out to be whether a single-layer or two-layer network will be used in our analysis, and, if the answer becomes two-layer network, then how many hidden units it will include. We determined what the answer should be through ten times ten-fold cross-validation illustrated in Figure 4.6. Each fold was trained using back-propagation algorithm discussed in Section 3.3.1. From Figure 4.7 we can conclude that a network with 14 hidden units yields the minimum validation error. Zero in the horizontal axis represent the single-layer network.

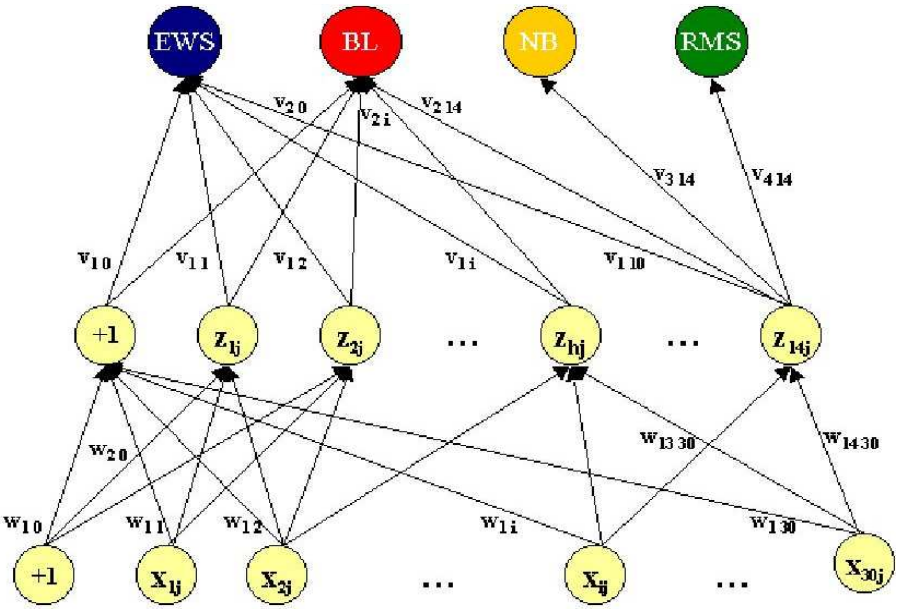

Figure 4.5. Architecture of a multilayer feedforward network including one hidden layer with 14 hidden units

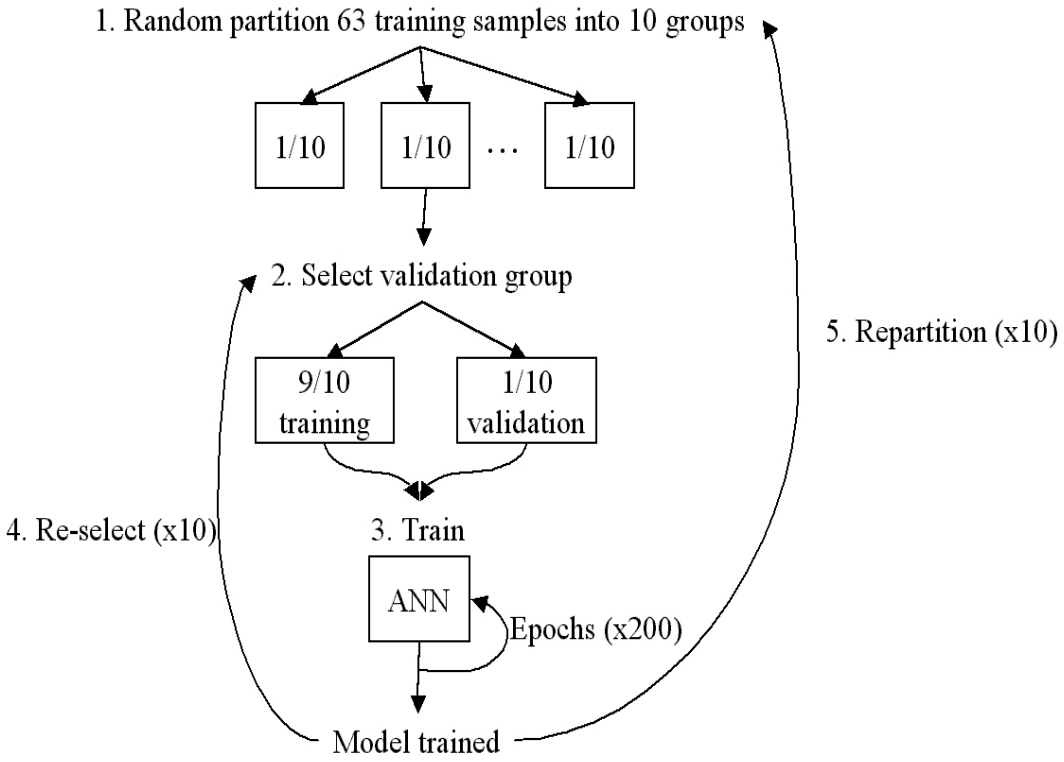

Figure 4.6. Schematic illustration of the training process

The other model parameters such as learning rate and momentum coefficient were also cross-validated by ten times tenfold cross-validation. In other words, nine tenths of the 63 training samples was used to train a model and remaining one tenths to validate the model.This was repeated ten times, each time randomly shuffling the 63 samples before dividing into ten folds. The validation error was taken into account in determining the appropriate values for these model parameters. The values that gives the smallest root mean squared error were preferred. The maximum number of epochs was determined to be 200. This number allows the convergence of validation error as well as showing no signs of overtraining as seen in Figure 4.10. Also, option that terminates learning when validation errors of ten successive epochs show no improvement was enabled.

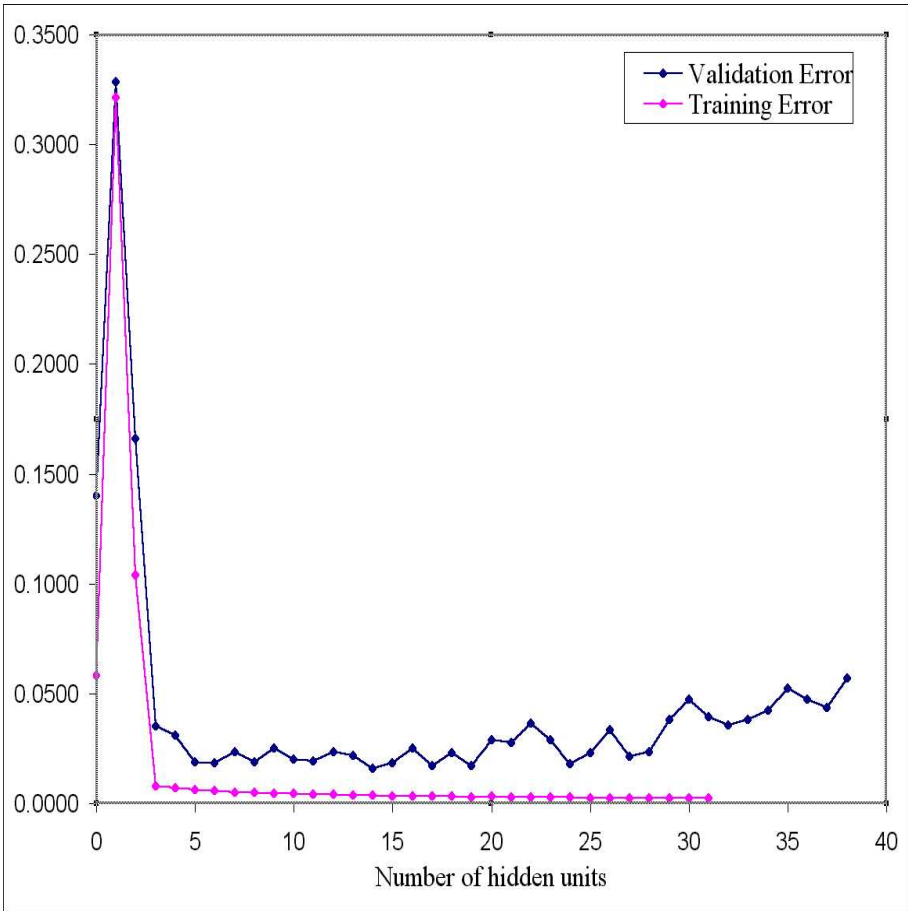

Figure 4.7. Changes in training and validation errors as the number of hidden units in the hidden layer is increased

100 different models were trained, each time shuffling the training samples before cross-validation and setting the initial weights between the units randomly. The average votes from these 100 models helped us to classify each sample into one of the diagnostic categories. Each sample was classified to the class for which it attains the highest output.

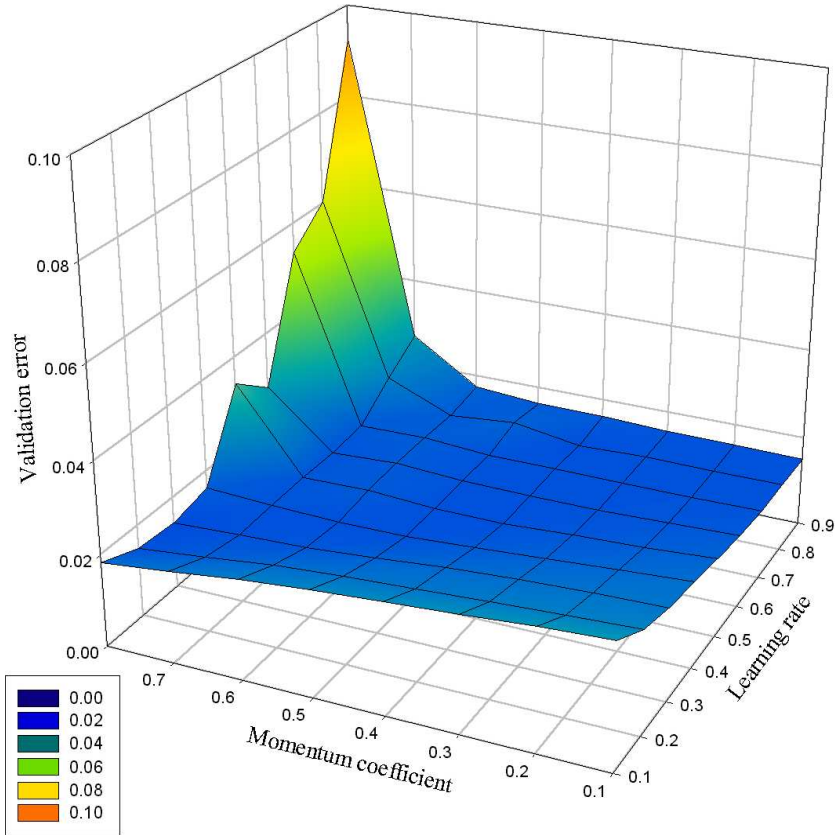

Figure 4.8. Interpolated validation error as a function of two predefined model parameters: learning rate and momentum coefficient

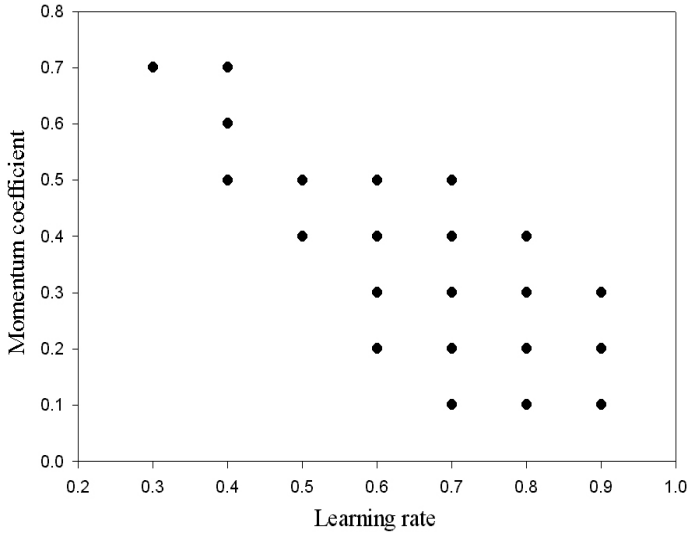

Figure 4.9. Optimal values for learning rate and momentum coefficient

Each row in Table 4.1 corresponds to a particular test instance. The values in the last four columns are the results obtained from sigmoid functions at the output

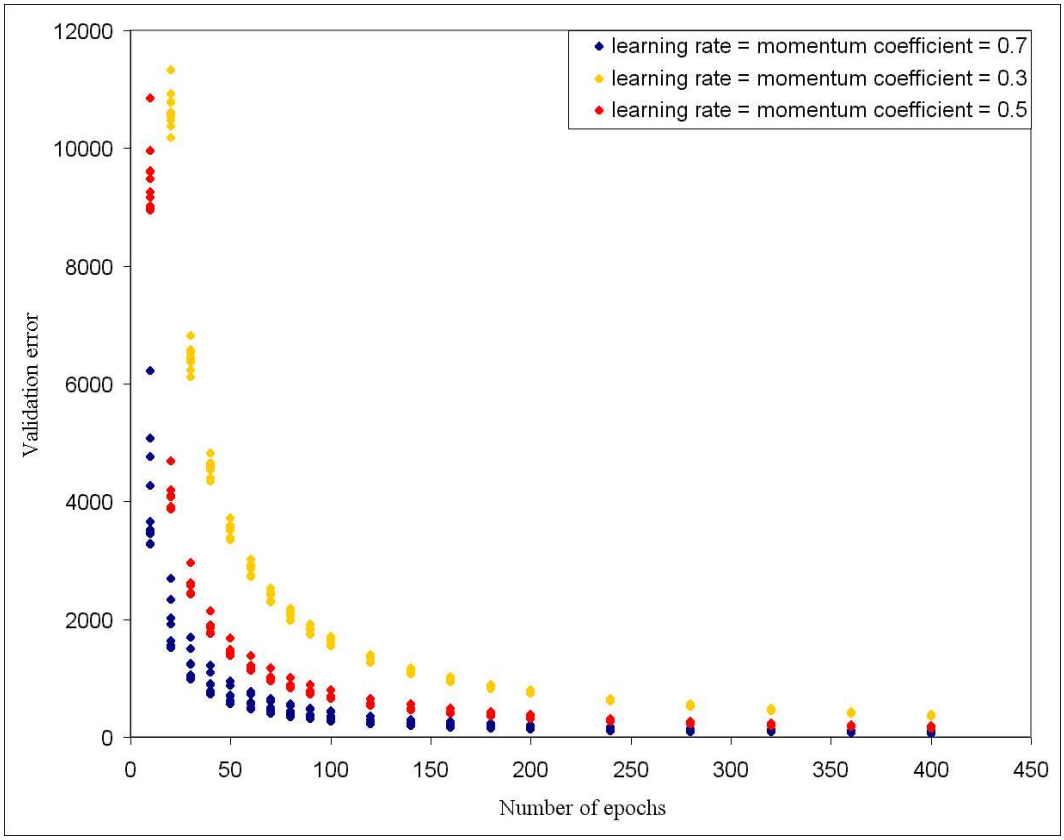

Figure 4.10. Behavior of validation error in our two layer feedforward network with 14 hidden units as the number of training epochs is increased: Yellow dots correspond to the mean validation errors of ten times repeated ten-fold cross-validation process where $\eta = \alpha = 0.3$; red dots correspond to the mean validation errors of ten times repeated ten-fold cross-validation process where $\eta = \alpha = 0.5$; dark blue dots correspond to the mean validation errors of ten times repeated ten-fold cross-validation process where $\eta = \alpha = 0.7$

units corresponding to different classes. These values were produced by averaging the results of the 100 previously trained models. The output unit with the highest sigmoid result was denoted by boldface and the corresponding test instance was classified into the class that is represented by that unit. Table 4.1 reveals that excluding the non-SRBCT samples from the test set, all the remaining samples were classified correctly. For example, for the instance#4 whose actual class was NB, sigmoid results at EWS, BL, NB, and RMS output units are 0.009, 0.006, 0.979, and 0.005, respectively. Consequently, it was classified into NB class as it got the highest sigmoid result at that unit.

Table 4.1. Average of all ANN outputs

Instance#	Actual Class	Predicted Class	EWS	BL	NB	RMS
1	-	RMS	0.032	0.044	0.001	**0.924**
2	-	BL	0.061	**0.770**	0.167	0.001
3	-	EWS	**0.915**	0.033	0.046	0.006
4	NB	NB	0.009	0.006	**0.979**	0.005
5	RMS	RMS	0.044	0.026	0.001	**0.928**
6	-	RMS	0.037	0.019	0.001	**0.943**
7	-	EWS	**0.640**	0.251	0.034	0.074
8	NB	NB	0.002	0.019	**0.937**	0.042
9	EWS	EWS	**0.958**	0.025	0.001	0.017
10	RMS	RMS	0.003	0.005	0.004	**0.987**
11	BL	BL	0.008	**0.975**	0.006	0.012
12	EWS	EWS	**0.920**	0.004	0.000	0.076
13	RMS	RMS	0.006	0.004	0.004	**0.985**
14	EWS	EWS	**0.988**	0.001	0.003	0.007
15	EWS	EWS	**0.970**	0.004	0.002	0.024
16	EWS	EWS	**0.594**	0.356	0.001	0.049
17	RMS	RMS	0.003	0.006	0.005	**0.987**
18	BL	BL	0.009	**0.976**	0.008	0.007
19	RMS	RMS	0.001	0.026	0.017	**0.956**
20	NB	NB	0.125	0.019	**0.855**	0.001
21	NB	NB	0.072	0.009	**0.918**	0.002
22	NB	NB	0.024	0.010	**0.964**	0.002
23	NB	NB	0.062	0.006	**0.930**	0.002
24	BL	BL	0.008	**0.976**	0.007	0.009
25	EWS	EWS	**0.991**	0.004	0.002	0.003

The next thing to be determined is whether the test instance#4 will be diagnosed to be NB or its diagnosis will be rejected. The Euclidean distance of the sample to the ideal NB sample was calculated using Equation 3.52. The ideal NB sample was

represented by the vector $(0, 0, 1, 0)^T$ whose elements correspond to the sigmoid results at EWS, BL, NB, and RMS, respectively. Then,

$$d = \sqrt{(0.009 - 0)^2 + (0.006 - 0)^2 + (0.979 - 1)^2 + (0.005 - 0)^2} = 0.024 \qquad (4.1)$$

is the distance between the instance#4 and the ideal NB sample. Since this distance is smaller than the threshold distance 0.034 for the NB class which is denoted by the blue line in Figure 4.11.h , the instance#4 was confidently diagnosed.

Repeating the calculation in the previous paragraph for each sample Table 4.1 was constructed. The ideal samples for EWS, BL, NB, RMS were represented by the vectors $(1, 0, 0, 0)^T$, $(0, 1, 0, 0)^T$, $(0, 0, 1, 0)^T$, and $(0, 0, 0, 1)^T$, respectively. Then taking the threshold distance of the class to which the sample was classified into account, it was diagnosed or its diagnosis was rejected. Threshold distance for each class was denoted by a blue line in Figure 4.11.

Table 4.2. Distances between the actual outputs and the desired outputs

Instance#	Actual Class	Predicted Class	d_{EWS}	d_{BL}	d_{NB}	d_{RMS}
1	-	RMS	-	-	-	0.093
2	-	BL	-	0.291	-	-
3	-	EWS	0.103	-	-	-
4	NB	NB	-	-	0.024	-
5	RMS	RMS	-	-	-	0.088
6	-	RMS	-	-	-	0.071
7	-	EWS	0.446	-	-	-
8	NB	NB	-	-	0.078	-
9	EWS	EWS	0.052	-	-	-
10	RMS	RMS	-	-	-	0.015
11	BL	BL	-	0.030	-	-
12	EWS	EWS	0.111	-	-	-
13	RMS	RMS	-	-	-	0.017
14	EWS	EWS	0.014	-	-	-
15	EWS	EWS	0 039	-	-	-
16	EWS	EWS	0.543	-	-	-
17	RMS	RMS	-	-	-	0.015
18	BL	BL	-	0.028	-	-
19	RMS	RMS	-	-	-	0.053
20	NB	NB	-	-	0.193	-
21	NB	NB	-	-	0.109	-
22	NB	NB	-	-	0.044	-
23	NB	NB	-	-	0.093	-
24	BL	BL	-	0.028	-	-
25	EWS	EWS	0.010	-	-	-

To conclude, all SRBCT test samples were correctly classified by the MLP that we trained. For the samples whose distance is greater than the largest training sample distance, the diagnosis was rejected. The diagnosis of four of all five non-SRBCT test

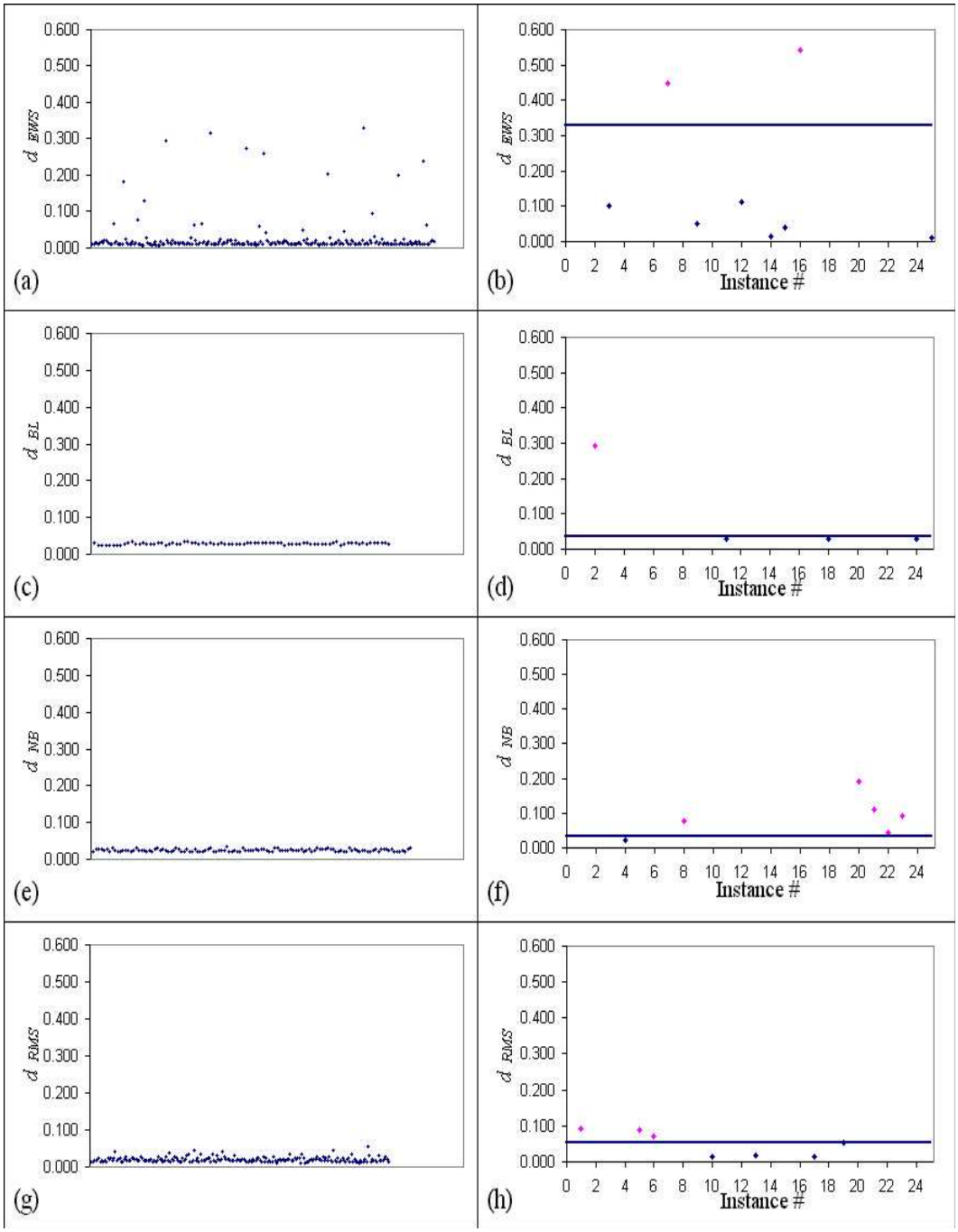

Figure 4.11. Distances between the actual outputs and the desired outputs: For the training samples (a) and the test samples (b) classified to the EWS class; for the training samples (c) and the test samples (d) classified to the BL class; for the training samples (e) and the test samples (f) classified to the NB class; for the training samples (g) and the test samples (h) classified to the RMS class

samples were rejected since they lie outside the threshold. Instance # 3 of Table 4.1 was diagnosed as EWS although it is one of the non-SRBCT test samples. Besides seven of the SRBCT test samples (one EWS, five NBs, and one EWS) could not be confidently diagnosed although they were correctly classified.

4.2. LMT Results

The algorithm discussed in Section 3.3.2 was used to build classification trees with logistic regression functions at the leaves.

The LogitBoost algorithm was used to iteratively fit simple linear regression functions at the root and the leaves. The optimum number of iterations to perform was determined by a five fold cross-validation. To save time, this number was cross-validated once at the root and then used again and again at the other nodes. When cross-validating the number of LogitBoost iterations misclassification error was minimized instead of the root mean squared error.

All SRBCT test samples were correctly classified by the LMTs that we trained. For the samples whose distance is greater than the largest training sample distance, the diagnosis was rejected. The diagnosis of all five non-SRBCT test samples were rejected since they lie outside the threshold. Four of the SRBCT test samples (four NBs, and two RMS') could not be confidently diagnosed although they were correctly classified.

Table 4.3. Average of all posterior probabilities generated by LMT

Instance#	Actual Class	Predicted Class	EWS	BL	NB	RMS
1	-	RMS	0.353	0.089	0.056	**0.502**
2	-	BL	0.207	**0.435**	0.240	0.118
3	-	EWS	**0.344**	0.255	0.248	0.153
4	NB	NB	0.088	0.128	**0.616**	0.168
5	RMS	RMS	0.342	0.059	0.054	**0.545**
6	-	RMS	0.350	0.088	0.076	**0.486**
7	-	EWS	**0.387**	0.206	0.179	0.228
8	NB	NB	0.095	0.173	**0.437**	0.295
9	EWS	EWS	**0.519**	0.244	0.118	0.120
10	RMS	RMS	0.055	0.025	0.029	**0.891**
11	BL	BL	0.122	**0.815**	0.027	0.036
12	EWS	EWS	**0.618**	0.056	0.061	0.265
13	RMS	RMS	0.159	0.033	0.051	**0.758**
14	EWS	EWS	**0.727**	0.029	0.066	0.178
15	EWS	EWS	**0.662**	0.031	0.041	0.266
16	EWS	EWS	**0.563**	0.105	0.058	0.275
17	RMS	RMS	0.100	0.042	0.044	**0.815**
18	BL	BL	0.075	**0.878**	0.031	0.016
19	RMS	RMS	0.168	0.089	0.110	**0.633**
20	NB	NB	0.258	0.156	**0.422**	0.164
21	NB	NB	0.293	0.106	**0.327**	0.273
22	NB	NB	0.158	0.147	**0.547**	0.148
23	NB	NB	0.285	0.097	**0.381**	0.237
24	BL	BL	0.103	**0.832**	0.032	0.032
25	EWS	EWS	**0.879**	0.041	0.039	0.041

Table 4.4. Distance between the actual and the ideal posterior probabilities

Instance#	Actual Class	Predicted Class	d_{EWS}	d_{BL}	d_{NB}	d_{RMS}
1	-	RMS	-	-	-	0.619
2	-	BL	-	0.659	-	-
3	-	EWS	0.761	-	-	-
4	NB	NB	-	-	0.446	-
5	RMS	RMS	-	-	-	0.575
6	-	RMS	-	-	-	0.632
7	-	EWS	0.709	-	-	-
8	NB	NB			0.665	-
9	EWS	EWS	0.565	-	-	-
10	RMS	RMS	-	-	-	0.128
11	BL	BL		0.227	-	-
12	EWS	EWS	0.472	-	-	-
13	RMS	RMS	-	-	-	0.296
14	EWS	EWS	0.334	-	-	-
15	EWS	EWS	0.433	-	-	-
16	EWS	EWS	0.530	-	-	-
17	RMS	RMS	-	-	-	0.219
18	BL	BL	-	0.147	-	-
19	RMS	RMS	-	-	-	0.428
20	NB	NB	-	-	0.672	-
21	NB	NB	-	-	0.790	-
22	NB	NB	-	-	0.523	-
23	NB	NB	-	-	0.727	-
24	BL	BL	-	0.202	-	-
25	EWS	EWS	0.139	-	-	-

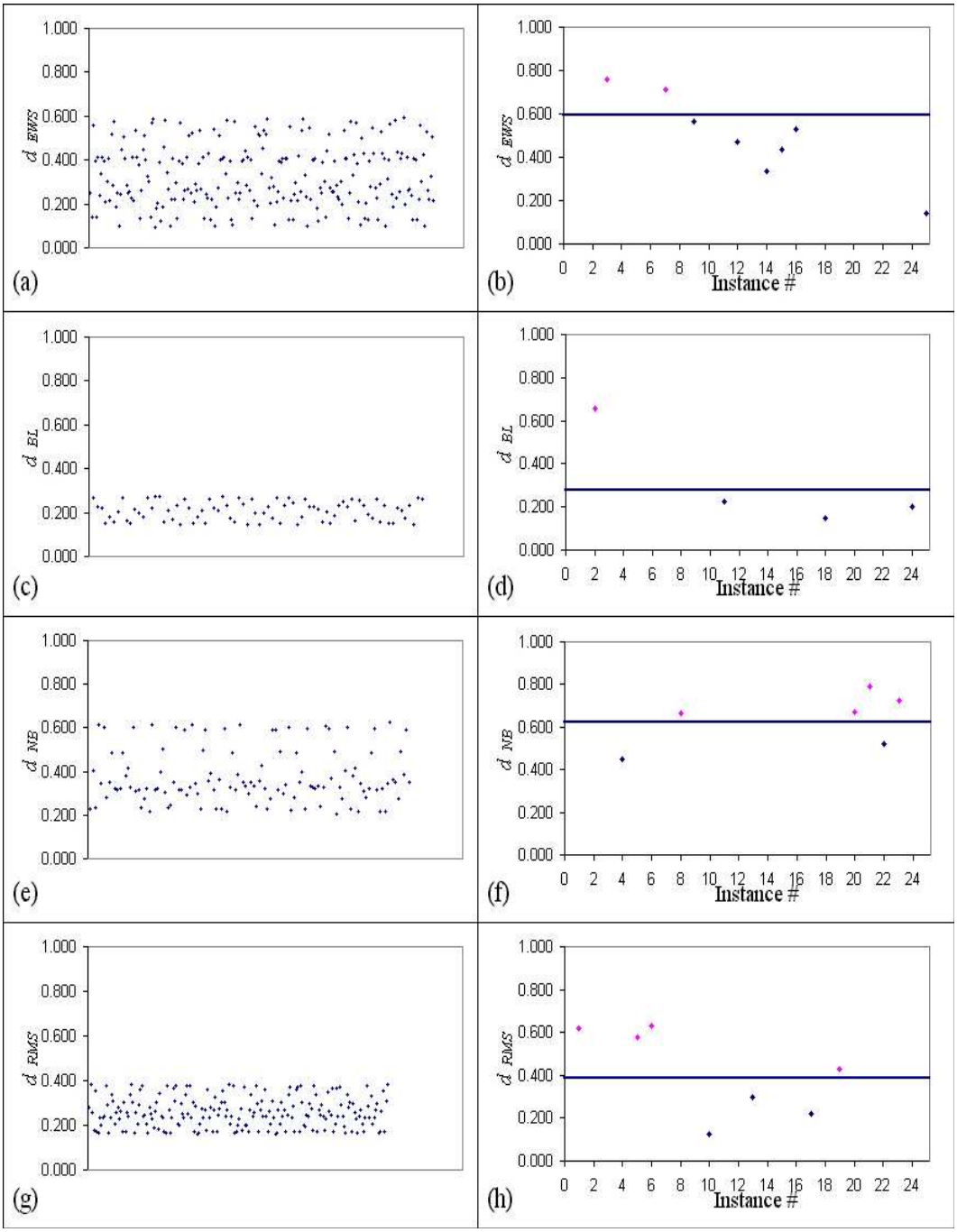

Figure 4.12. Distances between the actual posterior probabilities and the ideal posterior probabilities: For the training samples (a) and the test samples (b) classified to the EWS class; for the training samples (c) and the test samples (d) classified to the BL class; for the training samples (e) and the test samples (f) classified to the NB class; for the training samples (g) and the test samples (h) classified to the RMS class

5. CONCLUSIONS AND DISCUSSIONS

The models that we trained classified all the training and test samples correctly, i.e. classification was achieved with 100 per cent accuracy.

But, when it comes to diagnostic predictions, MLP we trained could not diagnose seven SRBCT samples confidently as they lie outside the distribution confined by the largest training sample distance threshold. It also diagnosed one of the five non-SRBCT samples into one of the SRBCT categories.

LMT results were slightly better than MLP results since LMT rejected the diagnosis of all non-SRBCT test samples. Besides it could diagnose larger number of SRBCT test samples compared to MLP.

Regarding that all the training samples and test samples excluding non-SRBCTs classified correctly we might conclude that both multilayer perceptron model and logistic regression tree could learn the SRBCT classes from the given samples. In other words, both models could determine he boundaries that separate the samples of different classes without loosing their generalization abilities.

On the other hand, the criterion we defined to judge the confidence level in classification of a particular sample did not work quite well. In favor of avoiding diagnosis of non-SRBCT samples, we desisted from diagnosing actual SRBCT samples. Diagnostic prediction should be done without deteriorating the unbiased nature of the learning process by defining subjective criteria on which the future decisions will be based. Instead, non-SRBCT samples should be introduced to the model in the training stage in order to teach the forbidden regions of the space to SRBCT samples. Since the non-SRBCT samples can be located at many distinct regions of the input space, non-SRBCT samples from diverse categories should be included in the training process.

In all the trained models, NBs suffered from not being diagnosed with quite

confidence. This problem was also obvious in Figure 4.4.b and Figure 4.4.f. Referring to these figures the training and the test samples belonging to the class NB were not perfectly coinciding in the input space. The reason for this may be that our models were trained using only NB cell lines while four of the NB test samples were tumor biopsy materials. On the other hand, the training samples of the other class were including both cell lines and tumor biopsy materials. For the other three classes, we may conclude that the models performed quite well.

APPENDIX A: PSEUDOCODE FOR LOGITBOOST ALGORITHM

1. Start with weights $w_{ij} = 1/n,\ i = 1, \ldots, n,\ j = 1, \ldots, J,\ F_j(x) = 0$ and $p_j(x) = 1/J\ \forall j$
2. Repeat for $m = 1, \ldots, M$:
   (a) Repeat for $j = 1, \ldots, J$:
      i. Compute working responses and weights in the j^{th} class

$$z_{ij} = \frac{y^*_{ij} - p_j(x_i)}{p_j(x_i)(1 - p_j(x_i))}$$

$$w_{ij} = p_j(x_i)(1 - p_j(x_i))$$

      ii. Fit the function $f_{mj}(x)$ by a weighted least-squares regression of z_{ij} to x_i with weights w_{ij}
   (b) Set $f_{mj}(x) \leftarrow \frac{J-1}{J}(f_{mj}(x) - \frac{1}{J}\sum_{k=1}^{J} f_{mk}(x)),\ F_j(x) \leftarrow F_j(x) + f_{mj}(x)$
   (c) Update $p_j(x) = \frac{e^{F_j(x)}}{\sum_{k=1}^{J} e^{F_k(x)}}$
3. Output the classifier $\text{argmax}_j\ F_j(x)$

The variables y^*_{ij} encode the observed class membership probabilities for an instance x_i, i.e.

$$y^*_{ij} = \begin{cases} 1 & \text{if } y_i = j, \\ 0 & \text{if } y_i \neq j. \end{cases} \tag{A.1}$$

The $p_j(x)$ are the estimates of the class probabilities for an instance x given by the model fit so far.

REFERENCES

1. Maggipinto, T., G. Nardulli, S. Dusini, F. Ferrari, I. Lazzizzera, A. Sidoti, A. Sartori and G. P. Tecchiolli, "Role of Neural Networks in the Search of the Higgs Boson at LHC", *Physics Letters B*, Vol. 409, pp. 517-522, 1997.

2. Garrido, L. and A. Juste, "On the Determination of Probability Density Functions by Using Neural Networks", *Computer Physics Communications*, Vol. 115, pp. 25-31, 1998.

3. Bass, S. A., A. Bischoff, J. A. Maruhn, H. Stöcker and W. Greiner, "Neural Networks for Impact Parameter Determination", *Physical Review C*, Vol. 53, pp. 2358-2363, 1996.

4. Rohde, D. J., M. J. Drinkwater, M. R. Gallagher, T. Downs and M. T. Doyle, "Applying Machine Learning to Catalogue Matching in Astrophysics", *Monthly Notices of the Royal Astronomical Society*, Vol. 360, pp. 69-75, 2005.

5. *W. M. Keck Foundation Biotechnology Resource Laboratory*, http://keck.med.yale.edu/affymetrix/technology.htm, 2006.

6. *Access Excellence the National Health Museum Resource Center*, http://www.accessexcellence.org/RC/VL/GG/index.html, 2006.

7. Berrar, D. P., W. Dubitzky and M. Granzow, *A Practical Approach to Microarray Data Analysis*, Kluwer Academic Publishers, Dordrecht, 2003.

8. *National Human Genome Research Institute*, http://www.genome.gov/10000533, 2006.

9. Hall, M. A., *Correlation-based Feature Selection for Machine Learning*, Ph.D. Thesis, The University of Waikato, 1999.

10. Fayyad, U. M. and K. B. Irani, "Multi-interval Discretization of Continuous-valued Attributes for Classification Learning", *Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence*, Chambry, 1993, pp. 1022-1027, Morgan Kaufmann, San Mateo, 1993.

11. Alpaydin, E., *Introduction to Machine Learning*, The MIT Press, Cambridge, 2004.

12. *Neural Nets by Kevin Gurney*, http://www.shef.ac.uk/psychology/gurney/notes/index.html, 2006.

13. Minsky, M. and S. Papert, *Perceptrons: An Introduction to Computational Geometry*, The MIT Press, Cambridge, 1969.

14. Rumelhart, D. E., G. E. Hinton and R. J. Williams, "Learning Representations by Backpropagating Errors", *Nature*, Vol. 323, pp. 533-536, 1986.

15. Hornik, K ., M. Stinchcombe, and H. White, "Multilayer Feedforward Networks are Universal Approximators", *Neural Networks*, Vol. 2, pp. 359-366, 1989.

16. Landwehr, N., M. Hall and E. Frank, "Logistic Model Trees", *Machine Learning* , Vol. 59, pp. 161-205, 2005.

17. Friedman, J., T. Hastie and R. Tibshirani, "Additive Logistic Regression: a Statistical View of Boosting", *Annals of Statistics* , Vol. 28, pp. 337-374, 2000.

18. Witten, I. H. and E. Frank, *Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations*, Morgan Kaufmann Publishers, San Francisco, 2000.

19. *WEKA*, http://www.cs.waikato.ac.nz/ml/weka/, 2006.

20. *NHGRI*, http://research.nhgri.nih.gov/microarray/Supplement/, 2006.

21. Khan, J ., J. S. Wei, M. Ringner, L. H. Saal, M. Ladanyi, F. Westermann, F.

Berthold, M. Schwab, C. R. Antonescu, C. Peterson and P. S. Meltzer, "Classification and Diagnostic Prediction of Cancers Using Gene Expression Profiling and Artificial Neural Networks", *Nature Medicine*, Vol. 7, pp. 673-679, 2001.