Improved tree species discrimination at leaf level with hyperspectral data: combining binary classifiers

BY

XOLANI DASTILE

SUPERVISOR: PROFESSOR G. JAGER

CO-SUPERVISOR: DOCTOR P. DEBBA

A THESIS SUBMITTED IN PARTIAL FULFILMENT

OF THE REQUIREMENTS

FOR MASTER IN SCIENCE IN

MATHEMATICAL STATISTICS DEGREE

IN THE FACULTY OF SCIENCE

RHODES UNIVERSITY

FACULTY OF SCIENCE

DEPARTMENT OF STATISTICS

FEBRUARY 2011

Abstract

The purpose of the present thesis is to show that hyperspectral data can be used for discrimination between different tree species. The data set used in this study contains the hyperspectral measurements of leaves of seven savannah tree species. The data is high-dimensional and shows large within-class variability combined with small between-class variability which makes discrimination between the classes challenging. We employ two classification methods: G-nearest neighbour and feed-forward neural networks. For both methods, direct 7-class prediction results in high misclassification rates. However, binary classification works better. We constructed binary classifiers for all possible binary classification problems and combine them with Error Correcting Output Codes. We show especially that the use of 1-nearest neighbour binary classifiers results in no improvement compared to a direct 1-nearest neighbour 7-class predictor. In contrast to this negative result, the use of neural networks binary classifiers improves accuracy by 10% compared to a direct neural networks 7-class predictor, and error rates become acceptable. This can be further improved by choosing only suitable binary classifiers for combination.

Contents

List of Figures

List of Tables

ACKNOWLEDGEMENTS

First and foremost, I give thanks to the Most High God for making things possible. I would like to thank my supervisor, Professor Gunther Jager, for his constant guidance and encouragement for the entire duration of the research. Without Professor Jager, this thesis would not have been undertaken. I would also like to thank my co-supervisor, Dr Pravesh Debba, for his constant support and invaluable contribution to this work.

To Rhodes University statistics department staff, thanks a lot for your support especially, Professor Sarah Radloff, for accepting me to do Masters in Statistics. To Rhodes statistics post graduate students, thanks for your motivations and support.

I would also like to thank The Council for Scientific and Industrial Research (CSIR) for financial support.

To my friends who helped me in one way or the other for completing this thesis, thanks a lot. Last but not least, I would like to thank my Mother and my Brother, for constant love and support throughout my years at university.

1 Introduction

The advent of hyperspectral remote sensors with high spectral resolution has enhanced the classification of earth's surface objects. Hyperspectral remote sensors record data of the earth's surface objects and this recorded data is called *hyperspectral data*. This data is recorded using many closely spaced wavelength bands. Hyperspectral remote sensors can record hyperspectral data of vegetation, soil and water (these are few earth's surface objects) at different altitudes above the ground. The hyperspectral data can then be used to classify the earth's surface objects. The interest of using hyperspectral data is due to the fact that hyperspectral data can be collected regularly using airplanes and satellites and this allows easy and regular access to the data. Possible applications of hyperspectral data are e.g. flood detection and monitoring [9], and forest monitoring [28].

The study that we are conducting focuses on classifying vegetation, specifically tree species at leaf level. By *leaf level* we mean that only the leaves of the trees were used for classification. The trees that we are classifying are deciduous trees.

The main purpose of this study was to investigate, if hyperspectral data can be used to discriminate between certain tree species. We show in this thesis that this is challenging but that it is possible.

The data that we used was collected in May 2008 at Kruger National Park in South Africa. It contains hyperspectral measurements of 7 different savannah tree species. Our data set is challenging for the following three reasons:

(1) it is high dimensional;
(2) it is small in size (total 148 samples, approximately 20 samples per species);
(3) it shows high within-class variability and small between-class variability.

For reason (1), usually some kind of "dimension reduction" is performed for hyperspectral data prior to classification. In this study we only used very simple dimension reduction. We chose every 10th (20th or 30th) band. For reason (2), it is generally difficult to make good classifiers and get a good estimate of the expected number of misclassifications at the same time. With

few samples, we decided to use K-fold cross validation as suggested in [3]. For reason (3), we decided to look at model-free classifiers, as they are usually not based on comparison of the means. The model-free approaches that we decided to use are G-nearest neighbour classifiers and feed forward neural networks classifiers. Both have good theoretical properties and are often used in applications [23]. Our results in using these classifiers for 7-class predictions (i.e. each species is considered as a class) were unsatisfactory. However, binary classification, where we e.g. distinguish one class versus the rest, showed acceptable misclassification rates. Therefore we decided to try and use binary classifiers and combine their outputs for 7-class predictions. There are different ways of combining binary classifiers to obtain a multiclass classifier. Often e.g. majority votes are suggested [15]. In 1995 a new approach for combining binary classifiers, Error Correcting Output Code was suggested in [7] and shown to be successful ([7],[10]). We therefore use this combining method for our problem.

In the course of our experiments we found that if the binary classifiers are 1-nearest neighbour classifiers, the Error Correcting Output Code returns the same error rate as a direct 7-class 1-nearest neighbour classifier. That this will always be the case is shown in this thesis. Furthermore, we can show that compared to a direct 7-class neural networks predictor, Error Correcting Output Code combination of binary neural networks classifiers yields an improvement of the error rate by roughly 10%. Classification of this data now seems to become possible. Looking at the error rates of the binary classifiers, a natural idea is to use only those binary classifiers which produce reasonable error rates. Our experiments suggest that further improvements of the classification accuracy can be obtained using this approach.

The thesis is organised as follows. In section 1, we briefly introduce hyperspectral remote sensing. In section 2 we discuss classification and the classifiers, and the estimation of the error probability of a classifier. In section 3 we discuss the binary classifiers and their combination using Error Correcting Output Codes. In section 4 we discuss the methods which are used in this study. In section 5 we state and explain the results and section 6 is the discussion and conclusion.

2 Hyperspectral Remote Sensing

Remote sensing is the method of obtaining information about an object or area through the analysis of data gathered by sensors that are not in physical contact with the object or area of interest [16]. The sensors detect electromagnetic energy reflected by the objects. The electromagnetic energy is an energy that is composed of both electric wave and magnetic wave. Both waves are perpendicular to each other and are travelling at the speed of light. In remote sensing, electromagnetic waves are characterised by their wavelength location within an electromagnetic spectrum [16]. The electromagnetic spectrum is shown in Figure 2.1. Remote sensing systems use one or several of the thermal infrared, reflected infrared, visible and microwave regions of the electromagnetic spectrum. The electromagnetic energy that is reflected by the earth surface objects will vary depending on the material type of the object. These differences provide a possibility to distinguish between different types of earth's surface objects [16].

There are two kinds of electromagnetic energy sensors, namely *active sensors* and *passive sensors*. Figure 2.2 shows an active sensor and a passive sensor. The active sensor provides its own source of energy radiation whereas the passive sensor depends on the sun for its source of energy radiation. The sensors that are used in remote sensing operate at different altitudes or platforms above the objects or the area of interest (e.g. the sensors are mounted on planes or satellites or handheld spectrometers).

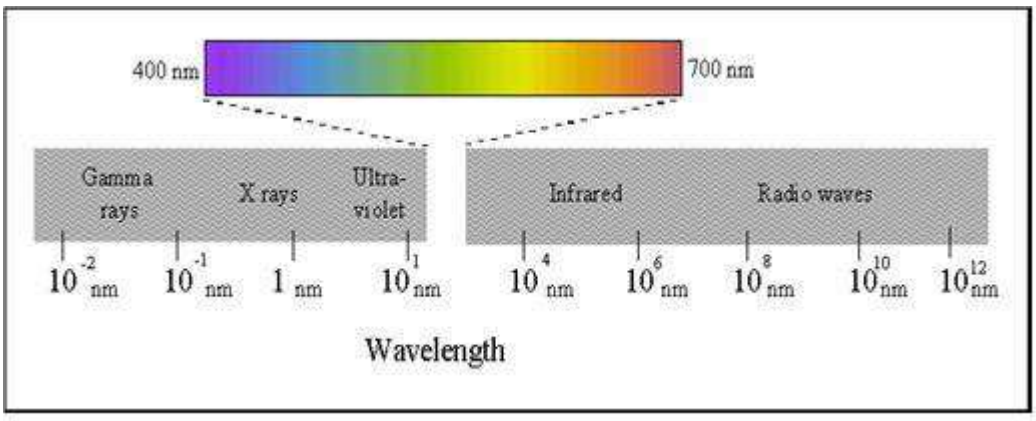

Figure 2.1: Electromagnetic spectrum [29]

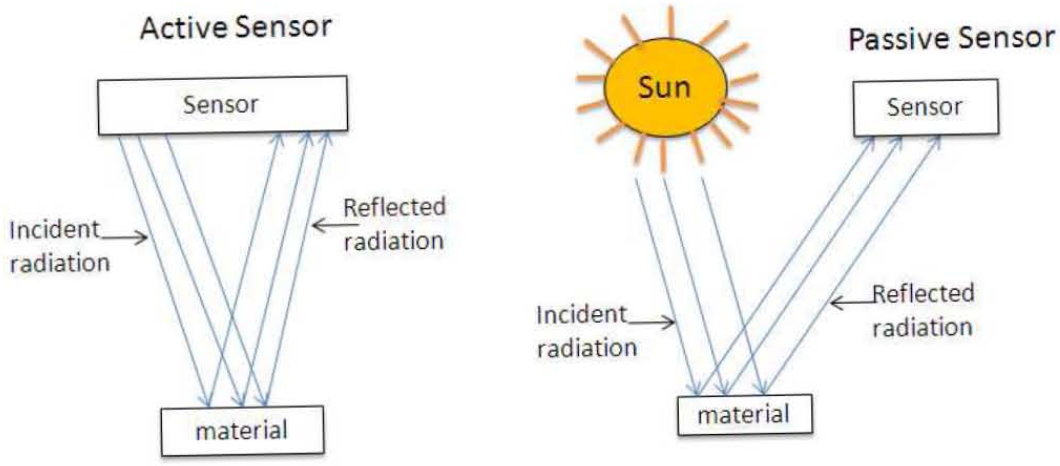

Figure 2.2: Hyperspectral remote sensors

2.1 Hyperspectral Data

Hyperspectral data are obtained from remote sensors using many narrow and closely spaced wavelength bands. The wavelength scales of hyperspectral remote sensors include visible range ($400nm$-$700nm$), near infra-red range ($700nm$-$1300nm$) and short wave infra-red range ($1300nm$-$2500nm$), where nm denotes nano meters. Due to the large number of different wavelengths, the data which is recorded by hyperspectral remote sensors is high dimensional data. Hyperspectral remote sensors record the ratio of the reflected radiation to the incident radiation [16], and this ratio is called

$$reflectance = \frac{reflected\ radiation}{incident\ radiation}.$$

Radiation is also known as light, thus hyperspectral remote sensors record the ratio of light which hits the object and is reflected by the object. Reflectances of an earth's surface object are measured at different wavelengths to form an (ideally) continuous reflectance curve. Different types of objects (i.e. materials) can often be distinguished by having different reflectance curves [16]. A typical reflectance curve for vegetation is shown in Figure 2.3. In general it is difficult to analyse hyperspectral data due to the large number of reflectances (i.e. reflectances in the wavelength range $400nm - 2500nm$). Usually a form of data reduction is performed before the analysis of hyperspectral data [22].

2.2 Vegetation discrimination with hyperspectral data

A typical spectral reflectance curve of vegetation in Figure 2.3 shows a "peak and valley" configuration. The valleys show reflectances, and the peaks show absorptions of the electromagnetic energy by vegetation. E.g the valleys in the visible portion ($400nm$ to $700nm$) of the spectrum are dictated by pigments in plant leaves. The chlorophyll in plant leaves absorbs electromagnetic energy in the blue and red regions of the electromagnetic spectrum and reflect electromagnetic energy in the green region of the spectrum. Vegetation reflects strongly in the near infra-red region of the electromagnetic spectrum ($700nm$ to $1300nm$, this is due to the internal structure of the leaves [16]). Beyond $1300nm$ of the electromagnetic spectrum, energy incident upon vegetation is absorbed (absorption is due to water present in plant leaves) or reflected. Reflectance beyond $1300nm$ is approximately inversely related to the total water present in a leaf. This total water present in a leaf is a function of both the moisture content and the thickness of a leaf [16].

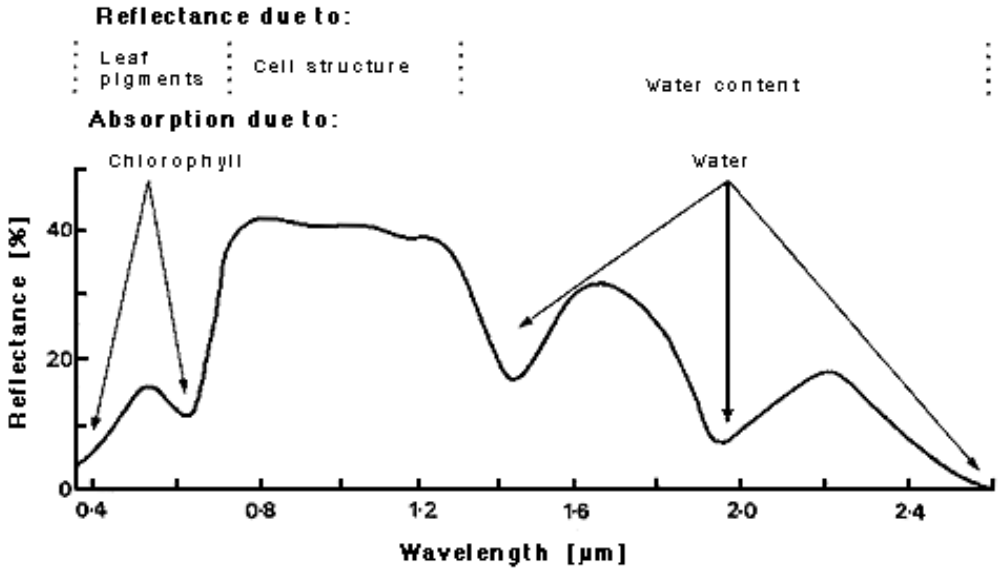

Figure 2.3: Reflectance curve of vegetation [27]

3 Classification and Classifiers

We refer to *classification* as the identification of an object with feature vector $x \in \mathbb{R}^n$ into one of the existing classes $\{\omega_1, \omega_2, \cdots, \omega_c\}$. A *feature vector* of an object consists of numerical measurements (i.e. features) which describe an object. For example, for remotely sensed data of vegetation with 2 different wavelength bands, a feature vector $x \in \mathbb{R}^2$ will consist of vegetation reflectances at these two wavelength bands. Figure 3.1 shows a classification system. So in general when we want to classify a new object, firstly we obtain several features of a new object, then we classify a new object into one class ω_i of the existing classes $\{\omega_1, \omega_2, \cdots, \omega_c\}$ based on its features. Features of an object are often given as real numbers. Classes are often given as integers but e.g. for Neural Networks we use orthogonal coding for our classes. E.g. for a 3-class classification problem, orthogonal coding for class 1 is denoted by the vector $\begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}$, class 2 is denoted by $\begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix}$ and class 3 is denoted by $\begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}$.

We will later use *supervised learning* to construct a classifier. In supervised learning, we assume that there is a supervisor who is able to classify the data without errors. Thus we get a "labelled data set"

$$\mathcal{D} = \left\{ \left(x^1, t_1\right), \left(x^2, t_2\right), \cdots, \left(x^N, t_N\right) \right\}$$

where $x^i \in \mathbb{R}^n$ denotes a feature vector and t_i denotes a class label from a set of classes $\{\omega_1, \omega_2, \cdots, \omega_c\}$. This "labelled data set",$\mathcal{D}$, is then used to construct a classifier.

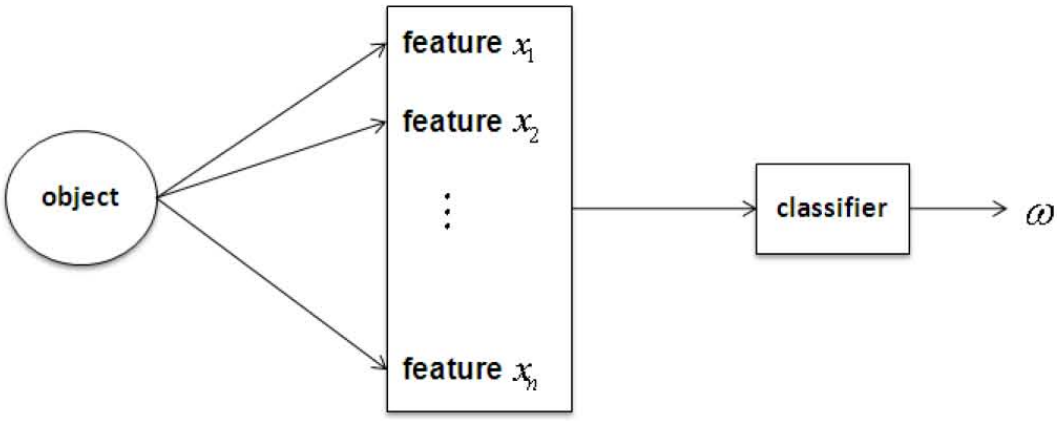

Figure 3.1: Classification System

3.1 Minimum error classification

In general there are different methods for constructing a classifier from data. We can use model approaches which assume underlying probability distributions of data or we can use model-free approaches which do not assume underlying probability distributions of data to construct a classifier. Hence using these different methods results in different classifiers. In order to decide which classifier is "better", it becomes necessary to estimate the error probability, $P(error)$, of each classifier. A natural criterion is that we want to use a classifier with as little misclassifications as possible, i.e. with a small error probability. The *Bayes classifier* is optimal, i.e. there can be no "better" classifier with smaller error probability. This Bayes classifier is based on Bayes decision rule [8]. Before we discuss Bayes decision rule we need to introduce the following notation:

- $P(\omega_i \mid x)$ is called the *a posteriori probability* (i.e. the probability that the object is of class ω_i knowing that the feature vector is x);
- $P(\omega_i)$ is called the *a priori probability* (i.e. the probability with which we expect the respective kinds of classes) ;
- $p(x \mid \omega_i)$ is called a *class conditional density* (i.e. the distribution (density) of the feature vector of class ω_i);
- $p(x)$ is called the *evidence* (it can be viewed as the scale factor that guarantees that the *posteriori probabilities* sum to one).

The Bayes decision rule [8] is defined as follows: *decide for class* ω_i *if for all* $j \neq i$

$$P(\omega_i \mid x) \geq P(\omega_j \mid x)$$

where

$$P(\omega_i \mid x) = \frac{p(x \mid \omega_i) P(\omega_i)}{p(x)}$$

with

$$p(x) = \sum_{j=1}^{c} p(x \mid \omega_j) P(\omega_j).$$

Suppose that we define ω_m as the class with maximum posteriori probability, i.e.

$$P(\omega_m \mid x) = \max_{1 \leq i \leq c} P(\omega_i \mid x). \tag{3.1}$$

Then the Bayes decision rule always selects ω_m (if there are ties the Bayes decision rule selects a class ω_i which e.g. comes first in ordering). The following derivation for possible minimum error probability is based on [8]. The error probability is given by

$$P(error) = \int P(error, x)\, dx = \int P(error \mid x)\, p(x)\, dx. \tag{3.2}$$

In (3.2), if for every x we ensure that the *conditional error probability*, $P(error \mid x)$, is as small as possible, then the integral must be as small as possible. Hence, the error probability, $P(error)$, will be minimized. For example in a classification problem with two classes, $\{\omega_1, \omega_2\}$, the conditional error probability is given by

$$P(error \mid x) = \begin{cases} P(\omega_1 \mid x) = 1 - P(\omega_2 \mid x) & if\ we\ decide\ for\ \omega_2 \\ P(\omega_2 \mid x) = 1 - P(\omega_1 \mid x) & if\ we\ decide\ for\ \omega_1 \end{cases}.$$

In this example we can minimize the conditional error probability by deciding for class ω_1 if $P(\omega_1 \mid x) \geq P(\omega_2 \mid x)$ and class ω_2 otherwise. Thus, Bayes decision rule minimizes $P(error)$ by minimizing $P(error \mid x)$ for every x. Suppose that we have c classes, $\{\omega_1, \omega_2, \cdots, \omega_c\}$, we let $P^*(error \mid x)$ be the minimum possible value of $P(error \mid x)$, and $P^*(error)$ be the minimum possible value of $P(error)$, then by using (3.1) and Bayes decision rule we have

$$P^*(error \mid x) = 1 - P(\omega_m \mid x). \tag{3.3}$$

Thus, the possible minimum error probability is given by

$$P^*\left(error\right) = \int P^*(error \mid x)p(x)dx. \tag{3.4}$$

3.2 Model-free Approaches

Model-free approaches do not assume the underlying probability distribution of data. In contrary to model approaches which are used e.g. to estimate the parameters of the class-conditional densities (such as maximum likelihood estimation of the parameters of normal models), model-free approaches do not estimate these parameters. The high-dimensionality of hyperspectral data restricts the application of model approaches. Suppose that we would want to estimate the class-conditional n-variate density using histograms. If we consider in each dimension 10 bins, then we need in case of

- $n = 1$: more than 10 samples;
- $n = 2$: more than 10^2 samples;
- $n = 3$: more than 10^3 samples;
- $\vdots$
- $n = 2101$: more than 10^{2101} samples

in order to assure that the bins will not be empty. Hence the number of samples which we need for estimation of the $n-$variate density using a histogram approach grows in the order of 10^n. This is an example of the so-called *curse of dimensionality* [11]:

- For a good discrimination between the classes we want to use many features.
- If the the feature number is high, then it becomes impossible to estimate the densities, as we do not have enough samples.

3.2.1 G-Nearest Neighbour

The *G-nearest neighbour classifier* [8] is constructed by using a labelled data set

$$\mathcal{D} = \{(x^1, t_1), (x^2, t_2), ..., (x^N, t_N)\},$$

where $x^i \in \mathbb{R}^n$ denotes a feature vector and t_i denotes a class label from the set of classes, $\{\omega_1, \omega_2, \cdots, \omega_c\}$. Note that a feature vector is now and then called a *sample* in contrast to the usage of this term in statistics. At presentation of a new object with feature vector $x \in \mathbb{R}^n$ we determine the G closest samples in the labelled data set $\mathcal{D}$. The class of the majority of these G closest samples is the class that will be assigned to the new object with feature vector x.

One possible measure of closeness of the feature vector x to the samples in the labelled data set is the *Euclidean distance*. That is at presentation of sample x, we determine the distances of x to the $x^1, x^2, \cdots, x^N$ denoted by

$$d_1 = d(x^1, x), ..., d_N = d(x^N, x)$$

where

$$d_i = d(x^i, x) = \sqrt{(x_1^i - x_1)^2 + (x_2^i - x_2)^2 + \cdots + (x_n^i - x_n)^2}$$

denotes the Euclidean distance between x^i and x for $i = 1, 2, \cdots, N$. A pseudo-code for the G-nearest neighbour classification is given in the following box.

G-nearest Neighbour pseudo-code

- Input: labelled samples $(x^1, t_1), (x^2, t_2), \cdots, (x^N, t_N)$, new object x, number of neighbours G.
- For $i = 1$ to N determine distances $d_i = d(x^i, x)$.
- Choose G samples with smallest d_i.
- Determine class ω of the majority of G closest samples.
- Assign class ω to x.
- Output: class label ω for x.

Figure 3.2 shows a 5-nearest neighbour classification. There are two classes and a sample with an unknown class. The aim is to assign this new sample into one of the two classes. The majority of the samples that are closest to the new sample are from class 1 (i.e. four of the samples are from class 1), hence the class of the new sample is class 1.

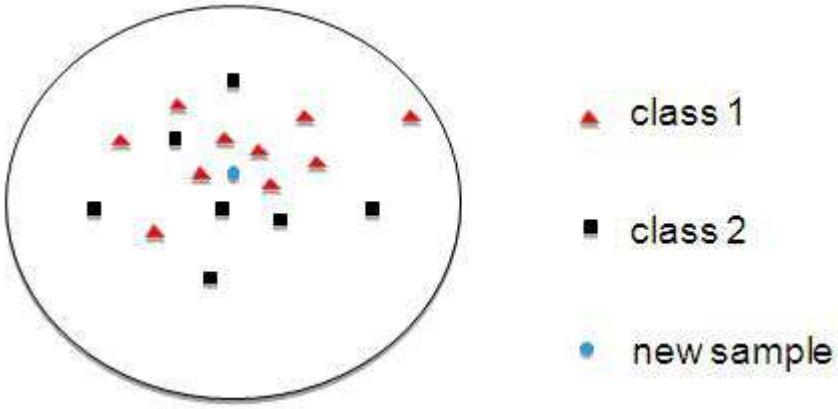

Figure 3.2: 5-Nearest Neighbour classification

For G= 1, the 1-nearest neighbour classifier is simply called the *nearest neighbour classifier*. The nearest neighbour classifier usually leads to an error rate greater than the minimum possible, the Bayes error rate [8] (see section 3.1). But it can be shown that with unlimited number of samples the error rate is never worse than twice the optimal Bayes error rate [8]. We now reproduce the argument of [8]. Suppose that $x' \in H$ is the sample in $\mathcal{D}$ nearest to the new sample x. The label t' associated with the nearest neighbour is a random variable, and the probability that $t' = \omega_i$ is a posteriori probability $P(\omega_i \mid x')$. When the number of samples in $\mathcal{D}$ is very large, it is reasonable to assume that x' is sufficiently close to x such that $P(\omega_i \mid x') \simeq P(\omega_i \mid x)$. Theorem 3.2.1 shows the convergence of the error probability of nearest neighbour decision rule.

Theorem 3.2.1 *If $P_N(error)$ is the N samples error probability of the nearest neigbour decision rule, and if for $N \to \infty$ we denote the error probability of the nearest neighbour decision rule, $P(error)$, by*

$$P(error) = \lim_{N \to \infty} P_N(error), \tag{3.5}$$

then

$$P^*(error) \leq P(error) \leq \left(2P^*(error) - \frac{c}{c-1}\left(P^*(error)\right)^2\right) \leq 2P^*(error) \tag{3.6}$$

where $P^*(error)$ denotes a minimum error probability of the Bayesian decision rule and c denotes the number of classes.

Proof: When different sets of N samples, $\left\{x^1, x^2, \cdots, x^N\right\}$, are used to classify x, different vectors x' will be obtained for the nearest neighbour of x. Because the decision rule depends on the nearest neighbour of x, we have a conditional error probability $P(error \mid x, x')$ that depends on both x and x'. By integrating over x', we obtain

$$\begin{aligned} P(error \mid x) &= \int P(error \mid x, x')p(x' \mid x)dx' \\ &= \int \frac{P(error, x, x')}{p(x, x')} \times \frac{p(x, x')}{p(x)}dx' \\ &= \int \frac{P(error, x, x')}{p(x)}dx' \\ &= \int P(error, x' \mid x)\, dx'. \end{aligned} \tag{3.7}$$

As $N \to \infty$ we expect $p(x' \mid x)$ to approach a "delta function" centered at x, making the evaluation of (3.7) trivial. A delta function is defined as follows:

$$\delta(x) = \begin{cases} 0 & if\ x \neq 0 \\ \infty & if\ x = 0 \end{cases}$$

with

$$\int \delta(x)dx = 1.$$

E.g. we can look at the Gaussian functions

$$\delta_n(x) = \frac{1}{\sqrt{2\pi}\sigma_n} e^{\left\{-\left(\frac{x}{\sigma_n}\right)^2\right\}}$$

with

$$\sigma_n = \frac{1}{n}.$$

The Gaussian functions are shown in Figure 3.3. We see that as n increases, the Gaussian functions approximate the delta function.

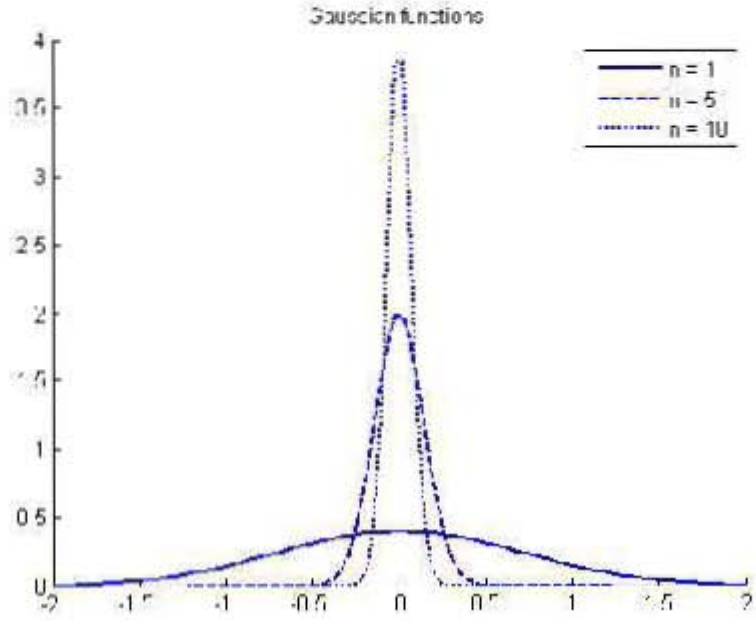

Figure 3.3: Gaussian functions

A delta function has a fundamental property [24] that for any continuous function, f,

$$\int f(x')\delta(x')dx' = f(0). \tag{3.8}$$

It is shown in [24] that (3.8) generalizes to

$$\int f(x')\delta(x'-a)dx' = f(a). \tag{3.9}$$

Suppose that at a given x, the probability density, $p(\cdot)$, is continuous and non-zero. The probability that any sample falls within a hypersphere S centered about x is some positive number P_s given by

$$P_s = \int_{x' \in S} p(x')dx'.$$

The graph of the sphere is shown in Figure 3.4, $\varepsilon > 0$ (i.e. a small positive value) is the *radius*.

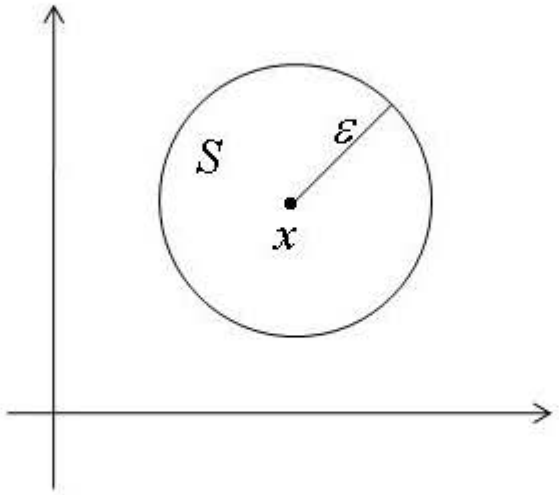

Figure 3.4: Hypersphere S

The probability that the sample falls outside this hypersphere is $1 - P_s$ with $P_s > 0$. Thus the probability that all N of the independently drawn samples fall outside this hypersphere is $(1 - P_s)^N$, which approaches zero as N approaches infinity, since $0 < 1 - P_s < 1$. Therefore

$$\lim_{N \to \infty} P\left(|x' - x| > \varepsilon\right) = \lim_{N \to \infty} \left(1 - P_s\right)^N = 0,$$

hence x' converges to x in probability, and $p\left(x' \mid x\right)$ become very peaked in the vicinity of x and very small elsewhere. Hence $p\left(x' \mid x\right)$ approaches a delta function. We now turn into the estimation of the conditional error probability $P_N\left(e \mid x, x'\right)$ for the Nearest Neighbour Rule. To estimate this conditional error probability $P_N\left(e \mid x, x'\right)$, we denote the nearest neighbour to x by x'_N. Suppose that we have N independently drawn random samples of pairs $\left(x^1, t_1\right), \left(x^2, t_2\right), \cdots, \left(x^N, t_N\right)$, where $t_i \in \left\{\omega_1, \omega_2, \cdots, \omega_c\right\}$. Suppose that during classification, a pair (x, t) is selected, and also suppose that x'_N, labelled t'_N, is the sample nearest to x. Since the pairs, $\left(x^i, t_i\right)$, are independent, we have

$$\begin{aligned} P\left(t, t'_N \mid x, x'_N\right) &= P\left(t \mid x\right) P\left(t'_N \mid x'_N\right) \\ &= \frac{P\left(t, x\right)}{P\left(x\right)} \times \frac{P\left(t'_N, x'_N\right)}{P\left(x'_N\right)} \\ &= \frac{P\left(t, x, t'_N, x'_N\right)}{P\left(x, x'_N\right)}. \end{aligned} \tag{3.10}$$

If we use a nearest neighbour decision rule, we make an error whenever $t \neq t'_N$. Thus the conditional error probability is given by

$$\begin{aligned} P_N\left(error \mid x, x'_N\right) &= 1 - \sum_{i=1}^{c} P\left(t = \omega_i, t'_N = \omega_i \mid x, x'_N\right) \\ &= 1 - \sum_{i=1}^{c} P\left(\omega_i \mid x\right) P\left(\omega_i \mid x'_N\right). \end{aligned} \tag{3.11}$$

To get $P_N\left(error\right)$, we substitute (3.11) in (3.7) for $P_N\left(error \mid x\right)$ and then integrate the results over x. As it was remarked earlier that the integration in (3.7) becomes trivial as N approaches infinity and $p\left(x'_N \mid x\right)$ approaches a delta function. If $P\left(\omega_i \mid x\right)$ is continuous at x, we thus obtain

$$\begin{aligned} \lim_{N\to\infty} P_N\left(error \mid x\right) &= \int [1 - \sum_{i=1}^{c} P\left(\omega_i \mid x\right) P\left(\omega_i \mid x'_N\right)] \delta\left(x'_N - x\right) dx'_N \\ &\overset{(3.9)}{=} 1 - \sum_{i=1}^{c} P^2\left(\omega_i \mid x\right). \end{aligned} \tag{3.12}$$

The asymptotic nearest neighbour error rate is given by

$$\begin{aligned} P\left(error\right) &= \lim_{N\to\infty} P_N\left(error\right) \\ &= \lim_{N\to\infty} \int P_N\left(error \mid x\right) p\left(x\right) dx \\ &= \int [1 - \sum_{i=1}^{c} P^2\left(\omega_i \mid x\right)] p\left(x\right) dx. \end{aligned} \tag{3.13}$$

We find bounds on $P\left(error\right)$ in terms of Bayes error rate $P^*\left(error\right)$. The lower bound on $P\left(error\right)$ is $P^*\left(error\right)$. For an upper bound, we must determine how large the nearest neighbour error rate $P\left(error\right)$ can become for a given Bayes error rate $P^*\left(error\right)$. Hence using (3.13), we must determine how small $\sum_{i=1}^{c} P^2\left(\omega_i \mid x\right)$ can be for a given $P\left(\omega_m \mid x\right)$. We write

$$\sum_{i=1}^{c} P^2\left(\omega_i \mid x\right) = P^2\left(\omega_m \mid x\right) + \sum_{i\neq m}^{c} P^2\left(\omega_i \mid x\right), \tag{3.14}$$

and then we bound this sum by minimizing the second term subject to the following constraints:

- $P(\omega_i \mid x) \geq 0$;
- $\sum_{i \neq m}^{c} P(\omega_i \mid x) = 1 - P(\omega_m \mid x) = P^*(error \mid x)$.

The summation $\sum_{i=1}^{c} P^2(\omega_i \mid x)$ is minimized if all the posterior probabilities except the mth are equal. Using the second constraint, suppose that $i \neq m$, then we use Lagrangian multilpliers to find the minimum. The Lagrangian function is given by

$$L = L(P(\omega_i \mid x), \lambda) = \sum_{i \neq m}^{c} P^2(\omega_i \mid x) - \lambda[\sum_{i \neq m}^{c} P(\omega_i \mid x) - P^*(error \mid x)].$$

If we differentiate L with respect to $P(\omega_i \mid x)$ and λ, and then equate the derivatives with zero, we get:

$$P(\omega_i \mid x) = \frac{P^*(error \mid x)}{(c-1)}.$$

Suppose that $i = m$, then the second constraint becomes

$$1 - P(\omega_i \mid x) - P^*(error \mid x) = 0.$$

Using Lagrange multipliers we get

$$L = \sum_{i=1}^{c} P^2(\omega_i \mid x) - \lambda[1 - P(\omega_i \mid x) - P^*(error \mid x)].$$

If we differentiate L with respect to $P(\omega_i \mid x)$ and λ, and then equate the derivatives with zero, we get:

$$P(\omega_i \mid x) = 1 - P^*(error \mid x).$$

Thus we have the inequality

$$
\begin{aligned}
&\sum_{i=1}^{c} P^2(\omega_i \mid x) \geq (1 - P^*(error \mid x))^2 + \sum_{i \neq m}^{c} \left[\frac{P^*(error \mid x)}{(c-1)}\right]^2 \\
\iff &\sum_{i=1}^{c} P^2(\omega_i \mid x) \geq (1 - P^*(error \mid x))^2 + (c-1)\frac{(P^*(error \mid x))^2}{(c-1)^2} \\
\iff &\sum_{i=1}^{c} P^2(\omega_i \mid x) \geq (1 - P^*(error \mid x))^2 + \frac{(P^*(error \mid x))^2}{(c-1)} \\
\iff &\sum_{i=1}^{c} P^2(\omega_i \mid x) \geq 1 - 2P^*(error \mid x) + (P^*(error \mid x))^2 + \frac{(P^*(error \mid x))^2}{c-1} \\
\iff &\sum_{i=1}^{c} P^2(\omega_i \mid x) - 1 \geq -2P^*(error \mid x) + \frac{cP^{2*}(error \mid x)}{c-1} \\
\iff &1 - \sum_{i=1}^{c} P^2(\omega_i \mid x) \leq 2P^*(error \mid x) - \frac{c}{c-1}P^{2*}(error \mid x) \leq 2P^*(error \mid x) \\
\iff &1 - \sum_{i=1}^{c} P^2(\omega_i \mid x) \leq 2P^*(error \mid x).
\end{aligned}
$$

If we substitute the above inequality in (3.13) and use (3.4), we get

$$
\begin{aligned}
P(error) = \int [1 - \sum_{i=1}^{c} P^2(\omega_i \mid x)] p(x)\, dx &\leq \int 2P^*(error \mid x)\, p(x)\, dx \\
\iff P(error) &\leq 2\int P^*(error \mid x)\, p(x)\, dx \\
\iff P(error) &\leq 2P^*(error).\blacksquare
\end{aligned}
$$

Hence we have shown that the error probability of the nearest neighbour is never worse than twice the Bayes error rate if the sample size of the training set $\mathcal{D}$ goes to infinite. This shows that in theory (and with enough samples), the nearest neighbour classifier is a "good" classifier, provided that $P^*(error)$ is small.

3.2.2 Neural Networks

An *Artificial Neural Network* [21] is a system which is motivated by biological neural network

systems. An artificial neural network emulates the way in which a biological neural network of the brain processes information. A simple biological neuron receives signals, then it processes these signals and it produces new signals. These new signals are then transferred to other neurons. In the same way an artificial neuron receives signals in the form of vector inputs, which are then evaluated in the accumulator and transferred to other neurons via weight connections. An input vector $(x_1, x_2, \cdots, x_n)'$ of real numbers is transported via weight connections to an accumulator. Each weight connection has a weight ν_k. The weight ν_k is multiplied by an input x_k and the resulting product is sent into the accumulator. In the accumulator these products of inputs and weights are summed up together with a bias b to give

$$b + \nu_1 x_1 + \nu_2 x_2 + \cdots + \nu_n x_n = b + \sum_{i=1}^{n} \nu_i x_i. \tag{3.15}$$

A transfer function f is applied to (3.15) to give an output $y = f\left(b + \sum_{i=1}^{n} \nu_i\right)$. This output y is either transported to other neurons via weight connections or is the final output of a neural network. Figure 3.5 shows as an example a graph of a single artificial neuron.

Usually we use *orthogonal coding* to code the classes for neural networks. This means that for a $c-$ class classification problem, class ω_i is represented by a $c \times 1$ column vector, that has a 1 (one) in row i and zeroes elsewhere. Thus a neural network will have c output neurons. E.g. for a 3-class classification problem, class 1 is denoted by $\begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}$, class 2 is denoted by $\begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix}$, and class 3 is denoted by $\begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}$. The reason why we prefer orthogonal coding to numerical coding (where we label the classes by $1, 2, \cdots, c$) is that the error at an ouput is measured by

$$error = (t - y)^2$$

where t is the target and y the output of the neural networks. If we had e.g. $t = 3$, then a neural networks output $y = 5$ is less wrong than a neural networks output $y = 7$. However, class labels usually come without ordering or numerical meaning, i.e. class labels are categorical. Therefore both $y = 5$ and $y = 7$ are equally wrong.

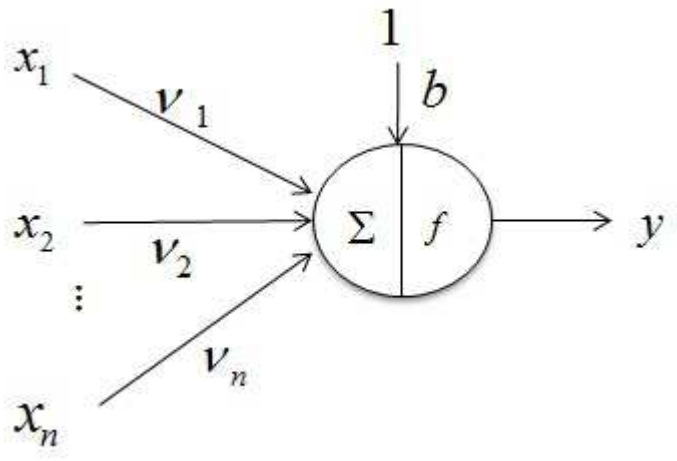

Figure 3.5: Single Artificial Neuron

Figure 3.6 shows as example a graph of a *multilayer feed-forward neural network*s, which has an input layer, two hidden layers and an output layer. Data flow from the input layer to the next hidden layer, from there to the second hidden layer and from there to the output layer. For multilayer feed-forward neural networks we introduce the following notation:

- i is the input number;
- j is the neuron number;
- k is the layer number;
- ν_{ij}^k is the weight of a connection from input i to neuron j in layer k;
- y_j^k is the output of neuron j in layer k;
- b_{kj} is the bias of neuron j in layer k;
- y is an output of a neural network.

In this example of a graph of multilayer feed-forward neural networks in Figure 3.6, an input vector $(x_1, x_2, \cdots, x_n)'$ in the input layer is transported into the first hidden layer in the following way

$$b_{1j} + \nu_{1j}^1 x_1 + \nu_{2j}^1 x_2 + \cdots + \nu_{nj}^1 x_n = b_{1j} + \sum_{i=1}^{n} \nu_{ij}^1 x_i \tag{3.16}$$

for $j = 1, 2, \cdots, m$ (where m is the number of hidden neurons in the first hidden layer). We apply transfer function in (3.16) for $j = 1, 2, \cdots, m$ and we get the outputs y_j^1. These outputs, y_j^1, are transported into the second hidden layer via weight connections in the following way

$$b_{2l} + \nu_{1l}^2 y_1^1 + \nu_{2l}^2 y_2^1 + \cdots + \nu_{ml}^2 y_m^1 = b_{2l} + \sum_{j=1}^{m} \nu_{jl}^2 y_j^1 \tag{3.17}$$

for $l = 1, 2, \cdots, k$ (where k is the number of hidden neurons in the second hidden layer). Again we apply a transfer function in (3.17) for $l = 1, 2, \cdots, k$ and we get the outputs y_l^2. These outputs, y_l^2, are transported into the output layer via weight connections in the following way

$$b_{3w} + \nu_{1w}^3 y_1^2 + \nu_{2w}^3 y_2^2 + \cdots + \nu_{kw}^3 y_m^2 = b_{3w} + \sum_{l=1}^{k} \nu_{lw}^3 y_l^2 \tag{3.18}$$

for $w = 1$ (because we only have one neuron in the output layer for this example). We apply a transfer function in (3.18) for $w = 1$ and we get an output y. Typical transfer functions (used later) are e.g. the sigmoid function (usually in the hidden layers), given by

$$f(x) = \frac{1}{1 + e^{-x}},$$

or the linear function (usually in the output layer), given by

$$f(x) = x.$$

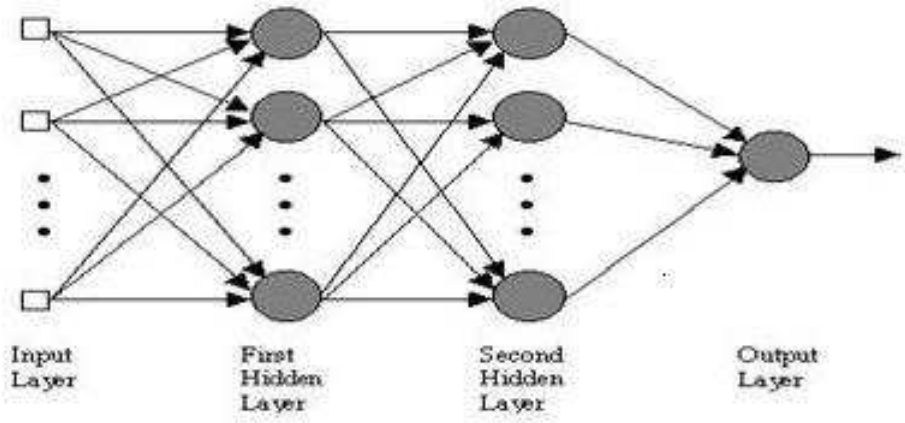

Figure 3.6: A multilayer feed-forward neural network

A neural network with n inputs and m outputs can be viewed as an input-output mapping

$$\Phi : \mathbb{R}^n \to \mathbb{R}^m.$$

This mapping $\Phi = \Phi(\cdot; Q)$ depends on parameters (i.e. weights and biases) collected in the vector Q. Given a labelled data set, $\left\{(x^1, t_1), (x^2, t_2), \cdots, \left(x^N, t_N\right)\right\}$, with feature vectors $x^i \in \mathbb{R}^n$, we try to find optimal parameters Q^* such that for all $i = 1, 2, \cdots, N$

$$\Phi\left(x^i; Q^*\right) \approx t_i.$$

Here we do not want equality. If we have equality, then it would imply that the error sum of squares is zero. In reality, the error sum of squares will not be equal to zero, because the data that is used to find optimal parameters is noisy. We may e.g. have $x^i = x^k$ but $t_i \neq t_k$. It is better to make the error sum of squares to be close to zero. This is similar to e.g. linear regression, where the deviation of the observed value $y_i = \beta_0 + \beta_i x_i$ from the corresponding predicted value $\hat{y}_i = \hat{\beta}_0 + \hat{\beta}_i x_i$ given by

$$e_i = y_i - \hat{y}_i$$

must be as small as possible so that the residual sum of squares is approximately equal to zero. This is due to the fact that the data comes with the noise hence there will be residuals, and the aim of linear regression is to minimize the sum of squares of these residuals. An interpretation

of neural networks from a statistical perspective can e.g. be found in [26].

Neural networks can be used to approximate a wide range of functions. This is shown for example by the following Theorem 3.2.2 (Universal Approximation Property of Multilayer Perceptrons [5]).

Theorem 3.2.2: *Let $f : [0,1]^n \to \mathbb{R}$ be a continuous function and let $\epsilon > 0$. Then there exists a multilayer perceptron with one hidden layer with sigmoid transfer functions and a linear transfer function in the output neuron such that for its input -output mapping $\Phi : [0,1]^n \to \mathbb{R}$ we have*

$$\mid f(x) - \Phi(x) \mid < \epsilon$$

for all $x \in [0,1]^n$.

For classification, the unknown mapping f is certainly not continuous (even worse: it is not a mapping because we may have objects with different classes but the same feature vector), but Theorem 3.2.2 shows the theoretical approximation capabilities of neural networks. Theorem 3.2.2 leaves us with the idea that a solution for a continuous function exists. In order to find this solution, we will have to experiment to find the appropriate architecture (i.e. the number of hidden neurons). Also we will need a learning algorithm which determines the weights and biases.

In order to quantify "goodness of approximation" of a neural network, we need an error function $E = E(Q)$ which is a function of weights and biases. The error function is given as the sum of squared errors [3]

$$E = E(Q) = \frac{1}{2} \sum_{i=1}^{N} \parallel \Phi\left(x^i; Q\right) - t_i \parallel^2 \tag{3.19}$$

where

$$\parallel x \parallel^2 = x_1^2 + x_2^2 + \cdots + x_n^2.$$

A neural networks determines a set of parameters (i.e. biases and weights), collected in a vector Q^*, which minimize the error function E. The value of the error function based on the new set of parameters is $E(Q^*)$ and if it is less than the value of the error function $E(Q)$ which is

based on the old set of parameters i.e.

$$E\left(Q^*\right) < E\left(Q\right),$$

then the old set of parameters is replaced by the new set of parameters, i.e.

$$Q \leftarrow Q^*.$$

We now say that parameters of a neural network are updated, thus a neural network has *learnt* the new parameters Q^* from the labelled data set. The learning process is repeated several times (i.e. the labelled data set is presented to the neural network again, and parameters are updated $Q^* \leftarrow Q^{**}$ according to the value of the new error function $E\left(Q^*\right)$). We stop the learning process when a certain minimum value of the error function is reached.

3.2.2.1 Learning Algorithms: steepest descent

Given a labelled data set $\mathcal{D} = \left\{\left(x^1, t_1\right), \left(x^2, t_2\right), \cdots, \left(x^N, t_N\right)\right\}$ where $x^i \in \mathbb{R}^n$ and $t_i \in \{\omega_1, \omega_2, \cdots, \omega_c\}$, we use the *steepest descent algorithm* to minimize the error function E. Note that the error function $E = E\left(Q\right)$ is a function of $|Q|$ (i.e. $|Q|$ denotes the number of elements in Q) arguments, i.e.

$$E : \mathbb{R}^{|Q|} \rightarrow \mathbb{R}$$

for one output neuron.

We look at a general function $f : \mathbb{R}^n \rightarrow \mathbb{R}$ for minimization. If we look at function graph of $f\left(x\right)$ in Figure 3.7, when using steepest descent algorithm we would *go downhill* in order to get the minimum value of $f\left(x\right)$. Note that in this case, x, is not a feature vector but an input variable of a function f. In our case, x, is a vector of weights and biases. For a function $f : \mathbb{R}^n \rightarrow \mathbb{R}$, we start at some point, $x \in \mathbb{R}^n$, we choose a direction $\upsilon \in \mathbb{R}^n$ and a step size $\alpha > 0$ such that $f\left(x + \alpha\upsilon\right) < f\left(x\right)$, then we update $x \leftarrow x + \alpha\upsilon$. The steepest descent algorithm determines the slope of a function graph at a point, say x, in all directions υ and chooses the strongest slope. The update rule is repeated several times and it stops after a certain number

of iterations. In [21] it is shown that the direction υ is given by the negative gradient

$$\upsilon = -grad\,(f)\,(x)$$

where $grad\,(f)\,(x) = \left(\frac{\partial f}{\partial x_1}(x), \frac{\partial f}{\partial x_2}(x), \cdots, \frac{\partial f}{\partial x_n}(x)\right)$ is the gradient of the function f at x. The steepest descent algorithm is also known as the gradient descent algorithm.

Steepest Descent Algorithm

- Input: function f, point $x \in \mathbb{R}^n$, and stepsize α;
- determine $grad\,(f)\,(x)$;
- update $x \leftarrow x - \alpha \times \frac{grad(f)(x)}{\|grad(f)(x)\|}$;
- output: x

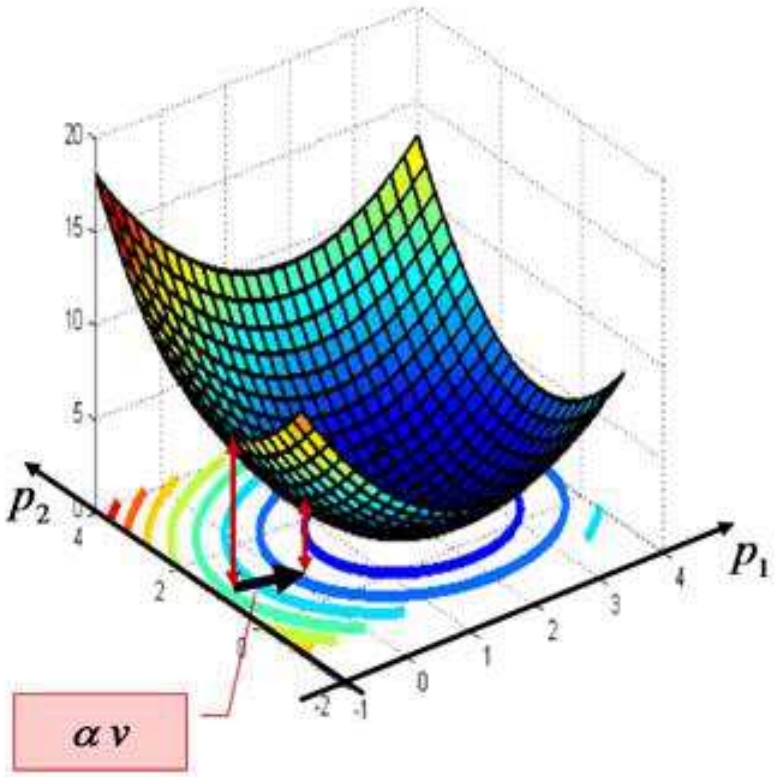

Figure 3.7: Function graph of f and steepest descent direction

3.2.2.2 Error back-propagation

Consider the last layer of a multilayer feed-forward neural networks. Suppose that an output neuron produces an incorrect response in a multilayer feed-forward neural networks when presented with an input vector. To solve this supposition, we have to determine which hidden neurons in the hidden layer are responsible for making an error, then the weights and biases connected to those hidden neurons get adjusted. The adjustment of the weights will produce a better response of the output neuron. If we consider a multilayer feed-forward neural networks with differentiable activation functions (i.e. transfer functions), then activation functions of the output neurons become differentiable functions of the weights and biases. Consider an error function given by (3.19), which is a differentiable function of the weights and biases, because $\Phi\left(x^i; Q\right)$ depends on the weights and biases. The derivatives of the error functions with respect

to the weights and biases are evaluated and these derivatives are used to find weights and biases that minimize the error function, by using the gradient descent agorithm. The steepest descent algorithm can be interpreted as propagating the error "backwards" through the network, hence the name *back-propagation* [2].

3.2.2.3 Resilient back-propagation algorithm

Resilient back-propagation [25] is simply a heuristic modification of the gradient descent algorithm. In contrast to the steepest descent algorithm, only the sign of the partial derivative is used to perform learning and adaptation of the weights. For each weight, an individual update-value $\triangle_{ij}$ is introduced, which determines the size of the weight update. The update-value is given as follows:

$$\triangle_{ij}(t) = \begin{cases} \eta^+ \times \triangle_{ij}(t-1) & if \quad \frac{\partial E(t-1)}{\partial \nu_{ij}} \times \frac{\partial E(t)}{\partial \nu_{ij}} > 0 \\ \eta^- \times \triangle(t-1) & if \quad \frac{\partial E(t-1)}{\partial \nu_{ij}} \times \frac{\partial E(t)}{\partial \nu_{ij}} < 0 \\ \triangle_{ij}(t-1) & else \end{cases}$$

where $0 < \eta^- < 1 < \eta^+$, t is the number of iterations, ν_{ij} is the weight of the connection from the neuron j to the output i and E is the error function. Verbally this adaptation rule can be explained as follows: every time the partial derivative of the corresponding weight ν_{ij} changes its sign, which indicates that the last update-value was too big and the algorithm has missed a local minimum, the update is decreased by the factor η^-. If the derivative does not change its sign, the update-value is slightly increased in order to accelerate convergence in shallow regions. Once the update-value for each weight is adapted, the weight-update follows a simple rule [25]: if the derivative is positive (i.e. an increasing error), the weight is decreased by its update-value, if the derivative is negative, the update-value is added: i.e. we have

$$\triangle\nu_{ij}(t) = \begin{cases} -\triangle_{ij}(t) & if \quad \frac{\partial E(t)}{\partial \nu_{ij}(t)} > 0 \\ +\triangle_{ij}(t) & if \quad \frac{\partial E(t)}{\partial \nu_{ij}(t)} < 0 \\ 0 & else \end{cases}$$

and with this we update as follows:

$$\nu_{ij}(t+1) = \nu_{ij}(t) + \triangle\nu_{ij}(t).$$

Resilient back-propagation usually learns faster than standard back-propagation (i.e. steepest descent), especially in regions where the error function is flat [25].

3.3 Estimation of Error Probability and Standard Error

The error probability of a classifier is defined as a probability of assigning an object to an incorrect class, and the error probability can usually only be estimated. Because we base our decision of choosing a better classifier on these esimates, we need to also know how "reliable" these estimates are. Therefore we also need the knowledge of the standard errors, i.e. the standard deviations of our estimates. The following derivations are mainly based on [3] and [14].

Consider a classifier

$$\Phi : \mathcal{F} \to \{\omega_1, \omega_2, \cdots, \omega_c\},$$

as a mapping from feature space $\mathcal{F} \subset \mathbb{R}^n$ to the set of class labels $\{\omega_1, \omega_2, \cdots, \omega_c\}$. We assume that we know a probability distribution on $\mathcal{F} \times C$, i.e. we assume that for any $B \subset \mathcal{F}$ and $\omega_i \in \mathcal{C}$, the probability

$$P(B, \omega) = P(x \in B, \omega = \omega_i)$$

is known. Then the error probability of the classifier $\Phi : \mathcal{F} \to C$ is defined by

$$P(error) = P(\Phi(x) \neq t)$$

for a randomly chosen sample $(x, t) \in \mathcal{F} \times \mathcal{C}$. Firstly we define a random variable on $\mathcal{F} \times C$, by

$$Z((x,t)) = \begin{cases} 1 & if \quad \Phi(x) \neq t \\ 0 & else \end{cases}.$$

We will later also denote $Z((x,t)) = L(\Phi(x), t)$ and call it the *zero-one loss* (see [3]). The

expected value of $Z = Z((x,t))$ then becomes

$$E(Z) = 1 \cdot P(Z=1) + 0 \cdot P(Z=0) = P(\Phi(x) \neq t) = P(error).$$

Suppose $Z_i = Z((x^i,t_i))$ for $i = 1,2,\cdots,N$ and $\left\{(x^1,t_1),(x^2,t_2),\cdots,\left(x^N,t_N\right)\right\}$ is a random sample (here we consider the term sample as is used in statistics) from $\mathcal{F}\times\mathcal{C}$. Then we can use the sample mean of the random variables $Z_1,Z_2,\cdots,Z_N$ as an estimator for $P(error)$. This estimator is given by

$$\hat{P} = \hat{P}(error) = \overline{Z} = \frac{1}{N}\sum_{i=1}^{N} Z\left(\left(x^i,t_i\right)\right),$$

which is the proportion of misclassifications in the data $\left\{x^1,x^2,\cdots,x^N\right\}$. As a sample mean, this esimator is unbiased:

$$E\left(\hat{P}\right) = E\left(\frac{1}{N}\sum_{i=1}^{N} Z_i\right) = \frac{1}{N}\sum_{i=1}^{N} E(Z_i) = \frac{1}{N}NE(Z) = P(error),$$

using the fact that $E(Z_1) = E(Z_2) = \cdots = E(Z_N) = E(Z)$ and the linearity of the expectation.

In order to determine the standard error of our estimator, we consider each (x^i,t_i) as a Bernoulli trial, where a success is a misclassification (i.e. $\Phi(x^i) \neq t_i$) and the probability of success is $P = P(error)$. Then the random variable

$$Y = \sum_{i=1}^{N} Z\left(\left(x^i,t_i\right)\right)$$

counts the number of successes and hence has a binomial distribution, $Y \sim b(N, P = P(error))$. The mean of Y is NP and the variance is $NP(1-P)$. Therefore the standard error is given

by

$$\begin{aligned} SE\left(\hat{P}\right) &= \sigma_{\hat{P}} = \sqrt{var\left(\hat{P}\right)} \\ &= \sqrt{var\left(\frac{1}{N}\sum_{i=1}^{N} Z\left((x^i, t_i)\right)\right)} \\ &= \sqrt{\frac{1}{N^2} var\left(Y\right)} \\ &= \sqrt{\frac{P\left(1-P\right)}{N}}. \end{aligned}$$

If the sample size is large enough, we can replace the unknown P by our estimate $\hat{P}$ in order to estimate this standard error. We consider the sample variance s^2 of the numbers $Z_i = Z\left((x^i, t_i)\right) \in \{0, 1\}$, then

$$\begin{aligned} s^2 &= \frac{1}{N}(\sum_{i=1}^{N}\left(Z_i - \frac{1}{N}\sum_{i=1}^{N} Z_i\right)^2) \\ &= \frac{1}{N}(\sum_{i=1}^{N}\left(Z_i - \overline{Z}\right)^2) \\ &= \frac{1}{N}(\sum_{i=1}^{N}\left(Z_i^2 - 2Z_i\overline{Z} + \overline{Z}^2\right)) \\ &= \frac{1}{N}(\sum_{i=1}^{N} Z_i^2 - 2N\overline{Z}\cdot\overline{Z} + N\overline{Z}^2) \\ &= \frac{1}{N}\sum_{i=1}^{N} Z_i^2 - \overline{Z}^2 \\ &= \frac{1}{N}\sum_{i=1}^{n} Z_i - \overline{Z}^2 \qquad since \quad (Z_i^2 = Z_i) \\ &= \overline{Z}\left(1-\overline{Z}\right). \end{aligned}$$

Hence the sample variance equals

$$s^2 = \hat{P}\left(1-\hat{P}\right),$$

and the standard error of our esimator for the error probability can be written as

$$SE\left(\hat{P}\right) = \sqrt{\frac{s^2}{N}}.$$

In practice, the underlying probability distribution on $\mathcal{F}\times\mathcal{C}$ is usually not known. Hence, classifiers are constructed from data. Let a learning task

$$\mathcal{L} = \left\{\left(x^1, t_1\right), \left(x^2, t_2\right), \cdots, \left(x^N, t_N\right)\right\}$$

be randomly sampled from $\mathcal{F}\times\mathcal{C}$. We construct a classifier $\Phi_{\mathcal{L}} : \mathcal{F} \to \mathcal{C}$ from $\mathcal{L}$. The learning task, $\mathcal{L}$, is therefore called *training set,* as we "train" the classifier on that set. A construction of a classifier from a labelled data set $\mathcal{L}$ usually involves the minimization of the number of misclassifications on $\mathcal{L}$. Hence, if we use the same data set, $\mathcal{L}$, to estimate the error probability $P\left(error\right)$, this estimate will be too optimistic. The way out is to randomly sample a second data set from $\mathcal{F}\times\mathcal{C}$, the *test set* $\mathcal{T}$ and use $\mathcal{T}$ to estimate $P\left(error\right)$ of $\Phi_{\mathcal{L}}$.

3.3.1 The Test Set Approach

In practice, the number of available data is often restricted and new ones either cannot be sampled or it is expensive to gather new data. The labelled data set $\mathcal{D}$ is therefore randomly split into two independent non-empty sets $\mathcal{L}$ and $\mathcal{T}$ with $\mathcal{L} \cup \mathcal{T} = \mathcal{D}$, $\mathcal{L} \cap \mathcal{T} = \emptyset$. These sets are used as training set $\mathcal{L}$ and test set $\mathcal{T}$. The training set is used to train a classier $\Phi_{\mathcal{L}}$ and the test set is used to estimate its error probability. The error probability estimate is used to assess the performance of the classifier $\Phi = \Phi_{\mathcal{L}}$. We define for a sample $x \in \mathcal{T}$ with target t as before the loss function

$$L\left(\Phi\left(x\right), t\right) = Z\left(\left(x, t\right)\right) = \begin{cases} 1 & if \quad \Phi\left(x\right) \neq t \\ 0 & else \end{cases}$$

and estimate the error probability $P\left(error\right)$ as the proportion of misclassifications on the test

set $\mathcal{T}$, i.e.

$$\begin{aligned}\hat{P} = \hat{P}\left(error\right) &= \frac{1}{\mid \mathcal{T} \mid}\left(number \quad of \quad missclassifications \quad on \quad \mathcal{T}\right) \\ &= \frac{1}{\mid \mathcal{T} \mid}\sum_{x\in\mathcal{T}} Z\left((x,t)\right)\end{aligned}$$

where $\mid \mathcal{T} \mid$ denotes the number of samples in the test set $\mathcal{T}$. In order to get the variance of our esimater $\hat{P}$, we proceed as in the previous section. We have

$$\begin{aligned}var\left(\hat{P}\right) &= \frac{1}{\mid \mathcal{T} \mid^2} var\left(\sum_{x\in\mathcal{T}} Z\left((x,t)\right)\right) \\ &= \frac{1}{\mid \mathcal{T} \mid^2} \mid \mathcal{T} \mid P\left(error\right)\left(1 - P\left(error\right)\right) \\ &\approx \frac{1}{\mid \mathcal{T} \mid}\hat{P}\left(1-\hat{P}\right).\end{aligned}$$

Consider the sample variance of the values $Z\left((x,t)\right) \in \{0,1\}$, i.e.

$$\begin{aligned}s^2 &= \frac{1}{\mid \mathcal{T} \mid}\sum_{x\in\mathcal{T}}\left(Z-\hat{P}\right)^2 \\ &= \frac{1}{\mid \mathcal{T} \mid}\sum_{x\in\mathcal{T}}\left(Z^2 - 2Z\hat{P} - \hat{P}^2\right) \\ &= \frac{1}{\mid \mathcal{T} \mid}\sum_{x\in\mathcal{T}} Z^2 - 2\frac{1}{\mid \mathcal{T} \mid}\sum_{x\in\mathcal{T}} Z\hat{P} - \frac{1}{\mid \mathcal{T} \mid}\sum_{x\in\mathcal{T}}\hat{P}^2 \\ &= \frac{1}{\mid \mathcal{T} \mid}\sum_{x\in\mathcal{T}} Z - 2\frac{1}{\mid \mathcal{T} \mid}\sum_{x\in\mathcal{T}} Z\hat{P} - \frac{1}{\mid \mathcal{T} \mid}\times \mid \mathcal{T} \mid \hat{P}^2 \\ &= \hat{P} - 2\hat{P}\hat{P} - \hat{P}^2 \\ &= \hat{P} - \hat{P}^2 \\ &= \hat{P}\left(1-\hat{P}\right)\end{aligned} \tag{3.20}$$

where $\hat{P} = \hat{P}\left(error\right)$. Hence we can estimate standard error of $\hat{P}$ as

$$SE\left(\hat{P}\right) = \sqrt{\frac{s^2}{\mid \mathcal{T} \mid}}.$$

3.3.2 K-fold cross validation

For a small data set $\mathcal{D}$ it becomes difficult to obtain a suitable classifier and a good estimate of its error probability at the same time. We need many data to "train" a good classifier, hence few data will be left for test set, which makes the estimate of $P(error)$ poor. If we use many data for the test set (giving a good estimate of $P(error)$), then the constructed classifier will be bad. In order to solve this problem, *K-fold cross validation* ([3], [14]) is suggested. Suppose again that we have a labelled data set $\mathcal{D}$. First we randomly divide $\mathcal{D}$ into K equally sized subsets (= *folds*) : $\mathcal{D}_1, \mathcal{D}_2, \cdots, \mathcal{D}_K$ such that

$$\bigcup_{k=1}^{K} \mathcal{D}_k = \mathcal{D}$$

and

$$\mathcal{D}_k \cap D_l = \emptyset \qquad (k \neq l)\,.$$

This means that the union of the K equally sized subsets is equal to the total data set $\mathcal{D}$ and the subsets are disjoint. The idea is to train K classifiers $\Phi_1, \Phi_2, \cdots, \Phi_K$ (i.e. $\Phi_k = \Phi_{\mathcal{L}_k}$ for $k = 1, 2, \cdots, K$) on the training sets $\mathcal{L}_k = \mathcal{D} \backslash \mathcal{D}_k$ for $k = 1, 2, \cdots, K$ and then estimate the error probabilities of the K classifiers on the test sets $\mathcal{T}_k = \mathcal{D}_k$ for $k = 1, 2, \cdots, K$, respectively. Figure 3.8 shows a 4-fold cross validation. The upper parts of the data set in Figure 3.8 are used for training the classifiers and the lower part is used as a test set for estimating the error probability.

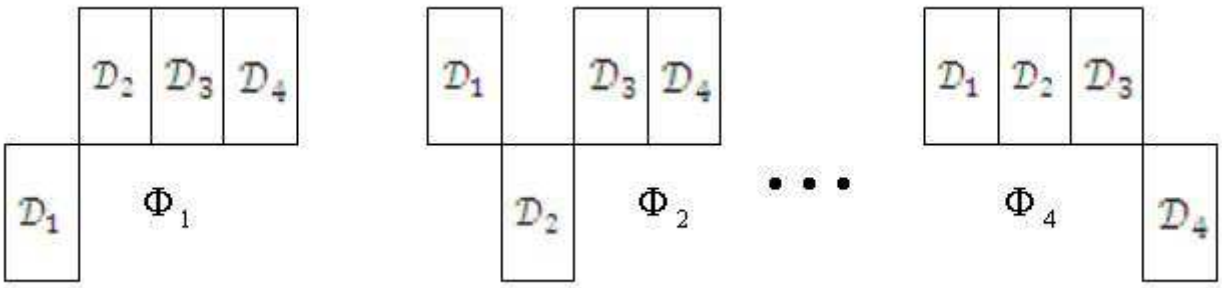

Figure 3.8: 4-fold cross validation

Basic assumption in cross validation ([3], **[14]):** *If $\Phi : \mathcal{F} \to \mathcal{C}$ is a classifier trained on the whole data set $\mathcal{D}$ and $\Phi_k : \mathcal{F} \to \mathcal{C}$ are classifiers trained on $\mathcal{L}_k = \mathcal{D} \setminus \mathcal{D}_k$, and if we denote*

the error probabilities of the classifiers Φ, Φ_k *by* $P(error;\Phi)$ *and* $P(error;\Phi_k)$, *resepectively, then*

$$P(error;\Phi) \approx P(error;\Phi_k)$$

for all $k = 1, 2, \cdots, K$.

In the following Theorem 3.3.2.1 we show that the estimate of the error probability of the classifier Φ is the average of the estimates for the error probabilities of the classifiers Φ_k.

Theorem 3.3.2.1: *The error probability when using* K *-fold cross validation can be estimated by*

$$\hat{P}(error;\Phi) \approx \frac{1}{K}\sum_{k=1}^{K} \hat{P}(error;\Phi_k).$$

Proof: We use the above basic assumption to obtain the estimate of the error probability $P(error;\Phi)$. We introduce the following notation

- $N^{(k)} \approx \frac{N}{K}$ number of samples in $\mathcal{T}_k$;
- N =total number of samples.

First we define a selector function s by

$$s : \begin{cases} \mathcal{D} \to \{1, 2, \cdots, K\} \\ x \longmapsto s(x) = k \Leftrightarrow x \in \mathcal{D}_k. \end{cases}$$

We define again for a sample x with target t a loss function

$$Z((x,t)) = L(\Phi(x), t) = \begin{cases} 1 & if \quad \Phi(x) \neq t \\ 0 & else \end{cases}.$$

Then by using the basic assumption in cross validation, we get

$$Z((x,t)) = \left\{ \begin{array}{ll} 1 & if \quad \Phi(x) \neq t \\ 0 & else \end{array} \right\} \approx L\left(\Phi_{s(x)}(x), t\right) = \begin{cases} 1 & if \quad \Phi_{s(x)}(x) \neq t \\ 0 & else \end{cases}$$

$L\left(\Phi_{s(x)}(x), t\right)$ is the *0-1-loss* if we classify a sample x. If $L\left(\Phi_{s(x)}(x), t\right)$ has a value 1, then we have a misclassification. Thus $L\left(\Phi_{s(x)}(x), t\right)$ can be used to count the number of misclassifications. For $x \in \mathcal{D}_k$ we have $\Phi_{s(x)} = \Phi_k$. We can then estimate the error probability $P(error)$ by

$$\begin{aligned}
\hat{P}(error) &= \frac{1}{N} \times (number\ of\ misclassifications\ made\ by\ \Phi) \\
&= \frac{1}{N} \sum_{x \in \mathcal{D}} Z((x,t)) \\
&\approx \frac{1}{N} \sum_{x \in \mathcal{D}} L\left(\Phi_{s(x)}(x), t\right) \\
&= \frac{1}{N} \sum_{x \in \bigcup_{k=1}^{K} \mathcal{D}_k} L\left(\Phi_{s(x)}(x), t\right) \\
&= \frac{1}{N} \sum_{k=1}^{K} \sum_{x \in \mathcal{D}_k} L\left(\Phi_k(x), t\right) \\
&= \frac{1}{N} \sum_{k=1}^{K} \frac{N^{(k)}}{N^{(k)}} \sum_{x \in \mathcal{D}_k} L\left(\Phi_k(x), t\right) \\
&= \frac{1}{N} \sum_{k=1}^{K} N^{(k)} \hat{P}^{(k)}
\end{aligned}$$

with $\hat{P}^{(k)} = \hat{P}(error; \Phi_k)$ the proportion of misclassifications on the test set $\mathcal{D}_k$ made by classifier Φ_k. With $N^{(k)} \approx \frac{N}{K}$, we devide both sides of the approximate equal sign by N , we get $\frac{N^{(k)}}{N} \approx \frac{1}{K}$. Thus

$$\hat{P}(error) \approx \frac{1}{K} \sum_{k=1}^{K} \hat{P}^{(k)} = \frac{1}{K} \sum_{k=1}^{K} \hat{P}(error; \Phi_k) .\blacksquare$$

We look at the estimate of the variance of $\hat{P}(error)$ assuming that the loss function values, $L\left(\Phi_{s(x)}(x), t\right)$ where $x \in \mathcal{D}$, are independent as suggested in [3]. Then, we have

$$var\left(\hat{P}(error)\right) = var\left(\frac{1}{N} \sum_{x \in \mathcal{D}} L\left(\Phi_{s(x)}(x), t\right)\right) \approx \frac{1}{N^2} \times N \times P(error)(1 - P(error)) .$$

We replace $P(error)$ by $\hat{P}(error)$, we get

$$var\left(\hat{P}(error)\right) \approx \frac{1}{N} \times \hat{P}(error)\left(1 - \hat{P}(error)\right).$$

Consider the sample variance s^2 of the values $L\left(\Phi_{s(x)}(x), t\right), \quad where\ x \in \mathcal{D}$. Using (3.20) we have

$$var\left(\hat{P}(error)\right) \approx \frac{1}{N} \times s^2.$$

Thus, the estimate of the standard error of $\hat{P}(error)$ is

$$SE\left(\hat{P}(error)\right) = \sqrt{\frac{s^2}{N}}.$$

Now we state the Central Limit Theorem [20]:

Theorem 3.3.2.2: *Let $X_1, X_2, \cdots, X_N$ be independent and identically distributed random variables having mean $E(X_i) = \mu$ and finite nonzero variance $V(X_i) = \sigma^2$. Let $S_n = X_1 + X_2 + \cdots + X_N$. Then*

$$\lim_{n\to\infty} P\left(\frac{S_n - N\mu}{\sigma\sqrt{N}} \leq u\right) = \Phi(u)$$

where $\Phi(u)$ is the probability that the standard normal random variable is less than u.

The estimate of the standard error can be used to obtain confidence intervals for the error probability based on the point estimate $\hat{P}$: If the sample size, N, is large enough, then by the Central Limit Theorem we have

$$\frac{\hat{P} - P}{\sigma_{\hat{P}}} \sim N(0, 1)$$

approximately. Thus, given the confidence level $1 - \alpha$, we determine z such that

$$P\left(-z \leq \frac{\hat{P} - P}{\sigma_{\hat{P}}} \leq z\right) = 1 - \alpha$$

and solve for $P = P(error)$:

$$P\left(\hat{P} - z\sigma_{\hat{P}} \leq P \leq \hat{P} + z\sigma_{\hat{P}}\right) = 1 - \alpha.$$

4 Combining binary classifiers

4.1 Binary classifiers versus multiclass predictors

A classification problem which involves only two classes is called a *binary classification problem*. Many classification methods yield binary classifiers, e.g. support vector machines [1] or linear discriminant methods [12]. An example of a binary classification is the medical diagnosis of a certain disease. In this example, the induced classifier uses clinical information from a patient to determine if he/she has a particular disease. The classes represent the presence or absence of the disease.

Many real problems involve dicrimination of more than two classes. These problems with more than two classes are called *multiclass classification problems*. With multiclass classification problems, we need to construct multiclass classifiers e.g. G-nearest neighbour classifiers and neural networks classifiers. Any classifier that can be used for a multiclass classification problem can be used as a binary classifier as well. An example of a multiclass classification problem is the classification of multiple types of tumor. A multiclass classification problem is more complex, since the induced classifier must be able to separate the data set into a higher number of classes, and this increases the chances of classification errors.

Possible target codings for multiclass classification problems are e.g.:

- numerical coding $1, 2, \cdots, c$ (suitable for G-nearest neighbour classifier);
- orthogonal coding $\begin{pmatrix} 1 \\ 0 \\ \vdots \\ 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix}, \cdots, \begin{pmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ 1 \end{pmatrix}$ where each column vector has a $c \times 1$ dimensions (suitable for neural networks) see section 3.2.2.

For c-class classification problem, when using neural networks we convert each numerical value into a column vector which consists of a single 1 (one) and $(c-1)$ zeroes. E.g a numerical value $i \in \{1, 2, \cdots, c\}$ is converted into a $c \times 1$ column vector which has 1 (one) in row i and

zeroes elsewhere. Hence a neural network must have c output neurons. The target coding for a binary classifier is given in the following section.

4.2 Decomposition of a multiclass problem into a set of binary classification problems

To reduce the chances of classification errors we suggest a decomposition of a multiclass classification problem into a set of binary classification problems by forming *metaclasses* C^+ and C^-. That is if $\mathcal{C} = \{\omega_1, \omega_2, \cdots, \omega_c\}$ is the set of class labels, then a partition of $\mathcal{C}$ into two non-empty subsets results in metaclasses C^+, C^-. Note that $C^- = \mathcal{C} \setminus C^+$ and $\emptyset \neq C^+ \neq \mathcal{C}$. This means that the metaclass C^- is the complement of the non-empty metaclass C^+ (i.e. these two metaclasses do not have elements in common). After the decomposition of a multiclass problem into binary classification problems, a binary classifier denoted by f_i is constructed for each of the binary classification problems. The usual target coding for a binary classifier is

$$t = \begin{cases} 0 & for\ a\ sample\ of\ metaclass\ C^- \\ 1 & for\ a\ sample\ of\ metaclass\ C^+ \end{cases} .$$

In Figure 4.1 we show a decomposition of a 7-class classification problem into metaclasses $C^+$and C^-. In this example we see that metaclass C^+ is formed by combining classes 1,2 and 4, and metaclass C^- is formed by combining classes 3,5,6 and 7.

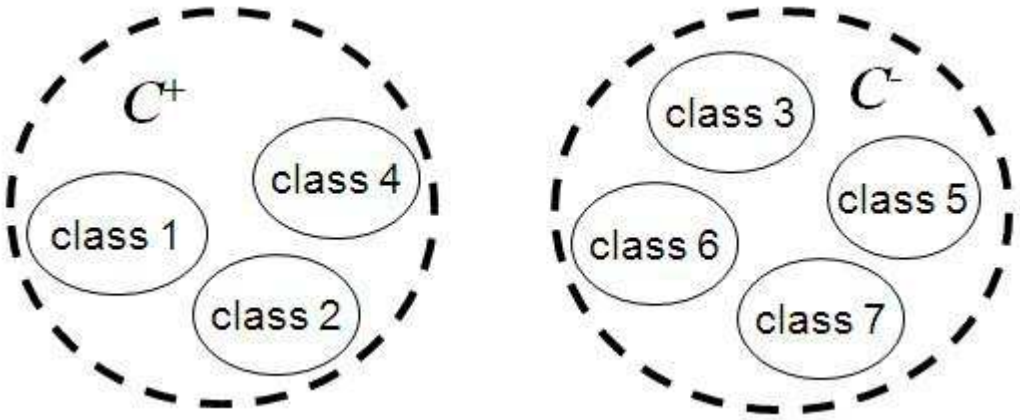

Figure 4.1: Metaclasses for a seven-class problem

Theorem 4.2.1: *For a c-class classification problem, there are at most* $2^{c-1}-1$ *different binary classification problems.*

Proof: Each partition of the set $\mathcal{C}=\{\omega_1,\omega_2,\cdots,\omega_c\}$ into two non-empty subsets C^+, C^- with $C^-=\mathcal{C}\setminus C^+$(i.e. the subsets are complements) gives rise to a binary classification problem. There are 2^c partitions of $\mathcal{C}=\{\omega_1,\omega_2,\cdots,\omega_c\}$. If we code each partition of $\{\omega_1,\omega_2,\cdots,\omega_c\}$ by a binary vector $(f_1,f_2,\cdots,f_c)$ with

$$f_i=\begin{cases}1 & if\ \omega_i\in C^+\\ 0 & if\ \omega_i\in C^-\end{cases}$$

then each partition corresponds to exactly one binary vector. Now,

$$(1,1,\cdots,1)-(f_1,f_2,\cdots,f_c)=(f_1',f_2',\cdots,f_c')$$

is complementary to $(f_1,f_2,\cdots,f_c)$, hence the subsets C^+, C^- are interchanged. So $(f_1,f_2,\cdots,f_c)$ and $(f_1',f_2',\cdots,f_c')$ code the same binary classification problem. Since half of these binary vectors are complementary to the other half, we have only $\frac{2^c}{2}=2^{c-1}$ binary vectors. One of these binary vectors either consists of all ones or zeroes, hence this binary vector cannot be used to code a binary classification problem (since it cannot distinguish between two classes). So we subtract 1 from 2^{c-1} and we get $2^{c-1}-1$ binary vectors. Since each binary vector is used to code a binary classification problem, we thus have $2^{c-1}-1$ binary classification problems.■

With $2^{c-1}-1$ binary classification problems, we need $2^{c-1}-1$ binary classifiers. For a new sample x we evaluate each of these binary classifiers to obtain the outputs which are then combined together to form a multiclass predictor for x. Basically the use of the decomposition strategy involves two steps. In the first step, the multiclass problem division into binary problems is performed, determining binary classifiers to be formed. The second step refers to how the outputs of the binary classifiers are combined to assign the class of a new sample.

4.3 Combining binary classifiers to solve a multiclass problem: Error Correcting Output Codes (ECOC)

Error Correcting Output Code (ECOC) [7] is one of the methods which are used to combine outputs of binary classifiers. Originally ECOC was suggested to allow binary classification methods to be used for multiclass problems [7]. It was shown e.g. in [7] that often the combination of binary classifiers by ECOC is superior to a direct multiclass predictor.

For ECOC we define a code matrix $M \in \{0,1\}^{c \times l}$, where c denotes the number of classes (i.e. $c = |\{\omega_1, \omega_2, \cdots, \omega_c\}|$) and l denotes the number of binary classifiers (i.e. $l = |\{f_1, f_2, \cdots, f_l\}|$) with f_i =binary classifier. We remember that the output coding for a binary classifier f_i when evaluating an object with feature vector x is

$$f_i(x) = \begin{cases} 0 & if\ x \in C^- \\ 1 & if\ x \in C^+ \end{cases} .$$

A code matrix M is composed of binary digits (i.e. ones and zeroes), and it has c rows and l columns. Each column of M represents the partition of $\mathcal{C} = \{\omega_1, \omega_2, \cdots, \omega_c\}$ into $\{C^-, C^+\}$ which is coded by 1_{C^+} with

$$1_{C^+}(\omega_i) = \begin{cases} 1 & if\ \omega_i \in C^+ \\ 0 & if\ \omega_i \in C^- \end{cases}$$

for $i = 1, 2, \cdots, c$. This means that for each column of M, if a class ω_i belongs in metaclass C^+ it is denoted by a 1, if another class ω_j where $i \neq j$ belongs in metaclass C^- it is denoted by a 0. Hence each binary classifier corresponds to a binary vector of length c. If all the binary classifiers predict class ω_i for $i = 1, 2, \cdots, c$ in the correct metaclasses, then each row of M can be intepreted as a class. Figure 4.2 shows an example of a code matrix for a 4-class problem with seven binary classifiers.

	f_1	f_2	f_3	f_4	f_5	f_6	f_7
class 1	1	1	1	1	1	1	1
class 2	0	0	0	0	1	1	1
class 3	0	0	1	1	0	0	1
class 4	0	1	0	1	0	1	0

Figure 4.2: ECOC code matrix

In this example of Figure 4.2 we will look only at the first two columns. Consider first the binary classifier f_1. We see that metaclass C^+ is formed by class 1 only, and metaclass C^- is formed by classes 2,3 and 4. For the binary classifier f_2 , the metaclass C^+ is formed by classes 1 and 4, the metaclass C^- is formed by classes 2 and 3. Now we look at the first two rows, ideally, if all $f_1, f_2, \cdots, f_7$ classify correctly and the ordered output list is $(f_1, f_2, \cdots, f_7) = (1, 1, \cdots, 1)$ (row 1 of M), then the object is of class 1. If e.g. the ordered output list is $(f_1, f_2, \cdots, f_7) = (0, 0, 0, 0, 1, 1, 1)$ (row 2 of M), the class of the object is of class 2.

If we want to classify a new object with feature vector x using ECOC, first we evaluate all the l binary classifiers for x, and we get a binary vector $\lambda = [f_1(x), f_2(x), \cdots, f_l(x)]$. Ideally for a sample (i.e. a feature vector) of class k , $f_i(x) = 1$ if class k is in metaclass C_i^+ else $f_i(x) = 0$. Then we compare each row M_i of M with λ by using some distance measure. One of the possible distance measures which is used is the *Hamming distance*, d_H.

Definition 4.3.1: Let $a, b \in \{0,1\}^{1\times l}$ be binary row vectors of length l. Then

$$\begin{aligned} d_H(a,b) &= \textit{number of components in which } a \neq b \\ &= \sum_{i=1}^{l} |a_i - b_i| \end{aligned}$$

is called the *Hamming distance between a and b.*

Basically the Hamming distance is the number of bit positions where two binary vectors differ

(e.g. in Figure 4.2 the Hamming distance, $d_H(r_1, r_2)$, between *row one* and *row two* is 4). After the comparison of each row M_i of M with λ, the row M_i which has the smallest Hamming distance to λ will be assigned as a class of a new object with feature vector x, since each row of M represents a class.

Figure 4.3 shows a classification of a new sample using ECOC for a 4-class classification problem, the binary vector is given by $\lambda = [\ 1 \quad 0 \quad 1 \quad 1 \quad 0 \quad 0 \quad 1\ ]$. We see that in Figure 4.3 the row of M which has the minimum Hamming distance is row 3, hence the class assigned to the new object is class 3.

λ	1	0	1	1	0	0	1	d_H
class 1	1	1	1	1	1	1	1	3
class 2	0	0	0	0	1	1	1	5
class 3	0	0	1	1	0	0	1	1
class 4	0	1	0	1	0	1	0	5
	f_1	f_2	f_3	f_4	f_5	f_6	f_7	

decision: class 3

Figure 4.3: New sample evaluation using ECOC

In [7] it is suggested that for a good ECOC, the rows and the columns of M must be well separated according to the Hamming distance. Hence the aim of ECOC is to make the Hamming distances between rows and between columns of M large. The row separation ensures that the classes are assigned different strings of binary digits because each class is different from the other class. The purpose of the column separation is to ensure that columns, say, i and j are not similar or identical. If these two columns i and j are similar or identical, then when a learning algorithm is applied to learn f_i and f_j, it will make similar (correlated) mistakes. Error Correcting Output Codes only succeed if the errors made in the bit positions are uncorrelated [7].

Now we determine the number of errors that ECOC can correct which is stated in the following Theorem 4.3.1. For Theorem 4.3.1, we need to prove the following Proposition 4.3.1. Before we prove this Proposition 4.3.1, we state the following definitions.

Definition 4.3.2: A codeword is a string of zeroes and ones. Hence each row of a code matrix M is a codeword.

Definition 4.3.3: $d_{H_{min}}(M)$ is the minimum Hamming distance between any pair of rows of M.

Lemma 4.3.1: The Hamming distance d_H is a metric [6], hence it satisfies the triangle inequality $d_H(a,b) + d_H(b,c) \geq d_H(a,c)$.

Proposition 4.3.1: [17] *A code matrix M can correct up to t errors in any codeword if $d_{H_{min}}(M) \geq 2t+1$.*

Proof: Suppose $d_{H_{min}}(M) \geq 2t+1$. Let a codeword s in M be the target codeword for the sample x. Suppose that x is evaluated by using ECOC which results in a codeword r that contains t or fewer errors. Then $d_H(s,r) \leq t$. Let s' be any codeword other than s in M. Then $d_H(s',s) \geq d_{H_{min}}(M) \geq 2t+1$. With the Hamming distances among s, s', and r that satisfy the triangle inequality

$$d_H(s,r) + d_H(s',r) \geq d_H(s,s'),$$

we get

$$d_H(s',r) \quad \geq \quad d_H(s,s') - d_H(s,r) \geq 2t+1-t \geq t+1.$$

Therefore

$$d_H(s,r) < d_H(s',r). \tag{4.1}$$

The inequality of (4.1) shows that s is the nearest codeword to r and hence x will be classified correctly.■

Theorem 4.3.1: [17] *If $d_{H_{min}}$ is the minimum Hamming distance between any pair of rows of M then the ECOC classification strategy can correct at least $\left\lfloor \frac{d_{H_{min}}-1}{2} \right\rfloor$ incorrect bits.*

Proof: Suppose that $d_{H_{min}}$ is the minimum Hamming distance between any pair of rows of M.

Then using proposition 4.3.1, we have

$$\begin{aligned} d_{H_{min}} &\geq 2t+1 \\ \Rightarrow t &\leq \frac{d_{H_{min}}-1}{2}. \end{aligned}$$

Since t must be a positive whole number, we take the floor of the fraction $\frac{d_{H_{min}}-1}{2}$, and we get

$$t = \left\lfloor \frac{d_{H_{min}}-1}{2} \right\rfloor.$$

This completes the proof.■

It is suggested in [7] that if a multiclass classification problem has c classes and that $3 \leq c \leq 7$, then *exhaustive codes* are used. An optimal exhaustive code is constructed in the following way (see [7]):

- Row 1 of M consists of all ones.
- Row 2 consists of 2^{c-2} zeros followed by $2^{c-2}-1$ ones.
- Row 3 consists of 2^{c-3} zeros, followed by 2^{c-3} ones, followed by $2^{c-3}-1$ ones.
- In row i, there are alternating runs of 2^{c-i} zeros and ones.

The use of exhaustive codes is to ensure that the columns as well as the rows of a code matrix are well separated. As noted before this results in good ECOC.

4.4 ECOC and 1-nearest neighbour

Suppose that we have a $c-$class classification problem. Then in Theorem 4.4.1 we will show that the 1-nearest neighbour classifier classifies exactly in the same way as the ECOC-1-nearest neighbour classifier (i.e. ECOC with all possible binary 1-nearest neighbour binary classifiers). First we introduce the following notation:

- M_{ik} is the entry of a code matrix M in row i column k;
- f_k is the k-th binary classifier;
- C_k^+ the metaclass of the k-th binary classifier f_k.

Then $M_{ik} = 1$ if and only if class $\omega_i \in \{\omega_1, \omega_2, \cdots, \omega_c\}$ is in metaclass C_k^+ of the binary classifier f_k. If class ω_i "wins" the nearest neighbour competition, then for the 1-nearest neighbour binary classifier f_k we have

$$classification = \begin{cases} 1 & if\ \omega_i \in C_k^+ \\ 0 & if\ \omega_i \in C_k^- \end{cases} \tag{4.2}$$

Theorem 4.4.1: T*he class assigned by 1-nearest neighbour classifier is the same class that is assigned by ECOC-1-nearest neighbour classifier.*

Proof: Suppose that the ECOC-1-nearest neighbour classifier is used to classify an object with feature vector x in a $c-$class classification problem. Let class ω_i be a "winner" class (i.e. the class nearest to the new object). Then using (4.2) we have classification 1 if and only if $\omega_i \in C_k^+$. But this means that the output vector of all $(f_1, f_2, \cdots, f_l)$ is just the $i-$th row of M and ECOC-1-nearest neighbour will assign class ω_i.■

As an illustration, consider an example of a 4-class classification problem in Figure 4.4. We want to classify a sample with an unknown class using 1-Nearest neighbour method. We see that the nearest sample to this new sample comes from class 2, hence class 2 is the winner class of the competition. This new sample will be assigned to class 2. Now we use the ECOC matrix in Figure 4.5 to classify this new sample. Then we use (4.2) for classification of the 1-nearest neighbour binary classifiers:

- for f_1 we have $C_1^+ = \{class\ 1\}$, C_1^- $\{class\ 2,\ class\ 3,\ class\ 4\}$, hence the classification is: 0;
- for f_2 we have $C_2^+ = \{class\ 1,\ class\ 4\}$, C_2^- $\{class\ 2,\ class\ 3\}$, hence the classification is: 0;
- for f_3 we have $C_3^+ = \{class\ 1,\ class\ 2\}$, C_3^- $\{class\ 3,\ class\ 4\}$, hence the classification is: 1;

- for f_4 we have $C_4^+ = \{class\ 1,\ class\ 2,\ class\ 3\}$, C_2^- $\{class\ 4\}$, hence the classification is: 1.

This results in a classification vector $\begin{bmatrix} 0 & 0 & 1 & 1 \end{bmatrix}$, which is identical to the second row in Figure 4.5. Hence the class of this new sample is class 2.

We have shown that ECOC with 1-nearest binary classifiers classifies in the same way as the 1-nearest neighbour classifier for a $c-$class classification problem. This implies that there will be no improvement in classification when using 1-nearest neighbour binary classifiers for ECOC, compared to $c-$class 1-nearest neighbour classification.

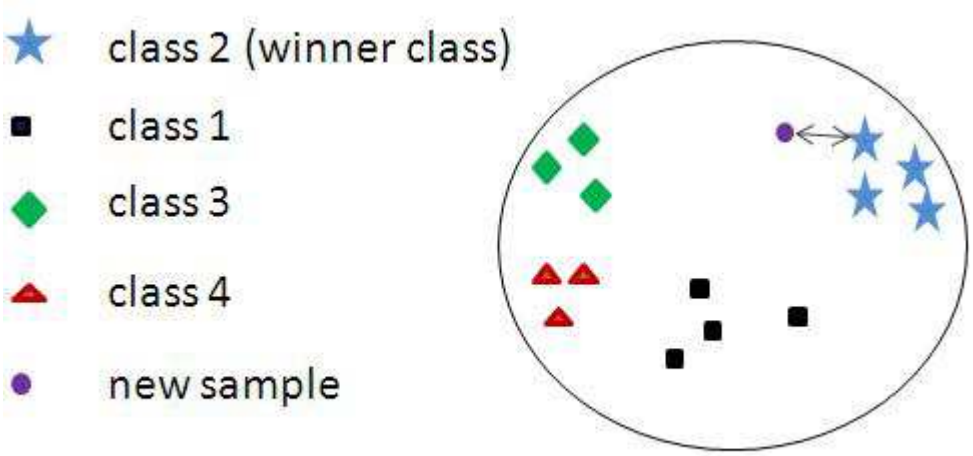

Figure 4.4: Example of ECOC and 1-nearest neighbour classification

$$\begin{array}{c} \\ class\ 1 \\ class\ 2 \\ class\ 3 \\ class\ 4 \end{array} \begin{array}{c} \begin{array}{cccc} f_1 & f_2 & f_3 & f_4 \end{array} \\ \begin{bmatrix} 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \end{bmatrix} \end{array}$$

Figure 4.5: Example of ECOC code matrix

5 Methods

Firstly we describe the data set that we are using. Then we discuss the details of the two classification methods that we used for this study. MATLAB (version 7.4.0.287(R2007a)) was used to programme both methods. The programme codes can be found in Appendix C.

5.1 Data Set

An Analytical Spectral Device (ASD) spectrometer (FieldSpec3 Pro FR) was used to record hyperspectral measurements of leaf samples taken from several different savannah tree species in the Kruger National Park in South Africa, in an attempt to assess tree species diversity in the park. The data was collected in May 2008. The leaves from the trees were placed into the leave clipper and the spectral signature was recorded. The study area is located in the "lowveld" savanna *biome* in the northeast South Africa. Figure 5.1 shows the study area.

Seven common plant tree species in the area were considered. The seven tree species include *Combretum apiculatum (CA), Combretum heroense (CH), Terminalia sericia (TS), Gymnospora sericia (GS), Lonchocarpus capassa (LC), Gymnospora buxifolia (GB), and Combretum zeyherrea (CZ)*. The number of observations for each species are 23, 20, 22, 18, 25, 21, and 19, respectively. The total data set therefore had 148 observations for the species measurements. Table 1 shows these different tree species. The wavelength range of the data is $400nm$ to $2500nm$ at a spectral resolution of $1nm$. Hence the hyperspectral data consists of 2101 spectral bands. The dimension of our data set is thus 2101×148, i.e. we have 148 samples and each sample has 2101 dimensions (or variables).

Figure 5.2 and Figure 5.3 show reflectance spectra of *Combretum apiculatum* and *Combretum heroense*, respectively. These reflectance spectra show a large within-species variability combined with a small between-species variability. If we superimpose reflectance spectra of *Combretum apiculatum* on the *Combretum heroense* reflectance spectra, there will be a strong overlap of the spectra. This strong overlap is underlined in Figure 5.4 which shows the mean reflectances for each of the 7 classes together with the $2\sigma-$bands (σ is the empirical standard deviation). Hence this overlap implies that there is a small between-species variability. The large within-species variability and the small between-species variability occurs for all the seven

tree species, this is shown by profile plots in the Appendix B.

class label	**species name**	**abbreviation**	**number of samples**
class 1	*Lonchocarpus Capassa*	*LC*	25
class 2	*Combretum Apiculatum*	*CA*	23
class 3	*Combretum Heroense*	*CH*	20
class 4	*Combretum Zeyherrea*	*CZ*	19
class 5	*Gymnospora Buxifolia*	*GB*	21
class 6	*Gymnospora Senegalensis*	*GS*	18
class 7	*Terminalia Sericia*	*TS*	22
Total number of samples			148

Table 1: 7 different tree species

Figure 5.1: Study Area [19]

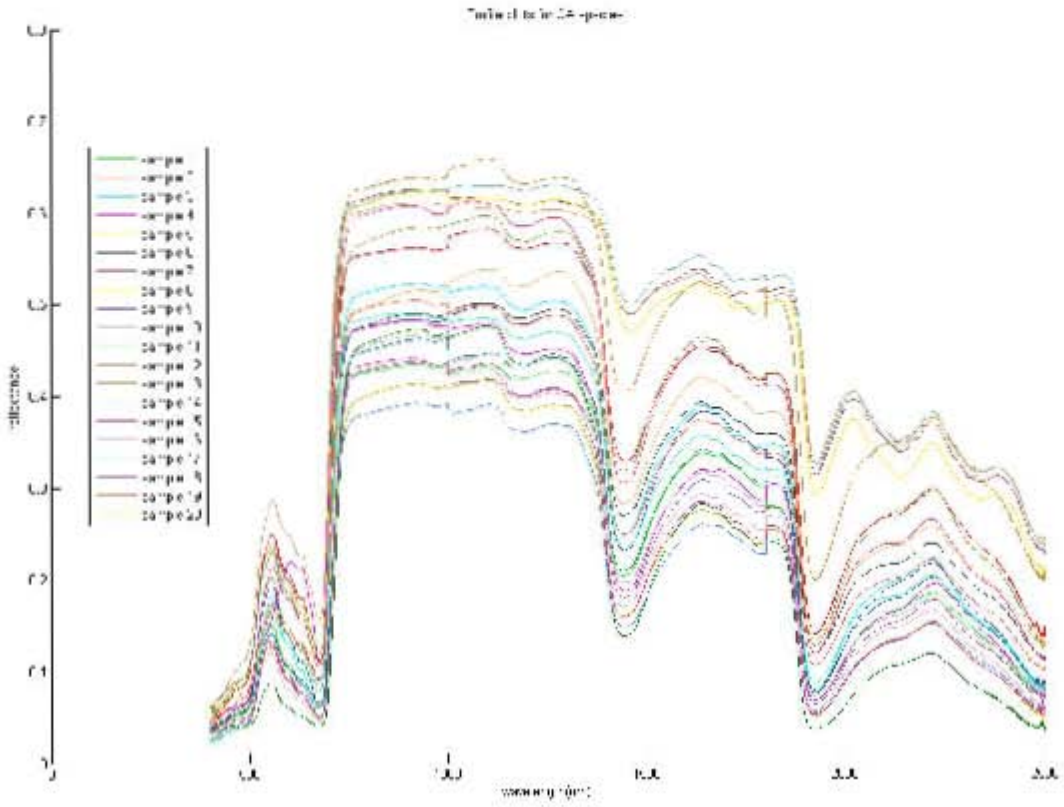

Figure 5.2: 20 samples reflectance curves of Combretum Apiculatum

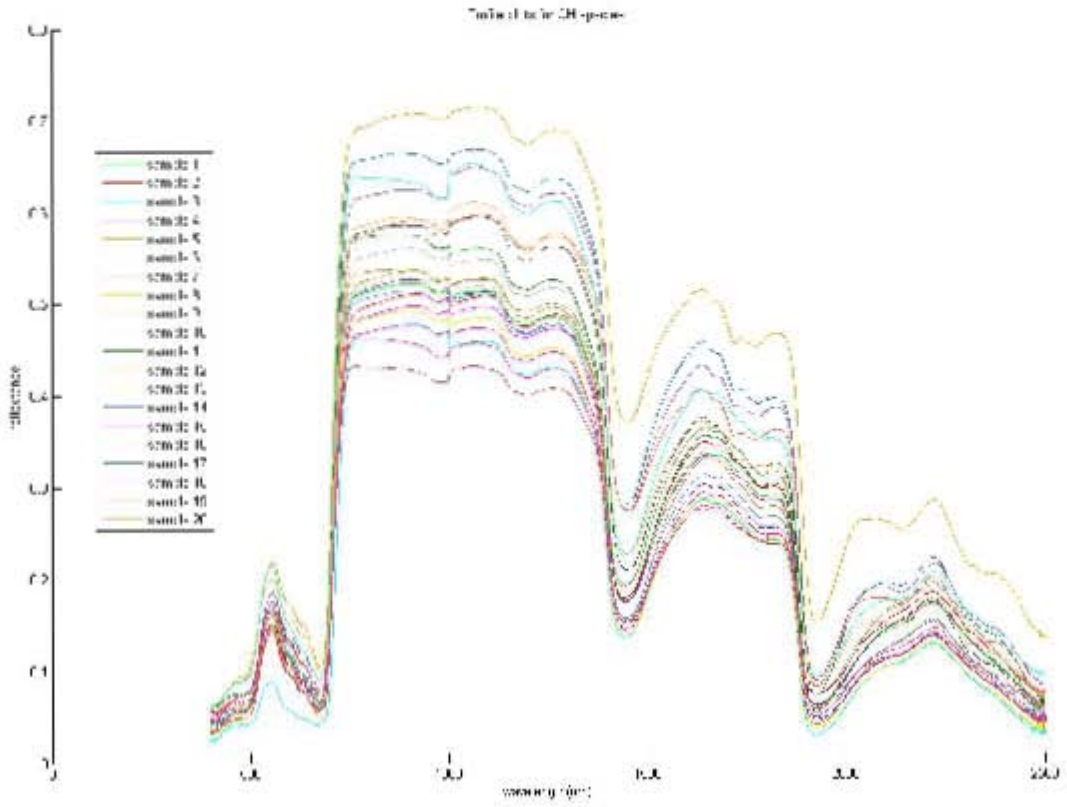

Figure 5.3: 20 samples reflectance curves of Combretum Heroense

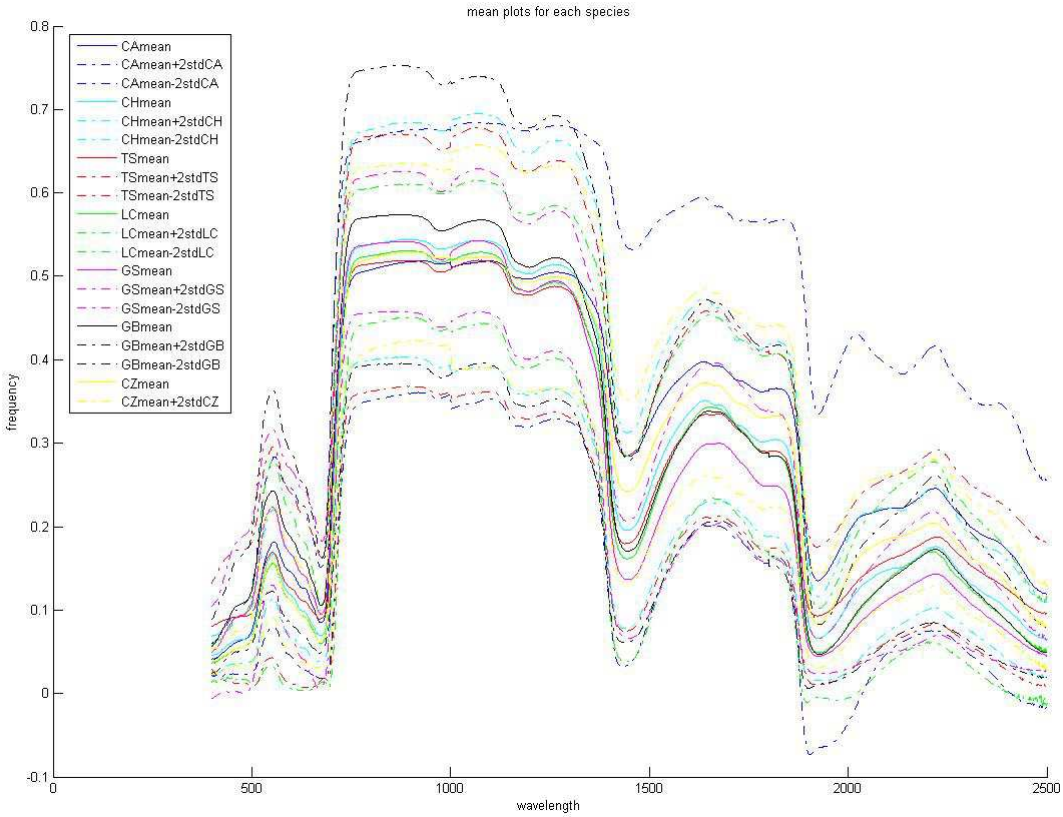

Figure 5.4: Mean reflectances of each species with 2σ bands

5.2 Approaches

We employed two methods to classify hyperspectral data: G-nearest Neighbour and Neural Networks classifiers. For both methods we randomly split the data into training sets and test sets using test-set approach to find opimal parameters of our classifiers and then K-fold cross validation approach to estimate the error rates of the classifiers with optimal parameters. We then decomposed the seven-class classification problem into $l = 2^{7-1} - 1 = 63$ binary classification problems. For each binary classification problem we constructed a binary classifier using the above mentioned methods. First we used all 2101 spectral bands for each experiment. Then we used sequential selection of spectral bands, for example we used every tenth wavelength band, every twentieth wavelength band and every thirtieth wavelength band for each experiment.

5.2.1 Test set approach

For the test-set approach we randomly split the data set into 70% training set and 30% test set. The training set was used to construct a classifier and the test set was used to estimate the error probability of a classifier. The performance of a classifier is based on the estimate of its error probability on the test set (see section 3.3.1). The test set acts as unseen independent data set. We performed ten (70%, 30%) random splits of the data for each experiment.

5.2.1.1 G-nearest neighbour method

The training data was used to construct a classifier using G-nearest neighbour method for different values of G in the set $\{1, 2, 5, 10, 12\}$, and its error probability was estimated on the test set. The test set approach was used to decide for the optimal number G that is used later.

5.2.1.2 Neural networks method

For neural networks, initial random values for weights and biases were assigned. The neural network toolbox (version 5.0.2 (R2007a))) of MATLAB was used. Then the training set was presented to the neural network to produce network outputs. Resilient backpropagation training was used. The default parameters of the MATLAB toolbox were used for the resilient backpropagation method. The training parameter "goal" was set to 0.03. The goal parameter determines when to stop training the neural network. This means that if the performance function (i.e. Mean Square Error) drops below the goal, the training stops. The number of times the training data was presented to the neural network was 1000 (i.e. 1000 epochs). Experience showed that these parameters (i.e. epochs and goal) are useful and we did not change these. It would be better, however, also to optimise these but due to lack of time we just kept them. We constructed two neural networks. The first neural networks had one hidden layer. The second neural networks had two hidden layers. The number of hidden neurons were taken from the set, $\{5, 10, 12, 15, 20\}$. For the neural networks with two hidden layers, we used the same number of hidden neurons for each layer. The test set approach was used to decide on the number of hidden layers and hidden neurons which will be used later.

5.2.2 K-fold cross validation approach

In this study we want to maintain the proportion of classes on each fold because we do not want to lose some classes in the folds. Hence we used stratified K-fold cross validation. The number of folds that we have used for cross validation is 7 (seven) and 10 (ten). After we have determined the optimal parameters from test set approach, we used K-fold cross validation to estimate the error probability of the seven-class classifier. Again, because of the random split in K-fold cross validation approach we performed 10 experiments and averaged the results.

5.2.3 Improvement using ECOC

The seven-class classification problem was decomposed into 63 binary classification problems. For each binary classification problem, we constructed a binary classifier using G-nearest neighbour and neural networks methods. We compared the ECOC combiner (which was obtained by combining neural networks binary classifiers) with the corresponding neural networks 7-class predictor. Also we compared the ECOC combiner (which was obtained by combining 1-nearest neighbour binary classifiers) with the corresponding 1-nearest neighbour 7-class predictor. We used the same parameters for the 7-class neural networks classifier and neural networks binary classifiers, i.e. we did not try to optimize these. In principle, for neural networks binary classifiers, different parameters could be optimal.

5.2.4 Leave away "bad binary classifiers" from ECOC

We estimated the error probability for each of the 63 binary classifiers which were formed by G-nearest neighbour and neural networks on the test set. Some of the binary classifiers produced very poor predictions, and we decided to leave these away and not to use them for ECOC combiner, hoping to improve prediction accuracy. If the error probability estimate of a binary classifier f_k was worse than a prescribed error probability cut-off value, then we deleted the binary classifier f_k. We used two error probability cut-off values, 10% and 15%. Ideally, the error probability of the resulting ECOC combiner should be estimated on an unseen data set but because of the limited size of our data set, we used the same test set again. Hence our estimates may be overly optimistic.

6 Results

The results of our experiments are summarized in Tables 2 to 26. The full details of the results are in Appendix A. In the first set of experiments, we use the test set approach (see section 3.3.1) to determine "optimal" parameters for the G-Nearest Neighbour classifier and the Neural Networks classifier. The optimal parameters that we are looking at are (i) the "optimal" number of nearest neighbours for the G-nearest neighbour classifier, (ii) the "optimal" number of hidden neurons, and (iii) the "optimal" number of hidden layers for the neural networks classifier. We choose optimal parameters according to the error probability estimate $P(error)$ of a classifier.

In the second set of experiments, we use these optimal parameters to construct the classifiers (i.e. G-nearest neighbour classifier and neural networks classifier). The hyperspectral data that we are using has 7 classes, hence we construct 7-class predictors (i.e. classifiers) using G-nearest neighbour method and neural networks method. Then we decompose the 7-class classification problem into a set of binary classification problems. We construct a classifier for each binary classification problem and then we combine the binary classifiers using Error Correcting Output Code to obtain a 7-class predictor. The K-fold cross validation approach (see section 3.3.2) was used to estimate the error probabilities of the classifiers.

6.1 Choice of "optimal" parameters

We run ten experiments using all the bands (i.e. 400nm-2500nm) and sequential selection of bands (i.e. every 10th band, every 20th band and every 30th band) and we estimate the error probability $P(error)$ of a classifier for each experiment. Then we average the ten error probability estimates, and collect these averages into the tables. The parameters which cause small averages of the error probability $P(error)$ estimates are considered as optimal in this study. Below we show the number of dimensions for the chosen bands:

- all bands = 2101 dimensions;
- every 10th band = 210 dimensions;
- every 20th band = 105 dimensions;

- every 30th band = 70 dimensions.

6.1.1 Nearest Neighbour

Table 2 shows the averages (over 10 experiments) of the error probability estimates $P(error)$ with averages (over 10 experiments) of the standard errors of $P(error)$ in brackets ( ) for different nearest neighbours G. We see that in Table 2, the optimal number of nearest neighbours for all the bands, every 10th band, every 20th band and every 30th band is 1 (one), as this is the only nearest neighbour number G which gives small averages of the error probability estimates. Table 3 shows 95% confidence intervals of the average of $P(error)$ for different nearest neighbours G. Most of these confidence intervals overlap, hence there is no significant difference between the average error probability estimates $P(error)$. Thus our choice of using 1 (one) as the optimal value for G is not the best choice. We could use any nearest neighbour number G as the optimal value, but we decided to choose 1 (one) based on the best point estimates. We also keep in mind that the estimate of the standard error of $P(error)$ is questionable, see [3], and that our sample size is small, which also make the application of the Central Limit Theorem dubious.

Number of nearest neighbours G	**Bands**			
	all bands	**every 10th**	**every 20th**	**every 30th**
1	**0.3182** (0.0701)	**0.3227** (0.0705)	**0.3159** (0.0700)	**0.3386** (0.0715)
2	0.3955 (0.0741)	0.4705 (0.0757)	0.3977 (0.0741)	0.3955 (0.0740)
5	0.4818 (0.0753)	0.4818 (0.0753)	0.4864 (0.0753)	0.4750 (0.0749)
10	0.5568 (0.0751)	0.5591 (0.0751)	0.5568 (0.0752)	0.5977 (0.0745)
12	0.6227 (0.0735)	0.6227 (0.0735)	0.6159 (0.0737)	0.5773 (0.0746)

Table 2: Averages of the error probabilities on the test set over 10 experiments for different nearest neighbours G (averages of the standard errors of the error probabilities in brackets)

95% Confidence Intervals				
Number of nearest neighbours G	**Bands**			
	all bands	**every 10th**	**every 20th**	**every 30th**
1	(0.1808; 0.4556)	(0.1845; 0.4609)	(0.1787; 0.4531)	(0.1985; 0.4787)
2	(0.2503; 0.5407)	(0.3221; 0.6189)	(0.2525; 0.5429)	(0.2505; 0.5405)
5	(0.3342; 0.6294)	(0.3342; 0.6294)	(0.3388; 0.6340)	(0.3282; 0.6218)
10	(0.4096; 0.7040)	(0.4119; 0.7063)	(0.4094; 0.7042)	(0.4517; 0.7437)
12	(0.4786; 0.7668)	(0.4786; 0.7668)	(0.4714; 0.7604)	(0.4311; 0.7235)

Table 3: 95% confidence intervals for $P(error)$ averages using different nearest neighbours G

6.1.2 Neural Networks

Table 4 shows the averages (over 10 experiments) of the error probability estimates $P(error)$ together with averages of the standard errors of $P(error)$ in brackets ( ). We see that, (i) if we use all the bands, we get a small average of the error probability estimate when the number of hidden neurons is 5, (ii) if we use every 10th band, the small average value of $P(error)$ occurs when we use 20 hidden neurons, (iii) if we use every 20th band, the small average value of $P(error)$ occurs when we use 15 hidden neurons, and (iv) if we use every 30th band, the small average value of $P(error)$ occurs when we use 20 hidden neurons. In Table 5 we see that, (i) if we use all the bands, we get a small average of the error probability estimate when the number of hidden neurons is 20, (ii) if we use every 10th band, the small average value of $P(error)$ occurs when we use 15 hidden neurons, (iii) if we use every 20th band, the small average value of $P(error)$ occurs when we use 20 hidden neurons, and (iv) if we use every 30th band, the small average value of $P(error)$ occurs when we use 12 hidden neurons. We then compare Tables 4 and 5 to determine the number of optimal hidden layers based on the low point estimate. We see that Table 4 gives averages of the error probability estimates that are smaller than those given by Table 5. Hence, we will use one hidden layer for our Neural Networks later. Here, for the neural networks classifier with two hidden layers we used the same number of hidden neurons in both layers. Again our choices of the optimal values are not the best, since the confidence intervals shown in Tables 6 and 7 overlap. Hence there is no significant difference between the average error probability estimates $P(error)$. Again we keep in mind that the standard error estimates and the confidence intervals may be unreliable.

Number of hidden neurons	**Bands**			
	all bands	**every 10th**	**every 20th**	**every 30th**
5	0.4023 (0.0742)	0.3364 (0.0711)	0.4182 (0.0741)	0.4182 (0.0750)
10	**0.2432** (0.0646)	0.2955 (0.0686)	0.2318 (0.0627)	0.2864 (0.0679)
12	0.2909 (0.0684)	0.2250 (0.0628)	0.2318 (0.0627)	0.2795 (0.0676)
15	0.2977 (0.0686)	0.2477 (0.0645)	**0.2205** (0.0626)	0.2659 (0.0669)
20	0.3114 (0.0699)	**0.2182** (0.0615)	0.2295 (0.0633)	**0.2409** (0.0643)

Table 4: Averages of error probabilities on the test set over 10 experiments for different number of hidden neurons for one hidden layer (averages of the standard errors of the error probabilities in brackets).

Number of hidden neurons	Bands			
	all bands	**every 10th**	**every 20th**	**every 30th**
5	0.4795 (0.0727)	0.4250 (0.0745)	0.4477 (0.0745)	0.4295 (0.0740)
10	0.4295 (0.0728)	0.3159 (0.0689)	0.3068 (0.0687)	0.2955 (0.0677)
12	0.3523 (0.0705)	0.3068 (0.0681)	0.3432 (0.0713)	**0.2591** (0.0660)
15	0.3523 (0.0726)	**0.2545** (0.0645)	0.3091 (0.0698)	0.3318 (0.0703)
20	**0.3159** (0.0701)	0.3045 (0.0683)	**0.2727** (0.0671)	0.2682 (0.0651)

Table 5: Averages of error probabilities on the test set over 10 experiments for different number of hidden neurons for two hidden layers (averages of the standard errors of the error probabilities in brackets).

95% Confidence Intervals				
Number of hidden neurons	**Bands**			
	all bands	**every 10th**	**every 20th**	**every 30th**
5	(0.2569; 0.5477)	(0.1970; 0.4758)	(0.2730; 0.5634)	(0.2712; 0.5652)
10	(0.1166; 0.3698)	(0.1610; 0.4300)	(0.1089; 0.3547)	(0.1533; 0.4195)
12	(0.1568; 0.4250)	(0.1019; 0.3481)	(0.1089; 0.3547)	(0.1470; 0.4120)
15	(0.1632; 0.4322)	(0.1213; 0.3741)	(0.0978; 0.3432)	(0.1348; 0.3970)
20	(0.1744; 0.4484)	(0.0977; 0.3387)	(0.1054; 0.3536)	(0.1149; 0.3669)

Table 6: 95% confidence intervals of $P\,(error)$ averages using different hidden neurons (1 hidden layer).

95% Confidence Intervals				
Number of hidden neurons	**Bands**			
	all bands	**every 10th**	**every 20th**	**every 30th**
5	(0.3370; 0.6220)	(0.2790; 0.5710)	(0.3017; 0.5937)	(0.2845; 0.5745)
10	(0.2868; 0.5722)	(0.1809; 0.4509)	(0.1721; 0.4415)	(0.1628; 0.4282)
12	(0.2141; 0.4905)	(0.1733; 0.4403)	(0.2035; 0.4829)	(0.1297; 0.3885)
15	(0.2100; 0.4946)	(0.1281; 0.3809)	(0.1723; 0.4459)	(0.1940; 0.4696)
20	(0.1785; 0.4533)	(0.1706; 0.4384)	(0.1412; 0.4042)	(0.1406; 0.3958)

Table 7: 95% confidence intervals of $P\,(error)$ averages using different hidden neurons (2 hidden layers).

6.2 Contrasting 7-class predictors : Nearest Neighbour vs Neural Networks

Tables 8 and 9 show the averages of the error probability estimates (over 10 experiments) of the 7-class classifiers with optimal parameters (i.e. 10 hidden neurons for all the bands, 20 hidden neurons for every 10th band, 15 hidden neurons for every 20th band, and 20 hidden neurons for every 20th band). These averages were obtained by using the K-fold cross validation approach (for K=10 and K=7). Tables 8 and 9 show that the neural network classifier outperforms the 1-nearest neighbour classifier by roughly 7%. However, even neural networks classifiers are not good enough for practical use. The fact that the 1-nearest nearest neighbour

is outperformed is in contrast with Theorem 3.2.1, which states that the 1-nearest neighbour classifier error probability is never worse than twice the optimal Bayes error rate. But Theorem 3.2.1 applies only when the number of samples approaches infinity. Hence the contrast may be due to the small sample size (i.e 148 samples) of this study.

The choice of the number of folds, K, was based on the average error probability estimates in Tables 8 and 9. The averages of the error probability estimates in Tables 8 and 9 are not significantly different (shown by the overlaping confidence intervals in Tables 10 and 11). Both estimates are close, which may indicate that the estimates are "good". Hence we decided to use K=10 for our subsequent experiments.

10-fold cross validation	**Bands**			
7-class classifiers	**all bands**	**every 10th**	**every 20th**	**every 30th**
Nearest Neighbour $P(error)$	0.3356 (0.0391)	0.3409 (0.0392)	0.3340 (0.0390)	0.3353 (0.0390)
Neural Networks $P(error)$	0.2772 (0.0370)	0.2401 (0.0352)	0.2582 (0.0359)	0.2528 (0.0358)

Table 8: Averages of the error probabilities on the test set for contrasting 7-class classifiers: Nearest Neighbour vs Neural Networks (10-fold cross validation) (averages of the standard errors of the error probabilities in brackets).

7-fold cross validation	**Bands**			
7-class classifiers	**all bands**	**every 10th**	**every 20th**	**every 30th**
Nearest Neighbour $P(error)$	0.3362 (0.0389)	0.3443 (0.0392)	0.3263 (0.0387)	0.3262 (0.0387)
Neural Networks $P(error)$	0.2878 (0.0373)	0.2598 (0.0360)	0.2475 (0.0354)	0.2218 (0.0341)

Table 9: Averages of the error probabilities on the test set for contrasting 7-class classifiers: Nearest Neighbour vs Neural Networks (7-fold cross validation) (averages of the standard errors of the error probabilities in brackets).

95% Confidence Intervals				
10-fold cross validation	**Bands**			
7-class classifiers	**all bands**	**every 10th**	**every 20th**	**every 30th**
Nearest Neighbour $P(error)$	(0.2590; 0.4122)	(0.2641; 0.4177)	(0.2576; 0.4104)	(0.2589; 0.4117)
Neural Networks $P(error)$	(0.2047; 0.3497)	(0.1711; 0.3091)	(0.1878; 0.3286)	(0.1826; 0.3230)

Table 10: 95% confidence intervals of $P(error)$ averages for contrasting 7-class classifiers: Nearest Neighbour vs Neural Networks (10-fold cross validation).

95% Confidence Intervals				
7-fold cross validation	**Bands**			
7-class classifiers	**all bands**	**every 10th**	**every 20th**	**every 30th**
Nearest Neighbour $P\,(error)$	(0.2600; 0.4124)	(0.2675; 0.4211)	(0.2504; 0.4022)	(0.2503; 0.4021)
Neural Networks $P\,(error)$	(0.2147; 0.3609)	(0.1892; 0.3304)	(0.1781; 0.3169)	(0.1550; 0.2886)

Table 11: 95% confidence intervals of $P\,(error)$ averages for contrasting 7-class classifiers: Nearest Neighbour vs Neural Networks (7-fold cross validation).

6.3 Improvement with ECOC

Table 12 shows the error probability estimates of some of the binary classifiers for both 1-nearest neighbour and neural networks. The first binary classifier tries to discriminate class 1 from the rest of the classes (i.e. classes 2, 3, 4, 5, and 7), the second binary classifier tries to discriminate classes 1, 5 and 6 from classes 2, 3, 4, and 7, the third binary classifier tries to discriminate classes 4, 5, and 7 from classes 2, 3, 5, and 6. In comparison to 7-class error probability estimates shown in Table 8, binary error probability estimates in Table 12 are better. So instead of using direct 7-class classifier, we decided to use binary classifiers (because they give "small" error probability estimates compared to 7-class classifier) and combine these binary classifiers using Error Correcting Output Code. We hope this combination of binary classifiers will improve classification of the seven savannah tree species.

	Binary classifiers		
	1 vs (2,3,4,5,6,7)	(1,5,6) vs (2,3,4,7)	(1,4,7) vs (2,3,5,6)
1-Nearest Neighbour $P\,(error)$	0.1400	0.1400	0.0700
Neural Networks $P\,(error)$	0.0800	0.0800	0.1700

Table 12: Some error probability estimates of the 1-nearest neighbour binary classifier and the neural networks on the test set (all bands).

6.3.1 The 7-class predictors versus ECOC combiners

6.3.3.1 Nearest Neighbour as binary classifiers

Table 13 shows the averages (over 10 experiments) of the error probability estimates together with the averages (over 10 experiments) of the standard errors of $P\,(error)$ when using 7-class 1-nearest neighbour classifier and ECOC combiner. We see that ECOC combiner and 1-nearest neighbour classifier give identical error probability estimates, hence there is no improvement in classification when using ECOC combiner with 1-nearest neighbour binary classifiers. This was explained in section 4.4. Table 14 shows that when using 5-nearest neighbours, ECOC

combiner does not improve classification. The increase in the number of nearest neighbours G caused the binary classifiers to perform bad. Hence ECOC combiner which is constructed by combining these bad 5-nearest neighbour binary classifiers will also perform bad. The cause of this bad perfomance of the ECOC combiner is going to be explained in detail in section 6.4.

6.3.3.2 Neural Networks as binary classifiers

Table 15 shows the averages of the error probability estimates (over 10 experiments) together with the averages of the standard errors of $P(error)$ for the 7-class neural networks classifier and ECOC combiner (using neural networks binary classifiers with same parameters as the 7-class neural networks classifiers). We see here that the ECOC combiner with neural networks binary classifiers improves the classification with a difference of about 13% when using all the bands. If we use every 10th band, classification improved by 8% difference, and if we use every 20th band, classification improved by 12% difference), and if we use every 30th band, classification improved by 10% difference. In Table 15, the averages of the error probability estimates for the 7-class classifier (which have approximately 25% error probability values) using the chosen bands are not acceptable, but we can perhaps live with ECOC combiner error probability estimates (which have approximately 15% error probability values). Again the 95% confidence intervals overlap, hence there is no significant difference between the average error probability estimates $P(error)$. In Table 18, we see that all the intervals overlap hence there is no significant difference between the error probability estimates of the 7-class classifier and ECOC predictor. Note that the standard error estimates may be unreliable, because we are using K-fold cross validation and that the sample size is small.

1-Nearest Neighbour classifiers	**Bands**			
	all bands	**every 10th**	**every 20th**	**every 30th**
7-class classifier $P(error)$	0.3356 (0.0391)	0.3409 (0.0392)	0.3340 (0.0390)	0.3353 (0.0390)
ECOC combiner $P(error)$	0.3356 (0.0391)	0.3409 (0.0392)	0.3340 (0.0390)	0.3353 (0.0390)

Table 13: Averages of the error probabilities on the test set: Improvement with Error Correcting Output Code when using 1- Nearest Neighbour binary classifiers (10-fold cross validation) (averages of the standard errors of the error probabilities in brackets).

5-Nearest Neighbour classifiers	**Bands**			
	all bands	**every 10th**	**every 20th**	**every 30th**
7-class classifier $P\,(error)$	0.4818 (0.0753)	0.4955 (0.0756)	0.4864 (0.0753)	0.4750 (0.0749)
ECOC combiner $P\,(error)$	0.4932 (0.0753)	0.5227 (0.0754)	0.4977 (0.0753)	0.4795 (0.0753)

Table 14: Averages of the error probabilities on the test set: Improvement with Error Correcting Output Code when using 5- Nearest Neighbour binary classifiers (10-fold cross validation) (averages of the standard errors of the error probabilities in brackets).

Neural Networks classifiers	**Bands**			
	all bands	**every 10th**	**every 20th**	**every 30th**
7-class classifier $P\,(error)$	0.2772 (0.0370)	0.2401 (0.0352)	0.2582 (0.0359)	0.2528 (0.0358)
ECOC combiner $P\,(error)$	0.1480 (0.0294)	0.1561 (0.0299)	0.1415 (0.0286)	0.1492 (0.0294)

Table 15: Averages of the error probabilities on the test set: Improvement with Error Correcting Output Code when using Neural Network binary classifiers (10-fold cross validation) (averages of the standard errors of the error probabilities in brackets).

95% Confidence Intervals				
1-Nearest Neighbour classifiers	**Bands**			
	all bands	**every 10th**	**every 20th**	**every 30th**
7-class classifier $P\,(error)$	(0.2590; 0.4122)	(0.2641; 0.4177)	(0.2576; 0.4104)	(0.2589; 0.4117)
ECOC combiner $P\,(error)$	(0.2590; 0.4122)	(0.2641; 0.4177)	(0.2576; 0.4104)	(0.2589; 0.4117)

Table 16: 95% confidence intervals of the $P\,(error)$ averages: Improvement with Error Correcting Output Code when using 1- Nearest Neighbour binary classifiers (10-fold cross validation).

95% Confidence Intervals				
5-Nearest Neighbour classifiers	**Bands**			
	all bands	**every 10th**	**every 20th**	**every 30th**
7-class classifier $P\,(error)$	(0.3342; 0.6294)	(0.3473; 0.6437)	(0.3388; 0.6340)	(0.3282; 0.6218)
ECOC combiner $P\,(error)$	(0.3456; 0.6408)	(0.3749; 0.6705)	(0.3501; 0.6453)	(0.3319; 0.6271)

Table 17: 95% confidence intervals of the $P\,(error)$ averages: Improvement with Error Correcting Output Code when using 5- Nearest Neighbour binary classifiers (10-fold cross validation).

95% Confidence Intervals				
Neural Networks classifiers	**Bands**			
	all bands	**every 10th**	**every 20th**	**every 30th**
7-class classifier $P\,(error)$	(0.2047; 0.3497)	(0.1711; 0.3091)	(0.1878; 0.3286)	(0.1826; 0.3230)
ECOC combiner $P\,(error)$	(0.0904; 0.2056)	(0.0975; 0.2147)	(0.0854; 0.1976)	(0.0916; 0.2068)

Table 18: 95% confidence intervals of the $P\,(error)$ averages: Improvement with Error Correcting Output Code when using Neural Networks binary classifiers (10-fold cross validation).

6.4 Leaving away "bad" classifiers

If a binary classifier misclassifies, it tends to add a "wrong bit" to the codeword. ECOC can only correct $h = \left\lfloor \frac{d_{H_{min}}-1}{2} \right\rfloor$ (in our case $\left\lfloor \frac{32-1}{2} \right\rfloor = 15$) wrong bits. If the number of incorrect bits h exceeds 15, then the ECOC combiner can not correct the wrong extra bits. Hence, the ECOC combiner will perform bad and will decide for the wrong class. We therefore try to leave away "bad" binary classifiers, in hoping that classification of the savannah tree species will be further improved. The binary classifiers with error probability estimates that are greater or equal to a cut-off value are considered as "bad" binary classifiers. Note that these estimates are obtained from the test sets, i.e. we used the test set in the construction of the combiner.

6.4.1 Binary classifier performances

In the following Tables 19 and 20, we show the error probability estimates of the 63 binary classifiers (i.e. 1-nearest neighbour and neural networks) of the 10th experiment in the 10th fold when using 10-fold cross validation for all the bands. The binary classifiers which perform "poor" are highlighted. E.g the 37th binary classifier in Table 20 gives 50% error probability estimate, this implies that the 37th binary classifier classifies incorrectly half of the time, hence this is a poor binary classifier.

From binary classifier 1 to binary classifier 21																					
0.14	0	0.14	0	0.14	0	0.14	0	0.21	0.07	0.21	0.07	0.21	0.07	0.21	0.07	0.14	0	0.14	0	0.14
From binary classifier 22 to binary classifier 42																					
0	0.14	0	0.21	0.07	0.21	0.07	0.21	0.07	0.21	0.07	0.21	0.07	0.21	0.07	0.21	0.07	0.21	0.07	0.14	0	
From binary classifier 43 to binary classifier 63																					
0.14	0	0.14	0	0.14	0	0.21	0.07	0.21	0.07	0.21	0.07	0.21	0.07	0.14	0	0.14	0	0.14	0	0.14	

Table 19: Error probability estimates on the test set for 1-nearest neighbour binary classifiers (all bands).

From binary classifier 1 to binary classifier 21																				
0.08	0.08	0.17	0.08	0.33	0.08	0.08	0.08	0.17	0.17	0.25	0.17	0	0.25	0.17	0.17	0.25	0.33	0.25	0.08	0.17
From binary classifier 22 to binary classifier 42																				
0.42	0.33	0.17	0.08	0.08	0.08	0.33	0.08	0.08	0.25	0.08	0.08	0.08	0.17	0.08	0.50	0.17	0.25	0.08	0.33	0.17
From binary classifier 43 to binary classifier 63																				
0.33	0.08	0.33	0.08	0.08	0.08	0.08	0.08	0.17	0.08	0.33	0.08	0.08	0.08	0.08	0	0.08	0	0.08	0	0

Table 20: Error probability estimates on the test set for neural networks binary classifiers (all bands).

6.4.2 1-Nearest neighbour binary classifiers

Table 21 shows that, when using 10% as a cut-off value, the error probability estimates got worse (i.e. there was no improvement in classification). For 15% cut-off value, the error probability estimates are slightly less than the error probability estimates when using ECOC combiner, but this difference is not significant (shown by the overlapping confidence intervals in Table 22). Hence we still do not have an improvement in classification. All these error probability estimates are not acceptable in practice. All the confidence intervals overlap, hence there is no significant difference between the averages of $P(error)$.

	Bands			
1-nearest neighbour	**all bands**	**every 10th**	**every 20th**	**every 30th**
ECOC combiner $P(error)$	0.3356 (0.0391)	0.3409 (0.0392)	0.3340 (0.0390)	0.3353 (0.0390)
10% cut-off $P(error)$	0.3608 (0.0391)	0.3656 (0.0398)	0.3598 (0.0397)	0.3623 (0.0391)
15% cut-off $P(error)$	0.3330 (0.0390)	0.3377 (0.0391)	0.3320 (0.0389)	0.3333 (0.0398)

Table 21: Averages of the error probabilities on the test set: Improvement with Error Correcting Output Code leaving bad classifiers using Nearest Neighbour classifier (10-fold cross validation) (averages of the standard errors of the error probabilities in brackets).

95% Confidence Intervals				
	Bands			
1-nearest neighbour	**all bands**	**every 10th**	**every 20th**	**every 30th**
ECOC combiner $P(error)$	(0.2590; 0.4177)	(0.2641; 0.4177)	(0.2576; 0.4104)	(0.2589; 0.4117)
10% cut-off $P(error)$	(0.2842; 0.4436)	(0.2876; 0.4436)	(0.2820; 0.4376)	(0.2857; 0.4389)
15% cut-off $P(error)$	(0.2569; 0.4143)	(0.2611; 0.4143)	(0.2558; 0.4082)	(0.2553; 0.4113)

Table 22: 95% confidence intervals of the $P(error)$ averages: Improvement with Error Correcting Output Code leaving bad classifiers using Nearest Neighbour classifier (10-fold cross validation).

6.4.3 Neural networks binary classifiers

Table 23 shows a further improvement (approximately 10% improvement) in classification for the 10% error probability estimate cut-off value. Table 24 show the confidence intervals. We see that when using 10% error probability estimate as the cut-off value for bad binary classifiers, the confidence intervals do not overlap with the confidence intervals of the average error probability estimates of ECOC combiner when using every 10th band and every 30th band. This implies that ECOC combiner with 10% error probability cut-off value results in the averages of $P(error)$ which are significantly different from the averages of the $P(error)$ for ECOC combiner with all the binary classifiers when using every 10th band and every 30th band. This is not true for the 15% cut-off value, because the confidence intervals of the averages of $P(error)$ overlap with the confidence intervals of the averages of the $P(error)$ for the ECOC combiner with all the binary classifier. But the point estimates are however by about 8% better. The further improvement here may be due to the fact that the same test set was used to remove bad classifiers, and after, it was also used to estimate the error probability $P(error)$. We do not have enough data for a further independent set, hence this estimate of the error probability, $P(error)$, may be overly optimistic.

Tables 25 and 26 show averages of the error probability estimates of neural networks binary classifiers. These averages of the error probabilities are those of the 10th experiment of the 10th fold in 10-fold cross validation approach. In Tables 25 and 26 we also show the binary classifiers which are removed (those that are highlighted) when using 10% error probability estimate and 15% error probability estimate as cut-off values, respectively.

In the 10th experiment of the 10-fold cross validation, we collected all the "bad" neural networks binary classifiers which perform badly on each fold. Then we produced the histograms in Figure 6.1. The histograms in Figure 6.1 show that binary classifier 45 was the most frequently chosen binary classifiers as "bad" binary classifiers for 10% error probability cut-off value. For 15% error probability cut-off value, the most freuquently chosen binary classifier as bad binary classifier was binary classifier 20.

Classifiers	Bands			
	all bands	**every 10th**	**every 20th**	**every 30th**
ECOC combiner $P\,(error)$	0.1480 (0.0294)	0.1561 (0.0299)	0.1415 (0.0286)	0.1492 (0.0294)
10% cut-off $P\,(error)$	0.0580 (0.0195)	0.0494 (0.0177)	0.0574 (0.0192)	0.0545 (0.0188)
15% cut-off $P\,(error)$	0.0765 (0.0221)	0.0802 (0.0224)	0.0721 (0.0213)	0.0755 (0.0219)

Table 23: Averages of the error probabilities on the test set: Improvement with Error Correcting Output Code leaving bad classifiers using Neural Network classifier (10-fold cross validation) (averages of the standard errors of the error probabilities in brackets).

95% Confidence intervals				
Classifiers	**Bands**			
	all bands	**every 10th**	**every 20th**	**every 30th**
ECOC combiner $P\,(error)$	(0.0904; 0.2056)	(0.0975; 0.2147)	(0.0854; 0.1976)	(0.0916; 0.2068)
10% cut-off $P\,(error)$	(0.0198; 0.0962)	(0.0147; 0.0841)	(0.0198; 0.0950)	(0.0177; 0.0913)
15% cut-off $P\,(error)$	(0.0332; 0.1198)	(0.0363; 0.1241)	(0.0304; 0.1138)	(0.0326; 0.1184)

Table 24: 95% confidence intervals of the $P\,(error)$ averages: Improvement with Error Correcting Output Code leaving bad classifiers using Neural Network classifier (10-fold cross validation).

From binary classifier 1 to binary classifier 21																					
0	0.08	0	0.08	0	0.25	0	0.08	0	0	0.08	0	0	0	0	0	0.08	0.17	0.08	0.25	0.17
From binary classifier 22 to binary classifier 42																					
0.08	0	0.08	0.08	0.17	0.08	0.08	0	0.08	0	0	0.08	0	0	0	0.25	0.08	0.08	0.17	0.33	0.08
From binary classifier 43 to binary classifier 63																					
0.25	0	0.33	0	0.25	0.08	0.08	0	0.17	0	0	0.08	0.17	0.17	0.08	0.08	0.17	0.17	0.25	0.08	0.08

Table 25: Error probabilities of neural network binary classifiers when using 10% $P\,(error)$ cut-off (every 10th band).

From binary classifier 1 to binary classifier 21																					
0	0.08	0	0.08	0	0.25	0	0.08	0	0	0.08	0	0	0	0	0	0.08	0.17	0.08	0.25	0.17
From binary classifier 22 to binary classifier 42																					
0.08	0	0.08	0.08	0.17	0.08	0.08	0	0.08	0	0	0.08	0	0	0	0.25	0.08	0.08	0.17	0.33	0.08
From binary classifier 43 to binary classifier 63																					
0.25	0	0.33	0	0.25	0.08	0.08	0	0.17	0	0	0.08	0.17	0.17	0.08	0.08	0.17	0.17	0.25	0.08	0.08

Table 26: Error probabilities of neural networks binary classifiers when using 15% $P\,(error)$ cut-off (every 10th band).

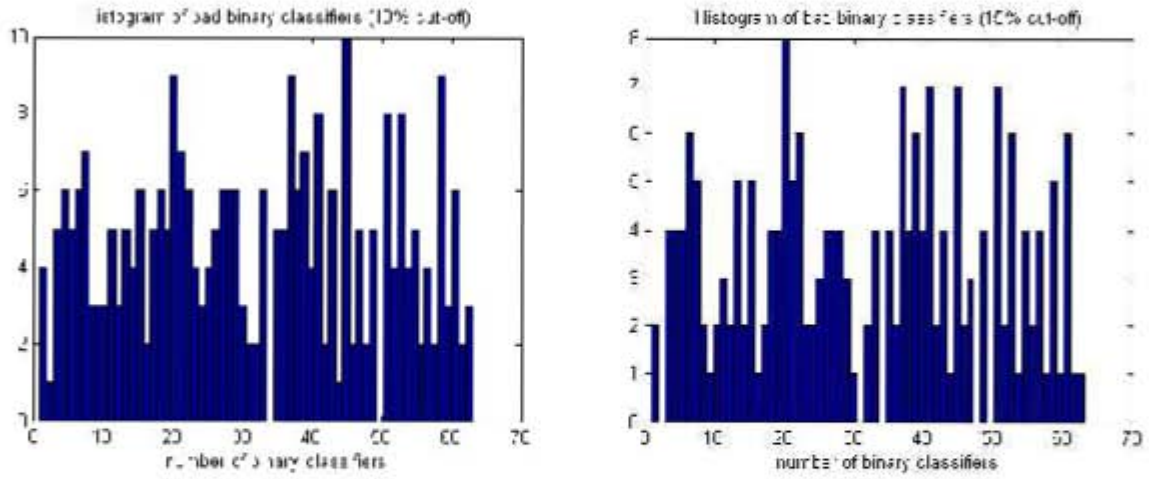

Figure 6.1: Histograms of bad neural networks classifiers when using 10% and 15% as cut-off values for the 10th experiment of 10-fold cross validation (every 10th band).

7 Discussion and Conclusions

The hyperspectral data of seven different tree species in Kruger National Park was used to construct 7-class predictors. The hyperspectral data showed large within-class variability and small between-class variability, hence mean based methods (e.g. Gaussian classifier) do not seem to be successful. We therefore used model-free approaches such as the neural networks classifier and the G-nearest neighbour classifier to classify the hyperspectral data of the seven different tree species in Kruger National Park.

This study has shown that the 7-class classifiers (i.e. 1-nearest neighbour classifier and neural networks classifier) performed poor, shown by the averages of the error probability estimates in Table 8 which are approximately 33% for 1-nerest neighbour classifier and 26% for neural networks classifier when using the chosen bands.

The way out was to decompose the 7-class classification problem into a set of binary classification problems. Then for each of these binary classification problems, a binary classifier was constructed. These binary classifiers were then combined using Error Correcting Output Code (ECOC) to form a 7-class predictor. We found that ECOC predictor which was formed by combining 1-nearest neighbour binary classifiers classified exactly in the same way as the 7-class 1-nearest neighbour classifier. The ECOC predictor which was formed by combining

neural networks binary classifiers has shown improvements of about 10% in the misclassification rate (see Table 15).

We noted that some of the binary classifiers produced very high error rates, hence we came up with the idea of removing those before combining the outputs by ECOC. A further improvement of about 10% was observed (see Table 23). These averages of the error probability estimates after we had deleted bad binary classifiers may, however, be overly optimistic. This is due to the fact that the same test set which was used to delete bad binary classifiers was also used to estimate the error probabilities of the ECOC predictors. To overcome this problem of having overly optimistic error probability estimates, we need a data set with more samples. The large number of samples will allow us to have an independent test set which will be used to estimate the error probability of the ECOC predictor after we have deleted bad binary classifiers.

One major limitation of the study is the small sample size (i.e 148 samples). This limitation results in not having a good classifier and a good estimate of its error probability at the same time. Again, with more samples we can possibly "learn" a good classifier and obtain a good estimate of its error probability. Another problem of this study is the small between-class variability combined with a high within-class variability (shown by Figure 5.4). This limitation makes classes to overlap, hence the classification of the hyperspectral data in this study was challenging. Despite all limitations, this study clearly shows that tree species discrimination using hyperspectral data is feasible.

7.1 Future Study

Due to the redundancy of the bands (i.e. features) in tree species hyperspectral data, we suggest feature extraction methods (e.g. Principal Components Analysis [13]) or feature selection methods (e.g. classification trees [3]) to the reduce dimensionality of the hyperspectral data for future study. A systematic way for selecting "useful bands" is e.g. presented in [22] for the discrimination between papyrus vegetation species.

We did not really optimize all parameters for the neural networks, especially not at all for the binary classifiers. Hence we suggest optimization of the parameters of the neural networks

binary classifiers for future study. We hope that this optimization will improve performance even more.

Also a systematic approach to delete "bad binary classifiers" could improve the final classification accuracy. We could use histograms such as in Figure 6.1 to find binary classifiers that should always be excluded. In order to do this properly, however, more data are necessary.

References

[1] Abe S., Support Vector machines for Pattern Classification, Springer-Verlag, London, 2005.

[2] Bishop C.M., Neural Networks for Pattern Recognition, Oxford University Press, Inc., New York, 1995.

[3] Breiman L., Friedman J.H. and Olshen R.A., Classification and Regression Trees, Chapman & Hall/CRC, 1998.

[4] Cochrane M.A., Using vegetation reflectance variability for species level classification of hyperspectral data, International Journal of Remote Sensing, 21 (2000), pp. 2075-2087.

[5] Cybenko G., Approximation by Superpositions of Sigmoidal Functions. Math. of Control, Signal and Systems 2 (1989), pp. 303-314.

[6] Diamond P. and Kloeden P., Metric Spaces of Fuzzy Sets Theory and Applications, World Scientific Publishing Co.Pte.Ltd., 1994.

[7] Dietterich T.G. and Bakiri G., Solving Multiclass Learning Problems via Error-Correcting Output Codes, Journal of Artificial Intelligence Research, 2 (1995), pp. 263-286.

[8] Duda R.O., Hart P.E. and Stork D.G., Pattern Classification, John Wiley and Sons, Inc., Second Edition, 2001.

[9] Felipe I.P., Dohm J.M., Baker V.R., Doggett T., Davies A.G., Castano R., Chien S., Cichy B., Greenley R., Sherwood R., Tran D. and Rabideau G., Flood detection and monitoring with the autonomous sciencecraft experiment onboard EO-1, Remote Sensing of Environment 101 (2006), pp. 463-481.

[10] Ghani R., Using Error-correcting Codes for Text Classification, ICML'00, Proceedings of the Seventeenth International Conference on Machine Learning, Morgan Kaufmann Publishers Inc., San Fransisco, 2000.

[11] Hand D.J., Discrimination and Classification, John Wiley & Sons Ltd, 1981.

[12] Huberty C.J., Applied Discriminant Analysis, John Wiley & Sons, Inc., 1994.

[13] Jolliffe I.T., Principal Components Analysis, Springer-Verlag, New York Inc., 1986.

[14] Kohavi R., A Study of Cross-Validation and Bootstrap for Accuracy Estimation and Model Selection, Proceedings IJCAI-95 (internationsal joint conference on artificial intelligence 1995), Montreal, Quebec, Canada, August 20-25, 1995, pp. 1137 - 1143.

[15] Krebel U.H.G., Pairwise classification and support vector machines, In: B. Schölkopf, C.J.C. Burges and A.J. Smola, Editors, Advances in kernel methods: Support vector learning, MIT Press, Cambridge, MA (1999), pp. 255–268.

[16] Lillesand T.M. and Kiefer R.W., Remote Sensing and Image Interpretation, John Wiley and Sons, Inc., 1979.

[17] Lin S., An Introduction To Error-Correcting Codes, Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1970.

[18] Lorena A.C., Carvalho A.C. and Gama J.M., A review on the combination of binary classifiers in multiclass problems, Artificial Intelligence Review, 30 (2008), pp. 19-37.

[19] Majeke B., Cho M., Debba P., Methieu R. and Ramoelo A., Species discrimination of African Savannah Trees at Leaf Level using Hyperspectarl remote Sensing, 6th EARSeL SIG IS Workshop, Tel Aviv University, Tel Aviv, Israel, March 16-19, 2009, pp. 1.

[20] Miller I and Miller M. John E., Freund's Mathematical Statistics with Applications , 7th Edition, Pearson Prentice Hall, 2004.

[21] Mitchell T.M., Machine Learning, The McGraw-Hill Companies, 1997.

[22] Mutanga O. and Adam E., Spectral discrimination of papyrus vegetation (Cyperus papyrus L.) in swamp wetlands using field spectrometry, ISPRS Journal of Photogrammetry and Remote Sensing 6 (2009), pp. 612-620.

[23] O'Farrell M., Lewis E., Flanagan C., Lyons W. and Jackman N., Comparison of k-NN and neural network methods in the classification of spectral data from an optical fibre-based sensor system used for quality control in the food industry, Sensors and Actuators B 111-112 (2005), pp. 354-362.

[24] Reddy B.D., Functional Analysis and Boundary-Value Problems: An introductory treatment, Longman, London and Wiley, New York, 1995.

[25] Riedmiller M. and Braun H., A Direct Adaptive Method for Faster Backpropagation Learning: The RPROP Algorithm, IEEE INTERNATIONAL CONFERENCE ON NEURAL NETWORKS, (1993), pp. 586-591.

[26] Ripley B.D., Neural Networks and Related Methods for Classification, Journal of the Royal Statistical Society, 56 (1994), pp. 409-456.

[27] Sanderson R., Introduction to Remote Sensing, New Mexico State University, no year, (http://spacegrant.nmsu.edu/statewide/projects/remote_sensing.pdf, (last accessed feb 2011)).

[28] Schmidt K.S and Skidmore A.K., Spectral discrimination of vegetation types in coastal wetland, Remote Sensing of Environment 26 (2003), pp.92-108.

[29] Shippert P., Introduction to Hyperspectral Image Analysis, Ph.D., Online Journal of Space Communication, Remote Sensing of Earth via Satellite 3 (2003), (http://spacejournal.ohio.edu/pdf/shippert.pdf, (last accessed feb 2011)).

Appendix A: Detailed results

The following Tables 28 to 34 show results of the error probability estimates of G-Nearest Neighbour classifiers (using different values of G) and their 95% confidence intervals over ten experiments for all the bands, every 10th band, every 20th band, and every 30th band. Note that we use G-NN in the tables for G-Nearest Neighbour.

G-NN	**Error Probability Estimates on the test set using all the bands**				
Neighbours:	**G=1**	**G=2**	**G=5**	**G=10**	**G=12**
Experiments					
1st	0.2045	0.4091	0.5000	0.5455	0.5682
2nd	0.2955	0.3636	0.5682	0.5455	0.5682
3rd	0.4545	0.5000	0.4773	0.5227	0.7045
4th	0.3409	0.4091	0.5909	0.5227	0.6591
5th	0.2727	0.3182	0.5000	0.5455	0.5909
6th	0.3864	0.4773	0.4318	0.6136	0.5909
7th	0.2727	0.3409	0.5000	0.5909	0.6364
8th	0.3182	0.3636	0.5000	0.5000	0.5909
9th	0.2500	0.3636	0.2955	0.7045	0.6136
10th	0.3864	0.4091	0.4545	0.4773	0.7045
average	**0.3182**	**0.3955**	**0.4818**	**0.5568**	**0.6227**
95% Confidence Intervals					
Experiments					
1st	(0.1034; 0.3057)	(0.2858; 0.5324)	(0.3746; 0.6254)	(0.4206; 0.6704)	(0.4439; 0.6924)
2nd	(0.1810; 0.4099)	(0.2430; 0.4843)	(0.4439; 0.6924)	(0.4206; 0.6704)	(0.4439; 0.6924)
3rd	(0.3296; 0.5794)	(0.3746; 0.6254)	(0.3520; 0.6026)	(0.3974; 0.6480)	(0.5901; 0.8190)
4th	(0.2220; 0.4598)	(0.2858; 0.5324)	(0.4676; 0.7142)	(0.3974;)0.6480	(0.5402; 0.7780)
5th	(0.1610; 0.3844)	(0.2013; 0.4350)	(0.3746; 0.6254)	(0.4206; 0.6704)	(0.4676; 0.7142)
6th	(0.2642; 0.5085)	(0.3520; 0.6026)	(0.3076; 0.5561)	(0.4915; 0.7358)	(0.4676; 0.7142)
7th	(0.1610; 0.3844)	(0.2220; 0.4598)	(0.3746; 0.6254)	(0.4676; 0.7142)	(0.5157; 0.7570)
8th	(0.2013; 0.4350)	(0.2430; 0.4843)	(0.3746; 0.6254)	(0.3746; 0.6254)	(0.4676; 0.7142)
9th	(0.1414; 0.3586)	(0.2430; 0.4843)	(0.1810; 0.4099)	(0.5901; 0.8190)	(0.4915; 0.7358)
10th	(0.2642; 0.5085)	(0.2858; 0.5324)	(0.3296; 0.5794)	(0.3520; 0.6026)	(0.5901; 0.8190)

Table 28: Error probability estimates for G-Nearest Neighbours (using different values of G) and their 95% confidence intervals (all bands).

G-NN	**Error Probability Estimates on the test set using every 10th band**				
Neighbours:	**G=1**	**G=2**	**G=5**	**G=10**	**G=12**
Experiments					
1st	0.2273	0.4318	0.5227	0.5682	0.5682
2nd	0.2955	0.4318	0.5455	0.5455	0.5682
3rd	0.4545	0.5227	0.4773	0.5227	0.7045
4th	0.3409	0.5000	0.5682	0.5227	0.6591
5th	0.2955	0.4091	0.5000	0.5455	0.5909
6th	0.3864	0.5000	0.4318	0.6136	0.5909
7th	0.2727	0.3864	0.5227	0.5909	0.6364
8th	0.3182	0.4545	0.5000	0.5000	0.5909
9th	0.2500	0.5455	0.2955	0.7045	0.6136
10th	0.3864	0.5227	0.4545	0.4773	0.7045
average	**0.3227**	**0.4705**	**0.4818**	**0.5591**	**0.6227**
95% Confidence Intervals					
Experiments					
1st	(0.1222; 0.3324)	(0.3076; 0.5561)	(0.3974; 0.6480)	(0.4439; 0.6924)	(0.4439; 0.6924)
2nd	(0.1810; 0.4099)	(0.3076; 0.5561)	(0.4206; 0.6704)	(0.4206; 0.6704)	(0.4439; 0.6924)
3rd	(0.3296; 0.5794)	(0.3974; 0.6480)	(0.3520; 0.6026)	(0.3974; 0.6480)	(0.5901; 0.8190)
4th	(0.2220; 0.4598)	(0.3746; 0.6254)	(0.4439; 0.6924)	(0.3974; 0.6480)	(0.5402; 0.7780)
5th	(0.1810; 0.4099)	(0.2858; 0.5324)	(0.3746; 0.6254)	(0.4206; 0.6704)	(0.4676; 0.7142)
6th	(0.2642; 0.5085)	(0.3746; 0.6254)	(0.3076; 0.5561)	(0.4915; 0.7358)	(0.4676; 0.7142)
7th	(0.1610; 0.3844)	(0.2642; 0.5085)	(0.3974; 0.6480)	(0.4676; 0.7142)	(0.5157; 0.7570)
8th	(0.2013; 0.2013)	(0.3296; 0.5794)	(0.3746; 0.6254)	(0.3746; 0.6254)	(0.4676; 0.7142)
9th	(0.1414; 0.1414)	(0.4206; 0.6704)	(0.1810; 0.4099)	(0.5901; 0.8190)	(0.4915; 0.7358)
10th	(0.2642; 0.2642)	(0.3974; 0.6480)	(0.3296; 0.5794)	(0.3520; 0.6026)	(0.5901; 0.8190)

Table 30: Error probability estimates for G-Nearest Neighbours (using different values of G) and their 95% confidence intervals (every 10th band).

G-NN	**Error Probability Estimates on the test set using every 20th band**				
Neighbours:	**G=1**	**G=2**	**G=5**	**G=10**	**G=12**
Experiments					
1st	0.2045	0.4091	0.5227	0.5682	0.5455
2nd	0.2955	0.3636	0.5682	0.5455	0.5455
3rd	0.4545	0.5000	0.4773	0.5227	0.7045
4th	0.3409	0.4091	0.5909	0.5227	0.6591
5th	0.2955	0.3182	0.5000	0.5455	0.5909
6th	0.3864	0.5000	0.4318	0.6136	0.5909
7th	0.2727	0.3409	0.5000	0.5909	0.6136
8th	0.2955	0.3636	0.5000	0.5000	0.5909
9th	0.2500	0.3636	0.2955	0.6818	0.6136
10th	0.3636	0.4091	0.4773	0.4773	0.7045
average	**0.3159**	**0.3977**	**0.4864**	**0.5568**	**0.6159**
95% Confidence Intervals					
Experiments					
1st	(0.1034; 0.3057)	(0.2858; 0.5324)	(0.3974; 0.6480)	(0.4439; 0.6924)	(0.4206; 0.6704)
2nd	(0.1810; 0.4099)	(0.2430; 0.4843)	(0.4439; 0.6924)	(0.4206; 0.6704)	(0.4206; 0.6704)
3rd	(0.3296; 0.5794)	(0.3746; 0.6254)	(0.3520; 0.6026)	(0.3974; 0.6480)	(0.5901; 0.8190)
4th	(0.2220; 0.4598)	(0.2858; 0.5324)	(0.4676; 0.7142)	(0.3974; 0.6480)	(0.5402; 0.7780)
5th	(0.1810; 0.4099)	(0.2013; 0.4350)	(0.3746; 0.6254)	(0.4206; 0.6704)	(0.4676; 0.7142)
6th	(0.2642; 0.5085)	(0.3746; 0.6254)	(0.3076; 0.5561)	(0.4915; 0.7358)	(0.4676; 0.7142)
7th	(0.1610; 0.3844)	(0.2220; 0.4598)	(0.3746; 0.6254)	(0.4676; 0.7142)	(0.4915; 0.7358)
8th	(0.1810; 0.4099)	(0.2430; 0.4843)	(0.3746; 0.6254)	(0.3746; 0.6254)	(0.4676; 0.7142)
9th	(0.1414; 0.3586)	(0.2430; 0.4843)	(0.1810; 0.4099)	(0.5650; 0.7987)	(0.4915; 0.7358)
10th	(0.2430 ; 0.4843)	(0.2858; 0.2858)	(0.3520; 0.6026)	(0.3520; 0.6026)	(0.5901; 0.8190)

Table 32: Error probability estimates for G-Nearest Neighbours (using different values of G) and their 95% confidence intervals (every 20th band).

G-NN	**Error Probability Estimates on the test set using every 30th band**				
Neighbours:	**G=1**	**G=2**	**G=5**	**G=10**	**G=12**
Experiments					
1st	0.4091	0.3864	0.6818	0.5455	0.5000
2nd	0.2955	0.4091	0.4318	0.6364	0.7045
3rd	0.3864	0.5000	0.5000	0.6591	0.5227
4th	0.3182	0.4091	0.4545	0.5909	0.5682
5th	0.3182	0.3182	0.3864	0.6364	0.5909
6th	0.4545	0.4773	0.3864	0.5227	0.5227
7th	0.2500	0.3182	0.4545	0.5909	0.6136
8th	0.2500	0.3636	0.4318	0.6364	0.5682
9th	0.3409	0.3636	0.4318	0.5909	0.6818
10th	0.3636	0.4091	0.5909	0.5682	0.5000
average	**0.3386**	**0.3955**	**0.4750**	**0.5977**	**0.5773**
95% Confidence Intervals					
Experiments					
1st	(0.2858; 0.5324)	(0.2642; 0.5085)	(0.5650; 0.7987)	(0.4206; 0.6704)	(0.3746; 0.6254)
2nd	(0.1810; 0.4099)	(0.2858; 0.5324)	(0.3076; 0.5561)	(0.5157; 0.7570)	(0.5901; 0.8190)
3rd	(0.2642; 0.5085)	(0.3746; 0.6254)	(0.3746; 0.6254)	(0.5402; 0.7780)	(0.3974; 0.6480)
4th	(0.2013; 0.4350)	(0.2858; 0.5324)	(0.3296; 0.5794)	(0.4676; 0.7142)	(0.4439; 0.6924)
5th	(0.2013; 0.4350)	(0.2013; 0.4350)	(0.2642; 0.5085)	(0.5157; 0.7570)	(0.4676; 0.7142)
6th	(0.3296; 0.5794)	(0.3520; 0.6026)	(0.2642; 0.5085)	(0.3974; 0.6480)	(0.3974; 0.6480)
7th	(0.1414; 0.3586)	(0.2013; 0.4350)	(0.3296; 0.5794)	(0.4676; 0.7142)	(0.4915; 0.7358)
8th	(0.1414; 0.3586)	(0.2430; 0.4843)	(0.3076; 0.5561)	(0.5157; 0.7570)	(0.4439; 0.6924)
9th	(0.2220; 0.4598)	(0.2430; 0.4843)	(0.3076; 0.5561)	(0.4676; 0.7142)	(0.5650; 0.7987)
10th	(0.2430; 0.4843)	(0.2858; 0.5324)	(0.4676; 0.7142)	(0.4439; 0.6924)	(0.3746; 0.6254)

Table 34: Error probability estimates for G-Nearest Neighbours (using different values of G) and their 95% confidence intervals (every 30th band).

The following Tables 36 to 42 show results of the error probability estimates of Neural Networks classifiers (with 1 hidden layer using different hidden neurons) and their 95% confidence intervals over ten experiments for all the bands, every 10th band, every 20th band, and every 30th band. Note that we use NeuralNets for Neural Networks in the tables.

NeuralNets	**Error Probability Estimates on the test set using all the bands**				
Neurons:	**5**	**10**	**12**	**15**	**20**
Experiments					
1st	0.3636	0.1818	0.3409	0.1818	0.3636
2nd	0.3636	0.2955	0.1818	0.3636	0.3182
3rd	0.3409	0.2045	0.2955	0.4545	0.3636
4th	0.4545	0.1591	0.2273	0.2727	0.2500
5th	0.4545	0.1818	0.3409	0.3182	0.3182
6th	0.4091	0.3409	0.3409	0.2273	0.2045
7th	0.5227	0.2955	0.3636	0.2955	0.4318
8th	0.3182	0.2500	0.3636	0.2500	0.2955
9th	0.4091	0.2273	0.2500	0.3636	0.2727
10th	0.3864	0.2955	0.2045	0.2500	0.2955
average	**0.4023**	**0.2432**	**0.2909**	**0.2977**	**0.3114**
95% Confidence Intervals					
Experiments					
1st	(0.2430**;** 0.4843)	(0.0851**;** 0.2786)	(0.2220**;** 0.4598)	(0.0851**;** 0.2786)	(0.2430; 0.4843)
2nd	(0.2430**;** 0.4843)	(0.1810**;** 0.4099)	(0.0851**;** 0.2786)	(0.2430**;** 0.4843)	(0.2013; 0.4350)
3rd	(0.2220**;** 0.4598)	(0.1034**;** 0.3057)	(0.1810**;** 0.4099)	(0.3296**;** 0.5794)	(0.2430; 0.4843)
4th	(0.3296**;** 0.5794)	(0.0673**;** 0.2508)	(0.1222**;** 0.3324)	(0.1610**;** 0.3844)	(0.1414; 0.3586)
5th	(0.3296**;** 0.5794)	(0.0851**;** 0.2786)	(0.2220**;** 0.4598)	(0.2013**;** 0.4350)	(0.2013; 0.4350)
6th	(0.2858**;** 0.5324)	(0.2220**;** 0.4598)	(0.2220**;** 0.4598)	(0.1222**;** 0.3324)	(0.1034; 0.3057)
7th	(0.3974**;** 0.6480)	(0.1810**;** 0.4099)	(0.2430**;** 0.4843)	(0.1810**;** 0.4099)	(0.3076; 0.5561)
8th	(0.2013**;** 0.4350)	(0.1414**;** 0.3586)	(0.2430**;** 0.4843)	(0.1414**;** 0.3586)	(0.1810; 0.4099)
9th	(0.2858**;**0.5324)	(0.1222**;** 0.3324)	(0.1414**;** 0.3586)	(0.2430**;** 0.4843)	(0.1610; 0.3844)
10th	(0.2642**;** 0.5085)	(0.1810**;** 0.4099)	(0.1034**;** 0.3057)	(0.1414**;** 0.3586)	(0.1810; 0.4099)

Table 36: Error probability estimates for Neural Networks (with 1 hidden layer using different hidden neurons) and their 95% confidence intervals (all bands).

NeuralNets	**Error Probability Estimates on the test set using every 10th band**				
Neurons:	**5**	**10**	**12**	**15**	**20**
Experiments					
1st	0.2955	0.2500	0.3409	0.2045	0.2045
2nd	0.4091	0.2955	0.2045	0.3182	0.2500
3rd	0.2045	0.2727	0.1591	0.2273	0.2045
4th	0.3182	0.2045	0.2955	0.0909	0.2273
5th	0.3864	0.2727	0.1818	0.2273	0.2273
6th	0.2273	0.2500	0.2045	0.2727	0.3636
7th	0.3864	0.2955	0.2045	0.2045	0.1818
8th	0.4091	0.4773	0.1591	0.2727	0.1136
9th	0.4091	0.3636	0.2955	0.3409	0.1136
10th	0.3182	0.2727	0.2045	0.3182	0.2955
average	**0.3364**	**0.2955**	**0.2250**	**0.2477**	**0.2182**
95% Confidence Intervals					
Experiments					
1st	(0.1810; 0.4099)	(0.1414; 0.3586)	(0.2220; 0.4598)	(0.1034; 0.3057)	(0.1034; 0.3057)
2nd	(0.2858; 0.5324)	(0.1810; 0.4099)	(0.1034; 0.3057)	(0.2013; 0.4350)	(0.1414; 0.3586)
3rd	(0.1034; 0.3057)	(0.1610; 0.3844)	(0.0673; 0.2508)	(0.1222; 0.3324)	(0.1034; 0.3057)
4th	(0.2013; 0.4350)	(0.1034; 0.3057)	(0.1810; 0.4099)	(0.0188; 0.1630)	(0.1222; 0.3324)
5th	(0.2642; 0.5085)	(0.1610; 0.3844)	(0.0851; 0.2786)	(0.1222; 0.3324)	(0.1222; 0.3324)
6th	(0.1222; 0.3324)	(0.1414; 0.3586)	(0.1034; 0.3057)	(0.1610; 0.3844)	(0.2430; 0.4843)
7th	(0.2642; 0.5085)	(0.1810; 0.4099)	(0.1034; 0.3057)	(0.1034; 0.3057)	(0.0851; 0.2786)
8th	(0.2858; 0.5324)	(0.3520; 0.6026)	(0.0673; 0.2508)	(0.1610; 0.3844)	(0.0340; 0.1932)
9th	(0.2858; 0.5324)	(0.2430; 0.4843)	(0.1810; 0.4099)	(0.2220; 0.4598)	(0.0340; 0.1932)
10th	(0.2013; 0.4350)	(0.1610; 0.3844)	(0.1034; 0.3057)	(0.2013; 0.4350)	(0.1810; 0.4099)

Table 38: Error probability estimates for Neural Networks (with 1 hidden layer using different hidden neurons) and their 95% confidence intervals (every 10th).

NeuralNets	**Error Probability Estimates on the test set using every 10th band**				
Neurons:	**5**	**10**	**12**	**15**	**20**
Experiments					
1st	0.4318	0.2045	0.3409	0.2273	0.2727
2nd	0.5455	0.3182	0.1136	0.2500	0.2273
3rd	0.4773	0.2500	0.2955	0.2273	0.1364
4th	0.4318	0.1591	0.1591	0.2500	0.1818
5th	0.4091	0.1818	0.2500	0.1364	0.2500
6th	0.2273	0.3182	0.2500	0.2500	0.2500
7th	0.4318	0.1818	0.2955	0.2045	0.1591
8th	0.4318	0.1136	0.3182	0.2727	0.3182
9th	0.3409	0.3864	0.1591	0.2500	0.2045
10th	0.4545	0.2045	0.1364	0.1364	0.2955
average	**0.4182**	**0.2318**	**0.2318**	**0.2205**	**0.2295**
95% Confidence Intervals					
Experiments					
1st	(0.3076; 0.5561)	(0.1034; 0.3057)	(0.2220; 0.4598)	(0.1222; 0.3324)	(0.1610; 0.3844)
2nd	(0.4206; 0.6704)	(0.2013; 0.4350)	(0.0340; 0.1932)	(0.1414; 0.3586)	(0.1222; 0.3324)
3rd	(0.3520; 0.6026)	(0.1414; 0.3586)	(0.1810; 0.4099)	(0.1222; 0.3324)	(0.0503; 0.2224)
4th	(0.3076; 0.5561)	(0.0673; 0.2508)	(0.0673; 0.2508)	(0.1414; 0.3586)	(0.0851; 0.2786)
5th	(0.2858; 0.5324)	(0.0851; 0.2786)	(0.1414; 0.3586)	(0.0503; 0.2224)	(0.1414; 0.3586)
6th	(0.1222; 0.3324)	(0.2013; 0.4350)	(0.1414; 0.3586)	(0.1414; 0.3586)	(0.1414; 0.3586)
7th	(0.3076; 0.5561)	(0.0851; 0.2786)	(0.1810; 0.4099)	(0.1034; 0.3057)	(0.0673; 0.2508)
8th	(0.3076; 0.5561)	(0.0340; 0.1932)	(0.2013; 0.4350)	(0.1610; 0.3844)	(0.2013; 0.4350)
9th	(0.2220; 0.4598)	(0.2642; 0.5085)	(0.0673; 0.2508)	(0.1414; 0.3586)	(0.1034; 0.3057)
10th	(0.3296; 0.5794)	(0.1034; 0.3057)	(0.0503; 0.2224)	(0.0503; 0.2224)	(0.1810; 0.4099)

Table 40: Error probability estimates for Neural Networks (with 1 hidden layer using different hidden neurons) and their 95% confidence intervals (every 20th).

NeuralNets	**Error Probability Estimates on the test set using every 10th band**				
Neurons:	**5**	**10**	**12**	**15**	**20**
Experiments					
1st	0.4773	0.2955	0.3636	0.2955	0.2500
2nd	0.3636	0.3182	0.2727	0.2500	0.2727
3rd	0.3636	0.1364	0.2273	0.1818	0.1364
4th	0.3864	0.2045	0.2045	0.2273	0.2500
5th	0.4091	0.3409	0.2727	0.2727	0.2045
6th	0.4545	0.3636	0.2500	0.2727	0.1591
7th	0.4318	0.2727	0.2727	0.2955	0.2500
8th	0.4318	0.3409	0.4318	0.2500	0.2955
9th	0.4318	0.2727	0.2727	0.3636	0.3409
10th	0.4318	0.3182	0.2273	0.2500	0.2500
average	**0.4182**	**0.2864**	**0.2795**	**0.2659**	**0.2409**
95% Confidence Intervals					
Experiments					
1st	(0.3520; 0.6026)	(0.1810; 0.4099)	(0.2430; 0.4843)	(0.1810; 0.4099)	(0.1414; 0.3586)
2nd	(0.2430; 0.4843)	(0.2013; 0.4350)	(0.1610; 0.3844)	(0.1414; 0.3586)	(0.1610; 0.3844)
3rd	(0.2430; 0.4843)	(0.0503; 0.2224)	(0.1222; 0.3324)	(0.0851; 0.2786)	(0.0503; 0.2224)
4th	(0.2642; 0.5085)	(0.1034; 0.3057)	(0.1034; 0.3057)	(0.1222; 0.3324)	(0.1414; 0.3586)
5th	(0.2858; 0.5324)	(0.2220; 0.4598)	(0.1610; 0.3844)	(0.1610; 0.3844)	(0.1034; 0.3057)
6th	(0.3296; 0.5794)	(0.2430; 0.4843)	(0.1414; 0.3586)	(0.1610; 0.3844)	(0.0673; 0.2508)
7th	(0.3076; 0.5561)	(0.1610; 0.3844)	(0.1610; 0.3844)	(0.1810; 0.4099)	(0.1414; 0.3586)
8th	(0.3076; 0.5561)	(0.2220; 0.4598)	(0.3076; 0.5561)	(0.1414; 0.3586)	(0.1810; 0.4099)
9th	(0.3076; 0.5561)	(0.1610; 0.3844)	(0.1610; 0.3844)	(0.2430; 0.4843)	(0.2220; 0.4598)
10th	(0.3076; 0.5561)	(0.2013; 0.4350)	(0.1222; 0.3324)	(0.1414; 0.3586)	(0.1414; 0.3586)

Table 42: Error probability estimates for Neural Networks (with 1 hidden layer using different hidden neurons) and their 95% confidence intervals (every 30th).

The following Tables to show results of the error probability estimates of the G-Nearest Neighbour classifiers and Neural Networks classifiers with "optimal" parameters and their 95% confidence intervals. These tables show the error probabilities of the 7-class classifiers, ECOC combiners and ECOC combiners with deleted binary classifiers (i.e. we deleted binary classifiers that have error rates that exceeds 10% and 15%) .

10-fold cross validation 1-Nearest Neighbour	**Error Probability Estimates on the test set using all the bands**			
		ECOC		
	seven-class	**Exhaustive**	**Non.Exhaust (15%)**	**Non.Exhaust (10%)**
Experiments				
1st	0.3493	0.3493	0.3489	0.3542
2nd	0.3240	0.3240	0.3286	0.3652
3rd	0.3407	0.3407	0.3215	0.3415
4th	0.3546	0.3546	0.3350	0.3475
5th	0.3378	0.3378	0.3504	0.4064
6th	0.3375	0.3375	0.3304	0.3304
7th	0.3335	0.3335	0.3254	0.3689
8th	0.3198	0.3198	0.3261	0.3419
9th	0.3231	0.3231	0.3164	0.3663
10th	0.3357	0.3357	0.3472	0.3860
average	**0.3356**	**0.3356**	**0.3330**	**0.3608**
95% Confidence Intervals				
Experiments				
1st	(0.2845;0.4140)	(0.2845;0.4140)	(0.2841; 0.4136)	(0.2892; 0.4193)
2nd	(0.2605; 0.3875)	(0.2605; 0.3875)	(0.2648; 0.3925)	(0.2996; 0.4307)
3rd	(0.2762; 0.4051)	(0.2762; 0.4051)	(0.2580; 0.3850)	(0.2770; 0.4059)
4th	(0.2896; 0.4197)	(0.2896; 0.4197)	(0.2708; 0.3991)	(0.2827; 0.4122)
5th	(0.2734; 0.4023)	(0.2734; 0.4023)	(0.2853; 0.4154)	(0.3394; 0.4733)
6th	(0.2730; 0.4019)	(0.2730; 0.4019)	(0.2663; 0.3946)	(0.2663; 0.3946)
7th	(0.2694; 0.3977)	(0.2694; 0.3977)	(0.2616; 0.3893)	(0.3031; 0.4347)
8th	(0.2563; 0.3833)	(0.2563; 0.3833)	(0.2622; 0.3899)	(0.2771; 0.4066)
9th	(0.2596; 0.3866)	(0.2596; 0.3866)	(0.2533; 0.3796)	(0.3007; 0.4318)
10th	(0.2712; 0.4001)	(0.2712; 0.4001)	(0.2821; 0.4122)	(0.3194; 0.4526)

Table 43: Error probability estimates for 1-Nearest neighbour and their 95% confidence intervals (all bands).

10-fold cross validation 1-Nearest Neighbour	**Error Probability Estimates on the test set using every 10th band**			
		ECOC		
	seven-class	**Exhaustive**	**Non.Exhaust (15%)**	**Non.Exhaust (10%)**
Experiments				
1st	0.3564	0.3564	0.3560	0.3614**0.0574**
2nd	0.3299	0.3299	0.3345	0.3711
3rd	0.3484	0.3484	0.3292	0.3491
4th	0.3546	0.3546	0.3350	0.3475
5th	0.3437	0.3437	0.3504	0.4064
6th	0.3427	0.3427	0.3357	0.3357
7th	0.3402	0.3402	0.3321	0.3755
8th	0.3270	0.3270	0.3332	0.3490
9th	0.3308	0.3308	0.3241	0.3740
10th	0.3357	0.3357	0.3472	0.3860
average	**0.3409**	**0.3409**	**0.3377**	**0.3656**
95% Confidence Intervals				
Experiments				
1st	(0.2914; 0.4215)	(0.2914; 0.4215)	(0.2910; 0.4211)	(0.2961; 0.4267)
2nd	(0.2660; 0.3937)	(0.2660; 0.3937)	(0.2703; 0.3987)	(0.3053; 0.4369)
3rd	(0.2836; 0.4131)	(0.2836; 0.4131)	(0.2653; 0.3930)	(0.2844; 0.4139)
4th	(0.2896; 0.4197)	(0.2896; 0.4197)	(0.2708; 0.3991)	(0.2827; 0.4122)
5th	(0.2789; 0.4085)	(0.2789; 0.4085)	(0.2853; 0.4154)	(0.3394; 0.4733)
6th	(0.2780; 0.4075)	(0.2780; 0.4075)	(0.2712; 0.4002)	(0.2712; 0.4002)
7th	(0.2757; 0.4047)	(0.2757; 0.4047)	(0.2679; 0.3963)	(0.3095; 0.4416)
8th	(0.2631; 0.3908)	(0.2631; 0.3908)	(0.2691; 0.3974)	(0.2840; 0.4141)
9th	(0.2670; 0.3947)	(0.2670; 0.3947)	(0.2606; 0.3876)	(0.3082; 0.4398)
10th	(0.2712; 0.4001)	(0.2712; 0.4001)	(0.2821; 0.4122)	(0.3194; 0.4526)

Table 44: Error probability estimates for 1-Nearest neighbour and their 95% confidence intervals (every 10th band).

10-fold cross validation 1-Nearest Neighbour	**Error Probability Estimates on the test set using every 20th band**			
		ECOC		
	seven-class	**Exhaustive**	**Non.Exhaust (15%)**	**Non.Exhaust (10%)**
Experiments				
1st	0.3452	0.3452	0.3448	0.3501
2nd	0.3246	0.3246	0.3292	0.3658
3rd	0.3421	0.3421	0.3292	0.3491
4th	0.3480	0.3480	0.3283	0.3408
5th	0.3378	0.3378	0.3504	0.4064
6th	0.3356	0.3356	0.3285	0.3285
7th	0.3340	0.3340	0.3259	0.3630
8th	0.3193	0.3193	0.3255	0.3413
9th	0.3241	0.3241	0.3175	0.3673
10th	0.3294	0.3294	0.3409	0.3860
average	**0.3340**	**0.3340**	**0.3320**	**0.3598**
95% Confidence Intervals				
Experiments				
1st	(0.2807; 0.4096)	(0.2807; 0.4096)	(0.2803; 0.4092)	(0.2853; 0.4149)
2nd	(0.2611; 0.3881)	(0.2611; 0.3881)	(0.2654; 0.3931)	(0.3002; 0.4314)
3rd	(0.2776; 0.4066)	(0.2776; 0.4066)	(0.2653; 0.3930)	(0.2844; 0.4139)
4th	(0.2832; 0.4127)	(0.2832; 0.4127)	(0.2644; 0.3921)	(0.2763; 0.4053)
5th	(0.2734; 0.4023)	(0.2734; 0.4023)	(0.2853; 0.4154)	(0.3394; 0.4733)
6th	(0.2711; 0.4001)	(0.2711; 0.4001)	(0.2644; 0.3927)	(0.2644; 0.3927)
7th	(0.2698; 0.3981)	(0.2698; 0.3981)	(0.2620; 0.3897)	(0.2975; 0.4286)
8th	(0.2558; 0.3828)	(0.2558; 0.3828)	(0.2617; 0.3894)	(0.2765; 0.4061)
9th	(0.2606; 0.3876)	(0.2606; 0.3876)	(0.2543; 0.3806)	(0.3018; 0.4329)
10th	(0.2652; 0.3936)	(0.2652; 0.3936)	(0.2761; 0.4057)	(0.3194; 0.4526)

Table 45: Error probability estimates for 1-Nearest neighbour and their 95% confidence intervals (every 20th band).

10-fold cross validation 1-Nearest Neighbour	**Error Probability Estimates on the test set using every 30th band**			
		ECOC		
	seven-class	**Exhaustive**	**Non.Exhaust (15%)**	**Non.Exhaust (10%)**
Experiments				
1st	0.3514	0.3514	0.3510	0.3564
2nd	0.3246	0.3246	0.3292	0.3658
3rd	0.3421	0.3421	0.3292	0.3491
4th	0.3480	0.3480	0.3283	0.3408
5th	0.3378	0.3378	0.3504	0.4064
6th	0.3356	0.3356	0.3285	0.3285
7th	0.3402	0.3402	0.3321	0.3818
8th	0.3193	0.3193	0.3255	0.3413
9th	0.3241	0.3241	0.3175	0.3673
10th	0.3294	0.3294	0.3409	0.3860
average	**0.3353**	**0.3353**	**0.3333**	**0.3623**
95% Confidence Intervals				
Experiments				
1st	(0.2867; 0.4162)	(0.2867; 0.4162)	(0.2862; 0.4158)	(0.2913; 0.4214)
2nd	(0.2611; 0.3881)	(0.2611; 0.3881)	(0.2654; 0.3931)	(0.3002; 0.4314)
3rd	(0.2776; 0.4066)	(0.2776; 0.4066)	(0.2653; 0.3930)	(0.2844; 0.4139)
4th	(0.2832; 0.4127)	(0.2832; 0.4127)	(0.2644; 0.3921)	(0.2763; 0.4053)
5th	(0.2734; 0.4023)	(0.2734; 0.4023)	(0.2853; 0.4154)	(0.3394; 0.4733)
6th	(0.2711; 0.4001)	(0.2711; 0.4001)	(0.2644; 0.3927)	(0.2644; 0.3927)
7th	(0.2757; 0.4047)	(0.2757; 0.4047)	(0.2679; 0.3963)	(0.3156; 0.4480)
8th	(0.2558; 0.3828)	(0.2558; 0.3828)	(0.2617; 0.3894)	(0.2765; 0.4061)
9th	(0.2606; 0.3876)	(0.2606; 0.3876)	(0.2543; 0.3806)	(0.3018; 0.4329)
10th	(0.2652; 0.3936)	(0.2652; 0.3936)	(0.2761; 0.4057)	(0.3194; 0.4526)

Table 46: Error probability estimates for 1-Nearest neighbour and their 95% confidence intervals (every 30th band).

10-fold cross validation Neural Networks	**Error Probability Estimates on the test set using all the bands**			
		ECOC		
	seven-class	**Exhaustive**	**Non.Exhaust (15%)**	**Non.Exhaust (10%)**
Experiments				
1st	0.2681	0.1602	0.0677	0.0347
2nd	0.2615	0.1528	0.1023	0.0763
3rd	0.2809	0.1646	0.0858	0.0628
4th	0.2948	0.1467	0.0735	0.0476
5th	0.2974	0.1558	0.0821	0.0641
6th	0.2727	0.1395	0.0944	0.0869
7th	0.3049	0.1433	0.0896	0.0577
8th	0.2722	0.1343	0.0530	0.0514
9th	0.2600	0.1438	0.0555	0.0530
10th	0.2599	0.1388	0.0606	0.0453
average	**0.2772**	**0.1480**	**0.0765**	**0.0580**
95% Confidence Intervals				
Experiments				
1st	(0.2078; 0.3283)	(0.1102; 0.2102)	(0.0321; 0.1033)	(0.0079; 0.0614)
2nd	(0.2017; 0.3213)	(0.1036; 0.2019)	(0.0601; 0.1444)	(0.0393; 0.1134)
3rd	(0.2201; 0.3416)	(0.1146; 0.2146)	(0.0474; 0.1242)	(0.0304; 0.0952)
4th	(0.2328; 0.3568)	(0.0985; 0.1950)	(0.0379; 0.1090)	(0.0188; 0.0764)
5th	(0.2350; 0.3598)	(0.1066; 0.2049)	(0.0451; 0.1192)	(0.0317; 0.0965)
6th	(0.2120; 0.3334)	(0.0922; 0.1869)	(0.0547; 0.1341)	(0.0485; 0.1253)
7th	(0.2421; 0.3677)	(0.0950; 0.1915)	(0.0499; 0.1293)	(0.0252; 0.0901)
8th	(0.2115; 0.3329)	(0.0879; 0.1807)	(0.0223; 0.0837)	(0.0207; 0.0821)
9th	(0.1998; 0.3203)	(0.0955; 0.1921)	(0.0231; 0.0880)	(0.0206; 0.0854)
10th	(0.2002; 0.3197)	(0.0924; 0.1851)	(0.0281; 0.0930)	(0.0165; 0.0741)

Table 47: Error probability estimates for Neural Network and their 95% confidence intervals (all bands).

10-fold cross validation Neural Networks	**Error Probability Estimates on the test set using every 10th band**			
		ECOC		
	seven-class	**Exhaustive**	**Non.Exhaust (15%)**	**Non.Exhaust (10%)**
Experiments				
1st	0.2343	0.1574	0.0662	0.0482
2nd	0.2120	0.1528	0.0778	0.0399
3rd	0.2898	0.1796	0.0849	0.0610
4th	0.2492	0.1551	0.0800	0.0493
5th	0.2483	0.1649	0.0683	0.0483
6th	0.2419	0.1668	0.0992	0.0656
7th	0.2382	0.1354	0.0678	0.0415
8th	0.2322	0.1385	0.0700	0.0517
9th	0.2529	0.1350	0.0795	0.0478
10th	0.2022	0.1757	0.1080	0.0411
average	**0.2401**	**0.1561**	**0.0802**	**0.0494**
95% Confidence Intervals				
Experiments				
1st	(0.1772; 0.2913)	(0.1082; 0.2065)	(0.0322; 0.1003)	(0.0194; 0.0770)
2nd	(0.1556; 0.2685)	(0.1036; 0.2019)	(0.0408; 0.1148)	(0.0131; 0.0666)
3rd	(0.2282; 0.3514)	(0.1272; 0.2319)	(0.0465; 0.1233)	(0.0286; 0.0934)
4th	(0.1905; 0.3079)	(0.1060; 0.2043)	(0.0429; 0.1170)	(0.0205; 0.0781)
5th	(0.1896; 0.3071)	(0.1149; 0.2149)	(0.0342; 0.1024)	(0.0195; 0.0771)
6th	(0.1842; 0.2995)	(0.1168; 0.2168)	(0.0594; 0.1389)	(0.0332; 0.0980)
7th	(0.1800; 0.2964)	(0.0891; 0.1818)	(0.0338; 0.1019)	(0.0147; 0.0682)
8th	(0.1746; 0.2899)	(0.0912; 0.1859)	(0.0344; 0.1056)	(0.0210; 0.0824)
9th	(0.1941; 0.3116)	(0.0886; 0.1813)	(0.0425; 0.1165)	(0.0190; 0.0766)
10th	(0.1476; 0.2567)	(0.1241; 0.2274)	(0.0658; 0.1501)	(0.0143; 0.0678)

Table 48: Error probability estimates for Neural Network and their 95% confidence intervals (every 10th band).

10-fold cross validation Neural Networks	**Error Probability Estimates on the test set using every 20th band**			
		ECOC		
	seven-class	**Exhaustive**	**Non.Exhaust (15%)**	**Non.Exhaust (10%)**
Experiments				
1st	0.3102	0.1392	0.0712	0.0578
2nd	0.2422	0.1321	0.0878	0.0614
3rd	0.2535	0.1281	0.0504	0.0700
4th	0.2354	0.1355	0.0868	0.0668
5th	0.2870	0.1477	0.0798	0.0535
6th	0.2848	0.1657	0.0678	0.0543
7th	0.2597	0.1284	0.0644	0.0388
8th	0.2642	0.1626	0.0627	0.0693
9th	0.2473	0.1536	0.0864	0.0526
10th	0.1980	0.1218	0.0633	0.0499
average	**0.2582**	**0.1415**	**0.0721**	**0.0574**
95% Confidence Intervals				
Experiments				
1st	(0.2478; 0.3726)	(0.0929; 0.1856)	(0.0372; 0.1053)	(0.0271; 0.0885)
2nd	(0.1840; 0.3004)	(0.0867; 0.1775)	(0.0494; 0.1262)	(0.0290; 0.0939)
3rd	(0.1947; 0.3122)	(0.0828; 0.1735)	(0.0198; 0.0811)	(0.0330; 0.1071)
4th	(0.1777; 0.2930)	(0.0891; 0.1819)	(0.0484; 0.1252)	(0.0327; 0.1008)
5th	(0.2254; 0.3486)	(0.0994; 0.1959)	(0.0427; 0.1168)	(0.0228; 0.0842)
6th	(0.2237; 0.3460)	(0.1157; 0.2157)	(0.0337; 0.1018)	(0.0236; 0.0850)
7th	(0.2004; 0.3190)	(0.0830; 0.1738)	(0.0303; 0.0984)	(0.0120; 0.0656)
8th	(0.2050; 0.3235)	(0.1126; 0.2126)	(0.0302; 0.0951)	(0.0353; 0.1034)
9th	(0.1885; 0.3060)	(0.1045; 0.2028)	(0.0480; 0.1248)	(0.0219; 0.0833)
10th	(0.1442; 0.2519)	(0.0786; 0.1651)	(0.0309; 0.0957)	(0.0211; 0.0787)

Table 49: Error probability estimates for Neural Network and their 95% confidence intervals (every 20th band).

10-fold cross validation Neural Networks	**Error Probability Estimates on the test set using every 30th band**			
		ECOC		
	seven-class	**Exhaustive**	**Non.Exhaust (15%)**	**Non.Exhaust (10%)**
Experiments				
1st	0.2528	0.1260	0.0719	0.0577
2nd	0.2527	0.1708	0.0746	0.0590
3rd	0.2435	0.1395	0.0804	0.0585
4th	0.2937	0.1397	0.0766	0.0574
5th	0.2493	0.1770	0.0707	0.0505
6th	0.2475	0.1751	0.1001	0.0423
7th	0.2532	0.1406	0.0791	0.0536
8th	0.2652	0.1415	0.0756	0.0688
9th	0.2335	0.1432	0.0647	0.0564
10th	0.2364	0.1388	0.0612	0.0411
average	**0.2528**	**0.1492**	**0.0755**	**0.0545**
95% Confidence Intervals				
Experiments				
1st	(0.1935; 0.3121)	(0.0807; 0.1714)	(0.0364; 0.1075)	(0.0252; 0.0901)
2nd	(0.1935; 0.3120)	(0.1200; 0.2216)	(0.0390; 0.1101)	(0.0266; 0.0914)
3rd	(0.1853; 0.3017)	(0.0922; 0.1869)	(0.0434; 0.1175)	(0.0278; 0.0891)
4th	(0.2317; 0.3557)	(0.0923; 0.1870)	(0.0396; 0.1137)	(0.0250; 0.0898)
5th	(0.1911; 0.3075)	(0.1262; 0.2278)	(0.0367; 0.1048)	(0.0218; 0.0793)
6th	(0.1887; 0.3062)	(0.1235; 0.2268)	(0.0591; 0.1410)	(0.0155; 0.0690)
7th	(0.1945; 0.3120)	(0.0932; 0.1879)	(0.0420; 0.1161)	(0.0229; 0.0843)
8th	(0.2050; 0.3255)	(0.0932; 0.1897)	(0.0385; 0.1126)	(0.0332; 0.1044)
9th	(0.1758; 0.2911)	(0.0950; 0.1915)	(0.0306; 0.0987)	(0.0240; 0.0889)
10th	(0.1793; 0.2935)	(0.0925; 0.1852)	(0.0288; 0.0936)	(0.0144; 0.0679)

Table 50: Error probability estimates for Neural Network and their 95% confidence intervals (every 30th band).

Appendix B: **Plots of data**

The following plots are the reflectance curves of the samples of the species. They show high within species variability, and if we superimpose them, they all overlap.

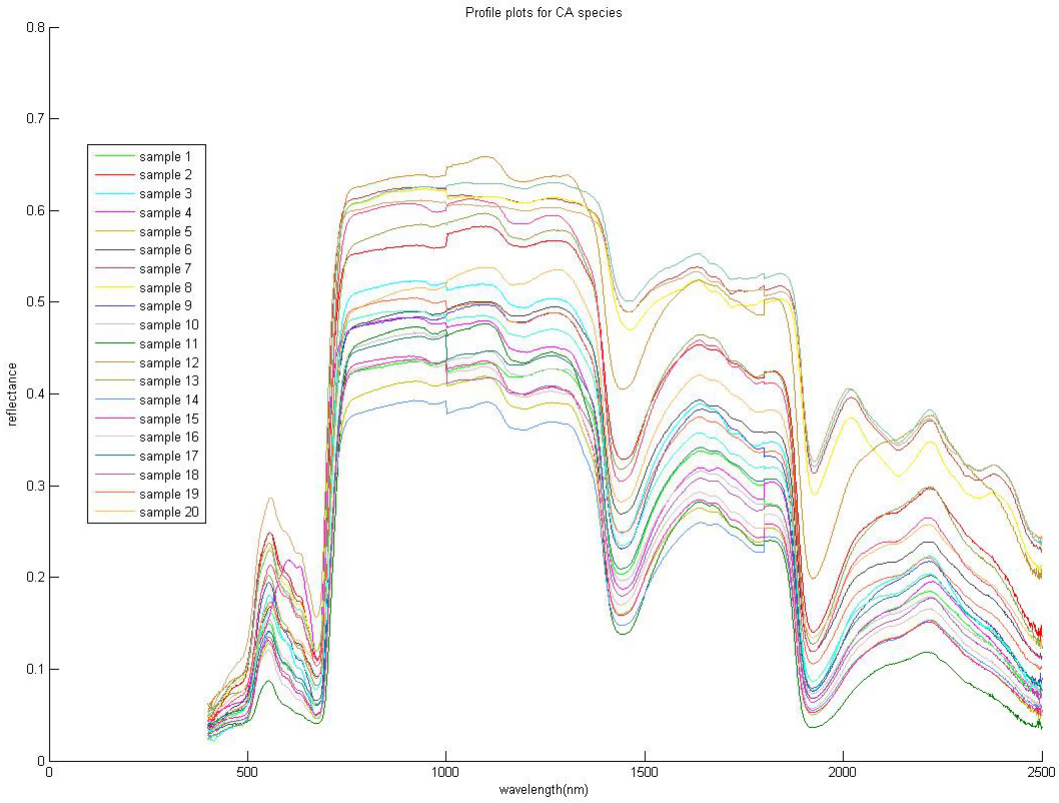

Figure B.1: 20 samples reflectance curves of Combretum Apiculatum

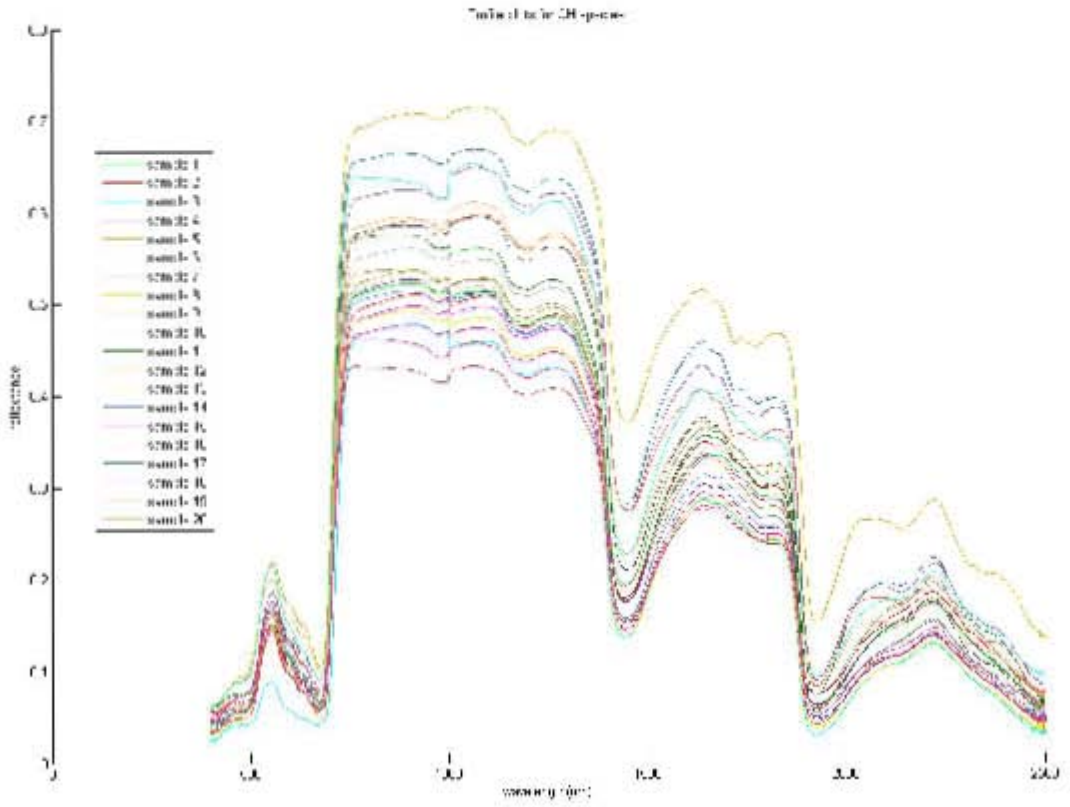

Figure B.2: 20 samples reflectance curves of Combretum Heroense

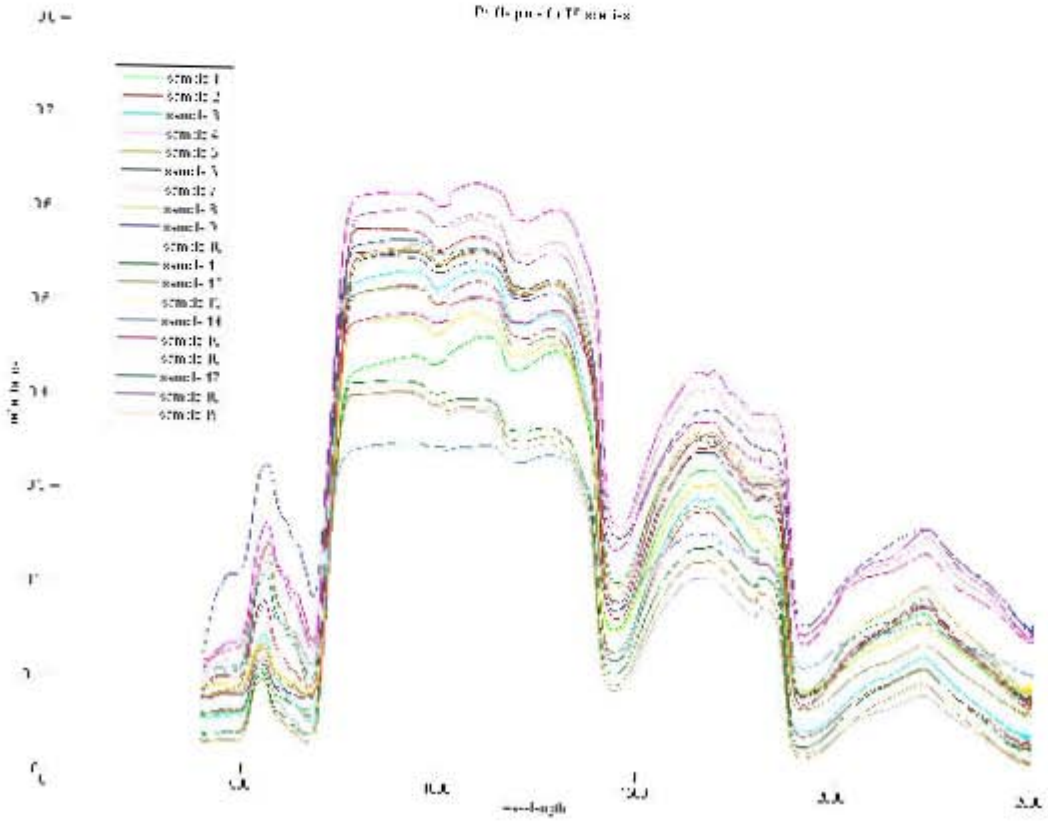

Figure B.3: Reflectance curves of Terminalia Sericia.

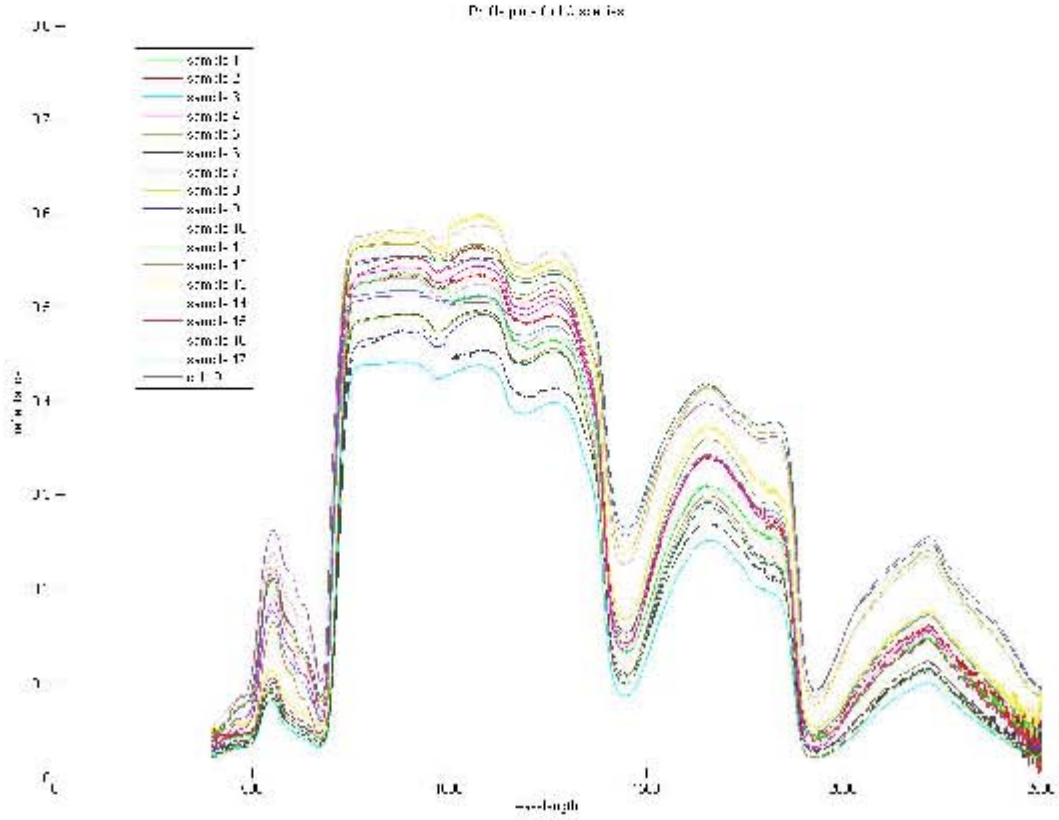

Figure B.4: Refelctance curves of Lonchocarpus Capassa.

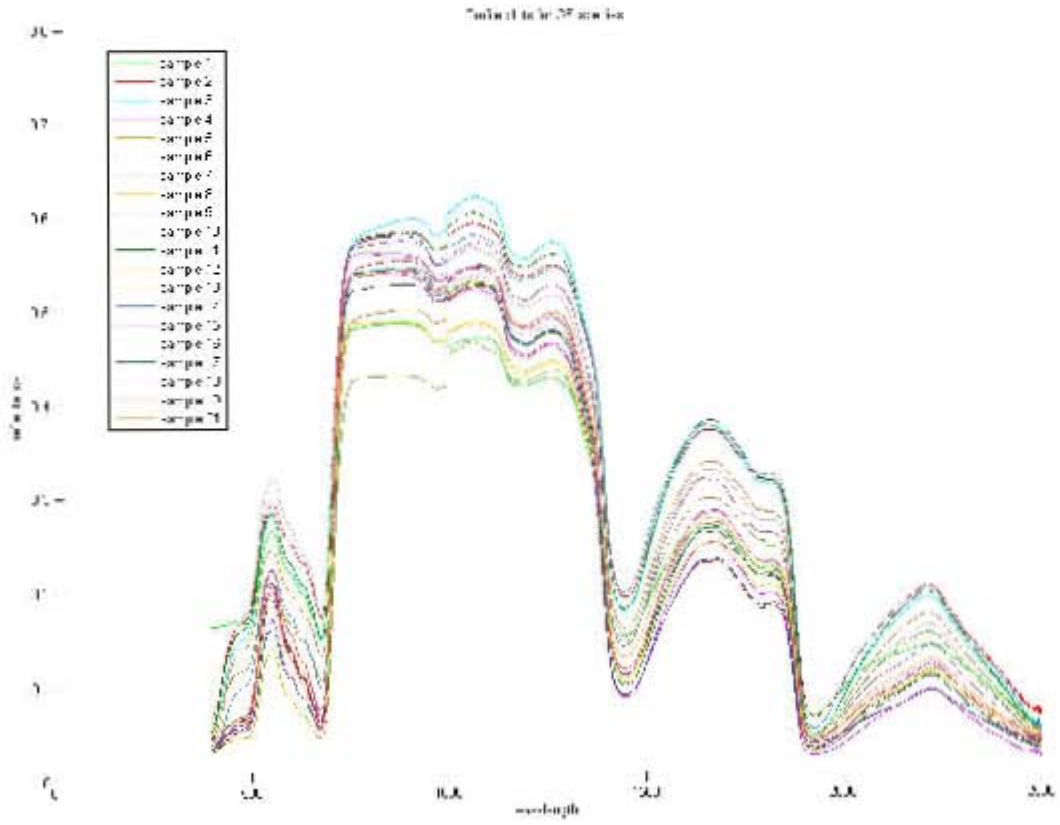

Figure B.5: Reflectance curves of Gymnospora Senegalensis.

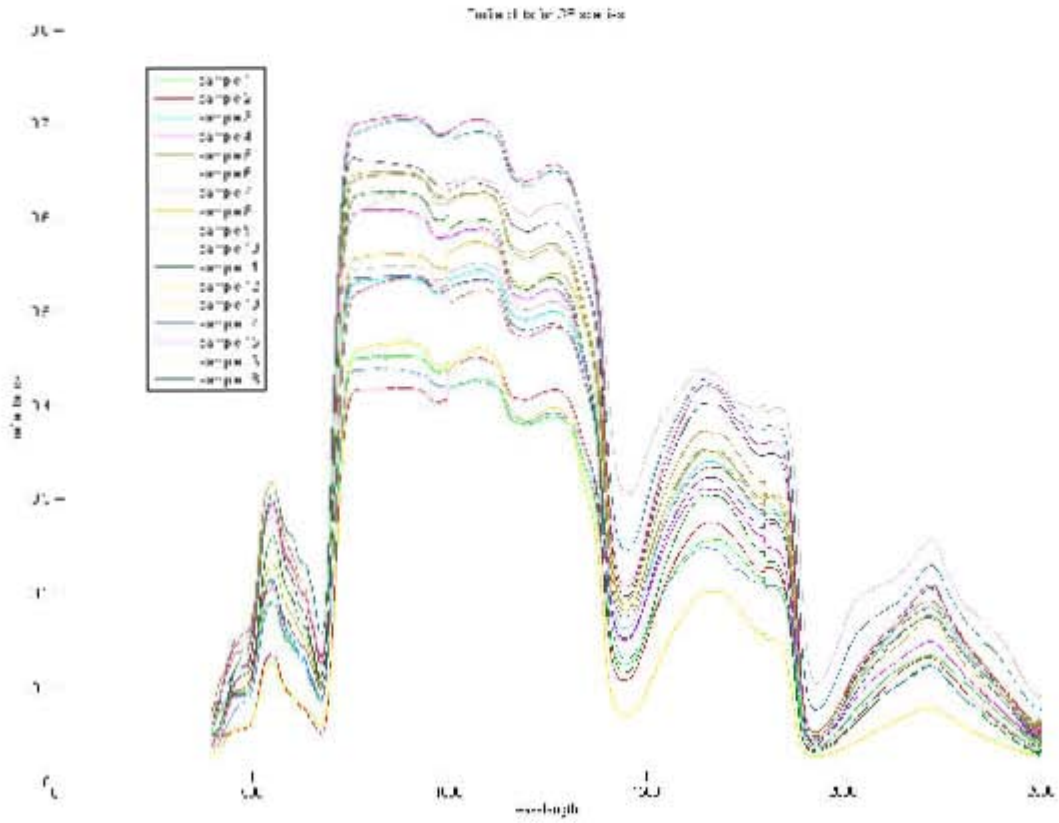

Figure B.6: Reflectance curves of Gymnospora Buxifolia.

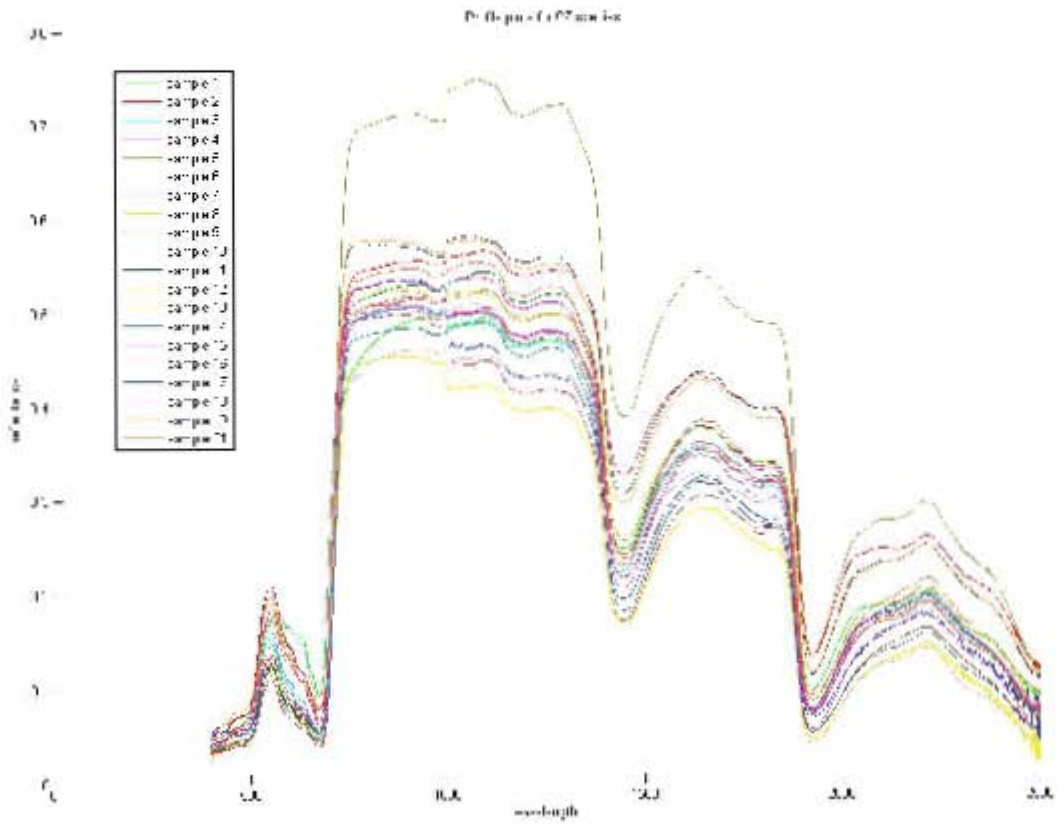

Figure B.7: Reflectance curves of Combretum Zeyherrea.

Appendix C: MATLAB programs

The following program classifies the hyperspectral data using test set approach to determine optimal number of nearest neighbours. Note that we use, %, for the comments on the programs.

```
% classify hyperspectral data using test set approach to determine optimal
% number of nearest neighbours.
clear all
close all
diary L:\Thesis_results\diary_test_nearest_exp2.txt
disp('************************************************************')
disp('************************************************************')
nearest_numbers=[1,2,5,10,12];
for neighbours=1:5
  for runs=1:10
       level = 0.95; % 1-alpha confidence level
       z_norm = norminv(1-(1-level)/2);
       disp('************************************************')
       disp(['run number :', num2str(runs)])
       text = ['experiment starts:  ',num2str(fix(clock))];
       %disp(text)
       %disp('')
       filename = 'L:\Thesis data_set\data.xls'; % change
       %this to your file
       train_prop = 0.7; % proportion of training data
       neighbour_number = nearest_numbers(neighbours); % number of nearest
       % neighbours
       % index of features we look at (frequencies considered)
       feature_index = 1:10:2101; % if we every 10th feature
       % feature_index = 1:20:2101; % if we every 20th feature
       %feature_index = 1:30:2101; % if we every 30th feature
       %feature_index = 1:2101; % if we use all features
```

```
        number_of_features = length(feature_index);
        % make training and test data (7 classes)
        %disp('make training and test data...  ');
        range = {'B2:Z2102','B2:X2102','B2:U2102','B2:T2102','B2:V2102',...
        'B2:S2102','B2:W2102'};
        data_all = [];
        targets_all = []; % targets contain class labels 1,2,3,....,7
%_________________________________________________________________
% read all data
        for class = 1:7
             data_all_new = xlsread(filename,class,char(range(class)));
             data_all = [data_all,data_all_new];;
             targets_all = [targets_all,class*ones(1,...
             length(data_all_new(1,:)))]; % targets contain class labels 1,2
             d(class)=length(data_all_new(1,:));
        end
%_________________________________________________________________
% split into training and test set
        number_of_data = length(data_all(1,:));
        rand_index = randperm(number_of_data);
        train_index = rand_index(1:round(train_prop*number_of_data));
        test_index = setdiff(rand_index,train_index);
        data_train = data_all(:,train_index);
        targets_train_all = targets_all(train_index);
        data_test = data_all(:,test_index);
        targets_test_all = targets_all(test_index);
%_________________________________________________________________
%disp('seven class prediction')
        %disp('classify training data...')
        class_train7 = k_nn_classifier(data_train(feature_index,:)',...
        data_train(feature_index,:)',targets_train_all',neighbour_number);
        check = find(class_train7 ~= targets_train_all);
        error_prob_train7(runs) = length(check)/length(targets_train_all);
```

```
        loss_vector_train7 = (class_train7 ~= targets_train_all); %for SE
        SE_train7(runs) =...
        sqrt(var(loss_vector_train7)/length(targets_train_all)); % SE
        %disp('classify test data...')
        class_test7 = k_nn_classifier(data_test(feature_index,:)',...
        data_train(feature_index,:)',targets_train_all',neighbour_number);
        check7 = find(class_test7 ~= targets_test_all);
        number_misclassifs7 = length(check7);
        number_test_samples = length(targets_test_all);
        error_prob_test7(runs) = length(check7)/length(targets_test_all);
        loss_vector_test7 = (class_test7 ~= targets_test_all); %for SE
        SE_test7(runs) = sqrt(var(loss_vector_test7)/number_test_samples);%SE
  end
disp('*****************************************************************')
disp(['Number of neighbours:',num2str(nearest_numbers(neighbours))])
disp('Error probability estimates when using seven class classifier')
error_prob_train7
error_prob_test7
disp('means of error probabilities for 7-class classifier')
average_error_prob_train7=mean(error_prob_train7)
average_error_prob_test7=mean(error_prob_test7)
disp(' Standard error for seven class classifier')
SE_train7
SE_test7
disp('Confidence intervals of 7-class classifiers')
conf_int_train7_left=(error_prob_train7-z_norm*SE_train7)
conf_int_train7_right=(error_prob_train7+z_norm*SE_train7)
conf_int_test7_left=(error_prob_test7-z_norm*SE_test7)
conf_int_test7_right=(error_prob_test7+z_norm*SE_test7)
end %different neighbours
diary off
```

The following program classifies the hyperspectral data using test set approach to determine the optimal number of hidden neurons. Here we showed the code that uses only one hidden layer. The code with two hidden layers is similar but uses two hidden layers for the net7 variable.

```
% classify hyperspectral data using test set approach to determine optimal
% number of hidden neurons.
% classify hyperspectral data using a
% neural network classifier with orthogonal coding.
clear all
close all
diary L:\Thesis_results\diary_test_set_neuro_exp.txt
disp('************************************************************')
disp('************************************************************')
hidden_neurons=[5,10,12,15,20];
for neurons=1:5
   for runs=1:10
       level = 0.95; % 1-alpha confidence level
       z_norm = norminv(1-(1-level)/2);
       disp('******************************************************')
       disp(['run number :', num2str(runs)])
       text = ['experiment starts:  ',num2str(fix(clock))];
       %disp(text)
       %disp('')
       filename = 'L:\Thesis data_set\data.xls'; % change this
       % to your file
       train_prop = 0.7; % proportion of training data
       number_of_hidden_neurons = hidden_neurons(neurons); % number
       % of hidden
       % neurons
       % index of features we look at (frequencies considered)
       feature_index = 1:10:2101; % if we every 10th feature
```

```
        %feature_index = 1:20:2101; % if we every 20th feature
        %feature_index = 1:30:2101; % if we every 30th feature
        %feature_index = 1:2101; % if we use all features
        number_of_features = length(feature_index);
%_____________________________________________________________________
        % make training and test data (7 classes)
        %disp('make training and test data...  ')
        range = {'B2:Z2102','B2:X2102','B2:U2102','B2:T2102','B2:V2102',...
        'B2:S2102','B2:W2102'};
        data_all = [];
        targets_all = []; % targets contain class labels 1,2,3,....,7
%_____________________________________________________________________
% read all data
        for class = 1:7
             data_all_new = xlsread(filename,class,char(range(class)));
             data_all = [data_all,data_all_new];;
             targets_all = [targets_all,...
             class*ones(1,length(data_all_new(1,:)))]; % targets
             %contain class
             %labels 1,2
        end
%_____________________________________________________________________
%orthogonal coding of targets
        targets_all_orth = zeros(7,length(targets_all));
        for class=1:7
             index_class = find(targets_all == class);
             targets_all_orth(class,index_class) = 1;
        end
%_____________________________________________________________________
% split into training and test set
        number_of_data = length(data_all(1,:));
        rand_index = randperm(number_of_data);
        test_index = setdiff(rand_index,train_index);
```

```
        data_train = data_all(:,train_index);
        targets_train_all = targets_all_orth(:,train_index);
        data_test = data_all(:,test_index);
        targets_test_all = targets_all_orth(:,test_index);
%_____________________________________________________________________
%disp('seven class prediction')
        net7 = newff(minmax(data_train(feature_index,:)),....
        [number_of_hidden_neurons,number_of_hidden_neurons,7],...
        {'logsig','purelin'},'trainrp');
        net7.trainParam.epochs = 1000;
        net7.trainParam.goal = 0.03;
        net7.trainParam.show = nan; % supress display
        net7 = train(net7,data_train(feature_index,:),targets_train_all);
        %disp('classify training data...')
        class_train = sim(net7,data_train(feature_index,:));
        [m,class_train] = max(class_train);
        [m,targets_train_all1] = max(targets_train_all);
        loss_vector_train7 = (class_train ~= targets_train_all1); %for SE
        check = find(class_train ~= targets_train_all1);
        error_prob_train7(runs) =...
        length(check)/length(targets_train_all1); % error prob
        SE_train7(runs) = ...
        sqrt(var(loss_vector_train7)/length(targets_train_all1)); % SE
        %disp('classify test data...')
        class_test = sim(net7,data_test(feature_index,:));
        [m,class_test] = max(class_test);
        [m,targets_test_all1] = max(targets_test_all);
        loss_vector_test7 = (class_test ~= targets_test_all1); %for SE
        check = find(class_test ~= targets_test_all1);
        number_misclassifs = length(check);
        number_test_samples = length(targets_test_all1);
        error_prob_test7(runs) = ...
        length(check)/length(targets_test_all1); % error prob
```

```
        SE_test7(runs) = ....
        sqrt(var(loss_vector_train7)/length(targets_train_all1)); % SE
  end
disp('****************************************************************')
disp(['Number of hidden neurons:', num2str(hidden_neurons(neurons))])
disp('Error probability estimates when using seven class classifier')
error_prob_train7
error_prob_test7
disp('means of error probabilities for 7-class classifier')
average_error_prob_train7=mean(error_prob_train7)
average_error_prob_test7=mean(error_prob_test7)
disp(' Standard error for seven class classifier')
SE_train7
SE_test7
disp('Confidence intervals of 7-class classifiers')
conf_int_train7_left=(error_prob_train7-z_norm*SE_train7)
conf_int_train7_right=(error_prob_train7+z_norm*SE_train7)
conf_int_test7_left=(error_prob_test7-z_norm*SE_test7)
conf_int_test7_right=(error_prob_test7+z_norm*SE_test7)
end %different hidden neurons
diary off % close diary file
```

The following program classifies hyperspectral data using G-nearest neighbour classifier with optimal number of nearest neighbours. We use K-fold cross validation.

```
% classify hyperspectral data using G-nearest neighbour classifier with optimal
% nearest neighbour number.
% of nearest neighbours.
% classify hyperspectral data using a
% nearest neighbour classifier
% classifies one meta class against another meta class
% use ECOC for combiner
% use cross validation for error prob estimate
clear all
close all
%diary L:\Thesis data_set\diary_cross_val_exp.txt
diary L:\Thesis_results\diary_cross_val_exp_nearest.txt
disp('************************************************************')
disp('************************************************************')
for runs=1:10
     level = 0.95; % 1-alpha confidence level
     z_norm = norminv(1-(1-level)/2);
     neighbour_number=1;
     disp('************************************************************')
     disp(['run number :'  ,num2str(runs)])
     text = ['experiment starts:  ',num2str(fix(clock))];
     %disp(text);
     %disp('');
     filename = 'L:\Thesis data_set\data.xls'; % change this to your file
     number_of_folds = 10; % K-fold cross-validation, K = number_of_folds
     feature_index = 1:10:2101; % if we every 10th feature
     %feature_index = 1:20:2101; % if we every 20th feature
     %feature_index = 1:30:2101; % if we every 30th feature
     %feature_index = 1:2101; % if we use all features
```

```
    number_of_features = length(feature_index);
%_____________________________________________________________________
% make ecoc matrix
    %disp('make ecoc matrix...');
    totalColumn = 63; % total number of columns
    totalRow = 7; %total number of rows ( each row represents a class)
    ECOC = ones(1,totalColumn); % The first row consists of all ones
    num_of_column = length(ECOC); % length of the code
    % Row 2 to row 7 have alternating runs of zeros and ones
    for row=2:totalRow
        n = 1; % we start at the first column
        rem = mod(num_of_column,2); % remainder when we divide the number
        % of columns by 2
        num_of_column = floor(num_of_column/2); % number of columns when we
        %divide by 2
        % a while loop to assure the we do not exceed the number of columns
        while n < totalColumn
            for i=1:(num_of_column + rem) % a for loop for the zero entries
            % of ECOC matrix code
                if n > totalColumn % if we exceed number of columns we
                    break; % break out of the for loop
                end
                ECOC(row,n) = 0;
                n = n+1;
            end
            for i=(i+1):(num_of_column + rem+num_of_column) % for loop
                                                            %for ones
                if n > totalColumn
                    break;
                end
                ECOC(row,n) = 1;
                n = n+1;
            end
```

```
          end
          num_of_column = num_of_column +rem;
     end
     ECOC;
%_____________________________________________________________________
% make training and test data (7 classes)
     %disp('make training and test data...  ');
     range = {'B2:Z2102','B2:X2102','B2:U2102','B2:T2102','B2:V2102',...
     'B2:S2102','B2:W2102'};
%_____________________________________________________________________
% read all data
     for class = 1:7
          data_class = xlsread(filename,class,char(range(class)));
          number_of_data(class) = length(data_class(1,:)); % number of samples
                                                                %per class
                                                        %(needed later for SE)
          randindex = randperm(number_of_data(class)); % permute the data
          data_class = data_class(:,randindex);
          number_per_fold = floor(number_of_data(class)/number_of_folds);
          for k=1:number_of_folds % data{class,fold}
          %contains data of class in fold
          data{class,k} = data_class(:,...
          number_per_fold*(k-1)+1:number_per_fold*k);
     end
          if number_per_fold*number_of_folds < number_of_data(class) % randomly
                                                                %distribute
                                                                 %leftover
                                                        %data into folds
               rest = number_per_fold*number_of_folds+1:number_of_data(class);
               for k=1:length(rest)
                    rand_k = randperm(number_of_folds);
                    data{class,rand_k(k)} = [data{class,rand_k(k)},...
                    data_class(:,rest(k))];
```

```
            end
        end
    end
    for k = 1:number_of_folds
        data_fold{k} = [];
        targets_fold{k} = [];
            for class = 1:7
                data_fold{k} = [data_fold{k},data{class,k}];
                targets_fold{k} = [targets_fold{k},...
                class*ones(1,length(data{class,k}(1,:)))];
        end
    end
    number_of_data; % display for diary file
    total_number_of_data = sum(number_of_data); % should 148
%_____________________________________________________________________
%orthogonal coding of targets
    for k = 1:number_of_folds
        targets_fold_orth{k} = zeros(7,length(targets_fold{k}));
        for class=1:7
            index_class = find(targets_fold{k} == class);
            targets_fold_orth{k}(class,index_class) = 1;
        end
    end
%_____________________________________________________________________
% for each fold:  make classifiers and evaluate...
    loss_vector_train7 = []; % initialize for estimation of standard error
    loss_vector_test7 = [];
    loss_vector_train_ecoc = [];
    loss_vector_test_ecoc = [];
    loss_vector_train_ecoc_new = [];
    loss_vector_test_ecoc_new = [];
    loss_vector_train_ecoc_new_alt = [];
    loss_vector_test_ecoc_new_alt = [];
```

```
loss_vector_trainc = [];
loss_vector_testc = [];
for fold = 1:number_of_folds
    %-------------------------------------------------------------------
    % split into training and test set
    test_fold = fold;
    train_folds = setdiff(1:number_of_folds,test_fold);
    data_train = [];
    targets_train_all_orth = [];
    for i = 1:length(train_folds)
        data_train = [data_train,data_fold{train_folds(i)}];
        targets_train_all_orth = [targets_train_all_orth,...
        targets_fold_orth{train_folds(i)}];
    end
    data_test = data_fold{test_fold};
    targets_test_all_orth = targets_fold_orth{test_fold};
    [m,targets_train_all] = max(targets_train_all_orth);
    [m,targets_test_all] = max(targets_test_all_orth);
    %-------------------------------------------------------------------------
    %disp(['seven class prediction on fold ',num2str(fold)]);
    % disp('classify training data...');
    class_train7 = k_nn_classifier(data_train(feature_index,:)',...
    data_train(feature_index,:)',targets_train_all',neighbour_number);
    loss_vector_train7 = [loss_vector_train7,class_train7 ~= ...
    targets_train_all]; %make vector for estimate of SE
    check = find(class_train7 ~= targets_train_all);
    error_prob_train7(runs,fold) = length(check)/length(targets_train_all);
    %disp('classify test data...');
    class_test7 = k_nn_classifier(data_test(feature_index,:)',...
    data_train(feature_index,:)',targets_train_all',neighbour_number);
    loss_vector_test7 = [loss_vector_test7,class_test7 ~=...
    targets_test_all];
    check7 = find(class_test7 ~= targets_test_all);
```

```
number_misclassifs = length(check7);
number_test_samples = length(targets_test_all);
error_prob_test7(runs,fold) = length(check7)/length(targets_test_all);
%-------------------------------------------------------------------------
%disp(['binary classifiers on fold ',num2str(fold), '...']);
class_train_array = [];
class_test_array = [];
error_prob_train = [];
error_prob_test = [];
for class1 = 1:totalColumn
    %disp(['column ',num2str(class1),' of 63...']);
    class1_index = find(ECOC(:,class1)==1)';
    class2_index = find(ECOC(:,class1)==0)';
    index_train_class = [];
    for class2 = class1_index
        index_train_class_new = find(targets_train_all == class2);
        index_train_class = [index_train_class,index_train_class_new];
    end
    index_train_rest = setdiff(1:length(targets_train_all(1,:)),...
    index_train_class);
    targets_train = zeros(1,length(targets_train_all));
    targets_train(index_train_class) = 1;
    targets_train(index_train_rest) = 2;
    index_test_class = [];
    for class2 = class1_index
    index_test_class_new = find(targets_test_all == class2);
    index_test_class = [index_test_class,index_test_class_new];
end
index_test_rest = setdiff(1:length(targets_test_all(1,:)),...
index_test_class);
targets_test = zeros(1,length(targets_test_all));
targets_test(index_test_class) = 1;
targets_test(index_test_rest) = 2;
```

```
        %set up nearest neighbour classifier
        %disp('classify training data...');
        class_train = k_nn_classifier(data_train(feature_index,:)',...
        data_train(feature_index,:)',targets_train',neighbour_number);
        class_train_ecoc=class_train;
        class_train_ecoc(class_train == 2) = 0;
        class_train_array(class1,:)  = class_train_ecoc;
        check = find(class_train ~= targets_train);
        error_prob_train(runs,class1) = length(check)/length(targets_train);
        %disp('classify test data...');
        class_test = k_nn_classifier(data_test(feature_index,:)',...
        data_train(feature_index,:)',targets_train',neighbour_number);
        class_test_ecoc = class_test;
        class_test_ecoc(find(class_test == 2)) = 0; % create 0's for
        % class 2
        class_test_array(class1,:)  = class_test_ecoc; % store
        %classification
        %results in row
        check = find(class_test ~= targets_test);
        number_misclassifs = length(check);
        number_test_samples = length(targets_test);
        error_prob_test(runs,class1) = length(check)/length(targets_test);
    end
%_____________________________________________________________________
    % use ecoc
    % training set
    %disp(['evaluate ecoc on fold ',num2str(fold),' ...']);
    class_train_ecoc_comb = [];
    class_test_ecoc_comb = [];
    for k = 1:length(class_train_ecoc)
    %determine hamming distance between columns of classification
    %results and ecoc matrix
    vec1 = class_train_array(:,k)'; % column k (as row)
```

```
for i=1:7
     vec2 = ECOC(i,:); % row i of ecoc matrix
     d(i) = sum(mod(vec1+vec2,2)); % hamming distance
end
[m,class_train_ecoc_comb(k)] = min(d); % find smallest
%hamming distance
end
loss_vector_train_ecoc = [loss_vector_train_ecoc,...
class_train_ecoc_comb ~= targets_train_all];
check_ecoc = find(class_train_ecoc_comb ~= targets_train_all);
error_prob_train_ecoc(runs,fold) = ...
length(check_ecoc)/length(targets_train_all);
% test set
for k = 1:length(class_test_ecoc)
     %determine hamming distance between columns of classification
     %results and ecoc matrix
     vec1 = class_test_array(:,k)'; % column k (as row)
          for i=1:7
               vec2 = ECOC(i,:); % row i of ecoc matrix
               d(i) = sum(mod(vec1+vec2,2)); % hamming distance
          end
          [m,class_test_ecoc_comb(k)] = min(d); % find smallest hamming
          %distance
end
loss_vector_test_ecoc = [loss_vector_test_ecoc,...
class_test_ecoc_comb ~= targets_test_all];
check_ecoc = find(class_test_ecoc_comb ~= targets_test_all);
error_prob_test_ecoc(runs,fold) = ...
length(check_ecoc)/length(targets_test_all);
%-------------------------------------------------------------------------
% ecoc without bad classifiers
class_test_ecoc_comb_new = [];
class_test_ecoc_comb_new_alt = [];
```

```
good_index = find(error_prob_test(runs,:)  <=0.15);
ECOC_new = ECOC(:,good_index);
class_test_array_new = class_test_array(good_index,:);
good_index_alt = find(error_prob_test(runs,:)  <=0.10);
ECOC_new_alt = ECOC(:,good_index_alt);
class_test_array_new_alt = class_test_array(good_index_alt,:);
% when using 15% cut
for k = 1:length(class_test_ecoc)
%determine hamming distance between columns of classification results
%and ecoc matrix
     vec1 = class_test_array_new(:,k)'; % column k (as row)
     for i=1:7
          vec2 = ECOC_new(i,:); % row i of ecoc matrix
          d(i) = sum(mod(vec1+vec2,2)); % hamming distance
     end
     [m,class_test_ecoc_comb_new(k)] = min(d); % find smallest
     % hamming distance
end
loss_vector_test_ecoc_new = [loss_vector_test_ecoc_new,...
class_test_ecoc_comb_new ~= targets_test_all];
check_ecoc_new = find(class_test_ecoc_comb_new ~= targets_test_all);
error_prob_test_ecoc_new(runs,fold) = ...
length(check_ecoc_new)/length(targets_test_all);
number_of_classifiers_15percent(fold) = length(good_index);
number_of_classifiers_10percent(fold) = length(good_index_alt);
% when using 10% cut
for k = 1:length(class_test_ecoc)
%determine hamming distance between columns of classification
%results and ecoc matrix
     vec1_alt = class_test_array_new_alt(:,k)'; % column k (as row)
     for i=1:7
          vec2_alt = ECOC_new_alt(i,:); % row i of ecoc matrix
          d_alt(i) = sum(mod(vec1_alt+vec2_alt,2)); % hamming distance
```

```
        end
        [m,class_test_ecoc_comb_new_alt(k)] = min(d_alt); % find smallest
        %hamming distance
    end
    loss_vector_test_ecoc_new_alt = [loss_vector_test_ecoc_new_alt,...
    class_test_ecoc_comb_new_alt ~= targets_test_all];
    check_ecoc_new_alt =...
    find(class_test_ecoc_comb_new_alt ~= targets_test_all);
    error_prob_test_ecoc_new_alt(runs,fold) = ...
    length(check_ecoc_new_alt)/length(targets_test_all);
end % folds
%_____________________________________________________________________
% estimates of error probabilities and standard errors
% 7 class predictor (nearest neighbour)
error_prob_train7_kfold(runs) = mean(error_prob_train7(runs,:));
se_train7 (runs)= sqrt(var(loss_vector_train7)/total_number_of_data);
error_prob_test7_kfold(runs) = mean(error_prob_test7(runs,:));
se_test7(runs) = sqrt(var(loss_vector_test7)/total_number_of_data);
% ecoc combiner
error_prob_train_ecoc_kfold(runs) =   mean(error_prob_train_ecoc(runs,:));
se_train_ecoc(runs) =   sqrt(var(loss_vector_train_ecoc)/total_number_of_data);
error_prob_test_ecoc_kfold(runs) = mean(error_prob_test_ecoc(runs,:));
se_test_ecoc(runs) =   sqrt(var(loss_vector_test_ecoc)/total_number_of_data);
% ecoc combiner without bad classifiers using 15% cut
error_prob_test_ecoc_new_kfold(runs) =...
mean(error_prob_test_ecoc_new(runs,:));
se_test_ecoc_new(runs) = ...
sqrt(var(loss_vector_test_ecoc_new)/total_number_of_data);
% ecoc combiner without bad classifiers using 10% cut
error_prob_test_ecoc_new_kfold_alt(runs) =...
mean(error_prob_test_ecoc_new_alt(runs,:));
se_test_ecoc_new_alt(runs) = ...
sqrt(var(loss_vector_test_ecoc_new_alt)/total_number_of_data);
```

```
  end
  disp('Seven class prediction on each fold using the training set and test set')
  error_prob_train7
  error_prob_test7
  disp('Binary prediction on each fold using the training set and test set')
  error_prob_train
  error_prob_test
  disp('ECOC prediction on each fold using the training set and test set')
  error_prob_train_ecoc
  error_prob_test_ecoc
  disp('ECOC without bad classifiers prediction on each fold using the test set
using 15% cut')
  error_prob_test_ecoc_new
  disp('ECOC without bad classifiers prediction on each fold using the test set
using 10% cut')
  error_prob_test_ecoc_new_alt
  disp('Overall predictions and standard errors for seven class classifiers')
  error_prob_train7_kfold
  se_train7
  error_prob_test7_kfold
  se_test7
  disp('means for test and training sets when using 7-class classifiers')
  average_error_prob_train7_kfold=mean(error_prob_train7_kfold)
  average_error_prob_test7_kfold=mean(error_prob_test7_kfold)
  disp('Confidence intervals for 7-class classifiers')
  conf_int_train7_kfold_left=(error_prob_train7_kfold-z_norm*se_train7)
  conf_int_train7_kfold_right=(error_prob_train7_kfold+z_norm*se_train7)
  conf_int_test7_kfold_left=(error_prob_test7_kfold-z_norm*se_test7)
  conf_int_test7_kfold_right=(error_prob_test7_kfold+z_norm*se_test7)
  disp('Overall predictions and standard errors for ECOC')
  error_prob_train_ecoc_kfold
  se_train_ecoc
  error_prob_test_ecoc_kfold
```

```
  se_test_ecoc
  disp('means for both train and test sets for ECOC')
  average_train_ecoc_kfold=mean(error_prob_train_ecoc_kfold)
  average_test_ecoc_kfold=mean(error_prob_test_ecoc_kfold)
  %confidence intervals
  disp('Confidence intervals for ECOC with bad classifiers')
  conf_int_train_ecoc_kfold_left=...
  (error_prob_train_ecoc_kfold-z_norm*se_train_ecoc)
  conf_int_train_ecoc_kfold_right=...
  (error_prob_train_ecoc_kfold+z_norm*se_train_ecoc)
  conf_int_test_ecoc_kfold_left=...
  (error_prob_test_ecoc_kfold-z_norm*se_test_ecoc)
  conf_int_test_ecoc_kfold_right=...
  (error_prob_test_ecoc_kfold+z_norm*se_test_ecoc)
  disp('Overall predictions and standard errors for ECOC without bad classifiers
on the test set using 15% cut')
  error_prob_test_ecoc_new_kfold
  se_test_ecoc_new
  disp('mean for 15% cut using ECOC')
  average_test_ecoc_new_kfold=mean(error_prob_test_ecoc_new_kfold)
  disp('Confidence intervals for ECOC without bad classifiers')
  conf_int_test_ecoc_new_kfold_left=...
  (error_prob_test_ecoc_new_kfold-z_norm*se_test_ecoc_new)
  conf_int_test_ecoc_new_kfold_right=...
  (error_prob_test_ecoc_new_kfold+z_norm*se_test_ecoc_new)
  disp('Overall predictions and standard errors for ECOC without bad classifiers
on the test set using 10% cut')
  error_prob_test_ecoc_new_kfold_alt
  se_test_ecoc_new_alt
  disp('mean for 10% cut using ECOC')
  average_test_ecoc_new_kfold_alt=mean(error_prob_test_ecoc_new_kfold_alt)
  disp('Confidence intervals for ECOC without bad classifiers')
  conf_int_test_ecoc_new_kfold_alt_left=...
```

```
(error_prob_test_ecoc_new_kfold_alt-z_norm*se_test_ecoc_new_alt)
conf_int_test_ecoc_new_kfold_alt_right=...
(error_prob_test_ecoc_new_kfold_alt+z_norm*se_test_ecoc_new_alt)
diary off % close the diary file
```

The following program classifies the hyperspectral data using Neural Networks with optimal number of hidden neurons an hidden layers. We used 10-fold cross validation.

```
% classify_hyper_data_1 using Neural networks with optimal parameters.
% classify hyperspectral data using a
% neural network classifier with orthogonal coding.
% classifies one meta class against another meta class
% use ECOC for combiner
% use cross validation for error prob estimate
clear all
close all
%diary L:\Thesis data_set\diary_cross_val_exp.txt
diary L:\Thesis_results\diary_cross_val_neuro_exp_new.txt
disp('************************************************************')
disp('************************************************************')
for runs=1:10
hist_values_15per=[];
hist_values_10per=[];
level = 0.95; % 1-alpha confidence level
z_norm = norminv(1-(1-level)/2);
disp('************************************************************')
disp(['run number :', num2str(runs)])
text = ['experiment starts:  ',num2str(fix(clock))];
%disp(text)
%disp('')
filename = 'L:\Thesis data_set\data.xls'; % change this to your file
number_of_folds = 10; % K-fold cross-validation, K = number_of_folds
number_of_hidden_neurons = %10 for all band
%20 for every 10th band
%15 for every 20th band
%20 for every 30th band
feature_index = 1:10:2101; % if we every 10th feature
```

```
%feature_index = 1:20:2101; % if we every 20th feature
%feature_index = 1:30:2101; % if we every 30th feature
%feature_index = 1:2101; % if we use all features
number_of_features = length(feature_index);
%_____________________________________________________________________
% make ecoc matrix
%disp('make ecoc matrix...')
totalColumn = 63; % total number of columns
totalRow = 7; %total number of rows ( each row represents a class)
ECOC = ones(1,totalColumn); % The first row consists of all ones
num_of_column = length(ECOC); % length of the code
% Row 2 to row 7 have alternating runs of zeros and ones
for row=2:totalRow
         n = 1; % we start at the first column
         rem = mod(num_of_column,2); % remainder when we divide the number
                                                          % of columns by 2
         num_of_column = floor(num_of_column/2); % number of columns when we
                                                              %divide by 2
         % a while loop to assure the we do not exceed the number of columns
         while n < totalColumn
              for i=1:(num_of_column + rem) % a for loop for the zero entries
                                                    % of ECOC matrix code
                   if n > totalColumn % if we exceed number of columns we
                        break; % break out of the for loop
                   end
                   ECOC(row,n) = 0;
                   n = n+1;
              end
              for i=(i+1):(num_of_column + rem+num_of_column) % for loop for ones
                   if n > totalColumn
                        break;
                   end
                   ECOC(row,n) = 1;
```

```
                    n = n+1;
                end
            end
            num_of_column = num_of_column +rem;
    end
    ECOC;
    %------------------------------------------------------------------------
    % make training and test data (7 classes)
    %disp('make training and test data...  ')
    range = {'B2:Z2102','B2:X2102','B2:U2102','B2:T2102','B2:V2102',...
    'B2:S2102','B2:W2102'};
    %------------------------------------------------------------------------
    % read all data
    for class = 1:7
            data_class = xlsread(filename,class,char(range(class)));
            number_of_data(class) = length(data_class(1,:)); % number of samples
                                                                %per class
                                                          %(needed later for SE)
            randindex = randperm(number_of_data(class)); % permute the data
            data_class = data_class(:,randindex);
            number_per_fold = floor(number_of_data(class)/number_of_folds);
            for k=1:number_of_folds % data{class,fold} contains data of class in
fold
                data{class,k} = data_class(:,number_per_fold*(k-1)+...
                1:number_per_fold*k);
            end
            if number_per_fold*number_of_folds < number_of_data(class) % randomly
                                                                  %distribute
                                                                    %leftover
                                                                %data into folds
                rest = number_per_fold*number_of_folds+...
                1:number_of_data(class);
                for k=1:length(rest)
```

```
                    rand_k = randperm(number_of_folds);
                    data{class,rand_k(k)} = [data{class,rand_k(k)},...
                    data_class(:,rest(k))];
                end
            end
end
  for k = 1:number_of_folds
    data_fold{k} = [];
    targets_fold{k} = [];
    for class = 1:7
         data_fold{k} = [data_fold{k},data{class,k}];
         targets_fold{k} = [targets_fold{k},...
         class*ones(1,length(data{class,k}(1,:)))];
    end
  end
number_of_data; % display for diary file
total_number_of_data = sum(number_of_data) ; % should 148
%_____________________________________________________________________
%orthogonal coding of targets
  for k = 1:number_of_folds
    targets_fold_orth{k} = zeros(7,length(targets_fold{k}));
    for class=1:7
         index_class = find(targets_fold{k} == class);
         targets_fold_orth{k}(class,index_class) = 1;
    end
  end
%_____________________________________________________________________
% for each fold:  make classifiers and evaluate...
loss_vector_train7 = []; % initialize for estimation of standard error
loss_vector_test7 = [];
loss_vector_train_ecoc = [];
loss_vector_test_ecoc = [];
loss_vector_train_ecoc_new = [];
```

```
loss_vector_test_ecoc_new = [];
loss_vector_train_ecoc_new_alt = [];
loss_vector_test_ecoc_new_alt = [];
loss_vector_trainc = [];
loss_vector_testc = [];
for fold = 1:number_of_folds
%_____________________________________________________________________
% split into training and test set
test_fold = fold;
train_folds = setdiff(1:number_of_folds,test_fold);
data_train = [];
targets_train_all = [];
     for i = 1:length(train_folds)
          data_train = [data_train,data_fold{train_folds(i)}];
          targets_train_all = [targets_train_all,...
          targets_fold_orth{train_folds(i)}];
     end
data_test = data_fold{test_fold};
targets_test_all = targets_fold_orth{test_fold};
%_____________________________________________________________________
%disp(['seven class prediction on fold ',num2str(fold)])
net7 = newff(minmax(data_train(feature_index,:)),...
[number_of_hidden_neurons,7],{'logsig','purelin'},'trainrp');
net7.trainParam.epochs = 1000;
net7.trainParam.goal = 0.03;
net7.trainParam.show = nan; % suppress display
net7 = train(net7,data_train(feature_index,:),targets_train_all);
%disp('classify training data...')
class_train = sim(net7,data_train(feature_index,:));
[m,class_train] = max(class_train);
[m,targets_train_all1] = max(targets_train_all);
loss_vector_train7 = [loss_vector_train7,...
class_train ~= targets_train_all1]; %make vector for estimate of SE
```

```
check = find(class_train ~= targets_train_all1);
error_prob_train7(runs,fold) = length(check)/length(targets_train_all1);
%disp('classify test data...')
class_test = sim(net7,data_test(feature_index,:));
[m,class_test] = max(class_test);
[m,targets_test_all1] = max(targets_test_all);
loss_vector_test7 = [loss_vector_test7,class_test ~= targets_test_all1];
check = find(class_test ~= targets_test_all1);
number_misclassifs = length(check);
number_test_samples = length(targets_test_all1);
error_prob_test7(runs,fold) = length(check)/length(targets_test_all1);
%disp('strike any key to continue')
%pause % wait for user interaction
%_____________________________________________________________________
%disp(['binary classifiers on fold ',num2str(fold), '...'])
class_train_array = [];
class_test_array = [];
error_prob_train = [];
error_prob_test = [];
  for class1 = 1:totalColumn
    %disp(['column ',num2str(class1),' of 63...'])
    class1_index = find(ECOC(:,class1)==1)';
    class2_index = find(ECOC(:,class1)==0)';
    index_train_class1 = [];
    for class2 = class1_index
         index_train_class1_new =...
         (find(targets_train_all(class2,:)  == 1);
         index_train_class1 = ...
         [index_train_class1,index_train_class1_new];
    end
    index_train_rest = setdiff(1:length(targets_train_all(1,:)),...
    index_train_class1);
    targets_train = ...
```

```
zeros(2,length(targets_train_all(1,:))); %orthogonal coding
targets_train(1,index_train_class1) = 1;
targets_train(2,index_train_rest) = 1;
index_test_class1 = [];
for class2 = class1_index
     index_test_class1_new = (find(targets_test_all(class2,:)  == 1);
     index_test_class1 = [index_test_class1,index_test_class1_new];
end
index_test_rest = setdiff(1:length(targets_test_all(1,:)),...
index_test_class1);
targets_test = zeros(2,length(targets_test_all(1,:)));
targets_test(1,index_test_class1) = 1;
targets_test(2,index_test_rest) = 1;
%set up neural network
net = newff(minmax(data_train(feature_index,:)),...
[number_of_hidden_neurons,2],{'logsig','purelin'},'trainrp');
net.trainParam.epochs = 1000;
net.trainParam.goal = 0.03;
net.trainParam.show = nan; % supress display
%train neural net
net = train(net,data_train(feature_index,:),targets_train);
%disp('classify training data...')
class_train = sim(net,data_train(feature_index,:));
[m,class_train] = max(class_train);
class_train_ecoc = class_train;
class_train_ecoc(find(class_train == 2)) = 0; % create 0's
%for class 2
class_train_array(class1,:)  = class_train_ecoc; % store
%classification
%results in row
[m,targets_train1] = max(targets_train);
check = find(class_train ~= targets_train1);
error_prob_train(runs,class1) = ...
```

```
    length(check)/length(targets_train1);
    %disp('classify test data...')
    class_test = sim(net,data_test(feature_index,:));
    [m,class_test] = max(class_test);
    class_test_ecoc = class_test;
    class_test_ecoc(find(class_test == 2)) = 0; % create 0's
    %for class 2
    class_test_array(class1,:)  = class_test_ecoc; % store
                                                          %classification
                                                          %results in row
    [m,targets_test1] = max(targets_test);
    check = find(class_test ~= targets_test1);
    number_misclassifs = length(check);
    number_test_samples = length(targets_test1);
    error_prob_test(runs,class1) = length(check)/length(targets_test1);
  end
%_____________________________________________________________________
% use ecoc
% training set
%disp(['evaluate ecoc on fold ',num2str(fold),' ...'])
class_train_ecoc_comb = [];
class_test_ecoc_comb = [];
for k = 1:length(class_train_ecoc)
%determine hamming distance between columns of classification results
%and ecoc matrix
vec1 = class_train_array(:,k)'; % column k (as row)
  for i=1:7
    vec2 = ECOC(i,:); % row i of ecoc matrix
    d(i) = sum(mod(vec1+vec2,2)); % hamming distance
  end
[m,class_train_ecoc_comb(k)] = min(d); % find smallest hamming
%distance
end
```

```
loss_vector_train_ecoc = [loss_vector_train_ecoc,....
class_train_ecoc_comb ~= targets_train_all1];
check_ecoc = find(class_train_ecoc_comb ~= targets_train_all1);
error_prob_train_ecoc(runs,fold) = ....
length(check_ecoc)/length(targets_train_all1);
% test set
  for k = 1:length(class_test_ecoc)
    %determine hamming distance between columns of classification
    %results and ecoc matrix
    vec1 = class_test_array(:,k)'; % column k (as row)
    for i=1:7
         vec2 = ECOC(i,:); % row i of ecoc matrix
         d(i) = sum(mod(vec1+vec2,2)); % hamming distance
    end
    [m,class_test_ecoc_comb(k)] = min(d); % find smallest hamming
    %distance
  end
loss_vector_test_ecoc = [loss_vector_test_ecoc,...
class_test_ecoc_comb ~= targets_test_all1];
check_ecoc = find(class_test_ecoc_comb ~= targets_test_all1);
error_prob_test_ecoc(runs,fold) =...
length(check_ecoc)/length(targets_test_all1);
%-----------------------------------------------------------------
% ecoc without bad classifiers
class_test_ecoc_comb_new = [];
class_test_ecoc_comb_new_alt = [];
index_all=1:63;
good_index = find(error_prob_test(runs,:)  <=0.15);
bad_index_15per=setdiff(index_all,good_index);
%number of bad classifiers
number_of_bad_f_k_15per(runs,fold)=length(bad_index_15per);
hist_values_15per=[hist_values_15per,bad_index_15per];
ECOC_new = ECOC(:,good_index);
```

```
class_test_array_new = class_test_array(good_index,:);
good_index_alt = find(error_prob_test(runs,:)  <=0.10);
bad_index_alt=setdiff(index_all,good_index_alt );
%number of bad classifiers
number_of_bad_f_k_10per(runs,fold)=length(bad_index_alt);
hist_values_10per=[hist_values_10per,bad_index_alt];
ECOC_new_alt = ECOC(:,good_index_alt);
class_test_array_new_alt = class_test_array(good_index_alt,:);
% when using 15% cut
  for k = 1:length(class_test_ecoc)
    %determine hamming distance between columns of classification
    %results and ecoc matrix
    vec1 = class_test_array_new(:,k)'; % column k (as row)
    for i=1:7
         vec2 = ECOC_new(i,:); % row i of ecoc matrix
         d(i) = sum(mod(vec1+vec2,2)); % hamming distance
    end
    [m,class_test_ecoc_comb_new(k)] = min(d); % find smallest hamming
                                                                %distance
  end
loss_vector_test_ecoc_new = [loss_vector_test_ecoc_new,...
class_test_ecoc_comb_new ~= targets_test_all1];
check_ecoc_new = find(class_test_ecoc_comb_new ~= targets_test_all1);
error_prob_test_ecoc_new(runs,fold) = ...
length(check_ecoc_new)/length(targets_test_all1);
number_of_classifiers_15percent(fold) = length(good_index);
number_of_classifiers_10percent(fold) = length(good_index_alt);
% when using 10% cut
  for k = 1:length(class_test_ecoc)
    %determine hamming distance between columns of classification results
    %and ecoc matrix
    vec1_alt = class_test_array_new_alt(:,k)'; % column k (as row)
    for i=1:7
```

```
        vec2_alt = ECOC_new_alt(i,:); % row i of ecoc matrix
        d_alt(i) = sum(mod(vec1_alt+vec2_alt,2)); % hamming distance
    end
    [m,class_test_ecoc_comb_new_alt(k)] = min(d_alt); % find smallest
                                                        %hamming distance
  end
loss_vector_test_ecoc_new_alt = [loss_vector_test_ecoc_new_alt,...
class_test_ecoc_comb_new_alt ~= targets_test_all1];
check_ecoc_new_alt = find(class_test_ecoc_comb_new_alt ~= ...
targets_test_all1);
error_prob_test_ecoc_new_alt(runs,fold) = ...
length(check_ecoc_new_alt)/length(targets_test_all1);
  end % folds
figure
hist(hist_values_10per,63)
xlabel('number of binary classifiers')
title('Histogram of bad binary classifiers (10% cut-off)')
figure
hist(hist_values_15per,63)
xlabel('number of binary classifiers')
title('Histogram of bad binary classifiers (15% cut-off)')
%_____________________________________________________________________
% estimates of error probabilities and standard errors
% 7 class predictor (nearest neighbour)
error_prob_train7_kfold(runs) = mean(error_prob_train7(runs,:));
se_train7 (runs)= sqrt(var(loss_vector_train7)/total_number_of_data);
error_prob_test7_kfold(runs) = mean(error_prob_test7(runs,:));
se_test7(runs) = sqrt(var(loss_vector_test7)/total_number_of_data);
% ecoc combiner
error_prob_train_ecoc_kfold(runs) =...
mean(error_prob_train_ecoc(runs,:));
se_train_ecoc(runs) = ...
sqrt(var(loss_vector_train_ecoc)/total_number_of_data);
```

```
  error_prob_test_ecoc_kfold(runs) =...
  mean(error_prob_test_ecoc(runs,:));
  se_test_ecoc(runs) = ...
  sqrt(var(loss_vector_test_ecoc)/total_number_of_data);
  % ecoc combiner without bad classifiers using 15% cut
  error_prob_test_ecoc_new_kfold(runs) = ...
  mean(error_prob_test_ecoc_new(runs,:));
  se_test_ecoc_new(runs) = ...
  sqrt(var(loss_vector_test_ecoc_new)/total_number_of_data);
  % ecoc combiner without bad classifiers using 10% cut
  error_prob_test_ecoc_new_kfold_alt(runs) = ...
  mean(error_prob_test_ecoc_new_alt(runs,:));
  se_test_ecoc_new_alt(runs) = ...
  sqrt(var(loss_vector_test_ecoc_new_alt)/total_number_of_data);
  % combining neural network
  error_prob_trainc_kfold(runs) = mean(error_prob_trainc(runs,:));
  se_trainc (runs)= sqrt(var(loss_vector_trainc)/total_number_of_data);
  error_prob_testc_kfold(runs) = mean(error_prob_testc(runs,:));
  se_testc(runs) = ...
  sqrt(var(loss_vector_testc)/total_number_of_data);
  %clear all
  end
  disp('Seven class prediction on each fold using the training set and test set')
  error_prob_train7
  error_prob_test7
  disp('Binary prediction on each fold using the training set and test set')
  error_prob_train
  error_prob_test
  disp('ECOC prediction on each fold using the training set and test set')
  error_prob_train_ecoc
  error_prob_test_ecoc
  disp('ECOC without bad classifiers prediction on each fold using the test set
using 15% cut')
```

```
  error_prob_test_ecoc_new
  disp('ECOC without bad classifiers prediction on each fold using the test set
using 10% cut')
  error_prob_test_ecoc_new_alt
  disp('Overall predictions and standard errors for seven class classifiers')
  error_prob_train7_kfold
  se_train7
  error_prob_test7_kfold
  se_test7
  disp('means for test and training sets when using 7-class classifiers')
  average_error_prob_train7_kfold=mean(error_prob_train7_kfold)
  average_error_prob_test7_kfold=mean(error_prob_test7_kfold)
  disp('Confidence intervals for 7-class classifiers')
  conf_int_train7_kfold_left=(error_prob_train7_kfold-z_norm*se_train7)
  conf_int_train7_kfold_right=(error_prob_train7_kfold+z_norm*se_train7)
  conf_int_test7_kfold_left=(error_prob_test7_kfold-z_norm*se_test7)
  conf_int_test7_kfold_right=(error_prob_test7_kfold+z_norm*se_test7)
  disp('Overall predictions and standard errors for ECOC')
  error_prob_train_ecoc_kfold
  se_train_ecoc
  error_prob_test_ecoc_kfold
  se_test_ecoc
  disp('means for both train and test sets for ECOC')
  average_train_ecoc_kfold=mean(error_prob_train_ecoc_kfold)
  average_test_ecoc_kfold=mean(error_prob_test_ecoc_kfold)
  %confidence intervals
  disp('Confidence intervals for ECOC with bad classifiers')
  conf_int_train_ecoc_kfold_left=...
  (error_prob_train_ecoc_kfold-z_norm*se_train_ecoc)
  conf_int_train_ecoc_kfold_right=...
  (error_prob_train_ecoc_kfold+z_norm*se_train_ecoc)
  conf_int_test_ecoc_kfold_left=...
  (error_prob_test_ecoc_kfold-z_norm*se_test_ecoc)
```

```
  conf_int_test_ecoc_kfold_right=....
  (error_prob_test_ecoc_kfold+z_norm*se_test_ecoc)
  disp('Overall predictions and standard errors for ECOC without bad classifiers
on the test set using 15% cut')
  error_prob_test_ecoc_new_kfold
  se_test_ecoc_new
  disp('mean for 15% cut using ECOC')
  average_test_ecoc_new_kfold=mean(error_prob_test_ecoc_new_kfold)
  disp('Confidence intervals for ECOC without bad classifiers')
  conf_int_test_ecoc_new_kfold_left=...
  (error_prob_test_ecoc_new_kfold-z_norm*se_test_ecoc_new)
  conf_int_test_ecoc_new_kfold_right=...
  (error_prob_test_ecoc_new_kfold+z_norm*se_test_ecoc_new)
  disp('Overall predictions and standard errors for ECOC without bad classifiers
on the test set using 10% cut')
  error_prob_test_ecoc_new_kfold_alt
  se_test_ecoc_new_alt
  disp('mean for 10% cut using ECOC')
  average_test_ecoc_new_kfold_alt=mean(error_prob_test_ecoc_new_kfold_alt)
  disp('Confidence intervals for ECOC without bad classifiers')
  conf_int_test_ecoc_new_kfold_alt_left=...
  (error_prob_test_ecoc_new_kfold_alt-z_norm*se_test_ecoc_new_alt)
  conf_int_test_ecoc_new_kfold_alt_right=...
  (error_prob_test_ecoc_new_kfold_alt+z_norm*se_test_ecoc_new_alt)
  disp('number of bad binary classifier on each fold (row represents an experiment,
column represents a fold)')
  number_of_bad_f_k_10per
  number_of_bad_f_k_15per
  numb_average_10per=sum(number_of_bad_f_k_10per')
  numb_average_15per=sum(number_of_bad_f_k_15per')
  diary off % close the diary file
```