Neural Network Ensembles

Albert de Jongh

Thesis presented in partial fulfilment of the
requirements for the degree of Master of Commerce
at the University of Stellenbosch.

Professor Ian Cloete
April 2004

Declaration

I, the undersigned, hereby declare that the work contained in this thesis is my own original work and that I have not previously in its entirety or in part submitted it at any university for a degree.

Signature: Date:

Abstract

It is possible to improve on the accuracy of a single neural network by using an ensemble of diverse and accurate networks. This thesis explores diversity in ensembles and looks at the underlying theory and mechanisms employed to generate and combine ensemble members. Bagging and boosting are studied in detail and I explain their success in terms of well-known theoretical instruments. An empirical evaluation of their performance is conducted and I compare them to a single classifier and to each other in terms of accuracy and diversity.

Opsomming

Dit is moontlik om op die akkuraatheid van 'n enkele neurale netwerk te verbeter deur 'n ensemble van diverse en akkurate netwerke te gebruik. Hierdie tesis ondersoek diversiteit in ensembles, asook die meganismes waardeur lede van 'n ensemble geskep en gekombineer kan word. Die algoritmes "bagging" en "boosting" word in diepte bestudeer en hulle sukses word aan die hand van bekende teoretiese instrumente verduidelik. Die prestasie van hierdie twee algoritmes word eksperimenteel gemeet en hulle akkuraatheid en diversiteit word met 'n enkele netwerk vergelyk.

Acknowledgements

I am indebted to a very long list of people who supported me during this journey. A big thank you goes to:

- Professor Ian Cloete
- Jacolette Stemmet
- My family, especially my parents who instilled in me the passion to learn.
- My church cell group and friends for prayer support!
- Jehovah Jireh, the Almighty God.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The development of computing and communication technologies has produced a world that lives off information. This is both fascinating and scary. It has become possible for us to *google*[1] the internet for background on a suspicious neighbour, the price of a kite surfing wake board and the last time your friend got a parking ticket. You can check up on your doctor's diagnosis and read about all the side effects of the medicine the pharmacist has given you.

> But the most fascinating part is in what you don't know about yourself.

It has become possible for us to search for relationships between seemingly unconnected pieces of data. Data is the the recorded facts; the innocent events and numbers. Information is the set of patterns and expectations that are waiting to be discovered. Machine learning has made it possible for us to literally read between the lines to extract useful and important information.

A model generated from data by a machine learning algorithm can be regarded as an *expert*. This expert's quality depends on a lot of factors—the amount and quality of the training data, and whether the machine learning algorithm was suitable for the problem space.

When a wise king makes a decision, he usually takes into account the opinions of several wise people around him rather than relying on his own judgement or that of a single "trusted" advisor. Discussion of different viewpoints may produce a consensus; otherwise a vote may be called for. It is also important for this wise king to know that his committee of wise people have different opinions and that they represent the diversity of all the people in his kingdom.

An obvious extension of this would be to also use the opinion of more than one machine learning expert to extract information from data. Every-

[1]http://www.google.com

body knows the story of the Mars probe that crashed because of one team of scientists using imperial and another team using decimal units. Could this not have been prevented by a committee of machine learning experts on the probe working together and taking votes for decisions?

This thesis explores the notion of an ensemble of neural network experts. There are several aspects that I specifically want to investigate:

1. How do we build ensembles of neural networks? What is the underlying theory and mechanisms?

2. Can we determine if an ensemble of neural networks is more accurate than a single network? How do we measure the accuracy of an ensemble?

3. Can we explain the success of some of the well-known ensembling methods?

4. Does diversity in the ensemble members result in more accurate ensembles? How do we measure the diversity?

This chapter is devoted to introducing the concepts that are specific to the field of neural network ensembles. I explain the thesis layout in section 1.7.

1.1 What is an ensemble?

Suppose we have a supervised learning algorithm. The learning algorithm is given training examples from the training set $\mathbb{Z} = \{(\vec{z}, y)\}$ in the problem space Θ (i.e. $\mathbb{Z} \subseteq \Theta$) for some function that we are trying to approximate. The $\vec{z}$ values are vectors of the form $\langle z_1, \ldots, z_a \rangle$ consisting of discrete- or real-valued *features* or *attributes*. The y values are drawn from a discrete set of classes $\mathbb{L} = \{l_1, \ldots, l_m\}$ in the case of *classification* or from the set of real numbers in the case of *prediction*. For the purpose of this section I will only be referring to classification. The training set may be imperfect—it might have some *noisy* (incorrectly labelled) examples.

The learning algorithm outputs a *classifier* C after training on the set of training examples $\mathbb{Z}$. This classifier is a *hypothesis* of the true function. Given a new example $\vec{x}$ from the problem space, the classifier C will label it with a label $l \in \mathbb{L}$.

An *ensemble* of classifiers is a set of classifiers whose predictions are combined to produce a single classifier. This resulting classifier is generally more accurate than any of the base classifiers making up the ensemble.

1.2 Why are ensembles interesting?

An ensemble is usually more accurate than a single classifier. Different classifiers may also implicitly represent different aspects of the training set, while a single classifier cannot represent all useful aspects. An important condition for an ensemble to be more accurate than any of its individual members is that the base classifiers are both *accurate* (see section 1.3) and *diverse*. I discuss the diversity of ensembles in detail in chapter 2.

It is also possible to construct good ensembles that perform better than any of its base classifiers. There are three fundamental reasons (Dietterich (2000)) for this. See figure 1.1 for a depiction.

Statistical A learning algorithm's goal is to construct an approximation of a function $f(\vec{x})$. This process can be viewed as a search through the hypothesis space $\mathbb{H}$ to find the best approximation. This becomes a statistical problem when the number of available training examples are small compared to the size of the hypothesis space. In this case the learning algorithm will find many different approximations (classifiers) of $f(\vec{x})$ that have the same accuracy on the training examples. In the figure there are two curves denoting this situation. The outer curve is the total hypothesis space $\mathbb{H}$. The inner curve is the set of classifiers that has the same accuracy on the training examples. If we construct an ensemble out of C_1, C_2 and C_3 we can reduce the risk of choosing the wrong classifier.

Computational A learning algorithm usually trains by performing some kind of local search through the hypothesis space. This search may get stuck in a local maximum. For example, neural networks train by employing back propagation that uses a gradient descent search. Suppose we have enough training examples to eliminate the statistical problem. It may still be very difficult to find the best approximation of $f(\vec{x})$. If we construct an ensemble by locally searching from different starting points in $\mathbb{H}$ we have a better chance of not getting stuck in a local maximum.

Representational Our learning algorithm may be unable to approximate the function $f(\vec{x})$—i.e. $f(\vec{x})$ cannot be expressed by any of the hypotheses in $\mathbb{H}$. If a neural network uses linear activation functions and does not have enough hidden units it may be unable to estimate complex polynomial functions. An ensemble constructed out of the base classifiers C_1, C_2 and C_3 can enlarge the possible space of representable functions to include $f(\vec{x})$.

1.3 What is accuracy?

Accuracy is an important aspect of a classifier. For example, we use accuracy as one of the factors to determine whether an ensemble was "better" than a single classifier. Evaluating the accuracy of a classifier is also an integral component of many learning methods. It is therefore important to agree on its definition.

There are three important things to keep in mind when looking at the accuracy of a classifier or ensemble on a limited set of data.

1. How *biased* is the estimated accuracy? The accuracy of the classifier on the training examples is not a good estimate of its accuracy over unseen examples—the training accuracy is usually too optimistic since the classifier was derived from the training examples.

2. Suppose one classifier does better than another on the limited set of data—does this mean that this classifier is better in general?

3. What is the *variance* of the estimated accuracy? Even if the classifier accuracy is measured over an unbiased set of test examples independent of the training examples, the measured accuracy can still be different from the true accuracy. This depends on how close the distribution of the test set was to that of the function we are trying to learn. A small test set will lead to a greater expected variance.

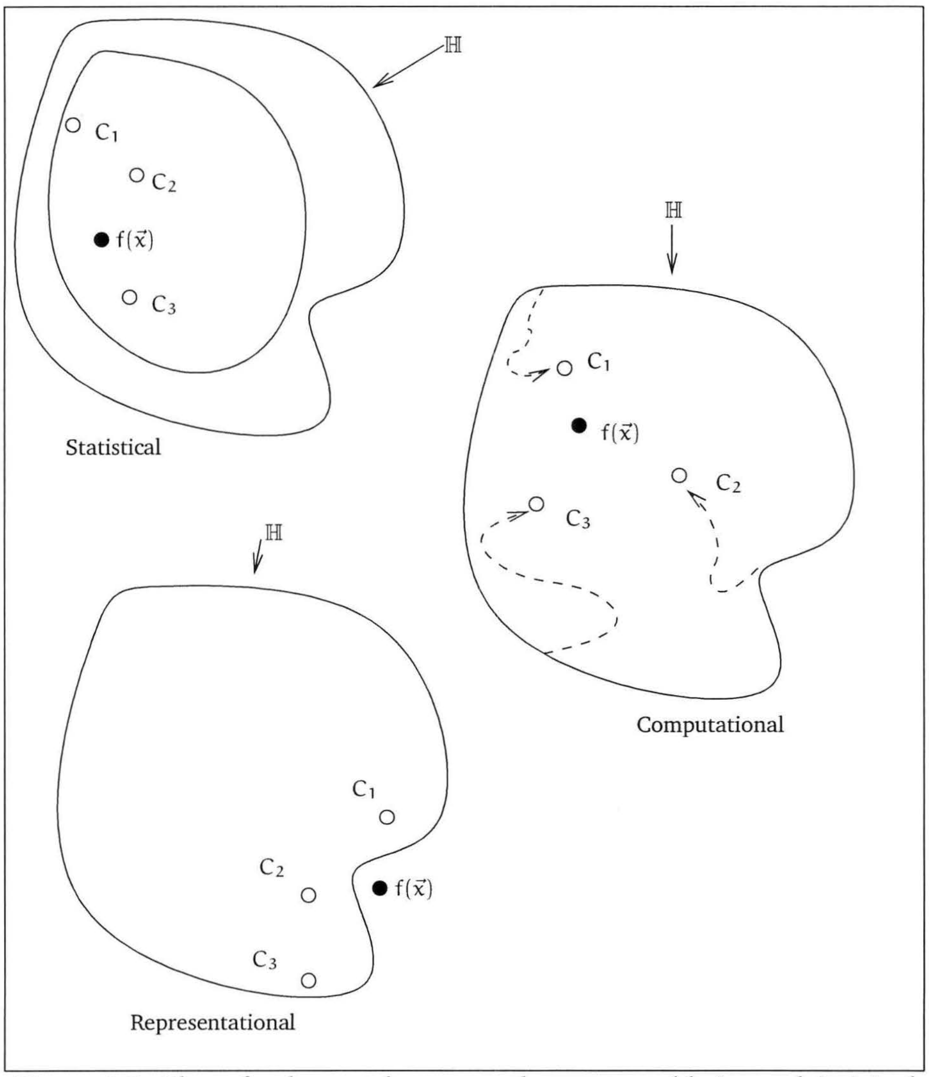


Figure 1.1: Three fundamental reasons why an ensemble is good (original figure in Dietterich (2000)).

This leads me to distinguish between *sample* and *true* accuracy, or equivalently, error. The **sample error rate** is the fraction of examples misclassified by the classifier C over the sample of data $\mathbb{Z}$:

$$\text{sample error} = \frac{1}{n}\sum_{\vec{z}\in\mathbb{Z}} \delta(f(\vec{z}), C(\vec{z})) \tag{1.1}$$

where f is the true function that we are trying to approximate and

$$\delta = \begin{cases} 1 & \text{when } f(\vec{z}) = C(\vec{z}) \\ 0 & \text{when } f(\vec{z}) \neq C(\vec{z}) \end{cases} \tag{1.2}$$

The **true error rate** of C is the probability that it will misclassify a single randomly drawn $\vec{x}$ from the input space Θ:

$$\text{true error} = \Pr_{\vec{x} \in \Theta} \left(f(\vec{x}) \neq C(\vec{x}) \right) \tag{1.3}$$

One would usually want to know the true error rate of a classifier; all that we can measure, however, is the sample error rate. Fortunately the sample error rate can be a good estimator of the true error rate, given that we make sure that the test set is independent from the training set and that it contains enough examples for how confident we want the estimation to be.

It might also be worth employing *cross-validation*. This has a computational impact with processing intensive learning algorithms like neural networks.

1.4 Bias-variance decomposition

The bias-variance decomposition is a useful theoretical tool for evaluating classifiers and ensembles. Several authors (Breiman (1996b), Opitz and Maclin (1999), Kohavi and Wolpert (1996), Witten and Frank (1999)) have used this as part of their proposed theories for the effectiveness of ensemble techniques like bagging and boosting. I will be referring back to this decomposition in later chapters.

The total expected error of a particular classifier on a specific target function and training set size has the following three components:

Bias The bias term measures how close the average classifier produced by the learning algorithm will be to the target function. Suppose we have an infinite number of independent training sets for a specific problem space. We can then use these training sets to set up and create an infinite number of classifiers. Take a random test instance and have it processed by all of the classifiers. Let the single ensemble answer be determined by the majority vote (or average if the class is numeric). Even in this ideal situation errors will still occur—no learning scheme is perfect. The error rate will depend on how well suited the machine learning method is to the specific problem. The learning algorithm's bias is defined as the averaged error rate over an infinite number of random test examples. If the bias term is zero we call the classifier *unbiased*.

The bias term is related to the *representational problem* in section 1.2.

Variance The variance term measures how much each of the classifier's classifications will vary with respect to each other and is related to the training set in use. The training set is usually finite and seldom completely representative of the distribution of the complete problem space. The expected value of this component of the total expected error of the learning algorithm over all the possible training sets is the variance.

Intrinsic target noise This term is defined as the minimum classification error associated with the Bayes optimal classifier for the problem. It is the lower bound of the expected cost associated with any learning algorithm. I explain the concept of a Bayes optimal classifier in the next section.

The bias-variance composition is also sometimes referred to as the *fundamental decomposition*.

Although this is a very interesting way to look at a specific learning algorithm it does have limitations when applying it to real-world problems. We need to know what the distribution is that we are trying to learn to estimate the bias, variance and target noise. This is of course unavailable for most real-world problems. Opitz and Maclin (1999) suggested holding out some of the data for this, but the training set size is greatly reduced if you want to get good estimates of the bias, variance and target noise.

1.5 Bayes optimal classifier

If is often interesting to compare our ensemble to the *best hypothesis* from the possible hypothesis space $\mathbb{H}$, given the set of training examples $\mathbb{Z}$. One way to explain what is meant by the "best" hypothesis is to say that we are searching for the *most probable* hypothesis, given the training data and any other initial knowledge that we know of the prior probabilities of the hypotheses in $\mathbb{H}$. The Bayes theorem provides a direct method for calculating such probabilities—calculate the probability of a hypothesis based on its *prior probability*, the probability of observing some data given the hypothesis and the observed data itself.

Bayes theorem is defined as

$$\Pr(h|\mathbb{Z}) = \frac{\Pr(\mathbb{Z}|h)\Pr(h)}{\Pr(\mathbb{Z})} \tag{1.4}$$

with the following definitions:

1. $\Pr(h)$ is the initial probability that a hypothesis h is true, before looking at the training data. $\Pr(h)$ is usually referred to as the *prior probability* of h and will include any background knowledge that may be available about the chance of h being the correct hypothesis. If no background knowledge is available, simply assign the same prior probability to every possible h.

2. $\Pr(\mathbb{Z})$ is the prior probability that $\mathbb{Z}$ will be observed; given no knowledge about which hypothesis is selected.

3. $\Pr(\mathbb{Z}|h)$ gives the probability that $\mathbb{Z}$ will be observed given the fact that hypothesis h holds.

$\Pr(h|\mathbb{Z})$ is the *posterior probability* of h—it is the confidence that h holds after training with the training examples $\mathbb{Z}$. $\Pr(h|\mathbb{Z})$ increases with $\Pr(h)$ and $\Pr(\mathbb{Z}|h)$. $\Pr(h|\mathbb{Z})$ decreases as $\Pr(\mathbb{Z})$ increases, because when there is a greater chance that $\mathbb{Z}$ will be observed independently of h, it also means that $\mathbb{Z}$ provides less "evidence" in support of h.

We are interested in finding the *best* or *most probable* hypothesis $h \in \mathbb{H}$, given the set of training examples $\mathbb{Z}$. Such a maximally probable hypothesis is called a *maximum a posteriori* (MAP) hypothesis. It is possible to determine the MAP hypotheses by using the Bayes theorem to calculate the posterior probability for each candidate hypothesis $h \in \mathbb{H}$. The MAP hypothesis h_{MAP} can be defined as

$$\begin{aligned} h_{\text{MAP}} &= \operatorname*{argmax}_{h \in \mathbb{H}} \Pr(h|\mathbb{Z}) \\ &= \operatorname*{argmax}_{h \in \mathbb{H}} \frac{\Pr(\mathbb{Z}|h)\Pr(h)}{\Pr(\mathbb{Z})} \\ &= \operatorname*{argmax}_{h \in \mathbb{H}} \Pr(\mathbb{Z}|h)\Pr(h) \end{aligned} \tag{1.5}$$

This far we have been trying to answer the question "what is the *most probable* hypothesis given the training data?" We are actually more interested in what is the most probable *classification* of a new instance given the training data. This question can be answered by feeding the new instance into the MAP hypothesis, but it is possible to do better.

Consider a hypothesis space containing three hypotheses, h_1, h_2 and h_3 (example taken from Mitchell (1996)). Let the posterior probabilities of these hypotheses be 0.4, 0.3 and 0.3. This means that h_1 is the MAP hypothesis (highest posterior probability). Take a new instance $\vec{x}$ and suppose h_1 classifies it positively and h_2 and h_3 classify it negatively. Taking all hypotheses into account, the probability that $\vec{x}$ is positive is 0.4 and the possibility that it is negative is $0.3 + 0.3 = 0.6$. The most probable (negative) classification in this case is not the classification generated by the MAP hypothesis.

Generally the most probable classification of a new instance is the combined predictions of all the hypotheses, weighted by their posterior probabilities. Take an example $\vec{z} \in \mathbb{Z}$ that can be labelled by a class label $l_k \in \mathbb{L}$. The probability $\Pr(l_k|\mathbb{Z})$ that the correct classification for $\vec{z}$ is l_k is

$$\Pr(l_k|\mathbb{Z}) = \sum_{h_i \in \mathbb{H}} \Pr(l_k|h_i)\Pr(h_i|\mathbb{Z}) \tag{1.6}$$

The optimal classification of $\vec{z}$ is the label l_k for which $\Pr(l_k|\mathbb{Z})$ is maximum. This is the **Bayes optimal classification**:

$$\operatorname*{argmax}_{l_k \in \mathbb{L}} \sum_{h_i \in \mathbb{H}} \Pr(l_k|h_i)\Pr(h_i|\mathbb{Z}) \tag{1.7}$$

No other classification method using the same hypothesis space and same prior knowledge can outperform this method on average. This method maximizes the probability that the new instance is classified correctly, given the set of training examples, the hypothesis space $\mathbb{H}$ and any known prior probabilities.

1.6 Design and implementation

The object oriented design and analysis process was used for the software developed. This included the normal stages:

- Requirements and initial analysis (setting up a problem statement and deducing the candidate objects, use cases and risks).
- Analysis
- Design
- Implementation

The process was adjusted slightly to fit my needs better—I used a more iterative version (similar to the *Extreme Programming* model).

1.6.1 Implementation

Implementation took place during several phases. I started out with a C++ neural network implementation and added the bagging ensemble method. During this time we decided to rather use Java. This had the pleasant result of us being able to run our programs on any platform without having to port code. I reused most of the bagging code and built a prototype neural network environment in Java. At this stage I was introduced to the WEKA (Waikato Environment for Knowledge Analysis) environment—a system developed by the Machine Learning group at the University of Waikato in New Zealand.

WEKA is a comprehensive toolkit for machine learning and data mining. Many learning algorithms have already been implemented within an object oriented Java framework. Regression, association rules and clustering algorithms have also been implemented. It includes a variety of tools for transforming and preprocessing datasets. It makes it easy for one to feed a dataset into a learning scheme, and to analyse the resulting classifier and its performance. It is enough to say that it is a comprehensive and powerful environment to conduct experiments within.

Rather than to build my own framework, I decided to reuse the existing functionality within WEKA, and to extend it where necessary. WEKA is an open source project and is distributed under the GNU general public license. This made it easy for me to extend WEKA. The licensing does have implications—if I were to sell my software I would have to provide the sources.

1.6.2 Tools

I employed a number of development tools during the course of this study.

Eclipse (http://www.eclipse.org) is an open source integrated development environment for Java (among others). It is used as the basis upon which tools like IBM's Websphere Studio for Application Development (WSAD) are built. I used it to develop, debug and distribute my extended WEKA application.

Ant is a general purpose Java build system. It is similar to the GNU `make` tool, but much more powerful. I used it to package and deploy my WEKA application. It is available from the Apache Software Foundation (http://jakarta.apache.org).

CVS is used for software configuration control and is an acronym for the *Concurrent Versions System* (available from http://www.cvshome.org). It was used for versioning of the software that was developed, as well as for the source files of this thesis.

Bash I wrote quite a lot of bash scripts (part of the Linux operating system) to automate some of the more mundane tasks—e.g. running sequences of experiments with different numbers of base classifiers in the ensembles.

Log4J is a logging framework for Java available from the Apache Software Foundation (http://jakarta.apache.org). WEKA did not use a proper logging framework and I have started to retrofit it with Log4J.

Linux was used as the operating system on most of the machines that I developed and deployed the WEKA package.

1.7 Thesis layout

I have attempted to arrange the material in this thesis to create a natural flow of ideas from the beginning to the end. The ideas introduced in this chapter are used throughout the rest of this work.

In chapter 2 I present a list of diversity measures that can be used to measure diversity (or similarity) in an ensemble of classifiers. Chapter 3 describes the underlying theory of ensemble methods and takes a look at some of the interesting methods that are available.

Bagging (chapter 4) and boosting (chapter 5) are two very well-known ensemble methods. They are described in detail, and I explain the different variations that are available. I test both methods on 11 data sets and present my empirical results—with specific reference to the diversity measures in chapter 2.

Chapter 6 concludes the thesis.

All the symbols used in the thesis are explained in Appendix A. The 11 data sets used for experiments are described in Appendix B.

Chapter 2

Diversity In Ensembles

An ensemble of classifiers are combined in the hope that the combination will be more accurate than a carefully constructed individual classifier. Chapter 3 will introduce a list of methods available for combining individual classifiers. Most of these methods have been shown to be very successful in improving on the accuracy of the individual base classifiers.

It only makes sense to combine classifiers that make their mistakes on different parts of the input space. A good example would be to take a committee of experts serving as consultants to the CEO of a company. If the committee members were to agree on every question asked of them the company would be better off by having just the best qualified expert in their service—and they would be saving a lot of money! Instead, if the experts were to have different opinions the CEO will be in a much better position to make balanced decisions.

The success of the combining methods introduced in chapter 3 is that, at least intuitively, they build an ensemble of *diverse* classifiers. Bagging generates data sets for each of the ensemble members by randomly selecting examples from the training set resulting in data sets that are related, but with random differences. There is no specific measure of diversity in this process, but it is assumed that the differences between the generated data sets are a key factor to the success of the bagging algorithm.

Quantifying this diversity is a first step towards trying to link diversity to the ensemble accuracy. The anticipation is also that diversity measures will help us in designing the members of the ensemble and the combination method.

In this chapter I present a list of diversity measures that may be used for measuring diversity in ensembles. There are four pairwise measures (section 2.2) and seven non-pairwise measures (section 2.3).

2.1 Definitions

Let $\mathbb{E} = \{C_1, C_2, \ldots, C_n\}$ be an ensemble of n classifiers that classifies examples from the input space Θ that has a attributes. Define $\mathbb{L} = \{l_1, l_2, \ldots, l_m\}$ to be a set of m class labels. Let $\vec{x} \in \Theta$ be a vector with a attributes.

All the measures in this chapter are discussed in terms of correct / incorrect decisions—the oracle output . The output $C_i(\vec{x})$ is 1 if $\vec{x}$ is correctly classified by C_i and 0 if C_i is wrong. This is an oracle output because it assumes that the correct class label of $\vec{x}$ is known.

Every measure can either be described as a *diversity* or *similarity* measure. The value of a diversity measure will increase if there is more diversity in an ensemble. A similarity measure's value will decrease with more diversity—it is the inverse of a diversity measure. I will categorize each of the measures as either a diversity or a similarity measure.

2.2 Pairwise measures

It is possible to derive many possible measures of the connection between two classifiers from statistics, but it is less clear when there are three or more classifiers. In this section we look at four pairwise measures and a way to find the averaged measure over all the classifiers.

2.2.1 Q Statistics

Let $\mathbb{Z} = \{\vec{z}_1, \vec{z}_2, \ldots, \vec{z}_N\}$ be the labelled set of training examples with each example $\vec{z}_j \in \Theta$. C_i either correctly or incorrectly classifies every $\vec{z}_j$. Let us represent this output of C_i as the binary vector $\vec{y}_i = \langle y_{1,i}, y_{2,i}, \ldots, y_{N,i}\rangle$, $i = 1, \ldots, n$, where $y_{j,i} = 1$ if C_i correctly classifies $\vec{z}_j$ and $y_{j,i} = 0$ if C_i incorrectly classifies $\vec{z}_j$.

Yule's Q Statistic (Yule (1900), Kuncheva and Whitaker (2003)) for two classifiers C_i and C_k is defined as

$$Q_{i,k} = \frac{N^{11}N^{00} - N^{01}N^{10}}{N^{11}N^{00} + N^{01}N^{10}} \tag{2.1}$$

where
N^{11} is the number of examples $\vec{z}_j$ for which $y_{j,i} = 1$ and $y_{j,k} = 1$,
N^{10} is the number of examples $\vec{z}_j$ for which $y_{j,i} = 1$ and $y_{j,k} = 0$,
N^{01} is the number of examples $\vec{z}_j$ for which $y_{j,i} = 0$ and $y_{j,k} = 1$ and
N^{00} is the number of examples $\vec{z}_j$ for which $y_{j,i} = 0$ and $y_{j,k} = 0$.

N^{11} can also be seen as the number of examples correctly classified by both classifiers, N^{10} as the number of examples correctly classified by C_i and incorrectly classified by C_k et cetera.

$$-1 \leq Q \leq 1 \tag{2.2}$$

Q has its maximum value of 1 when $N^{01}N^{10} = 0$—the classifiers correctly classify the same examples. If both classifiers always make their mistakes on different examples then $N^{11}N^{00} = 0$ and $Q = -1$. Classifiers that are similar will result in higher (positive) values of Q. Q is a *measure of similarity*.

2.2.2 Correlation coefficient

The correlation coefficient ρ between two classifiers C_i and C_k is

$$\rho_{i,k} = \frac{N^{11}N^{00} - N^{01}N^{10}}{\sqrt{(N^{11}+N^{10})(N^{01}+N^{00})(N^{11}+N^{01})(N^{10}+N^{00})}} \tag{2.3}$$

Q and ρ will always have the same sign and Kuncheva and Whitaker (2003) proved that $|\rho| \leq |Q|$.

$$-1 \leq \rho \leq 1. \tag{2.4}$$

ρ is also a *measure of similarity* and more diverse classifiers will result in smaller negative values of ρ.

2.2.3 Disagreement

The disagreement measure (Skalak (1996), Kuncheva and Whitaker (2003)) is the proportion of examples that only one classifier correctly classifies out of the total number of examples. Note that the total number of examples $N = N^{00} + N^{01} + N^{10} + N^{11}$.

$$\mathsf{Dis}_{i,k} = \frac{N^{10} + N^{01}}{N} \text{ for two classifiers } C_i \text{ and } C_k \tag{2.5}$$

Dis is a true *measure of diversity* as it will have higher (positive) values when the classifiers make their mistakes on different examples. Dis will have its maximum value of 1 when $N^{10} + N^{01} = N$ and $N^{11} + N^{00} = 0$—when there are no examples that both classifiers classify correctly and no examples that both classifiers make mistakes on. Similarly Dis will have its minimum value of 0 when $N^{11} + N^{00} = N$.

$$0 \leq \mathsf{Dis} \leq 1 \tag{2.6}$$

2.2.4 Double-fault

The double-fault measure is the proportion of examples that both classifiers C_i and C_k make mistakes on out of the total number of examples (Giacinto and Roli (2000), Kuncheva and Whitaker (2003)).

$$\mathsf{DF}_{i,k} = \frac{N^{00}}{N} \text{ for two classifiers } C_i \text{ and } C_k \tag{2.7}$$

Table 2.1 Matrix of pairwise Q statistics for an ensemble

	C_1	C_2	C_3	C_4	$\cdots$	C_n
C_1		$Q_{1,2}$	$Q_{1,3}$	$Q_{1,4}$	$\cdots$	$Q_{1,n}$
C_2			$Q_{2,3}$	$Q_{2,4}$	$\cdots$	$Q_{2,n}$
C_3				$Q_{3,4}$	$\cdots$	$Q_{3,n}$
$\vdots$					$\ddots$	$\vdots$
C_{n-1}						$Q_{n-1,n}$
C_n						

DF is a *measure of similarity* as it will have its highest value of 1 when both classifiers misclassify all examples ($N^{00} = N$). Then we can say

$$0 \leq \text{DF} \leq 1 \tag{2.8}$$

2.2.5 Pairwise measures for an ensemble

We would like to use the pairwise measures for an ensemble of more than two classifiers. If one were to calculate all the Q Statistics between all pairs of classifiers C_i and C_k in $\mathbb{E}$ the results can be presented in a matrix (see table 2.1). The Q Statistics measure is symmetrical and we only have to compute half of the matrix to be able to get an averaged measure. An averaged measure can then be computed for an ensemble of n classifiers as shown in equation 2.9.

$$Q_{avg} = \frac{2}{n(n-1)} \sum_{i=1}^{n-1} \sum_{k=i+1}^{n} Q_{i,k} \tag{2.9}$$

This method can be used to calculate all the averaged pairwise diversity measures.

2.2.6 Implementation

I implemented all the pairwise measures in the WEKA system. WEKA already had an `Evaluation` class that calculates all kinds of statistics on a classifier after it has been trained. It was logical that the diversity measures should also be calculated as part of the ensemble classifier evaluation process.

The diversity measures can only be used on ensemble classifiers like `Bagging` and `AdaBoost.M1` (package `weka.classifiers.meta` in WEKA). Since the diversity measures operate on the ensemble's base classifiers I created an interface that all meta (ensemble) classifiers that want their base classifiers to be evaluated for diversity must implement. The `Evaluation` class always checks if a classifier implements this interface before attempting to calculate the diversity measures on its base classifiers.

2.3 Non-pairwise measures

2.3.1 Entropy

Take an example $\vec{z}_j \in \mathbb{Z}$ from the input space Θ. An ensemble of classifiers has the highest diversity for this $\vec{z}_j$ when half of the classifiers ($\lfloor n/2 \rfloor$) in the ensemble correctly classify $\vec{z}_j$ and the other half ($n - \lfloor n/2 \rfloor$) misclassify $\vec{z}_j$.

Define $l(\vec{z}_j)$ to be the number of ensemble base classifiers (C_i) that correctly classifies a $\vec{z}_j$. The Entropy Measure E (Kuncheva and Whitaker (2003)) is defined in equation 2.10.

$$\mathsf{E} = \frac{1}{N} \sum_{j=1}^{N} \frac{\min\{l(\vec{z}_j), n - l(\vec{z}_j)\}}{n - \lfloor n/2 \rfloor} \tag{2.10}$$

E has its highest diversity value of 1 when $l(\vec{z}_j) = \lfloor n/2 \rfloor, \forall j$. E has its lowest value of 0 when $l(\vec{z}_j) = n$ or $l(\vec{z}_j) = 0, \forall j$.

$$0 \leq \mathsf{E} \leq 1 \tag{2.11}$$

E is a *measure of diversity*.

2.3.2 Kohavi-Wolpert variance

Kohavi and Wolpert (1996) defined the variance of the predicted class label y for $\vec{x} \in \Theta$ across training sets for a specific classifier as

$$\text{variance}_{\vec{x}} = \frac{1}{2} \left(1 - \sum_{i=1}^{m} \Pr(y = l_i|\vec{x})^2 \right) \tag{2.12}$$

Remember that $l_i \in \mathbb{L}$ was defined as one of m possible class labels for every $\vec{x}$ in section 2.1. For the oracle output we are only interested in two possible class labels: $\mathbb{L} = \{0, 1\}$. Kuncheva and Whitaker (2003) used this idea in the following way: look at the variance of the predicted class label y for the given training set using the base classifiers from the ensemble $\mathbb{E}$.

Kohavi and Wolpert (1996) estimated $\Pr(y = l_i)|\vec{x})$ as an average over the different data sets. For the oracle output we can estimate $\Pr(y = 0|\vec{x})$ and $\Pr(y = 1|\vec{x})$ for the training set over all the base classifiers C_i, $i = 1, 2, \ldots, n$ as

$$\hat{P}(y = 1|\vec{x}) = \frac{l(\vec{x})}{n} \text{ and } \hat{P}(y = 0|\vec{x}) = \frac{n - l(\vec{x})}{n} \tag{2.13}$$

If you substitute equation 2.13 into equation 2.12 you get

$$\begin{aligned} \text{variance}_{\vec{x}} &= \frac{1}{2}\left(1 - \hat{P}(y=1|\vec{x})^2 - \hat{P}(y=0|\vec{x})^2\right) \\ &= \frac{1}{2}\left(1 - \frac{l(\vec{x})^2}{n^2} - \frac{(n - l(\vec{x}))^2}{n^2}\right) \\ &= \frac{l(\vec{x})(n - l(\vec{x}))}{n^2} \qquad (2.14) \end{aligned}$$

Take the average of equation 2.14 over all the examples $z_j \in \mathbb{Z}$, and define the Kohavi-Wolpert variance (Kuncheva and Whitaker (2003)) to be

$$\mathsf{KW} = \frac{1}{N}\sum_{j=1}^{N} \frac{l(\vec{z}_j)(n - l(\vec{z}_j))}{n^2} \qquad (2.15)$$

KW has its smallest diversity value of 0 when $l(\vec{z}_j) = 0$ or $l(\vec{z}_j) = n, \forall j$. This will happen if all the classifiers classified all the $\vec{z}_j$ correctly or were wrong for all the examples. KW will have its largest value of $1/4$ when $l(\vec{z}_j) = \lfloor n/2 \rfloor, \forall j$.

$$0 \le \mathsf{KW} \le \frac{1}{4} \qquad (2.16)$$

KW is a *measure of diversity*.

2.3.3 Interrater agreement

This measure is derived from the measure of interrater reliability, κ, which is used to determine the level of agreement of raters assessing subjects. Kuncheva and Whitaker (2003) adjusted this measure to make it usable in our context—classifiers (raters) and training examples (subjects).

Let $\overline{p}$ be the average base classifier accuracy.

$$\begin{aligned} \overline{p} &= \frac{1}{N}\sum_{j=1}^{N}\frac{1}{n}\sum_{i=1}^{n} y_{j,i} \\ &= \frac{1}{N}\sum_{j=1}^{N}\frac{l(\vec{z}_j)}{n} \qquad (2.17) \end{aligned}$$

Then we can define

$$\begin{aligned} \kappa &= 1 - \frac{\frac{1}{n}\sum_{j=1}^{N} l(\vec{z}_j)(n - l(\vec{z}_j))}{N(n-1)\overline{p}(1-\overline{p})} \qquad (2.18) \\ &= 1 - \frac{n}{(n-1)\overline{p}(1-\overline{p})}\mathsf{KW} \qquad (2.19) \end{aligned}$$

The measure of interrater agreement is a *measure of similarity* since larger values of κ will occur when the ensemble base classifiers are more similar. Diverse base classifiers will result in negative values of κ. The value of κ is dependent on $\overline{p}$ and n. When $\overline{p} \to 0$ or $\overline{p} \to 1$ it will result in a very large factor for KW—resulting in a possible large negative (diverse) value of κ.

2.3.4 Difficulty

Define X to be a discrete random variable taking values in $\{0/n, 1/n, \ldots, 1\}$. The possible values of X describe the proportion of classifiers from $C_i \in \mathbb{E}$ correctly classifying a random $\vec{x} \in \Theta$. In other words—X tells us how *difficult* it was for the ensemble to classify a random $\vec{x}$. The measure of difficulty is based on the distribution of X.

Kuncheva and Whitaker (2003) suggested capturing this distribution shape by using the variance of X, σ_X^2, as the measure of difficulty θ.

We know that

$$\begin{aligned} \sigma_X^2 &= E[X^2] - (E[X])^2 \\ &= \mu_{X^2} - (\mu_X)^2 && (2.20) \\ \mu_X &= \sum_{x \in X} x \Pr(x|X) && (2.21) \end{aligned}$$

We estimate $\Pr(x|X)$ (equation 2.21) for all the training examples $\vec{z}_j \in \mathbb{Z}$ by building a histogram showing how many examples did $0/n, 1/n, \ldots, n/n$ classifiers correctly classify—define $h(x)$ to be the number of examples correctly classified by $x \in X$ classifiers. Then

$$\hat{P}(x|X) = \frac{h(x)}{N} \qquad (2.22)$$

from which we can calculate μ_X and μ_{X^2}.

Higher values of θ will mean a less diverse classifier team. The ideal (most diverse) ensemble will have $\theta = 0$. This implies that θ is a *measure of similarity*.

2.3.5 Generalized diversity

Krzanowski and Partridge (1997) proposed this measure as part of an article about diversity in software. They did a study on how different and diverse versions of mission-critical software systems (for example air craft guidance systems and nuclear reactor protection systems) could prevent software failure through inevitable errors.

Let Y be a discrete random variable taking values in $\{0/n, 1/n, \ldots, 1\}$. The possible values of Y describe the proportion of classifiers from $C_i \in \mathbb{E}$ incorrectly classifying a random $\vec{x} \in \Theta$ (Y is the inverse of X introduced in section 2.3.4). Define $p(i)$ to be the probability that i classifiers randomly picked from $\mathbb{E}$ misclassify a random $\vec{x}$.

Krzanowski and Partridge (1997) suggested that maximum diversity in a software system occurs when one randomly chosen part of the system failing results in another randomly chosen part not failing. In our case this would translate into maximum diversity when one randomly chosen $C_i \in \mathbb{E}$ fails and another $C_k \in \mathbb{E}$ correctly classifies a random example. The probability of both classifiers failing in this case is then $p(2) = 0$. Minimum diversity will be when failure of one classifier is always accompanied by failure of the other classifier. Then $p(2) = p(1)$.

Krzanowski and Partridge (1997) proved that

$$\begin{aligned} p(1) &= \sum_{i=1}^{n} \frac{i}{n} \Pr(Y = i/n) && (2.23) \\ p(2) &= \sum_{i=1}^{n} \frac{i(i-1)}{n(n-1)} \Pr(Y = i/n) && (2.24) \end{aligned}$$

We can estimate $\Pr(Y = i/n)$ similar to equation 2.22. Krzanowski and Partridge (1997) defined the generalized diversity measure as

$$\mathsf{GD} = 1 - \frac{p(2)}{p(1)} \tag{2.25}$$

GD will indicate maximum diversity of 1 when $p(2) = 0$. Minimum diversity occurs when $\mathsf{GD} = 0$ and $p(2) = p(1)$.

$$0 \le \mathsf{GD} \le 1 \tag{2.26}$$

GD is a *measure of diversity*.

2.3.6 Coincident failure diversity

Krzanowski and Partridge (1997) also proposed a modification to generalized diversity—coincident failure diversity. CFD will have a minimum value of 0 (no diversity) when all classifiers are always correct or when all classifiers are simultaneously correct or wrong. CFD will have its maximum value of 1 (very diverse ensemble) when at most one classifier will fail on any random chosen example.

$$\mathsf{CFD} = \begin{cases} 0, & \Pr(Y = 0/n) = 1 \\ \frac{1}{1-\Pr(Y=0/n)} \sum_{i=1} n \frac{n-i}{n-1} \Pr(Y = i/n), & \Pr(Y = 0/n) < 1 \end{cases} \tag{2.27}$$

Once again we can estimate $\Pr(Y = i/n)$ similar to equation 2.22.

$$0 \le \mathsf{CFD} \le 1 \tag{2.28}$$

CFD is also a *measure of diversity*.

2.3.7 Fitness

Opitz and Shavlik (1996) used the *fitness* measure as part of their ADDEMUP algorithm to help prune their generated ensembles of the least fittest base classifiers. They combined the base classifiers using a simple weighted sum of the base classifier outputs:

$$\hat{o} = \sum_{i=1}^{n} w_i \cdot C_i \text{ with } \sum_{i=1}^{n} w_i = 1$$

They then define a "diversity" measure d_i for classifier C_i

$$d_i = \sum_{j=1}^{N} [C_i(\vec{z}_j) - \hat{o}(\vec{z}_j)]^2 \tag{2.29}$$

The fitness measure F for classifier i is defined as

$$\begin{aligned} F_i &= \text{Accuracy}_i + \lambda d_i \\ &= (1 - \epsilon_i) + \lambda d_i \end{aligned} \tag{2.30}$$

with ϵ_i the error rate of classifier i and λ a trade-off between accuracy and diversity. Diverse and accurate ensemble members will have a greater value of F_i. Furthermore $F_i \geq 0$. This means that F is a *measure of diversity*.

We can determine the average fitness of the ensemble by averaging F_i over all the ensemble classifiers.

$$F = \frac{1}{n} \sum_{i=1}^{n} F_i \tag{2.31}$$

2.3.8 Implementation

I implemented all the non-pairwise measures in the WEKA system—similar to the pairwise measures. If a classifier implements the `MetaClassifier` interface its non-pairwise measures will be computed and displayed as part of the normal evaluation process.

Table 2.2 Summary of diversity measures. An up-arrow (↑) specifies that it is a measure of diversity. A down-arrow (↓) specifies that it is a measure of similarity.

Pairwise			
Q Statistics	Q	↓	$-1 \leq Q \leq 1$
Correlation coefficient	ρ	↓	$-1 \leq \rho \leq 1$
Disagreement	Dis	↑	$0 \leq \text{Dis} \leq 1$
Double-fault	DF	↓	$0 \leq \text{DF} \leq 1$
Non-pairwise			
Entropy	E	↑	$0 \leq E \leq 1$
Kohavi-Wolpert variance	KW	↑	$0 \leq \text{KW} \leq 1/4$
Interrater agreement	κ	↓	
Difficulty	θ	↓	$\theta \geq 0$
Generalized diversity	GD	↑	$0 \leq \text{GD} \leq 1$
Coincident failure diversity	CFD	↑	$0 \leq \text{CFD} \leq 1$
Fitness	F	↑	$F \geq 0$

2.4 Summary

Table 2.2 summarizes the eleven different measures of diversity introduced in this chapter. In the following chapters we will use these measures to determine if there is some kind of relationship between ensemble accuracy and diversity and if there are specific ensemble methods that are more suitable for generating diverse ensembles.

One has to be careful in using these measures—they should be used in conjunction with the normal accuracy measures. The fitness measure (section 2.3.7) is a good example of how I think one should be using the measures of diversity as part of the ensemble building process.

Chapter 3

Ensemble Methods

An ensemble of classifiers is a set of classifiers whose predictions (or classifications) are combined to produce a single superior classifier. The ensemble is usually more accurate than any single base classifier and may be able to better represent the classification problem.

There is a myriad of ensemble methods available today and they all have the same basic steps that take place as part of the ensemble process:

1. Generate the members of the ensemble.

2. Combine the members' predictions.

I explain this general process in section 3.1 and describe the basic techniques that are used in most methods.

The last part of the chapter (sections 3.2 to 3.9) introduces some well-known ensemble methods.

3.1 Process of building an ensemble

This section outlines the high-level ideas behind many of the ensemble methods in use.

3.1.1 Generate the base classifiers

Training examples

This is a common method for generating multiple base classifiers and is effective for *unstable* learning algorithms like neural networks and decision trees. The learning algorithm is run several times with a different set of examples based in some way on the original set of training examples:

1. Divide the training set into a number of disjoint subsets. Construct the base classifiers by running the learning algorithm on sets formed by leaving out a different disjoint subset for every iteration. This is similar to the process of *cross-validation* and ensembles constructed in this way is sometimes also called *cross-validated committees*.

2. Take bootstrap replicates of the training set by drawing random examples from it (with or without replacement). The most famous way of perturbing the training examples in this way must be bagging—it is explained in detail in chapter 4.

3. Add artificial noise to the training examples. This randomness is usually enough for unstable procedures to start their search at a different place in the hypothesis space. One can either add random noise to the training examples or add small amounts of noise to the input attributes, but leave the outputs as is. This is also known as *jittering*.

Input features

In cases where the input features (attributes) are highly redundant one can also manipulate the set of input features available to the learning algorithm. In cases where the classes are representable by entirely different feature domains the generation of random subsets out of a large feature set also appears to be successful.

Output targets

This general technique works by manipulating the class labels that are given to the learning algorithm. Dietterich and Bakiri (1995) developed a method called *error-correcting output coding* from this general technique.

Learning algorithm

An obvious method of generating different base classifiers is to manipulate the learning algorithms. This can be done in any of the following ways:

1. Initialize the learning algorithm with different parameters for every new base classifier. This can include random weights for neural networks and different parameter choices like the number of hidden nodes and layers. This is useful for injecting randomness into the learning algorithm.

2. Use heterogeneous sets of classifiers built using different learning algorithms in the same feature space and using the same training set. Different classifiers are able to express their opinions in different ways.

3.1.2 Combine the base classifiers

Voting

Voting is commonly used with classification and several variations exist:

1. An *unanimous* vote. The ensemble classifies a new example $\vec{x}$ as l only if all the base classifiers classified the new example with l.

2. *Modified unanimous vote*. The ensemble classifies a new example $\vec{x}$ as l if some base classifiers classified the new example as l and no other classifier labelled the new example with another class (rejects it).

3. A *majority* vote (weighted or unweighted). The ensemble decides that the new example $\vec{x}$ belongs to class l if more than half of the base classifiers support that $\vec{x}$ belongs to l. This rule is sometimes modified to require a different proportion of classifiers to agree.

4. *Threshold plurality*. The ensemble decides that $\vec{x}$ belongs to the class l if the number of classifiers that support it is considerably bigger than the number of base classifiers that support any other class.

Fixed rules

The fixed combining rules (Duin (2002)) make use of the fact that the outputs of the ensemble's base classifiers have a clear definition. It is not just numbers—it may be class labels, distances, probabilities and predictions. Suppose our base classifiers were to predict the probability of an example $\vec{x}$ being in a class l. There are several possible ways to combine these base classifiers into an ensemble.

1. The *product* rule is good if the individual base classifiers are independent. If performs best when the base classifiers are trained on different feature spaces.
$$\mathbb{E} \sim \prod_i C_i(\vec{x})$$
The rule also needs noise free and reliable confidence estimates and fails when the estimates are zero or very small.

2. The *sum* rule is often used with ensembles of predictors. It usually takes the form of an average of the outputs of the base classifiers or a weighted sum.
$$\mathbb{E} \sim \sum_i C_i(\vec{x})$$
The rule works well with a collection of similar base classifiers with independent noise behaviour—the errors in the probabilities are averaged out by the summation. This rule is equivalent to the product rule when the classifier outputs are similar, but still independent.

3. The *maximum* rule selects the classifier that is most confident. This fails when some of the base classifiers are over-trained.
$$\mathbb{E} \sim \max\{C_i\}$$
It is difficult to find applications for which this rule works well.

4. The *minimum* rule selects the classifier that has the least objection against a classifier; similarly to the maximum rule it is not easy to find applications for this rule.
$$\mathbb{E} \sim \min\{C_i\}$$

5. The *median* rule is similar to the sum rule, but is sometimes more robust.
$$\mathbb{E} \sim \text{median}\{C_i\}$$

Combining classifier

Instead of using voting or a fixed rule one can also use the training set and the outputs of the base classifiers to train another classifier to combine the base classifiers into an ensemble. One has to carefully consider how much and what subset of the training set to use for training the combining classifier as it is very easy to over-train the ensemble with this method.

An application of this method is discussed in more detail in section 3.6.

3.2 Select the single best classifier

The simplest form of an "ensemble" is simply selecting the single best or most accurate classifier from a set of base classifiers generated in some way. It is simple to use and easy to understand what is happening. It does however have the problems explained in sections 1.2 and 1.4.

One has to have an independent set of examples for evaluating the set of base classifiers and this set cannot be the same as the test set. If your set of labelled examples is small you might not have enough data to do this properly and you might select an over-trained classifier as the "best" classifier. This "best" classifier will in general perform worse on unseen data than some of the other generated classifiers.

A variation of this idea is discussed in section 3.8.

3.3 Linear combinations

Ensembles of predictors are often combined using the average or weighted sum of the outputs of the base classifiers. It is simple to construct and, depending on the method used to construct the base classifiers, often outperforms any single best base classifier. The ensemble members can be constructed in any way. A linear combination of classifiers is a *fixed summation rule*.

From a neural network perspective linearly combining the outputs of a number of trained neural networks is similar to setting up a single large neural network in which the trained neural networks are smaller networks operating in parallel. The combination weights are the connection weights of the output layer. For a given example $\vec{x}$ the output of the combined model is the weighted sum of the outputs of the component neural networks.

The general form of the linear combination is

$$\mathbb{E} = \sum_{i=1}^{n} w_i C_i$$

where the $\{w_i\}$ is a set of weights that may sum up to one. Using

$$w_i = \frac{1}{n}, \forall i$$

is the common method of averaging the outputs of the ensemble members.

Granger and Ramanathan (1984) considered three different approaches to obtaining linear combinations. They found that the best method is not to use the common practise of a weighted sum with the weights adding up to one. One should rather add a constant term to the sum and not constrain the weights to add up to unity:

$$\mathbb{E} = w_0 m + \sum_{i=1}^{n} w_i C_i \tag{3.1}$$

with m the mean of the outputs of the base classifiers. The weights $\{w_i\}$ are obtained by regression.

Linear combinations mainly work by attacking the statistical problem described in section 1.2.

3.4 Optimal linear combinations

Hashem et al. (1994) also proposed using an "optimal" linear combination of neural networks instead of a single best network. The optimal combination is constructed in a way similar to equation 3.1. Hashem et al. (1994) found that their optimal combination was better than that of the single best trained neural network and that of the simple averaging of the outputs of the base networks.

They set $m = 1$ and found that the optimal weights $\{w_i\}$ are equal to the ordinary least squares regression coefficients. The weights are not constrained to sum to unity. This algorithm is shown in figure 3.1.

Hashem et al. (1994) found that the *optimal linear combination* method performed better on poorly trained neural network base classifiers than on well-trained networks. For well-trained networks the combination weights tended to automatically sum to unity, with the constant term w_0 approaching 0. This can be explained by the fact that the base networks are close to the approximated function and have little bias. For the poorly trained networks the combination weights did not sum to unity while the constant term was significantly different from 0. In the case of the well-trained networks the optimal linear combination method operated in a "fine-tuning" role, while for the poorly trained networks it performed a significant modelling role.

3.5 Weighted majority

The weighted majority algorithm (Littlestone and Warmuth (1992)) is a simple and effective method that can construct an ensemble that is robust in the presence of errors in the data. The algorithm makes predictions by taking a weighted vote among a pool of prediction algorithms and builds the ensemble by altering the weight associated with every base classifier.

First we have to construct and train the set of base classifiers using methods derived from section 3.1.1. These classifiers make varying numbers of mistakes. The goal of the weighted majority algorithm is not to

```
*Ensemble generation*
**Require:** A labelled training set $\mathbb{Z}$. Base neural network architectures—may
  be the same or different.
  **for** $i = 1$ to $n$ **do**
    Construct neural network $C_i$ according to architecture settings and with
    random connection weights.
    Train this neural network with the whole training set $\mathbb{Z}$.
  Construct the ensemble

      $\mathbb{E} = w_0 + \sum_{i=1}^{n} w_i C_i$

  and estimate the combination weights by least squares regression.
*Classification procedure*
**Require:** An unseen example $\vec{x}$.
  Feed the example into the ensemble.
```

Figure 3.1: Algorithm for optimal linear combinations

make more mistakes than the best algorithm in the pool, even though it does not have any knowledge about the accuracy of the base classifiers.

The weighted majority algorithm begins by assigning a weight of 1 to every base classifier in the pool and then uses the set of training examples to modify these weights to reflect the accuracy of the base classifiers. Ensemble "learning" then proceeds in a sequence of iterations (see figure 3.2). In every iteration the algorithm takes an example from the set of training examples and feeds it to each of the base classifiers. Each classifier makes a prediction and these predictions are grouped together to enable the master algorithm to make its prediction. If the master algorithm misclassifies an example each base classifier that has misclassified that example has its weight reduced by a factor $0 \leq \beta \leq 1$. This makes it possible for the weighted majority algorithm to accommodate inconsistent training data. The algorithm never completely eliminates a base classifier but only reduces its weight. The weight of a classifier represents the "belief" of the master algorithm in the accuracy of the member.

If the weighted majority algorithm is used with equal weights and $\beta = 0$, it is identical to the *halving* algorithm. If $\beta > 0$, weighted majority gradually decreases the influence of base classifiers that make a large number of mistakes and gives the classifiers that make few mistakes high relative weights.

I have described the basic weighted majority algorithm where the predictions of the base classifiers and the ensemble as well as the labels of the examples are required to be binary. Various variants of the algorithm exist that that allow continuous predictions.

Littlestone and Warmuth (1992) proved that the number of mistakes m made by weighted majority

$$m < \frac{k \log_2 \frac{1}{\beta} + \log_2 n}{\log_2 \frac{2}{1+\beta}}$$

where k is the minimum number of mistakes made by any base classifier and n is the number of classifiers in the ensemble. This means that the number of mistakes made by weighted majority will never be greater than a constant factor times the number of mistakes made by the best classifier in the ensemble, plus a term that grows logarithmically with the size of the ensemble.

The prediction of the weighted majority vote is based on the weighted average of the predictions of the ensemble base classifiers. In that way it is similar to linearly combining predictions through a method like optimal linear combinations. Weighted majority works because it attacks the *statistical* problem (section 1.2) and reduces the *variance* of the bias-variance decomposition (section 1.4) of the total expected error.

3.6 Stacking

Stacked generalisation (Wolpert (1992)) can also be used for combining base classifiers (section 3.1.2). Stacking is generally used to combine base classifiers that are not of the same type—an ensemble of decision trees and neural networks, for example. Stacking introduces the notion of a *meta learner*—replacing the voting or averaging combining mechanisms that one finds in ensemble methods like bagging and boosting. With the meta learner stacking tries to learn how trustworthy each of the base classifiers in the ensemble is. The meta learner is another learning algorithm—for example a neural network—that trains on the outputs of the base classifiers. Some researchers also suggest using some of the inputs to the base classifiers in conjunction with the base classifiers' outputs; I am of the opinion that this will increase the chances of over-training the ensemble.

The meta learner tries to learn which of the base classifiers are more reliable—and thus how to best combine them. The meta learner has the same of number of inputs or attributes as the number of base classifiers—and these attribute values are simply the predictions of the base classifiers. During classification an instance is first fed into the base classifiers, and each one predicts a value. These predictions are then fed into the meta learner to be combined into a single final prediction.

Unfortunately we can't use the same training examples that have been used to train the base classifiers to also train the meta learner—for the same reason that we are unable to reuse the training examples to approximate the

```
*Ensemble generation*
Require: Labelled training set $\mathbb{Z}$. A learning algorithm for training the base
  classifiers. A weight adjustment factor $0 \leq \beta \leq 1$.
  for $i = 1$ to $n$ do {Train the base classifiers.}
    Construct classifier $C_i$ in some way (using a method derived from section 3.1.1).
    Train $C_i$ with the learning algorithm.
    Assign weight $w_i = 1$ to $C_i$.
  for all $\vec{z} \in \mathbb{Z}$ with binary label $y \in \mathbb{L} = \{0, 1\}$ do {Set up the weights.}
    Set $q_0 = 0$ and $q_1 = 0$.
    for all classifiers $C_i$ do
      if $C_i(\vec{z}) = 0$ then
        $q_0 = q_0 + w_i$
      else
        $q_1 = q_1 + w_i$
    if $q_1 > q_0$ then
      Set $c = 1$.
    else if $q_0 > q_1$ then
      Set $c = 0$.
    else
      Set $c$ at random to 1 or 0.
    if $c \neq y$ (not the correct binary label for $\vec{z}$) then
      for all classifiers $C_i$ do
        if $C_i(\vec{z}) \neq y$ then
          Update weight $w_i = w_i\beta$.
*Classification procedure*
Require: An unseen example $\vec{x}$.
  Set $q_0 = 0$ and $q_1 = 0$.
  for all classifiers $C_i$ in the ensemble $\mathbb{E}$ do
    if $C_i(\vec{z}) = 0$ then
      $q_0 = q_0 + w_i$
    else
      $q_1 = q_1 + w_i$
  if $q_1 > q_0$ then
    Predict 1.
  else if $q_0 > q_1$ then
    Predict 0.
  else
    Predict at random 1 or 0.
```

Figure 3.2: Algorithm for weighted majority vote. Assumes the base classifiers does binary classification.

true error rate. We are trying to use the meta learner to learn the reliability of the base classifiers. If we reuse the training examples we will be too optimistic—some of the base classifiers may be over-trained on the training examples and will receive an incorrect better rating. For this reason we have to have two sets of training examples—one for the base classifiers and another one for training the meta learner. The set used for training the meta classifier must never be used during training of the base classifier. In this way the base classifiers' predictions will be unbiased and the meta learner will be more accurate.

This process does make the training set even smaller. It is possible to incorporate the process of *cross-validation* in conjunction with stacking to make better use of the available data.

In his paper "The Combining Classifier: to Train or Not to Train", Duin (2002) has some interesting points that relate to stacking. He remarks that if the base classifiers have been trained independently (i.e. using different feature sets or different numbers of epochs) one must look carefully at their outputs. It may be necessary to calibrate or scale the outputs. Duin (2002) also proposed the following strategies for using the training data:

- Use a single training set for both the base classifiers and the meta model. Train the base classifiers carefully to avoid over-fitting. Reuse the training set for training the meta learner.

- Use a single training set for both the base classifiers and the meta model. Train the base classifiers weakly. Reuse the training set for training the meta learner.

- Separate the training examples into two sets. Use one set for training the base classifiers and another set for training the meta learner.

Stacking can help us find a solution for the *representational problem* described in section 1.2.

3.7 Mixture of expert models

Milidiú et al. (1999) described a system consisting of a mixture of different learning models suitable for time series forecasting. The training examples are partitioned into disjoint sets using a clustering algorithm. Every disjoint set is then used for training a number of different learning models. For every disjoint set a "winning" model is then chosen based upon accuracy on the independent test set. The strength of this method is that it always uses the most appropriate learning algorithm for the specific cluster of training data—reducing the bias (section 1.4).

The MEM system starts by doing a *Haar wavelets* transform on the training examples. This prepares the training set for the clustering algorithm

Isodata. The Isodata algorithm clusters the training examples into a predefined number of classes. Empty clusters are discarded.

Several learning methods are now used to generate a base classifier for every cluster using the training examples in that cluster. The learning methods may include neural networks, statistical models, decision trees—any learning algorithm that may be applicable to the kind of time series. The best model for every cluster must now be selected.

The independent test examples are first classified into the different clusters identified in the clustering step. This is done by selecting for each test example the corresponding centroid that has the minimum euclidian distance to it. Every kind of base classifier for every cluster is now evaluated using the test examples for the specific clusters. The "winning" classifier for every cluster is then selected.

Forecasting is done by first doing a Haar wavelet transform on the given unseen example. The example is then classified into a cluster similarly to the test examples. The winning classifier for that cluster is used to do the forecast.

The mixture of expert models method may also reduce the *representational problem* (section 1.2).

3.8 Overproduce and select

This is another strategy that may produce good ensembles. Generate a set of base classifiers and employ a mechanism for selecting a subset of base classifiers to be part of the ensemble. This results in a smaller ensemble that may generalize better.

3.8.1 GASEN

Zhou et al. (2002) used this strategy in a system they call GASEN—genetic algorithm based selective ensemble. They found that an ensemble that is built out of an appropriate subset is superior to an ensemble consisting of the whole set of base classifiers.

GASEN selects the "appropriate" subset by using a heuristic. A random weight is assigned to each of the base classifiers and a genetic algorithm is employed to evolve the weights to reflect the fitness of the neural networks. Only the neural networks with weights greater than λ are included in the ensemble. The fitness is linked to a base classifier's performance on a validation set that is bootstrap sampled from the training set. Zhou et al. (2002) explains the effectiveness of GASEN by noting that it decreases both the bias and the variance component of the bias-variance decomposition (section 1.4).

3.8.2 Image classification

Giacinto and Roli (2000) used a similar approach to design neural network ensembles for image classification. They found that it is a difficult task to directly generate a diverse ensemble, and select the subset of base classifiers that are most "error independent" or diverse. Their aim is to create an ensemble with the highest possible diversity.

Any method (section 3.1.1) can be used to produce the large set of base classifiers. The subsequent phase is to select the subset of classifiers that can be effectively combined. This phase takes place during a number of iterations.

For every iteration a large set of base classifiers is created. This set is grouped into clusters by a clustering algorithm—*compound error probability* is used as the distance function by the clustering algorithm. A cluster contains neural networks that make their errors on the same data. Neural networks belonging to different clusters are independent and make their errors on different parts of the input space. A candidate ensemble $\mathbb{E}^*$ is formed by taking one neural network from every cluster formed. A neural network is selected for the candidate ensemble by measuring its average distance from all the other clusters. The neural network with the greatest average distance wins and is selected for the candidate ensemble. Finally all the candidate ensembles are compared using an independent test set and the ensemble with the best performance is selected.

3.8.3 Generating accurate and diverse members

A good ensemble is one where the base classifiers are both accurate and make their mistakes on different parts of the input space. Opitz and Shavlik (1996) developed an algorithm, ADDEMUP, that uses genetic algorithms to generate a set of neural networks that are as accurate as possible, while at the same time have minimal overlap on where they make their errors. It works very much the same as the algorithms in the previous two sections—generate many base classifiers and keep a subset that has some special qualities. ADDEMUP currently uses neural networks, but can be easily extended to other learning algorithms that understand weighted examples.

ADDEMUP (algorithm shown in figure 3.3) starts by creating and training its base population of N neural networks. It then creates new neural networks using genetic operators such of crossover and mutation. The new neural networks are then trained and examples misclassified by previous ensemble members are emphasized (by loading the back-propagation cost function with a term measuring the total population error on the example). The new neural networks are added to the ensemble and their "fitness" is measured as shown in equation 2.30 in section 2.3.7. I repeat it here:

$$\begin{aligned} F_i &= \text{Accuracy}_i + \lambda d_i \\ &= (1 - \epsilon_i) + \lambda d_i \end{aligned} \tag{3.2}$$

with ϵ_i the error rate of neural network i and λ a trade-off constant between accuracy and diversity. Diverse and accurate ensemble members will have a greater value of F_i. Finally ADDEMUP prunes the population to the N fittest neural networks.

The accuracy term, $1 - \epsilon_i$, is network i's validation set accuracy. The trade-off constant's value is adjusted as follows:

- Do not change λ when the ensemble generalization error is decreasing.
- If the population error is not increasing and the population diversity is decreasing—increase λ.
- If the population error is increasing and the population diversity is not decreasing—decrease λ.

The predictions of the neural networks in the final ensemble are combined using a weighted sum of the output of each network. The weights are based on the validation-set accuracy of the corresponding base neural networks.

ADDEMUP is similar to boosting (chapter 5) in that it emphasizes examples that are misclassified by earlier ensemble members. It is different in its approach—rather than directly trying to create an ensemble of accurate and diverse base classifiers it repeatedly creates many base classifiers and then prunes it to the N *fittest* members.

3.9 Cooperative Modular Neural Networks

Cooperative modular neural networks (CMNN) is an architecture proposed by Auda and Kamel (1998) to combat partial over- and under-training of different regions in the feature space. They propose that the current generation of non-modular neural networks suffers from the "high coupling" of the hidden nodes—resulting in slow learning and over-fitting. Some regions in the class feature-space have high overlap due to the resemblance between two or more classes while some other regions show little or no overlap. The regions with little overlap will train faster than the more complex ones. This causes what Auda and Kamel (1998) call *partial over- and under-training*. This is the problem that they are trying to solve with CMNN. The idea is to decompose the classification task into homogeneous regions of groups of classes. The neural network architecture is modularized and a separate module is assigned to every region.

```
*Ensemble generation*
**Require:** Training set $\mathbb{Z}$. The trade-off constant, $\lambda > 0$.
  Create and train an initial population of N neural networks.
  **repeat**
    Use genetic operators (i.e. crossover) to generate M new networks.
    Train the new neural networks and add them to the population.
    Measure diversity of every network and calculate its fitness.
    Prune the population to the N fittest neural networks.
    Adjust the diversity trade-off constant $\lambda$.
  **until** some stopping criteria is met
  Output the final ensemble of N neural networks.
*Classification procedure*
**Require:** An unseen example $\vec{x}$.
  Ask each of the networks for a prediction or classification. Combine
  the predictions using a weighted sum, and the classifications by taking
  a weighted vote.
```

Figure 3.3: Algorithm for ADDEMUP

First an unsupervised network is used to decompose the problem into regions and subsets of classes are assigned to smaller modules. During training every module is trained to classify its own group of classes. The structure of a module's output layer consists of two kinds of output nodes—*class* outputs and *group* outputs. The number of class outputs is equal to the number of classes in the group, and the number of group outputs is equal to the number of other modules (or the total number of modules minus one). If the training example is in one of the module's classes its class output bit is high (one) while all other outputs (both for other classes in the module and for the groups) are low. If the training example is not in one of the module's classes, the group output unit representing the valid module is high and all other outputs are low. These high and low values are never strictly ones and zeroes—high outputs approach one and low outputs approach zero. For every module these group output values are used as votes of this module for the others.

The CMNN model is in effect restructuring a non-modular network to better utilize the learning scheme's capabilities. Auda and Kamel (1998) found that, since the modules are smaller and their tasks simpler, they are more accurate than the non-modular normal neural network.

3.10 Summary

In this chapter I have described the basic techniques that are used by most ensemble methods to build and combine the ensemble members. I introduced some of the well-known ensemble methods currently available and have tried to explain why they are effective—usually in terms of the bias-variance decomposition and the fundamental problem (section 1.2).

In chapters 4 and 5 I describe the ensemble methods bagging and boosting in much more detail.

Chapter 4

Bagging

Chapter 3 introduced various ensemble methods. In this chapter I will focus on *bootstrap aggregating*—or more commonly known as bagging (Breiman (1996a)).

Bagging is a bootstrap (see section 4.3) ensemble method that creates the base classifiers by training them on random redistributions of the training examples. This is such a simple process that one might expect the base classifiers to be very similar—in where they make their mistakes and how they classify new test instances. Surprisingly this is not the case. The bagging ensemble becomes more reliable by increasing the number of base classifiers.

I discuss the bagging algorithm in section 4.1 and explain how it increases accuracy in terms of the bias-variance decomposition in section 4.2. In section 4.3 I look at the specific method that bagging employs to generate the ensemble members' training sets. A variant of bagging, the *simple ensemble*, is discussed in section 4.4. The empirical results of experiments done with bagging are presented in section 4.5.

```
Ensemble generation
Require: A labelled training set $\mathbb{Z}$ with N training examples.
  for $i = 1$ to $n$ do
    Draw N examples randomly with replacement from $\mathbb{Z}$.
    Train classifier $C_i$ on the replicate data set.
Classification procedure
Require: An example $\vec{x} \in \Theta$.
  for all classifiers $C_i$ in the ensemble $\mathbb{E}$ do
    Predict the class of the example.
  Return the class that has been predicted most often.
```

Figure 4.1: Algorithm for bagging

4.1 How does it work?

Let $\mathbb{Z} = \{(\vec{z}_j, y_j), j = 1, 2, \dots, N\}$ be a labelled set of training examples with $\vec{z}_j \in \Theta$ and $y_j \in \mathbb{L}$ (as defined in appendix A). Assume also that we have a single trained classifier $C(\vec{z})$ that can predict y with a given $\vec{z}$. The goal is to create an ensemble of classifiers that will perform better than the single C.

Denote this ensemble of classifiers by $\mathbb{E} = \{C_1, C_2, \dots, C_n\}$. Every classifier in this ensemble is trained on a replicate of the original data set $\mathbb{Z}$. The replicate data sets $\mathbb{Z}_k$ are formed by randomly taking N examples from $\mathbb{Z}$. Once an example has been selected for a replicate set it is not removed from $\mathbb{Z}$, but remains to be possibly drawn again—the examples are drawn randomly, but with replacement. Each $(\vec{z}_j, y_j)$ may appear more than once or not at all in a particular replicate data set $\mathbb{Z}_k$. The $\{\mathbb{Z}_k\}$ are replicate training sets approximating the distribution of the original $\mathbb{Z}$.

If the class labels are numerical we replace the original classifier $C(\vec{z})$ with an average output of the ensemble. If the class labels are nominal we can replace $C(\vec{z})$ by letting the C_i vote—the class with the most votes is predicted by the ensemble. The bagging algorithm is shown in figure 4.1. An example run of the process is shown in table 4.1.

It is critical that the base classifiers of the ensemble are unstable—small changes in the input data will result in large changes in the classifiers. This is reasonable and in line with our reasoning in chapter 2—if the base classifiers of the ensemble were all very similar there would not be a big improvement over the single $C(\vec{z})$. Neural networks, classification trees and regression trees were found to be unstable in an instability study (Breiman (1996a)) while k-nearest neighbour methods were stable and not suitable for bagging.

Breiman (1996a) found that most of the improvement occurred with only 10 bootstrap replicates and that more than 25 replicates were not adding a lot of value. One could also vary the size of the replicate set—Breiman (1996a) used replicate sets of the same size as the original set and

Table 4.1 Example run of bagging

Original data set ($\mathbb{Z}$)	1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15
Bagging set 1 ($\mathbb{Z}_1$)	9, 2, 7, 7, 6, 1, 14, 1, 14, 14, 12, 3, 6, 1, 9
Bagging set 2 ($\mathbb{Z}_2$)	15, 7, 3, 14, 15, 12, 3, 10, 13, 7, 1, 15, 12, 14, 14
Bagging set 3 ($\mathbb{Z}_3$)	8, 9, 14, 11, 5, 13, 8, 2, 13, 6, 15, 10, 5, 13, 1
Bagging set 4 ($\mathbb{Z}_4$)	1, 12, 11, 1, 14, 8, 8, 9, 8, 1, 2, 3, 10, 4, 4
Bagging set 5 ($\mathbb{Z}_5$)	15, 13, 14, 2, 3, 3, 6, 8, 11, 5, 1, 5, 4, 5, 10
Bagging set 6 ($\mathbb{Z}_6$)	7, 2, 7, 15, 7, 2, 10, 8, 4, 12, 4, 2, 11, 1, 13
Bagging set 7 ($\mathbb{Z}_7$)	12, 10, 14, 8, 7, 11, 9, 6, 6, 12, 7, 13, 4, 12, 8
Bagging set 8 ($\mathbb{Z}_8$)	7, 12, 6, 2, 7, 1, 11, 4, 11, 4, 4, 8, 11, 12, 4
Bagging set 9 ($\mathbb{Z}_9$)	5, 13, 7, 7, 4, 11, 2, 15, 10, 6, 12, 11, 6, 11, 9
Bagging set 10 ($\mathbb{Z}_{10}$)	5, 1, 15, 3, 2, 6, 7, 15, 3, 3, 4, 10, 2, 5, 5
Bagging set 11 ($\mathbb{Z}_{11}$)	9, 4, 1, 15, 10, 8, 9, 14, 5, 14, 7, 8, 10, 8, 13
Bagging set 12 ($\mathbb{Z}_{12}$)	2, 12, 2, 6, 14, 7, 1, 10, 7, 13, 14, 7, 2, 2, 5
Bagging set 13 ($\mathbb{Z}_{13}$)	6, 10, 7, 9, 1, 10, 12, 8, 12, 4, 14, 4, 14, 14, 2
Bagging set 14 ($\mathbb{Z}_{14}$)	14, 5, 3, 14, 9, 10, 5, 4, 15, 13, 12, 12, 3, 14, 15
Bagging set 15 ($\mathbb{Z}_{15}$)	1, 8, 13, 14, 9, 4, 11, 7, 6, 7, 11, 9, 1, 14, 11

also tried replicate sets up to twice the size of the original set. There was no improvement in accuracy.

4.2 Bias-variance decomposition

It is useful to look at the bagging process through the *bias-variance decomposition* (see section 1.4). The bias-variance decomposition decomposes the total expected error of a classifier into three components: variance, bias and intrinsic target noise.

Bagging and other similar ensemble methods attempt to decrease the total expected error by reducing the variance component. Bagging does this by trying to neutralize the inherent instability of some learning methods by simulating the process of acquiring infinite training sets.

Bagging does not decrease the bias component notably as it does not directly try to adapt the learning algorithm to misclassified examples. Note that the bias component is the "mismatch" between a learning algorithm and the problem space that it is trying to approximate.

The intrinsic target noise is a lower bound on the expected error of any learning algorithm on the specific problem—it cannot be reduced.

4.3 0.632 bootstrap

It is interesting to look at the specific method that bagging employs to generate the ensemble members' training sets—bootstrap replicates. Bootstrapping is based on the statistical method of sampling with replacement. The specific variant used by bagging is also known as the *0.632 bootstrap*.

Assume that our original dataset has N examples. Sample this dataset N times with replacement to create the replicate set. A specific example has a $\frac{1}{N}$ chance of being picked each time and a probability of $1 - \frac{1}{N}$ not being selected. The chance of a specific instance to not be picked at all is

$$\left(1 - \frac{1}{N}\right)^N \approx \frac{1}{e} \approx 0.368$$

where e is the base of the natural logarithm.

For a dataset of reasonable size the replicate set will consist of approximately 63.2 % (0.632) of the original data set.

4.4 Simple ensemble

Bagging works because of the instability of the base classifiers in the ensemble. It creates an intuitively diverse ensemble by randomly varying the training sets of the individual classifiers. This led me to believe that one could maybe create the same effect with neural networks by trying some of the following techniques instead of disturbing the training examples—keep the training set the same for all of the base classifiers.

1. Vary the base networks by initializing it with different random weights.
2. Vary the base networks by constructing it differently—number of hidden units or layers.

I found that Opitz and Maclin (1999) did exactly this—they built a *simple ensemble* by initializing the base networks with different random weights. They found that in many cases this approach was just as effective as bagging.

1. How does the accuracy of a single neural network compare to that of a bagging ensemble?
2. What is the effect of ensemble size on ensemble accuracy?
3. What is the effect of ensemble size on ensemble diversity?
4. Can I determine a relationship between the ensemble accuracy and diversity?

Figure 4.2: Bagging: interesting problems

4.5 Empirical results

There were several interesting aspects of bagging that I wanted to study. They are shown in figure 4.2. I will try to answer these questions in this section.

The results of this section were also used to compare boosting to bagging (see section 5.5).

4.5.1 Methodology

I have used standard back-propagation as the learning algorithm for all the neural networks in the experiments (both for the single networks as well as for the base classifiers in the ensembles). The following process was repeated for every data set:

1. Randomize the data set and divide it into stratified[1] 70 % training, 20 % validation and 10 % test sets.
2. I used the following rule of thumb to determine the number of hidden units in every neural network: 1 hidden unit for every 10 input units and 1 hidden unit for every output unit. The number of epochs was determined based upon the learning rate chosen, number of examples and the number of hidden units. I then trained the neural network on the training set and used the validation set to tune the architecture and parameters. I never used the test set for the training or tuning of parameters. The architecture and parameters of the neural networks for every data set were then fixed for all the experiments (single networks and ensembles). The data set details and neural network architectures are shown in table 4.2.

[1]The distribution of examples in the training, test and validation sets are kept close to that of the original set.

Table 4.2 Data sets used for empirical analysis

	cases	classes	discrete	real	inputs	hidden	outputs	epochs	λ
soybean	683	19	35	0	84	23	19	30	0.3
breast-cancer	286	2	9	0	48	6	2	60	0.15
iris	150	3	0	4	4	4	3	80	0.05
balance-scale	625	3	0	4	4	5	3	60	0.3
heart-c	303	5	7	6	23	7	5	100	0.15
heart-h	294	5	7	6	23	7	5	100	0.15
lymph	148	4	16	3	38	7	4	50	0.1
mushroom	8124	2	22	0	125	14	2	20	0.3
hepatitis	155	2	13	6	23	4	2	80	0.15
horse-colic	348	3	15	7	58	9	3	40	0.15
labor	57	2	8	8	26	10	2	10	0.3

3. I used 10-fold *stratified cross-validation* to get a good estimate of the true error rate (section 1.3). The original data set was randomized and divided into 10 stratified "folds" or disjoint sets. I then removed the first fold, kept it separate, and trained the neural network on the 9 remaining folds. The fold that I removed was used as a test set to determine the error rate for this iteration. This process was repeated 10 times and every time a different fold was used as the test set. An average of the error rates over the 10 folds was then taken.

4. I set up 29 ensembles for every data set with ensemble size varying from 2 to 30 base neural networks. The connection weights for all the neural networks in every ensemble for a specific data set were the same. I used the same random seed for every network to get the same random connection weights. 10-fold cross-validation was also used to determine the accuracy of every ensemble[2]. The diversity measures were calculated on the test set fold for every one of the 10 iterations of the 10-fold cross-validation and then averaged.

4.5.2 Accuracy

Table 4.3 shows the single neural network error rates for all the data sets described in table 4.2. The single neural network achieved a perfect score on the mushroom dataset and I therefore did not use it in further experiments with bagging and boosting.

Figures 4.4 to 4.13 show the performance of bagging as a percentage in relation to the error rate of the single neural networks. Bagging showed an improvement on the single neural network in all but 3 of the 10 data

[2]This resulted in a lot of training: the balance-scale data set had a total of 174000000 back-propagation cycles!

Table 4.3 Error rates for single neural networks

soybean	8.5
breast-cancer	27.6
iris	4.7
balance-scale	5.4
heart-c	16.8
heart-h	20.7
lymph	16.9
mushroom	0
hepatitis	17.4
horse-colic	15.5
labor	8.8

sets used. It did remarkably well on the hepatitis dataset—an improvement of 20 % (figure 4.11). It is interesting to note that the improvement starts levelling off when more than 10 to 15 classifiers are in the ensemble. I give a possible explanation for this in the next section.

The three data sets that bagging could not improve on were breast-cancer, balance-scale and labor. The single neural network performance on balance-scale was already quite good and may have been approaching the *Bayes optimal classifier* performance—i.e. nothing could do better. Labor is a very small data set (only 57 instances) and I had difficulty training the single neural network on it—it over-trained very easily. The single network was trained for only 10 epochs.

The breast-cancer set was difficult for the single neural network, bagging and boosting (see figure 5.10).

4.5.3 Why does the accuracy improvement level off?

It is interesting to note that the accuracy improvement by bagging starts levelling off when more than 10 to 15 classifiers are in the ensemble. This can be explained by means of the *0.632 bootstrap* (section 4.3).

The 0.632 bootstrap method draws a new random replicate set of the original data set. A specific example thus has an approximate probability of 0.368 of not being picked for the first set. The probability of not being picked for two sets is $(0.368)^2 = 0.135$. Table 4.4 shows how quickly this probability gets less for 3 to 7 sets. It is highly unlikely that there are any examples that have not being trained on in an ensemble of seven neural networks.

I also set up an experiment to see how many examples are not selected by bagging on two real data sets—the results are shown in figure 4.3. One can expect that bagging cannot learn anything *new* when there are more than about seven classifiers in the ensemble. The other reason why it is

Table 4.4 Probabilities of an example not being picked by bagging

1	0.368
2	0.135
3	0.05
4	0.018
5	0.007
6	0.002
7	0.0009

still improving the accuracy until about 15 classifiers is related to the three fundamental reasons explained in section 1.2. Bagging uses the inherent instability present in some learning methods to decrease the computational problem by starting the search in different parts of the possible hypothesis space.

4.5.4 Diversity

All the diversity measures were calculated for every ensemble. The Q and ρ measurements are missing on some of the graphs—I found that their values were frequently undefined. Q, for example, frequently failed because of division by zero.

I have repeated table 4.5 from chapter 2. It shows all the diversity measures, whether it is a measure of diversity or similarity and its possible range of values.

Pairwise measures

- Q indicated high similarity (low diversity) for all the data sets.
- ρ also indicated high similarity for all the data sets that it was defined for. It got slightly worse (less diverse) with more networks being added to the ensembles.
- The disagreement measure, Dis, showed low diversity for all the data sets. For most of the data sets its value was below 0.15 and it did not change much as the ensembles got bigger.
- DF, the double-fault measure, interestingly, indicated low similarity for all the data sets. DF measures how often classifiers make the same mistakes. The DF measure proved to be rather useless.

The only data set with more diverse ensembles, as indicated by the pairwise measures, was breast-cancer. Breast-cancer was also the set with the highest error rates. This is not a good sign for our hypothesis that diversity might be

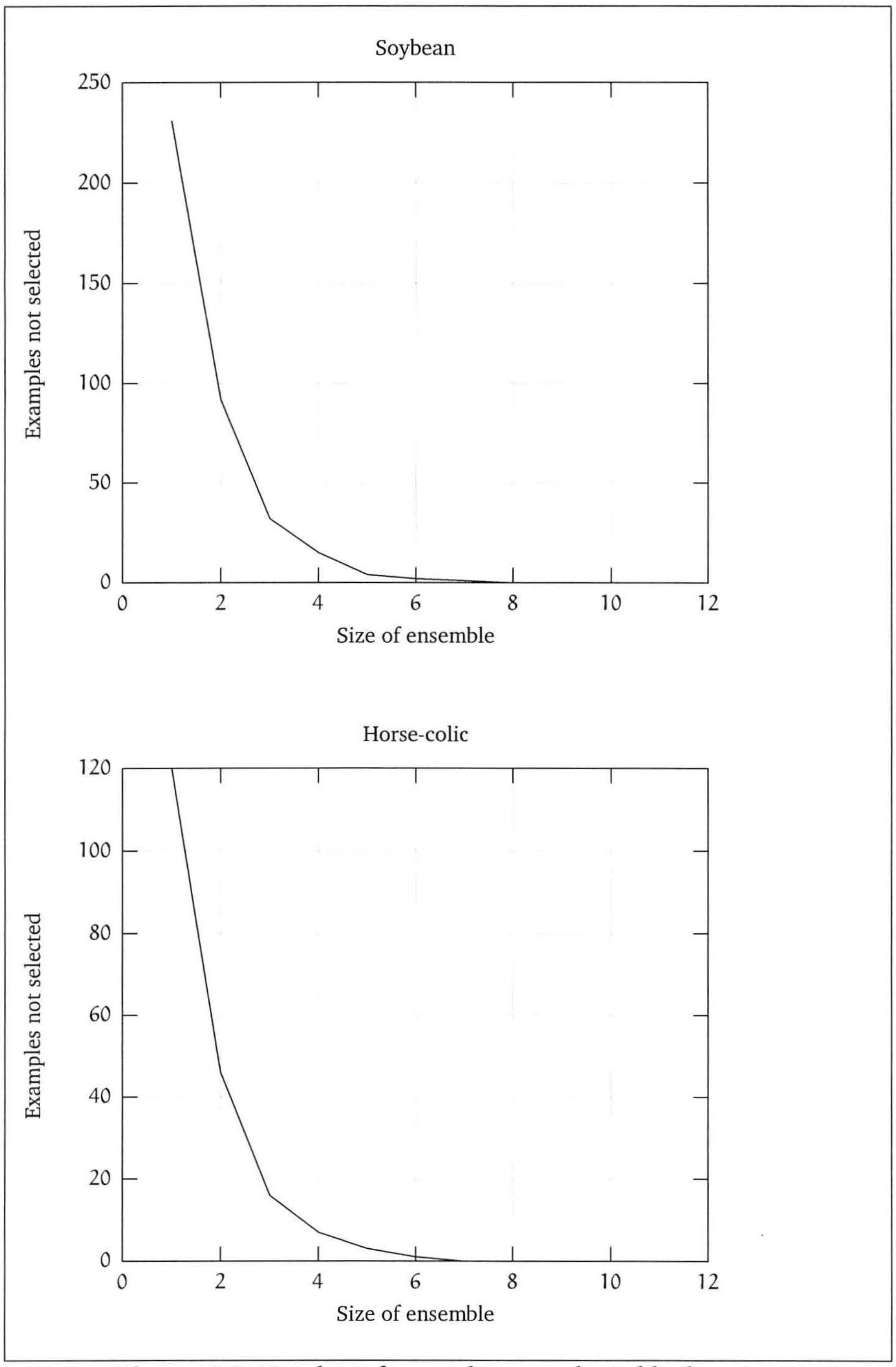


Figure 4.3: Number of examples not selected by bagging

Table 4.5 Summary of diversity measures. An up-arrow (↑) specifies that it is a measure of diversity. A down-arrow (↓) specifies that it is a measure of similarity.

Pairwise			
Q Statistics	Q	↓	$-1 \leq Q \leq 1$
Correlation coefficient	ρ	↓	$-1 \leq \rho \leq 1$
Disagreement	Dis	↑	$0 \leq \text{Dis} \leq 1$
Double-fault	DF	↓	$0 \leq \text{DF} \leq 1$
Non-pairwise			
Entropy	E	↑	$0 \leq E \leq 1$
Kohavi-Wolpert variance	KW	↑	$0 \leq \text{KW} \leq 1/4$
Interrater agreement	κ	↓	
Difficulty	θ	↓	$\theta \geq 0$
Generalized diversity	GD	↑	$0 \leq \text{GD} \leq 1$
Coincident failure diversity	CFD	↑	$0 \leq \text{CFD} \leq 1$
Fitness	F	↑	$F \geq 0$

related to ensemble accuracy. In this case the higher error rates might also have influenced the diversity scores.

Non-pairwise measures

- The entropy measure, E, measured low diversity for all the data sets.
- KW indicated low diversity for all the data sets and did not change much as the ensembles got bigger.
- κ indicated high similarity (low diversity). It was interesting to note the κ and ρ was very similar.
- The difficulty measure, θ, showed low similarity (high diversity) for all the data sets. θ is based on the distribution of difficulty—i.e. in this case all examples were equally difficult for all the classifiers.
- GD indicated low to medium diversity with the ensembles getting less diverse as they grew bigger.
- CFD indicated medium diversity for all the data sets.

The non-pairwise measures all indicated low (and some medium) diversity for all the data sets. The only exception was θ. Similarly to the pairwise measures, the non-pairwise measures also indicated higher (but not *high*) diversity for the breast-cancer data set.

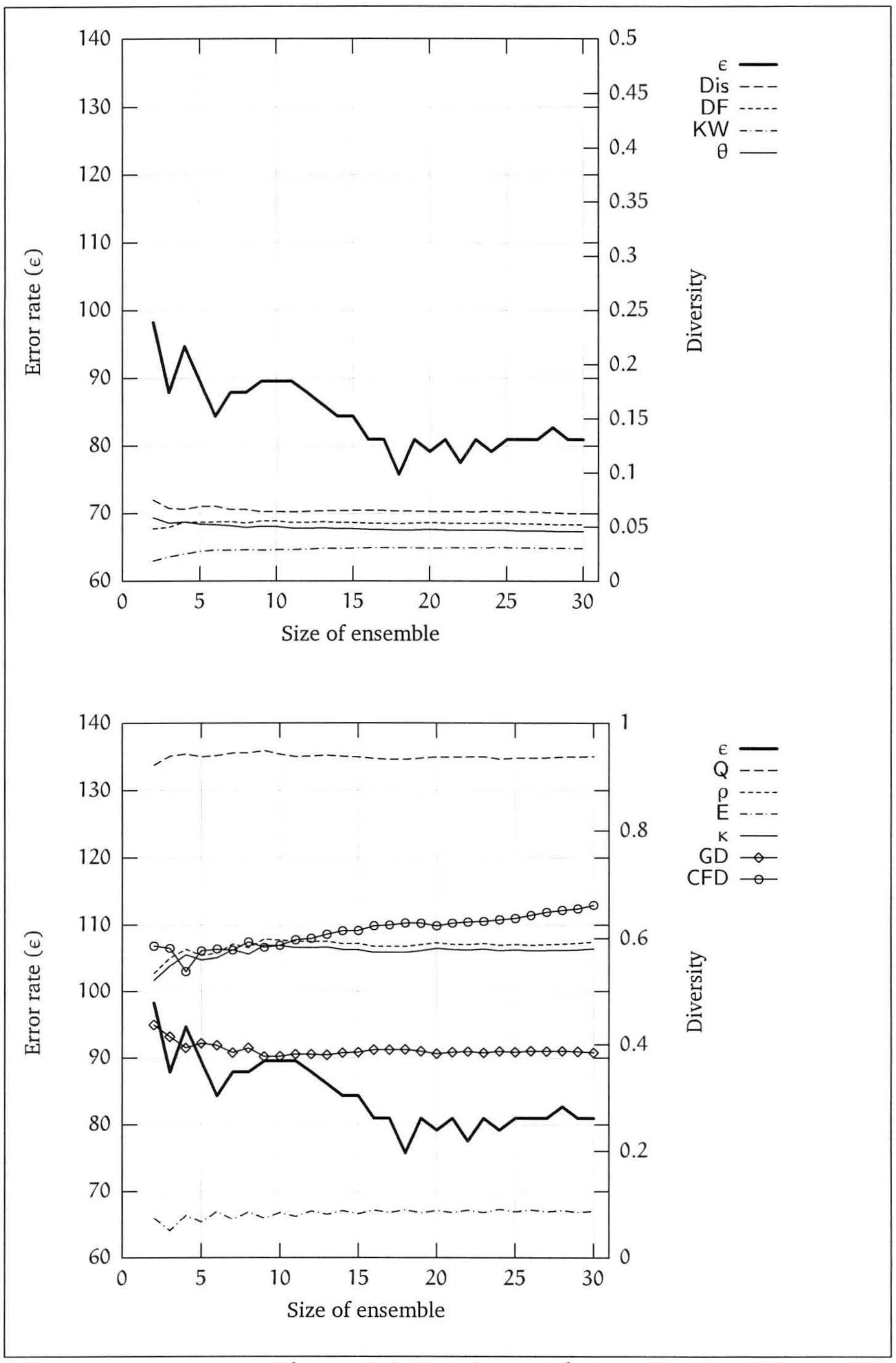


Figure 4.4: Bagging: soybean

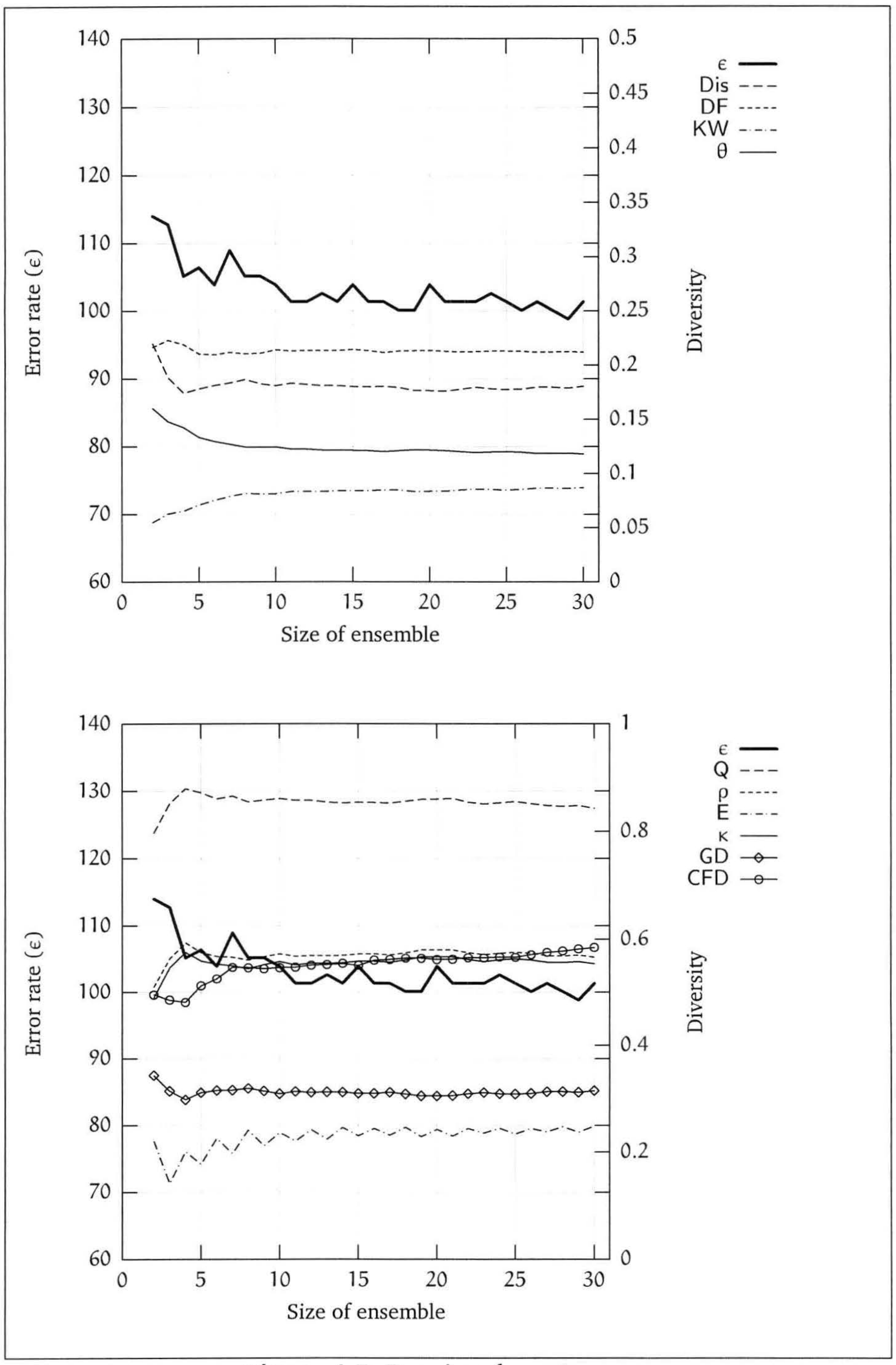


Figure 4.5: Bagging: breast-cancer

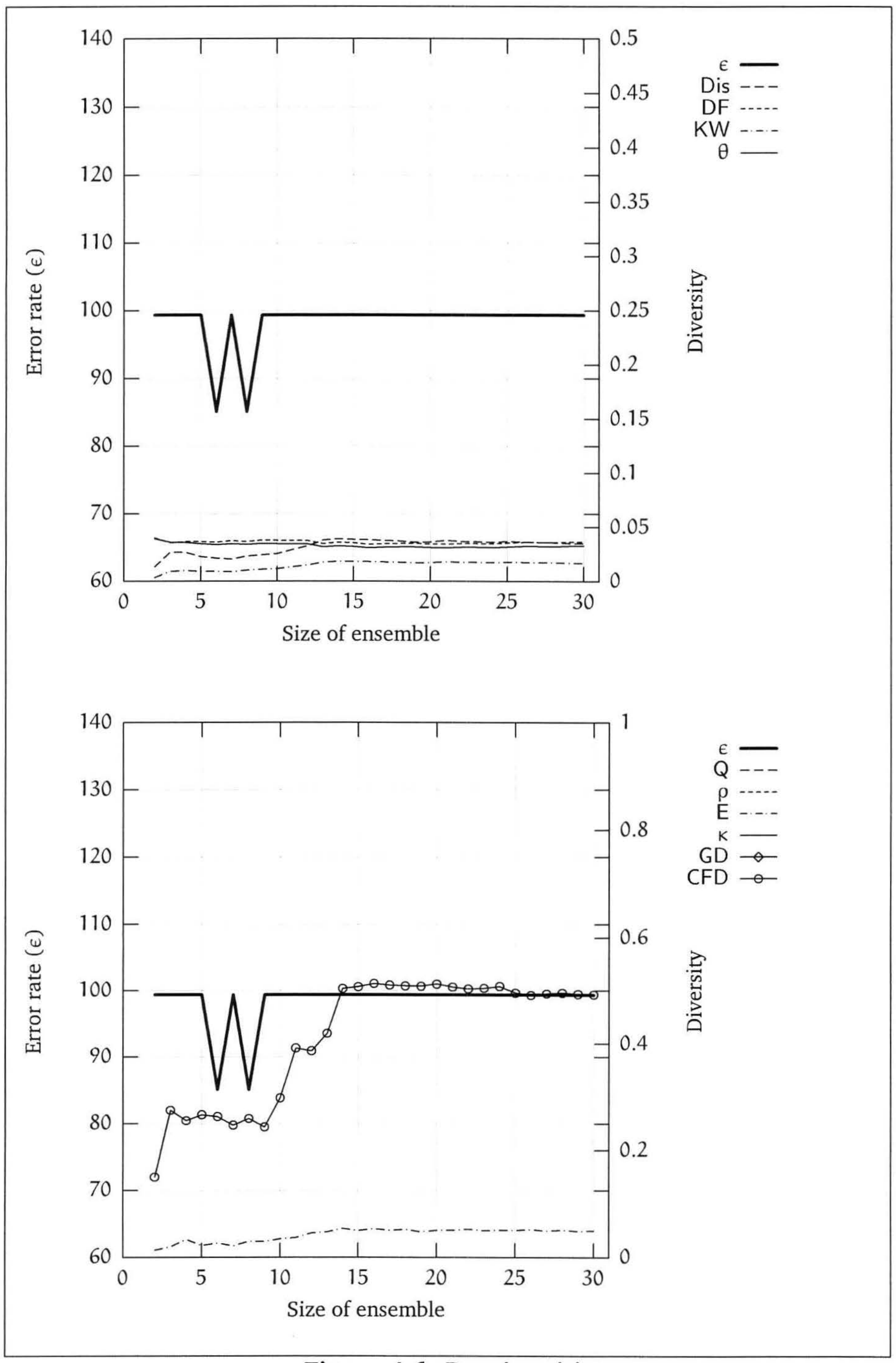


Figure 4.6: Bagging: iris

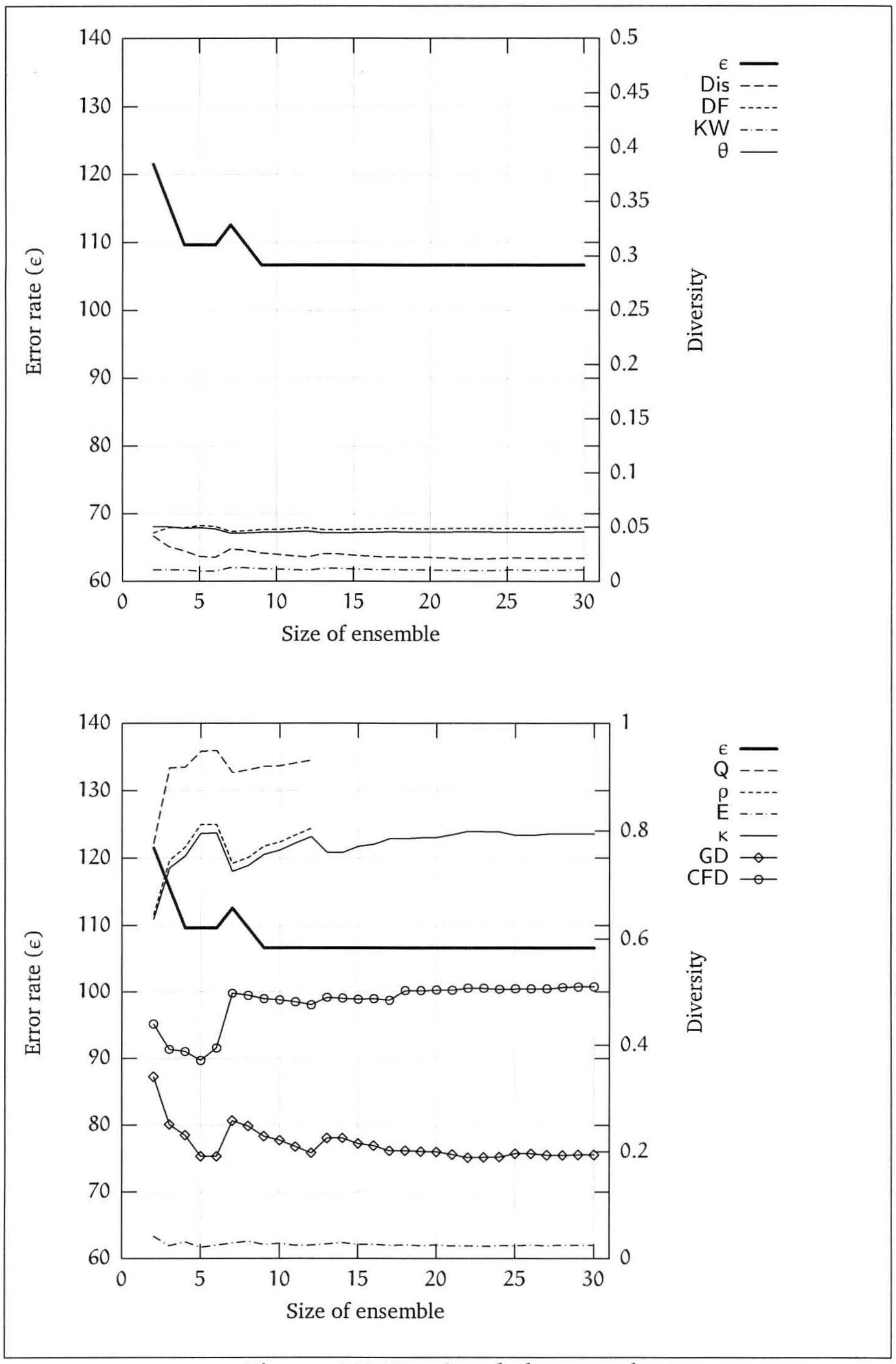


Figure 4.7: Bagging: balance-scale

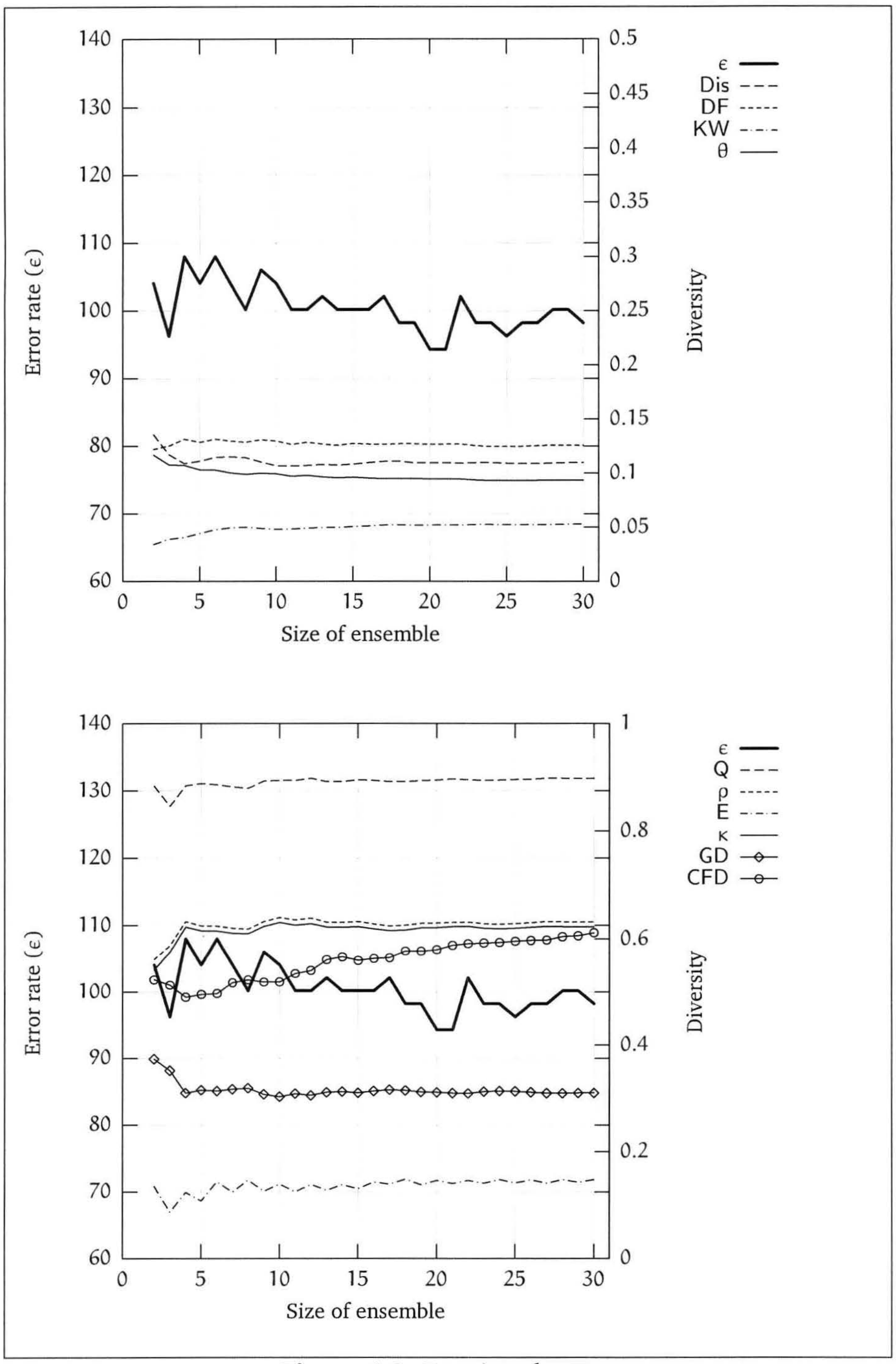


Figure 4.8: Bagging: heart-c

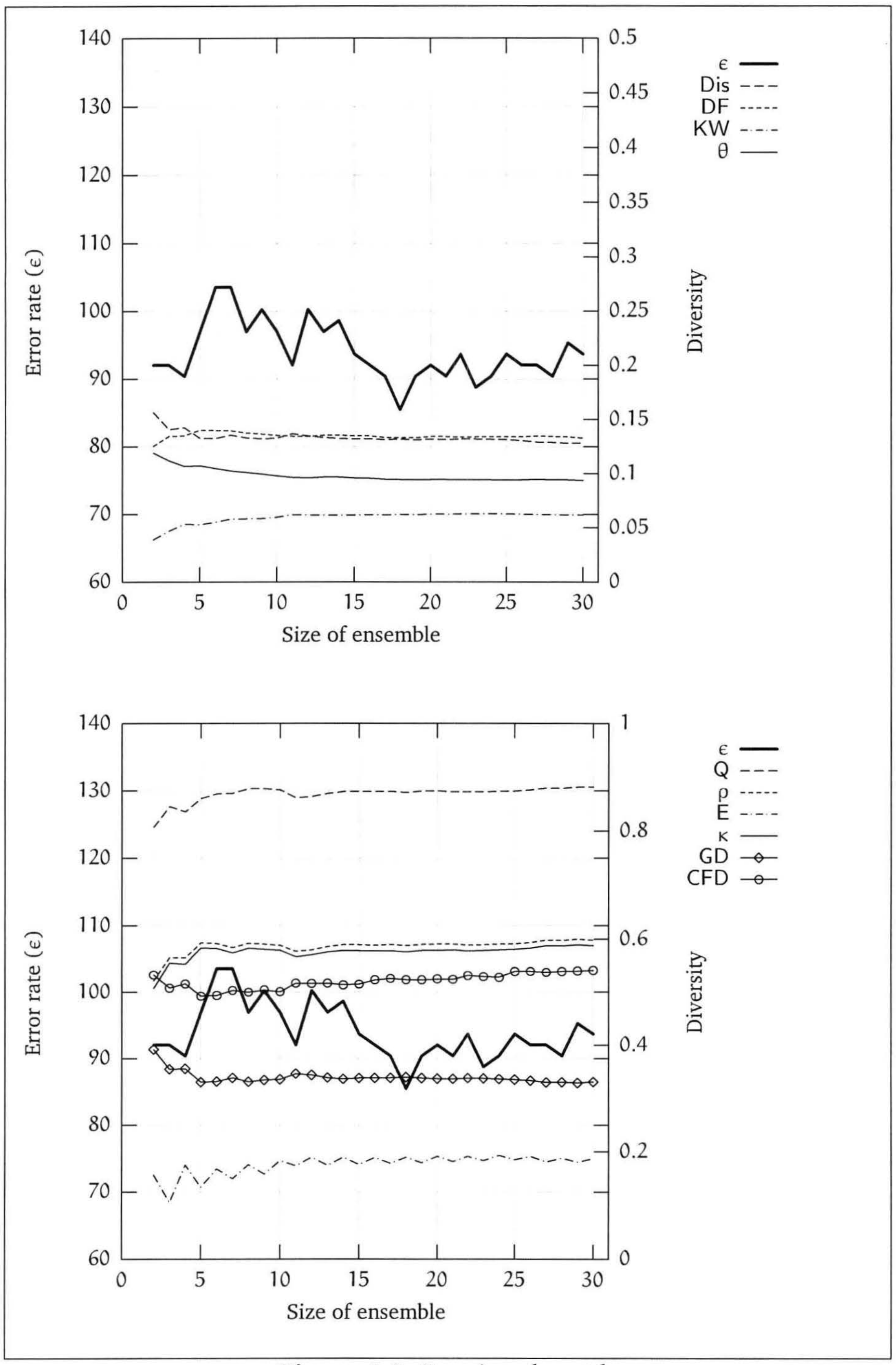


Figure 4.9: Bagging: heart-h

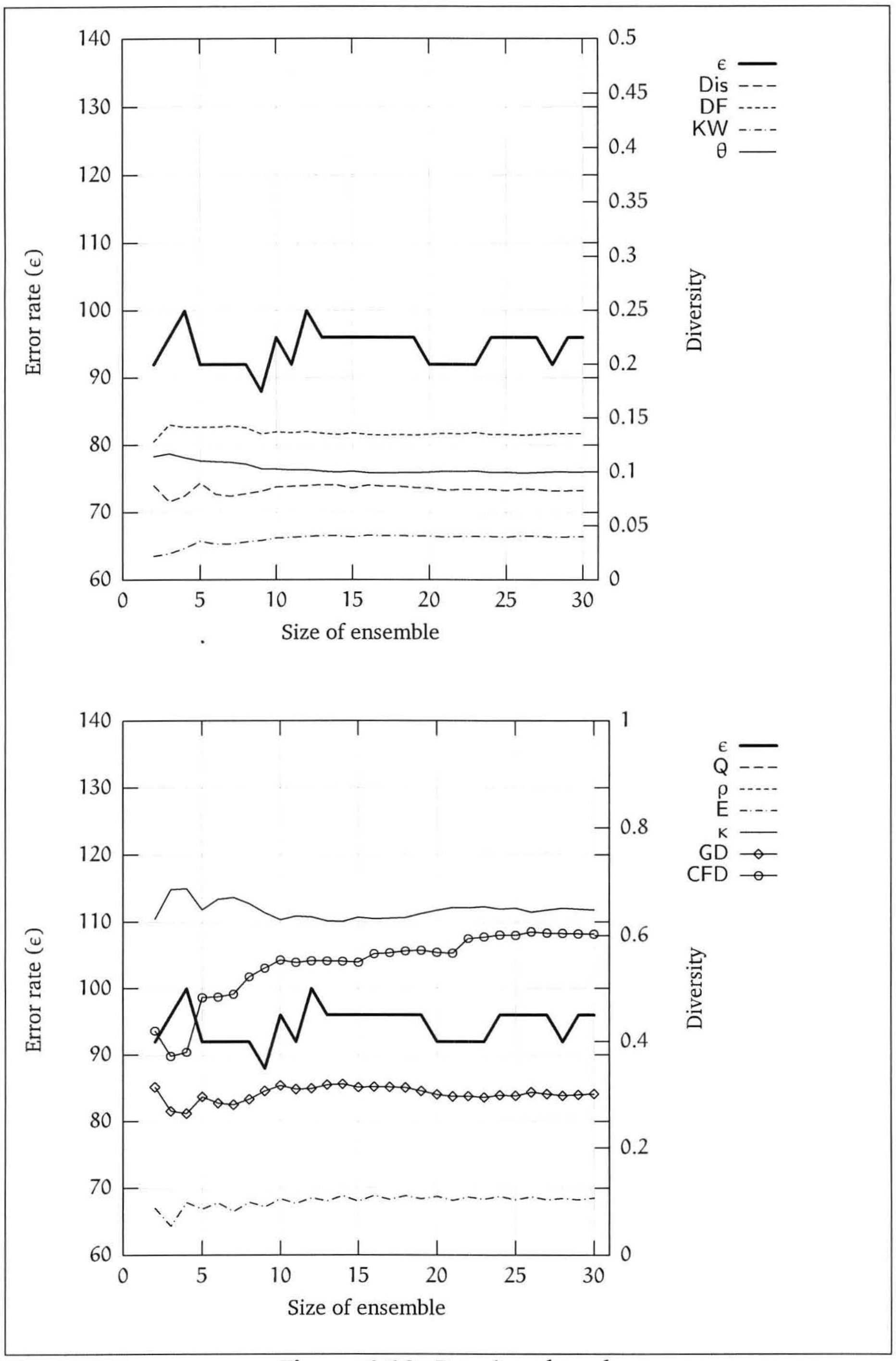


Figure 4.10: Bagging: lymph

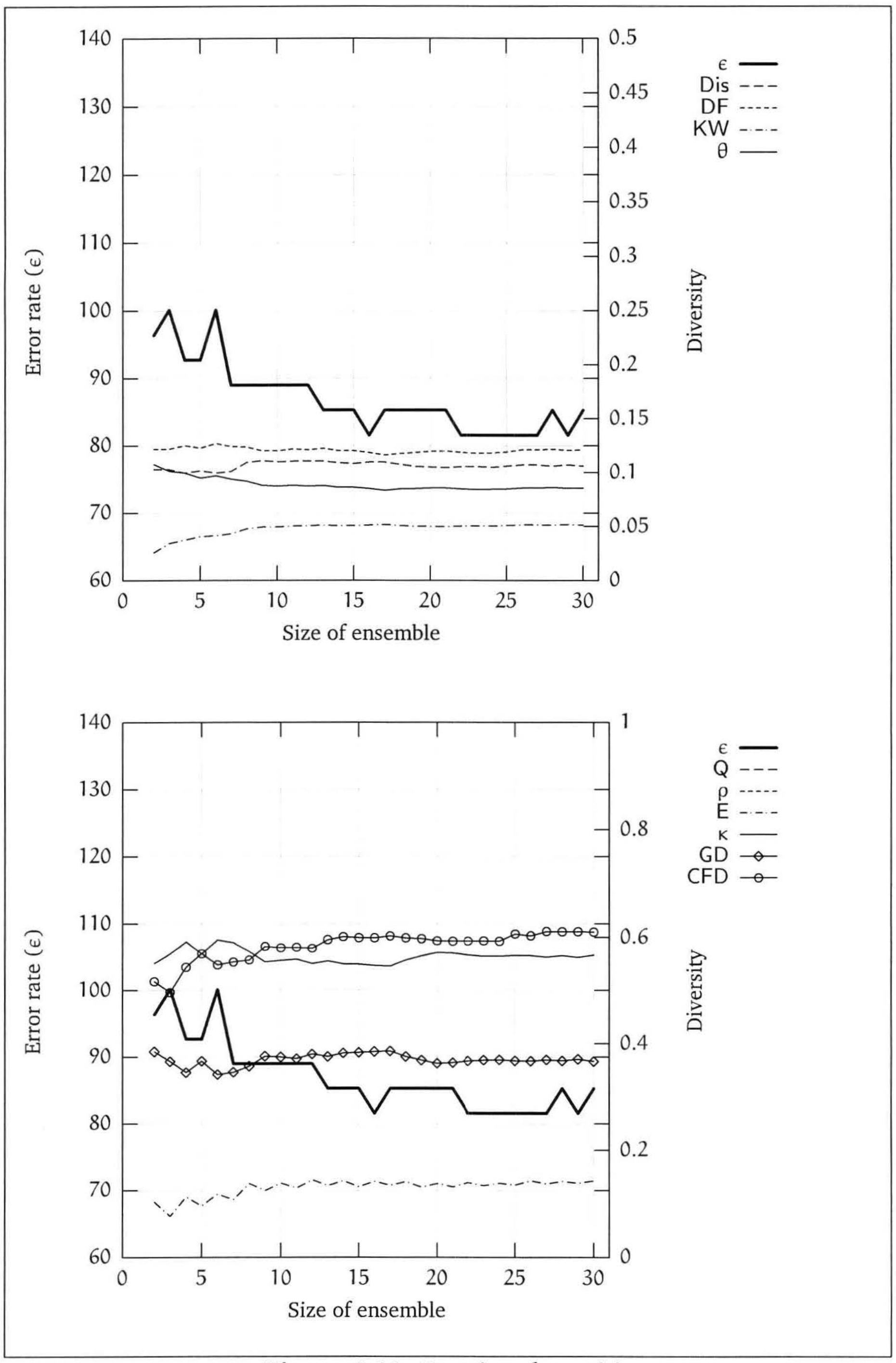


Figure 4.11: Bagging: hepatitis

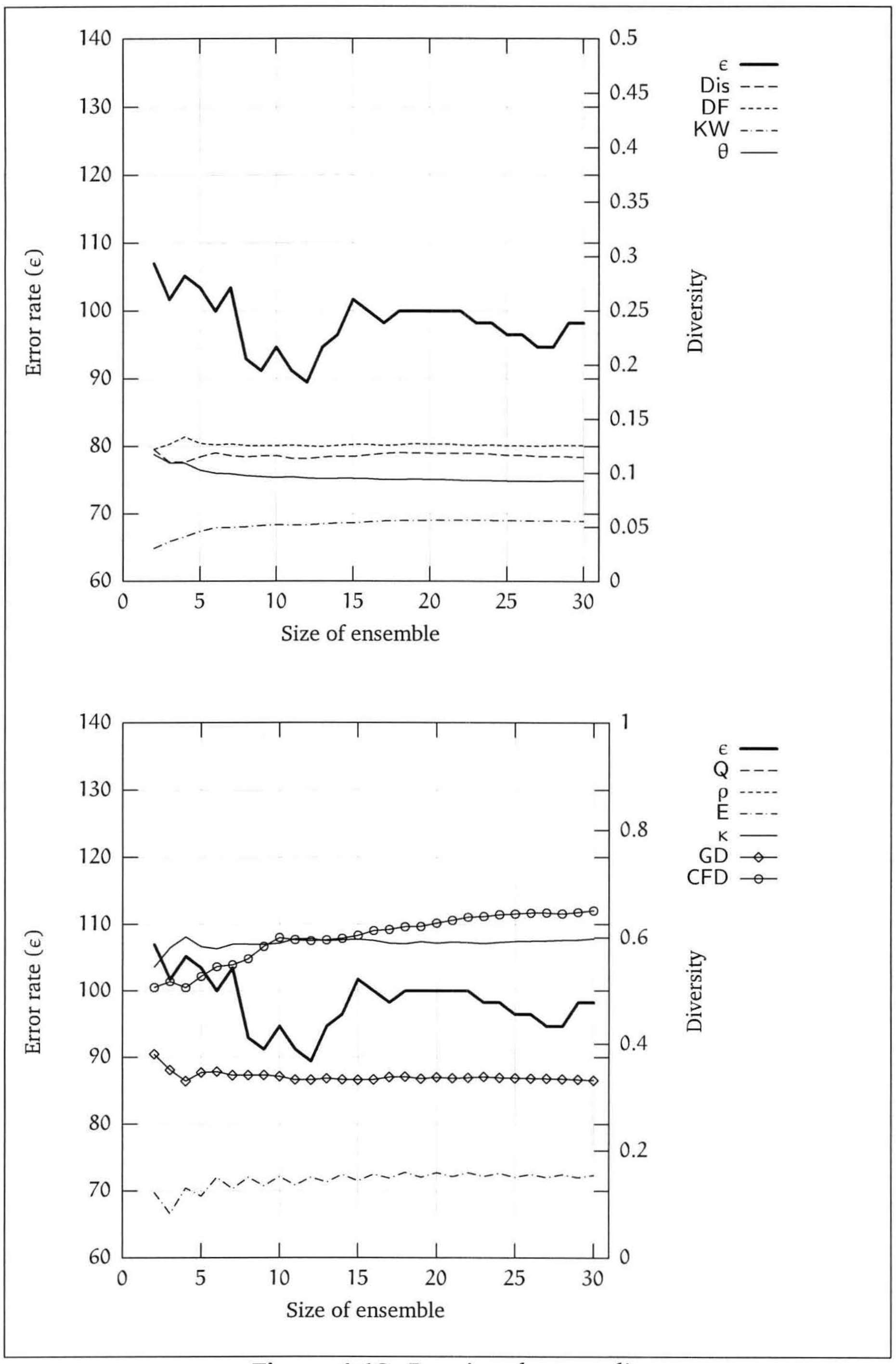


Figure 4.12: Bagging: horse-colic

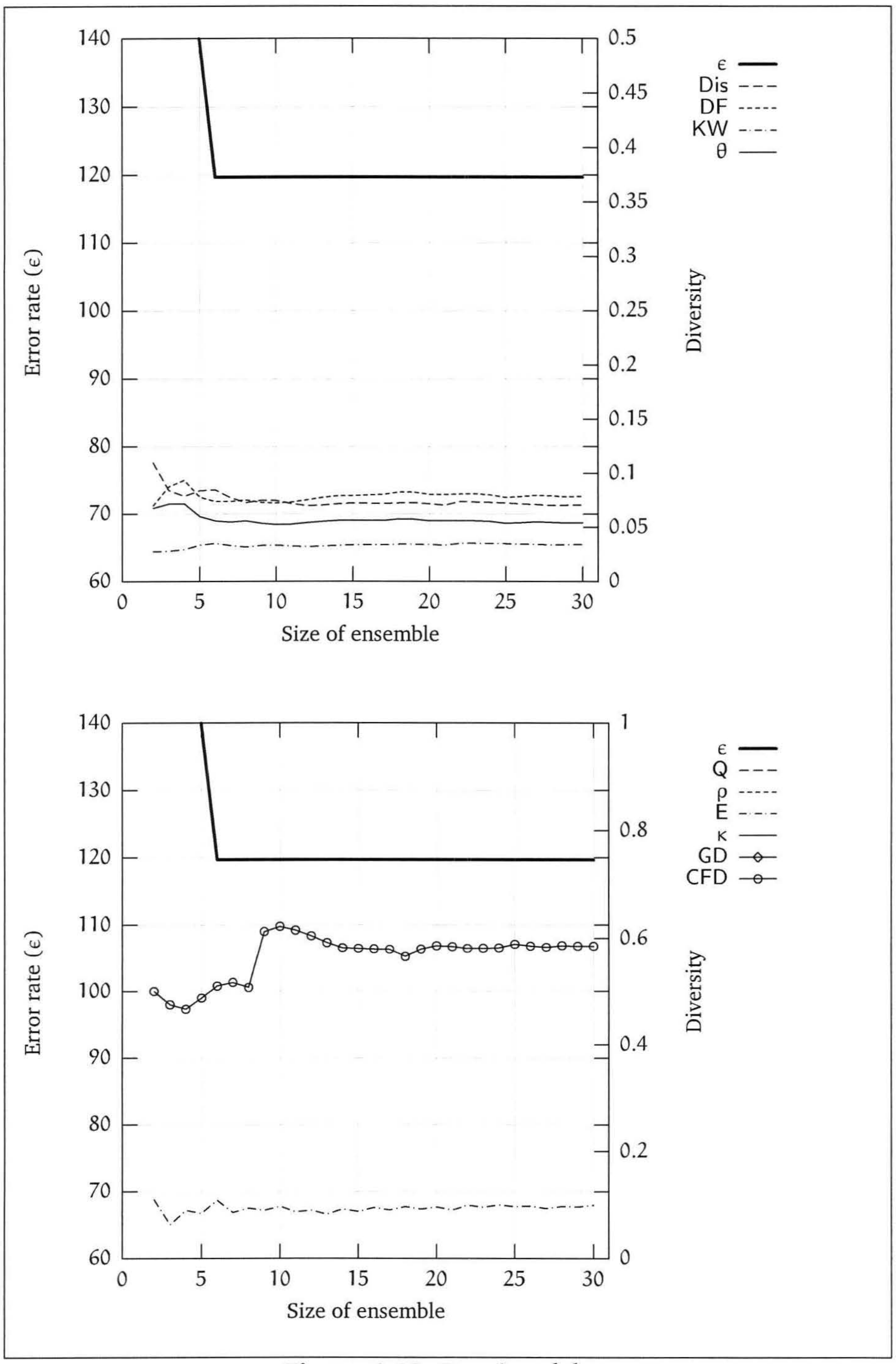


Figure 4.13: Bagging: labor

4.6 Summary

In this chapter I looked in detail at the process of bootstrap aggregating. I explained how it works, looked at reasons why it works and explained a variation of the idea. In section 4.5 I presented my empirical results. I answered the questions asked in figure 4.2:

1. How does the accuracy of a single neural network compare to that of a bagging ensemble? The bagging ensembles were more accurate on 7 occasions.
2. What is the effect of ensemble size on ensemble accuracy? The bagging ensembles' accuracy improvements started levelling off after 10 to 15 members were added to the ensembles.
3. What is the effect of ensemble size on ensemble diversity?
4. Can I determine a relationship between the ensemble accuracy and diversity?

The two last questions asked had rather disappointing answers (at least for the process of bagging ensembles)—I could not determine a relationship between ensemble size and diversity, nor could I see a clear relationship between the ensemble accuracy and diversity. In fact, some of the very accurate ensembles had low diversity scores.

In the next chapter I discuss the process of boosting and try to answer similar questions.

Chapter 5

Boosting

Boosting (Schapire (1990)) describes a family of learning methods. These methods produce a series of classifiers and are a general method for improving or *boosting* the accuracy of a given weak learning algorithm. A powerful ensemble can be built from very simple classifiers as long as they make less than 50 % errors.

Boosting is similar to bagging in that they both rely on feeding perturbed versions of the training examples to the base classifiers. Bagging follows an indirect approach by creating random and independent training sets. Boosting uses a direct approach in that the new training sets are created *deterministically* and *serially* where new models are influenced by the accuracy of previous ones. The new models become experts on examples incorrectly classified by previous models. I will explain this in section 5.2.

Bagging generates diverse classifiers only if the base learning algorithm is unstable. Boosting requires less instability in the base classifiers since it can make much larger changes in the training set by placing large weights on only a few of the examples.

Boosting is particularly susceptible to noise in the training data, unlike bagging—see section 5.3. The bias-variance decomposition is used in section 5.4 to explain boosting's success. In section 5.5 I present my experimental results with boosting and its variants.

1989	Michael Kearns, Leslie Valiant: the hypothesis boosting problem
1990	Robert Schapire provides the solution: Boost1
	Yoav Freund improves Boost1 and calls it boost-by-majority
1996	Robert Schapire, Yoav Freund: adaptive boosting
	Leo Breiman introduces arc-x4
1998	Jerome Friedman, Trevor Hastie, Robert Tibshirani: LogitBoost

Figure 5.1: Short history of boosting

5.1 Background

The idea of boosting arrived out of the *probably approximately correct* (PAC) or distribution-free learning model. It was developed as an answer to a theoretical question asked by Michael Kearns and Leslie Valiant within the PAC learning literature in 1989 (the question was termed the *hypothesis boosting problem*). They asked (paraphrased from Freund and Schapire (1998)):

> Suppose we have a computationally efficient learning algorithm that can generate a hypothesis which is slightly better than random guessing for any distribution over the inputs. Does the existence of such a *weak*[1] learning algorithm imply the existence of an efficient *strong*[2] learning algorithm that can generate arbitrarily accurate hypotheses?

Robert Schapire's answer (Schapire (1990)) to this question was: yes! His proof also described an efficient algorithm (Boost1, figure 5.2) that can transform any efficient weak algorithm into an efficient strong one. A year later Yoav Freund developed a more efficient and simpler algorithm called boost-by-majority (Freund (1990)). This still suffered from some practical drawbacks (Schapire (1999)). Freund and Schapire more recently described adaptive boosting, or AdaBoost (Freund and Schapire (1996)), solving many of the practical difficulties of the earlier boosting algorithms. In the same year Leo Breiman introduced arc-x4 (Breiman (1996b)), another variation of adaptive boosting. It is interesting to note that Breiman refers to boosting as Adaptive Resampling and Combining (Arcing). Friedman et al. (1998) made some minor modifications to AdaBoost to improve performance and called it *LogitBoost*.

[1]Weak learning algorithms only have to produce a classifier that performs slightly better than random guessing.

[2]Strong learning algorithms have to produce classifiers that are arbitrarily accurate given access to a source of training examples of the unknown distribution.

Table 5.1 Example run of boosting. This shows the case where the learning algorithm cannot handle weighted instances and with every iteration of boosting the hard examples have a higher probability of being sampled. Example 5 is a hard example.

Original data set ($\mathbb{Z}$)	$1, 2, 3, 4, 5$
Boosting iteration 1 ($\mathbb{Z}_1$)	$3, 2, 1, 4, 5$
Boosting iteration 2 ($\mathbb{Z}_2$)	$4, 2, 4, 5, 1$
Boosting iteration 3 ($\mathbb{Z}_3$)	$5, 2, 3, 1, 5$
Boosting iteration 4 ($\mathbb{Z}_4$)	$5, 3, 5, 5, 2$

5.2 How does it work?

Boosting algorithms can be organized into two categories.

Boosting by filtering concentrates on difficult examples by using previous members of the ensemble to organize the learning examples into easy and hard patterns. Train a first classifier normally. Pass all the training examples through this classifier and train the second classifier on the examples that consist of equal parts those that are classified correctly by the first classifier and those that are classified incorrectly. Train the third member of the ensemble on the examples that the first two classifiers disagree on.

Adaptive boosting picks the first set of examples randomly with replacement from the set of all the available training examples—in a similar way to bagging. Every iteration of the algorithm results in the downward adjustment of the probability of being picked of the correctly classified examples. In the later iterations the learning algorithm concentrates more on the hard examples.

5.2.1 Boost1

Boost1 falls into the boosting by filtering category. This is the first boosting algorithm developed by Schapire (1990) and it assumed that there was a large number of training examples available for which the class labels were known. The boost1 algorithm is shown in figure 5.2. C_1, C_2 and C_3 are three classifiers with individual error rates $\epsilon < 0.5$. Schapire proved that the resulting ensemble generated by boost1 will have an error rate of $3\epsilon^2 - 2\epsilon^3$. It is possible then to take boost1 and apply it iteratively to achieve an arbitrarily low error.

Boost1 forces C_2 and C_3 to concentrate on the difficult training examples. The training examples used for C_2 have a 50 % error rate with C_1. The training examples used for C_3 are those where C_1 and C_2 have different classifications.

```
Ensemble generation
Require: A large labelled training set $\mathbb{Z} = \{(\vec{z}_j, y_j), j = 1, \dots, N\}$ and a weak
  learning algorithm WeakLearn.
  Train $C_1$ on N randomly chosen examples from the training set $\mathbb{Z}$ using
  WeakLearn.
  repeat
    Flip a coin.
    if it is heads then
      Add the first randomly selected $\vec{z}_j \in \mathbb{Z}$ that is correctly classified by
      $C_1$ to the training set $\mathbb{Z}_1$.
    else
      Add the first randomly selected $\vec{z}_j \in \mathbb{Z}$ that is incorrectly classified by
      $C_1$ to the training set $\mathbb{Z}_1$.
  until there is N examples in $\mathbb{Z}_1$
  Train $C_2$ with the examples in $\mathbb{Z}_1$ using WeakLearn.
  repeat
    Take a random $\vec{z}_j \in \mathbb{Z}$.
    if $C_1$ and $C_2$ disagree on the classification then
      add $\vec{z}_j$ to the training set $\mathbb{Z}_2$.
  until there is N examples
  Train $C_3$ with the examples in $\mathbb{Z}_2$ using WeakLearn.
Classification procedure
Require: An example $\vec{x} \in \Theta$.
  if $C_1(\vec{x}) = C_2(\vec{x})$ then
    return $C_1(\vec{x})$
  else
    return $C_3(\vec{x})$
```

Figure 5.2: Algorithm for boost1

5.2.2 AdaBoost.M1

AdaBoost.M1 (Freund and Schapire (1996)) is widely used and specifically designed for classification. It is an extension of the original adaptive boosting algorithm to the multiclass case. Similarly to bagging it can be applied to any base classification algorithm. One of the main ideas of the algorithm is to keep a distribution of weights over the training set. The weight of the distribution on training example $\vec{z}_j$ on round i is defined as $D_i(j)$.

There are two ways (Dietterich (1999)) that AdaBoost can use this distribution of weights to construct new training sets to give to the base algorithm. If the base algorithm understands weighted examples one can use *boosting by weighting*. The entire training set with associated weight distribution is given to the base learning algorithm. If the base algorithm does not understand weighted examples one can use *boosting by sampling*—examples are drawn from with replacement from the original training set $\mathbb{Z}$ with probability proportional to their weights.

The presence of training example weights also changes the way in which a classifier's error is calculated. Instead of being the fraction of examples that is misclassified, it is the sum of the weights of the misclassified instances divided by the total weight of all the instances.

The algorithm (figure 5.3) begins by assigning equal weights to every training example. On each of the n iterations it calls the weak base learning algorithm—WeakLearn. The weights of the incorrectly classified training examples are increased by a factor related to the error rate ϵ_i of classifier C_i. Correctly classified examples' weights are not adjusted. The weights distribution is then normalized—resulting in the weights of the "easy" examples being decreased. In the next iteration a classifier is then built on the new reweighted training examples which focuses on classifying the difficult examples correctly. Then the examples' weights are again increased if they were classified incorrectly. After each iteration the weights distribution for the next iteration tells us how often training examples have been misclassified by previous classifiers. This procedure is an elegant way of generating a series of classifiers that complement each other. Another advantage of this algorithm is that one does not have to assume an oracle with an unlimited amount of training examples— instead we can continually recycle the training examples to drive the training set error rate to 0 (explained in section 5.2.3).

Classifying new instances are by way of a weighted vote. The error rate of a classifier for the training examples is used for determining the confidence that we should have in its vote. A weight of $-\log(\frac{\epsilon_i}{1-\epsilon_i})$ is used for weighing the output of classifier C_i.

There are various other extensions to adaptive boosting:

AdaBoost.M2 uses the concept of *pseudoloss* instead of error rate. AdaBoost.M1 may fail when there are more than two classes. In this case AdaBoost.M2 may be more suitable for the problem space. AdaBoost.M2 modifies the weight distribution not only by looking at whether the classifier correctly or incorrectly classified a training example, but also by looking at the *confidence* of the labelling.

AdaBoost.MH is also a solution for the multiclass problem. It works by creating a set of binary problems.

AdaBoost.R can be used for regression.

5.2.3 Training set error for adaptive boosting

The most important theoretical property of adaptive boosting is its ability to reduce the training set error (Schapire (1999)). Write the error of classifier

```
Ensemble generation
Require: A labelled training set $\mathbb{Z} = \{(\vec{z}_j, y_j), j = 1, \ldots, N\}$ with N examples
  and a weak learning algorithm WeakLearn.
  Assign an equal weight $\frac{1}{N}$ to every training example. This is the same as
  setting $D_1(j) = \frac{1}{N}$ for every example $\vec{z}_j$.
  for iterations $i = 1$ to $n$ do
    Train classifier $C_i$ on $\mathbb{Z}$ with weights $D_i$.
    Calculate the error $\epsilon_i$ made by this classifier.
    if $\epsilon_i = 0$ or $\epsilon_i \geq 0.5$ then
      Terminate model generation.
    for all examples $\vec{z}_j \in \mathbb{Z}$ do
      if $\vec{z}_j$ has been correctly classified by $C_i$ then
        Multiply its weight by $\frac{\epsilon_i}{1-\epsilon_i}$. Again, this is the same as setting
        $D_{i+1}(j) = D_i(j)\frac{\epsilon_i}{1-\epsilon_i}$.
    Normalize weights of all instances.
Classification procedure
Require: An example $\vec{x} \in \Theta$.
  Assign a weight of 0 to all possible classes.
  for each of the generated classifiers in the ensemble do
    Add $-\log(\frac{\epsilon_i}{1-\epsilon_i})$ to the weight of the class predicted by the model $C_i$.
  Return the class with the highest weight.
```

Figure 5.3: Algorithm for AdaBoost.M1

C_i as

$$\epsilon_i = \frac{1}{2} - \gamma_i \tag{5.1}$$

A classifier that guesses each example's class at random has an error rate of $\frac{1}{2}$ on binary problems. γ_i measures how much classifier C_i is better than random. Freund and Schapire (1996) proved the following for the training set error $\epsilon_{\text{combined}}$ of the combined classifier:

$$\epsilon_{\text{combined}} \leq \exp\left(-2\sum_i \gamma_i^2\right) \tag{5.2}$$

If every weak classifier is only slightly better than random so that $\gamma_i \geq \gamma$ for some $\gamma > 0$ then the training set error will drop exponentially fast.

AdaBoost.M1 guarantees that the *training error* goes to zero with enough iterations—however, the generalization performance *still* depends on the implementation and training of WeakLearn.

5.2.4 Arc-x4

Arc-x4 (Breiman (1996b)) started out as a mechanism for evaluating boosting where the resulting classifiers were combined without weighting the votes. Leo Breiman was testing adaptive boosting (or arc-fs as he calls it) and wanted to see what effect the increasing weights had on misclassified examples—he wanted to see whether it was the specific form of arc-fs that made it successful or whether it was the adaptive resampling.

Breiman then devised a simple update scheme for the weight distribution that relied on the number of misclassifications $m(\vec{x})$ by the ensemble of classifiers existing at iteration i (see figure 5.4 for a description of the algorithm). He tried different versions all of the form $1 + m(\vec{x})^h, h = 1, 2, 4$. The version with $h = 4$ did the best and became arc-x4.

This version does not have the weighted voting of adaptive boosting, but it still produces accurate ensembles (Opitz and Maclin (1999)).

5.2.5 LogitBoost

Friedman et al. (1998) showed that boosting can also be understood in terms of well known statistical principles—additive modelling and maximum likelihood. They came to the conclusion that adaptive boosting is a method for fitting an *additive* model $\mathbb{E}(\vec{x}) = \sum_i w_i C_i(\vec{x})$ in a forward stagewise manner. They proved that AdaBoost fits an additive logistic regression model using a criterion similar to the binomial log-likelihood. From this Friedman et al. (1998) derived *LogitBoost* that directly optimizes the binomial log-likelihood.

```
*Ensemble generation*
**Require:** A labelled training set $\mathbb{Z} = \{(\vec{z}_j, y_j), j = 1, \ldots, N\}$ with N examples
  and a weak learning algorithm WeakLearn.
  Set $D_1(j) = \frac{1}{N}$ for every example $\vec{z}_j$ (assign an equal weight to every
  example).
  **for** iterations $i = 1$ to $n$ **do**
    Train classifier $C_i$ on $\mathbb{Z}$ with weights $D_i$.
    Let $m(\vec{z}_j)$ be the number of misclassifications of $\vec{z}_j$ made by the classi-
    fiers $C_1, \ldots, C_i$.
    **for all** examples $\vec{z}_j \in \mathbb{Z}$ **do**
      Set $D_{i+1}(j) = 1 + m(\vec{z}_j)^4$.
    Normalize weights of all instances by dividing all the $D_{i+1}(j)$ by
    $\sum_j 1 + m(\vec{z}_j)^4$.
*Classification procedure*
**Require:** An example $\vec{x} \in \Theta$.
  Return the majority vote of the ensemble for this $\vec{x}$.
```

Figure 5.4: Algorithm for Arc-x4

The two-class version of LogitBoost is shown in figure 5.5. The weak learner for LogitBoost produces a mapping $C_i : \mathbb{Z} \longrightarrow [0, 1]$. The sign of C_i gives the classification with $|C_i|$ a measure of confidence in the prediction.

Friedman et al. (1998) also proposed a weight trimming enhancement to boosting. At each boosting iteration there is a distribution of weights over the training examples. A larger fraction of the training examples become correctly classified with increasing confidence—thereby receiving smaller weights. These examples have very little impact on the training of the base classifier. This suggests that at any specific iteration one can delete from the training examples these examples with a very low weight without having a big impact on the resulting classifiers. Computation is reduced since it tends to be proportional to the size of the training set—regardless of the weights distribution.

5.2.6 Ensemble size

Opitz and Maclin (1999) found that when bagging and boosting were applied to neural networks most of the error reduction took place after only ten to fifteen base classifiers. They came to a similar conclusion with bagging and decision trees (as did Breiman (1996a)).

AdaBoost and arc-x4 continued to measurably improve their test set accuracy until about twenty-five classifiers for decision trees. It was originally believed that boosting will continue to improve the test-set error indefinitely, but it was demonstrated that AdaBoost started overfitting with *very large*

```
*Ensemble generation*
**Require:** A labelled training set
    $$\mathbb{Z} = \{(\vec{z}_j, y_j), j = 1, \ldots, N \text{ and } y_j \in \mathbb{L} = \{0, 1\}\}$$
  and a weak learning algorithm WeakLearn.
  Set $w_j = \frac{1}{N}$ for every example $\vec{z}_j$ (assign an equal weight to every training example for the first iteration).
  Set $\mathbb{E}(\vec{z}) = 0$ and the probability estimates $p(\vec{z}_j) = \frac{1}{2}$ for every training example.
  **for** iterations $i = 1$ to $n$ **do**
    **for all** examples $\vec{z}_j \in \mathbb{Z}$ **do**
      Compute the working response and weights
        $$y_j = \frac{l_j - p(\vec{z}_j)}{p(\vec{z}_j)(1 - p(\vec{z}_j))}$$
        $$w_j = p(\vec{z}_j)(1 - p(\vec{z}_j))$$
    Train $C_i(\vec{z})$ on $\mathbb{Z}$ using the weighted (with weights $w_j$) working responses as class labels.
    Update
        $$\mathbb{E}(\vec{z}) = \mathbb{E}(\vec{z}) + \frac{1}{2}C_i(\vec{z})$$
    and
        $$p(\vec{z}) = \frac{e^{\mathbb{E}(\vec{z})}}{e^{\mathbb{E}(\vec{z})} + e^{-\mathbb{E}(\vec{z})}}$$
  Output the classifier
        $$\text{sign}[\mathbb{E}(\vec{z})] = \text{sign}\left[\sum_{i=1}^{n} C_i(\vec{z})\right]$$
*Classification procedure*
**Require:** An example $\vec{x} \in \Theta$.
  Return sign$[\mathbb{E}(\vec{x})]$.
```

Figure 5.5: Algorithm for LogitBoost (2 classes)

ensembles—10000 or more classifiers.

5.2.7 Weak learners

An important aspect to the successful application of boosting algorithms is the base weak learning algorithm (known as WeakLearn in all the algorithms shown so far). A weak learner that is *too* weak cannot guarantee that the ensemble will perform adequately (Schapire (1990)). An overly strong learner may again lead to overfitting, becoming more severe during later iterations of boosting. Meir and Rätsch (2002) empirically found that a base learner that already performs quite well, but is too simple for the data at hand, is best suited for using with boosting.

The following weak learners are all suitable for boosting:

Decision trees and stumps have been widely used for many years. Meir and Rätsch (2002) showed that boosting significantly enhanced the performance of decision trees and stumps[3].

Neural networks are also very successful with boosting. Neural networks can represent arbitrary continuous functions and it would seem that they could very easily overfit data when used with boosting. Drucker (1999) however found that neural networks are superior to decision trees when used with boosting. I am specifically interested in boosting neural networks and in section 5.5 I present my experimental results.

Kernel functions and linear combinations.

5.3 Training set noise

Boosting usually performs better than bagging, but it has been found that it is less resilient with regards to noise in the training data (Dietterich (2000), Opitz and Maclin (1999)). Freund and Schapire (1996) suggested that this sometimes poor performance of boosting results from overfitting the training set since later training sets may be over-emphasizing the training examples that are noise. This is the first possible reason. The second reason may be that the classifiers are combined using weighted voting (Opitz and Maclin (1999)). An unweighted scheme is generally more resilient to overfitting.

Bagging constructs each base classifier independently by manipulating the input data. It acts a bit like Bayesian voting (Dietterich (2000))—by sampling from the space of all possible hypotheses with a bias toward hypotheses that are accurate on the training data. This mainly addresses the underlying statistical problem. In contrast, boosting constructs each new

[3]A decision stump is simply a one-level decision tree—a classifier formed by splitting the input space once and then halting.

base classifier with the explicit goal of eliminating previous errors. This increases the risk of overfitting—especially with data that is noisy. Boosting puts more weight on the misclassified examples which leads to overfitting. Bagging does relatively well with noisy data as it increases the underlying statistical problem.

5.4 Bias-variance decomposition

I will try to explain boosting's success with the bias-variance decomposition (see section 1.4). Remember that the fixed portion of the bias-variance decomposition, the intrinsic target noise, cannot be reduced further.

Bagging is a pure variance reducing procedure—boosting is fundamentally different. Friedman et al. (1998) suggested that there is very little connection between weighted boosting and bagging. It appears to be a mainly bias reducing procedure by incorporating stable highly biased weak classifiers into a jointly fitted additive expansion.

Freund and Schapire (1998) found that boosting can reduce both variance and bias. They found it evident mostly in experiments with stumps—a learning algorithm with high bias. Freund and Schapire (1998) also found that in experiments with very specific data sets boosting actually *increased* the variance, while at the same time reduced the bias sufficiently to reduce the final error.

Freund and Schapire (1996) argued that boosting reduces the bias term since it focuses on the misclassified training examples—customising the learning algorithm to be closer to the target function. This makes it possible for boosting to construct a function that is not even producible by its base classifiers; using a linear base classifier to produce a combined classifier that can learn non-linear functions!

1. How does the accuracy of a single neural network compare to that of a boosting ensemble?
2. What is the effect of ensemble size on ensemble accuracy?
3. What is the effect of ensemble size on ensemble diversity?
4. Can I determine a relationship between the ensemble accuracy and diversity?
5. How does the accuracy of boosted ensembles compare to that of bagged ensembles?
6. Does boosting generated more *diverse* ensembles than bagging?

Figure 5.6: Boosting: interesting problems

5.5 Empirical results

There were several interesting aspects of boosting that I wanted to study. They were similar to those asked in chapter 4 (figure 4.2) and are shown in figure 5.6. Additionally, I was interested in comparing the accuracy and diversity of boosted and bagged ensembles.

5.5.1 Methodology

The methodology used to conduct the experiments was the same as explained in section 4.5.1. The experiments were conducted on the same datasets (table 5.2). I have used the AdaBoost.M1 version of boosting for all experiments.

As explained in section 5.2.2 one can either use *boosting by weighting* or *boosting by sampling*. The specific version of back-propagation that I have used for all the experiments understands weighted examples. A problem showed up early during my experimental research. The boosting process terminated often and early (with less than 5 networks in the ensemble) because $\epsilon = 0$ or $\epsilon \geq 0.5$. Breiman (1996b) had similar problems with experiments that he did and suggested the following: reset all the weights to be equal and continue. In this way I made sure that the boosted ensembles always had the same number of base networks as bagging. I adjusted AdaBoost.M1 to reflect this. Furthermore, because of the fact that a neural network is deterministic, I had to rather use the boosting by sampling method. If I used the boosting by weighting method the AdaBoost.M1 method would generate duplicate members—the next neural network would be the same as the first neural network in the ensemble and this would lead to the weights being changed in the same way for the second ensemble member, and for the

Table 5.2 Data sets used for empirical analysis

	cases	classes	discrete	real	inputs	hidden	outputs	epochs	λ
soybean	683	19	35	0	84	23	19	30	0.3
breast-cancer	286	2	9	0	48	6	2	60	0.15
iris	150	3	0	4	4	4	3	80	0.05
balance-scale	625	3	0	4	4	5	3	60	0.3
heart-c	303	5	7	6	23	7	5	100	0.15
heart-h	294	5	7	6	23	7	5	100	0.15
lymph	148	4	16	3	38	7	4	50	0.1
mushroom	8124	2	22	0	125	14	2	20	0.3
hepatitis	155	2	13	6	23	4	2	80	0.15
horse-colic	348	3	15	7	58	9	3	40	0.15
labor	57	2	8	8	26	10	2	10	0.3

rest of the ensemble.

5.5.2 Accuracy

Table 5.3 shows the single neural network error rates for all the data sets described in table 5.2. Figures 5.9 to 5.18 show the performance of boosting as a percentage in relation to the error rate of the single neural networks for the corresponding data sets.

Boosting did better than the single neural network on 5 of the 10 data sets. It showed really big improvements on those 5 sets: between 10 % and 30 % ! It had more or less the same performance on the hepatitis data set and performed worse on the breast-cancer, heart-c, colic and labor data sets. I found that when boosting improved the accuracy on a data set, it improved more on the single network accuracy than bagging. Furthermore, boosting kept on improving on the single network error rate beyond the ensemble size where bagging's improvements levelled off. I saw improvements on soybean, balance-scale, heart-h and lymph even when the ensemble grew bigger than 15 classifiers.

5.5.3 Diversity

All the diversity measures were calculated for every ensemble. I have repeated table 5.4 from chapter 2. It shows all the diversity measures, whether it is a measure of diversity or similarity and its possible range of values.

Pairwise measures

- Q indicated that the neural networks in the ensembles for all the data sets were statistically independent (i.e. Q was close to 0) and had high diversity. It showed higher diversity as the ensembles got bigger.

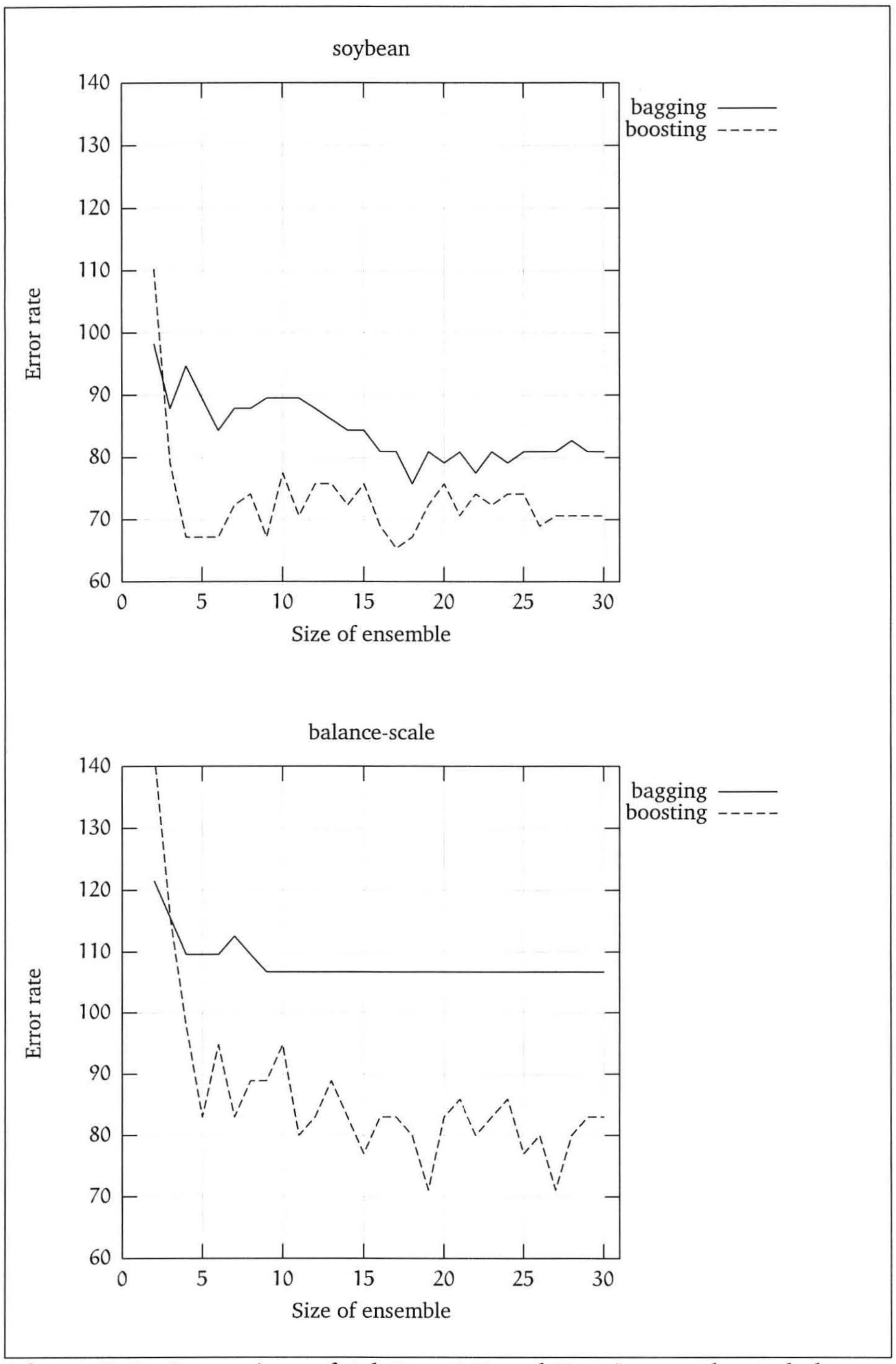
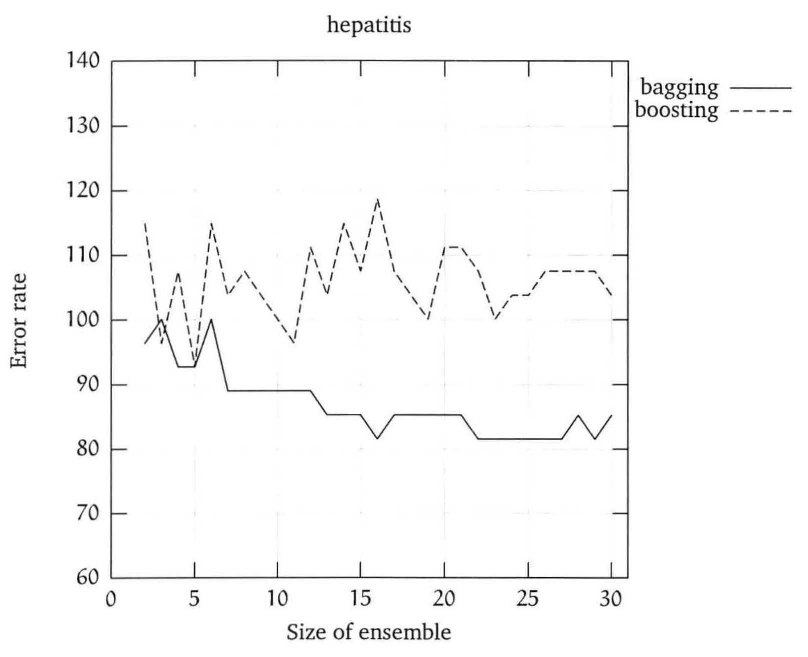


Figure 5.7: Comparison of AdaBoostM1 and Bagging: soybean, balance-scale



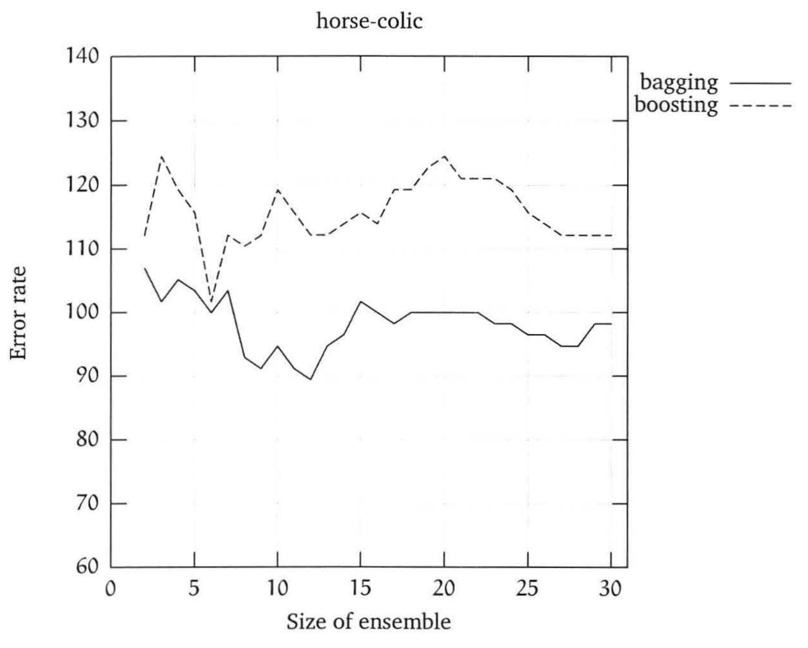


Figure 5.8: Comparison of AdaBoostM1 and Bagging: hepatitis, horse-colic

Table 5.3 Error rates for single neural networks

soybean	8.5
breast-cancer	27.6
iris	4.7
balance-scale	5.4
heart-c	16.8
heart-h	20.7
lymph	16.9
mushroom	0
hepatitis	17.4
horse-colic	15.5
labor	8.8

- ρ also indicated high diversity.
- The disagreement measure, Dis, increased as the ensembles grew bigger. It indicated medium diversity.
- DF indicated medium similarity (or diversity) in the ensembles. It did not really change much as the ensembles grew bigger.

The pairwise measures (except for DF) all indicated that the boosted ensembles were diverse.

Non-pairwise measures

- The entropy measure, E, indicated medium to high diversity for most of the data sets. Its value increased as the ensembles grew larger.
- KW measured high diversity for all the data sets—its value also increased as the ensembles grew bigger.
- κ indicated low similarity for all the data sets. Its value decreased slightly (more for two of the data sets) as the ensembles grew bigger.
- θ indicated low similarity.
- GD measured medium to high diversity and did not change much as the ensembles grew.
- CFD indicated slightly higher diversity than GD for all the data sets. It is understandable as CFD is related to GD.

The non-pairwise measures all indicated medium to high diversity for all the data sets.

Table 5.4 Summary of diversity measures. An up-arrow (↑) specifies that it is a measure of diversity. A down-arrow (↓) specifies that it is a measure of similarity.

Pairwise			
Q Statistics	Q	↓	$-1 \leq Q \leq 1$
Correlation coefficient	ρ	↓	$-1 \leq \rho \leq 1$
Disagreement	Dis	↑	$0 \leq Dis \leq 1$
Double-fault	DF	↓	$0 \leq DF \leq 1$
Non-pairwise			
Entropy	E	↑	$0 \leq E \leq 1$
Kohavi-Wolpert variance	KW	↑	$0 \leq KW \leq 1/4$
Interrater agreement	κ	↓	
Difficulty	θ	↓	$\theta \geq 0$
Generalized diversity	GD	↑	$0 \leq GD \leq 1$
Coincident failure diversity	CFD	↑	$0 \leq CFD \leq 1$
Fitness	F	↑	$F \geq 0$

Most of the measures also grew more diverse as the ensembles grew bigger—indicating that there is definitely a link between a boosted ensemble's size and diversity.

5.5.4 Comparison to bagging

I have included two figures (figure 5.7 and figure 5.8) that compare boosting and bagging on four data sets—soybean, balance-scale, hepatitis and horse-colic. I chose them to illustrate that when boosting was more accurate than bagging it was much more accurate. It was also interesting to note that when boosting was less accurate, it was usually less accurate by a big margin. Bagging can be applied with more confidence to all data sets and will usually perform better than a single network. Boosting will do better than bagging, but performs worse when the dataset has noise.

It was interesting to note that AdaBoost.M1 produced more diverse ensembles than bagging. This was the case for all the data sets. This can be explained by looking at how boosting produces its ensembles—it is a *direct* search for members that can correctly classify examples that previous members have misclassified. One would expect it to have more diverse ensembles and I was happy to discover that this was in fact the case.

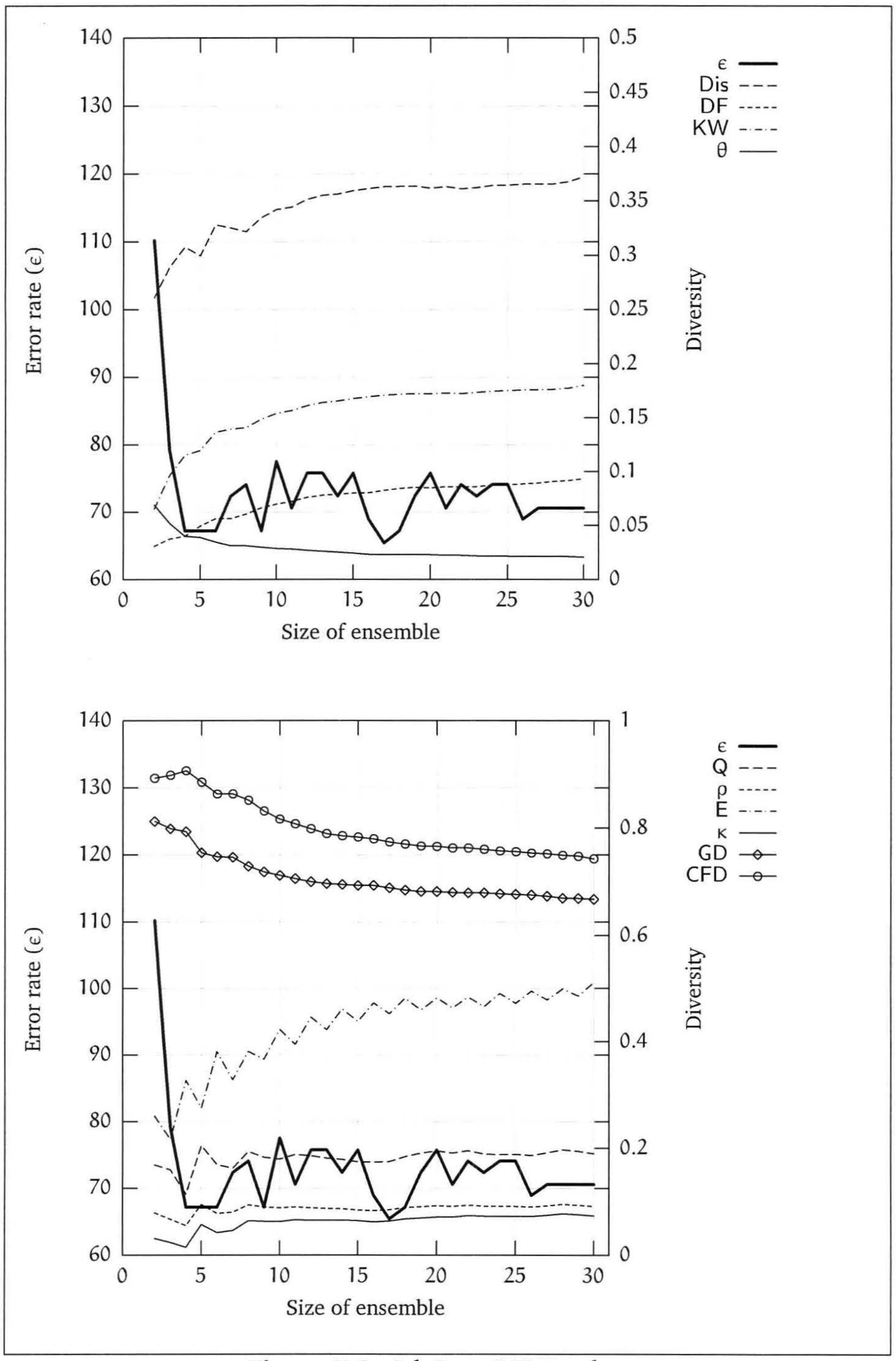


Figure 5.9: AdaBoostM1: soybean

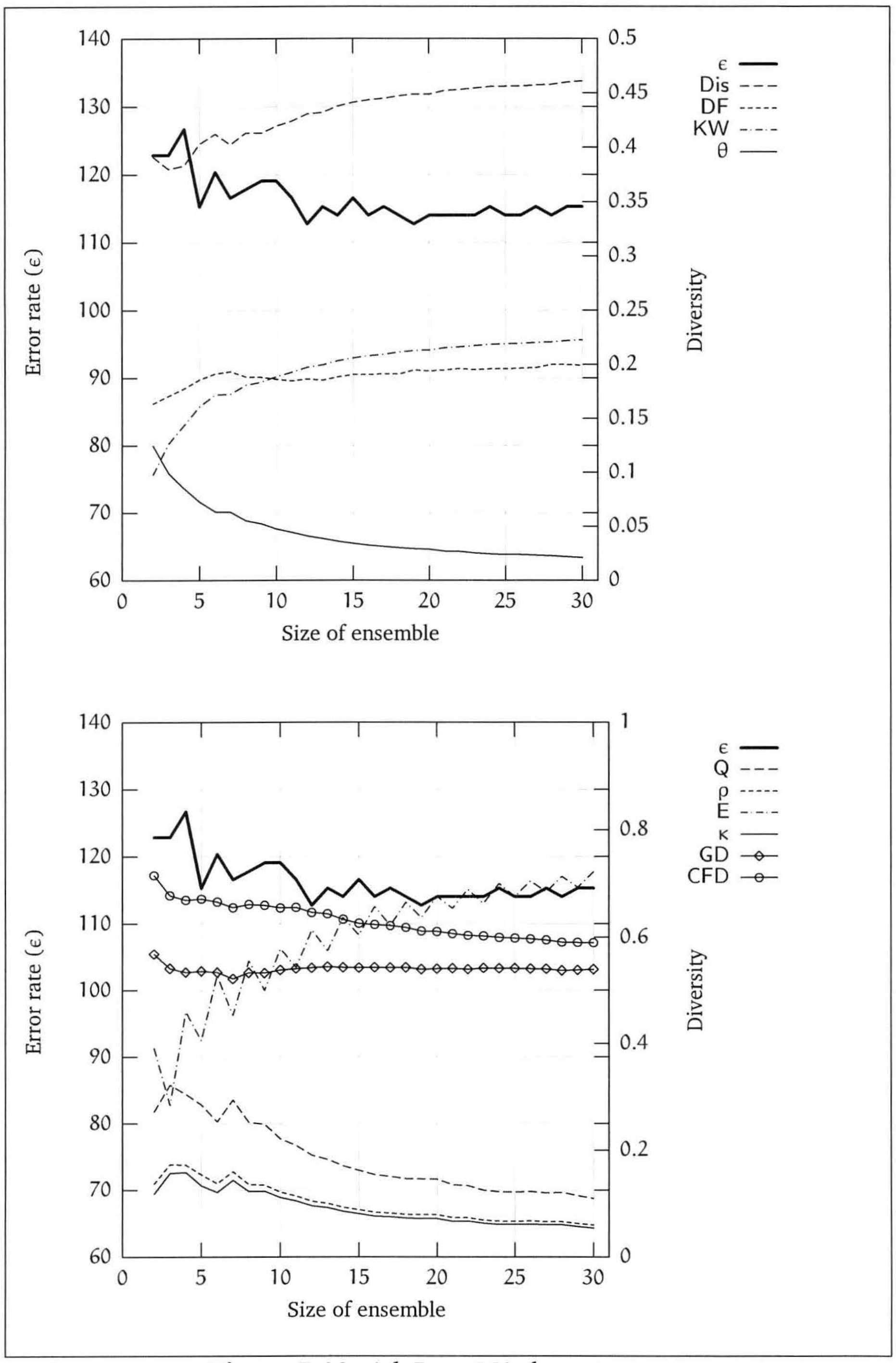


Figure 5.10: AdaBoostM1: breast-cancer

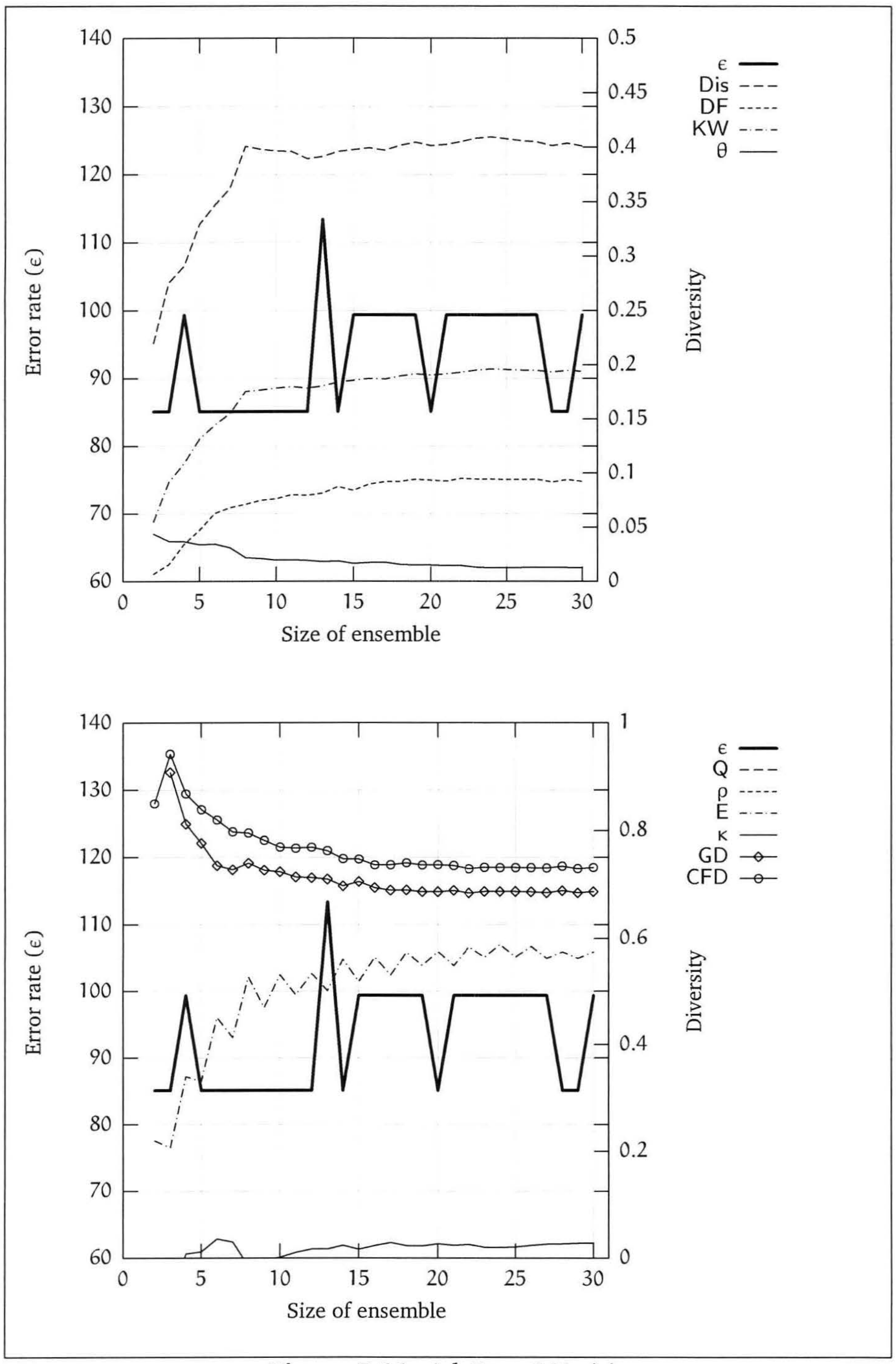


Figure 5.11: AdaBoostM1: iris

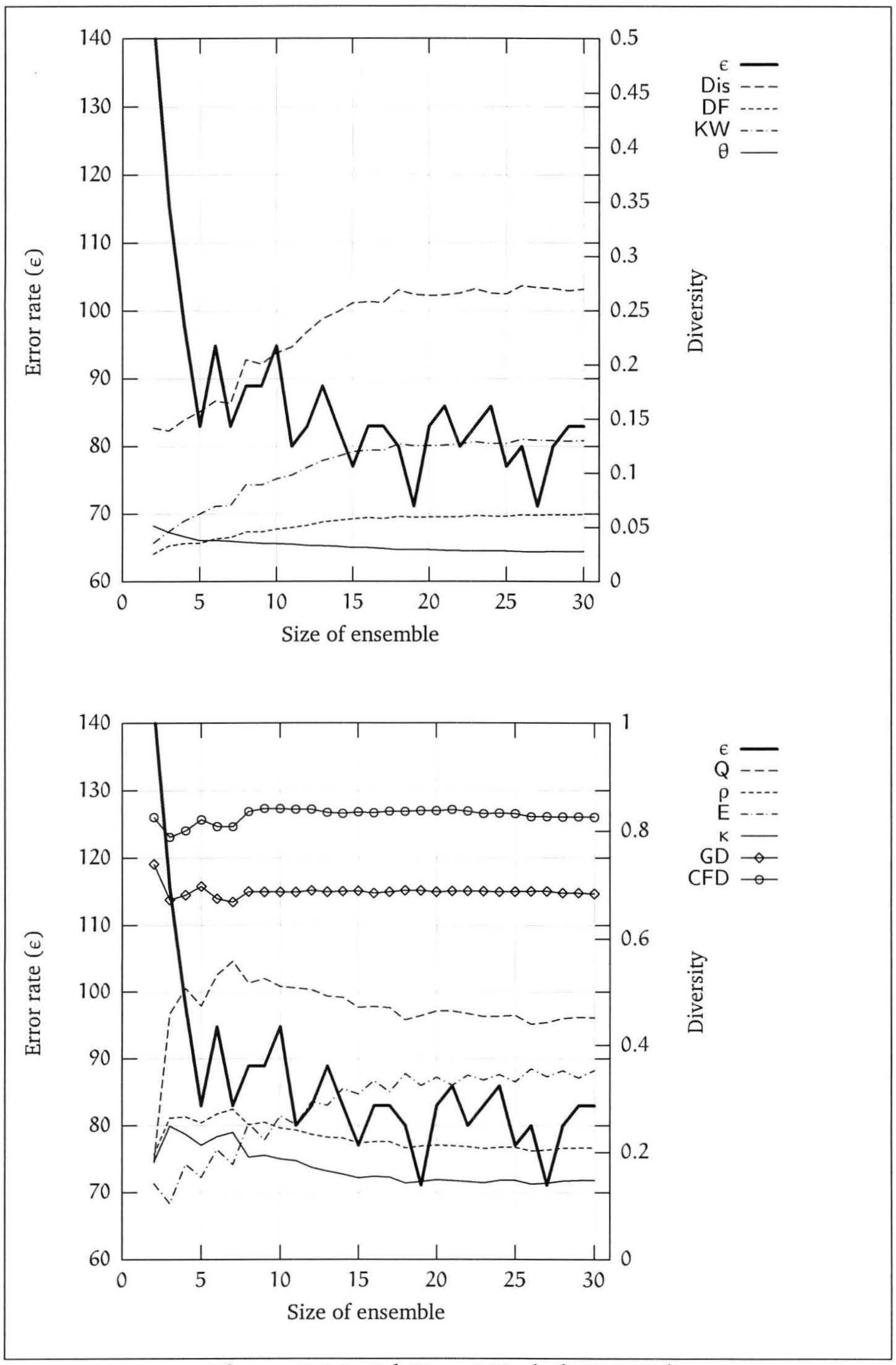


Figure 5.12: AdaBoostM1: balance-scale

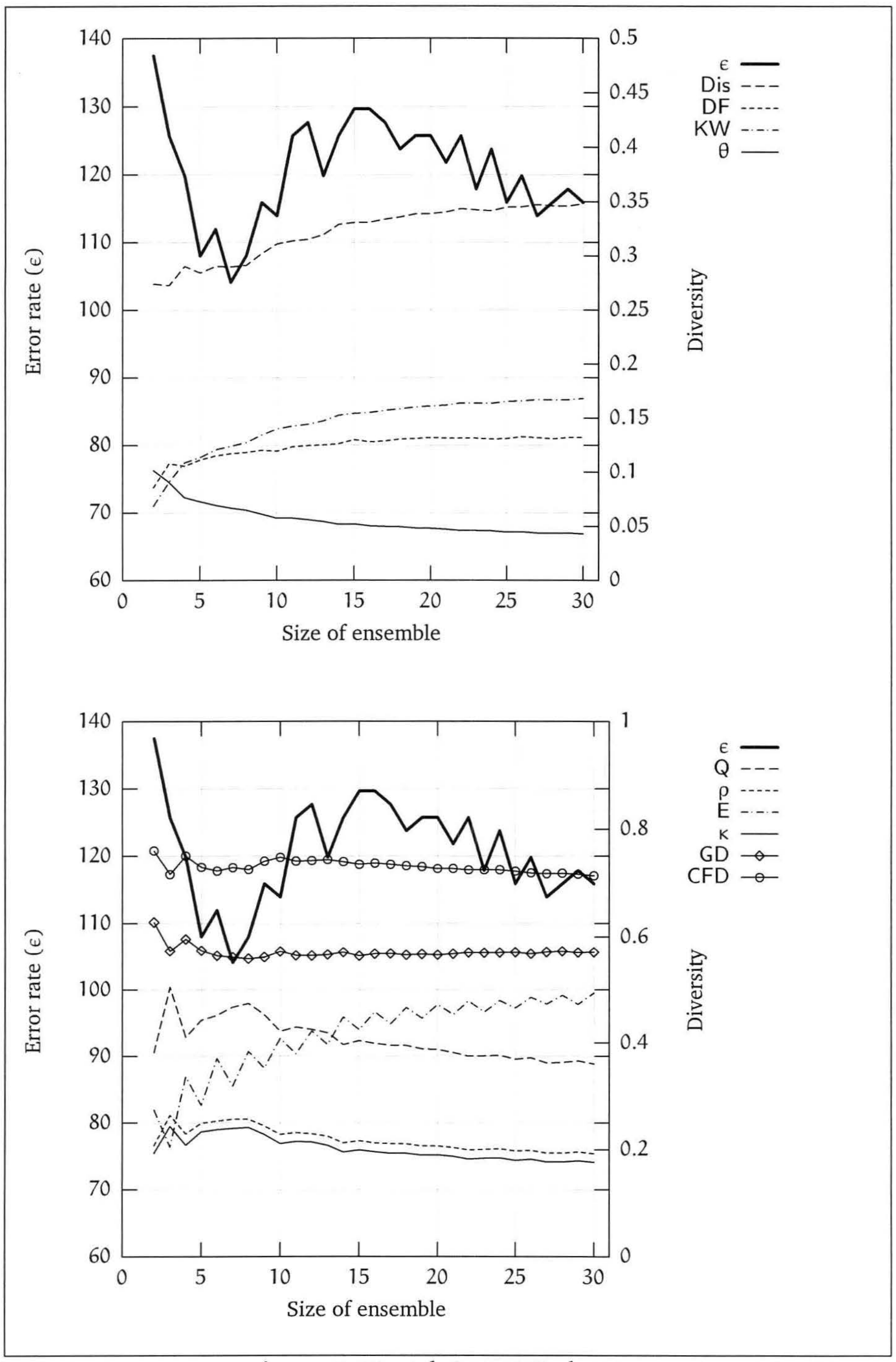


Figure 5.13: AdaBoostM1: heart-c

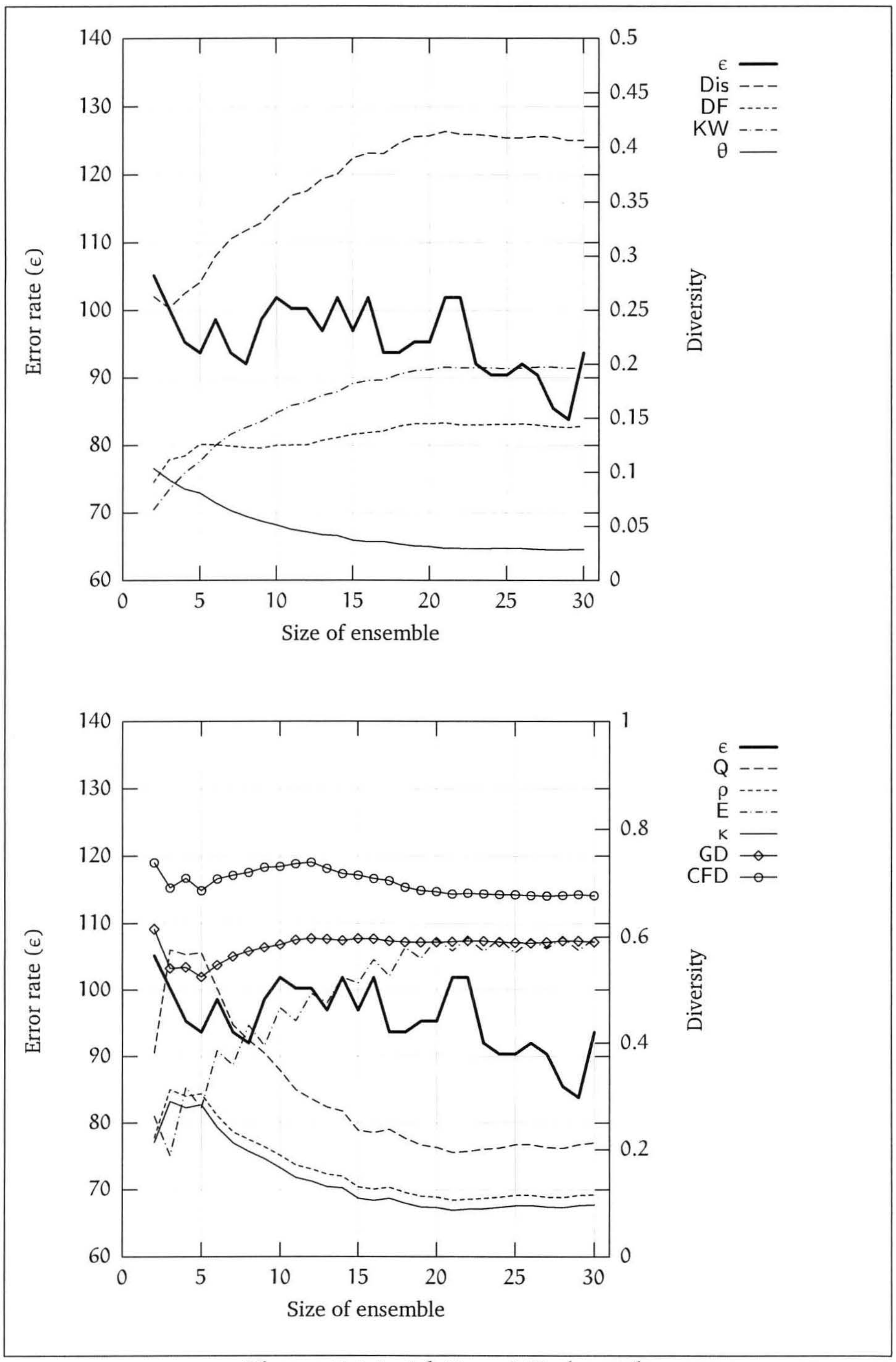


Figure 5.14: AdaBoostM1: heart-h

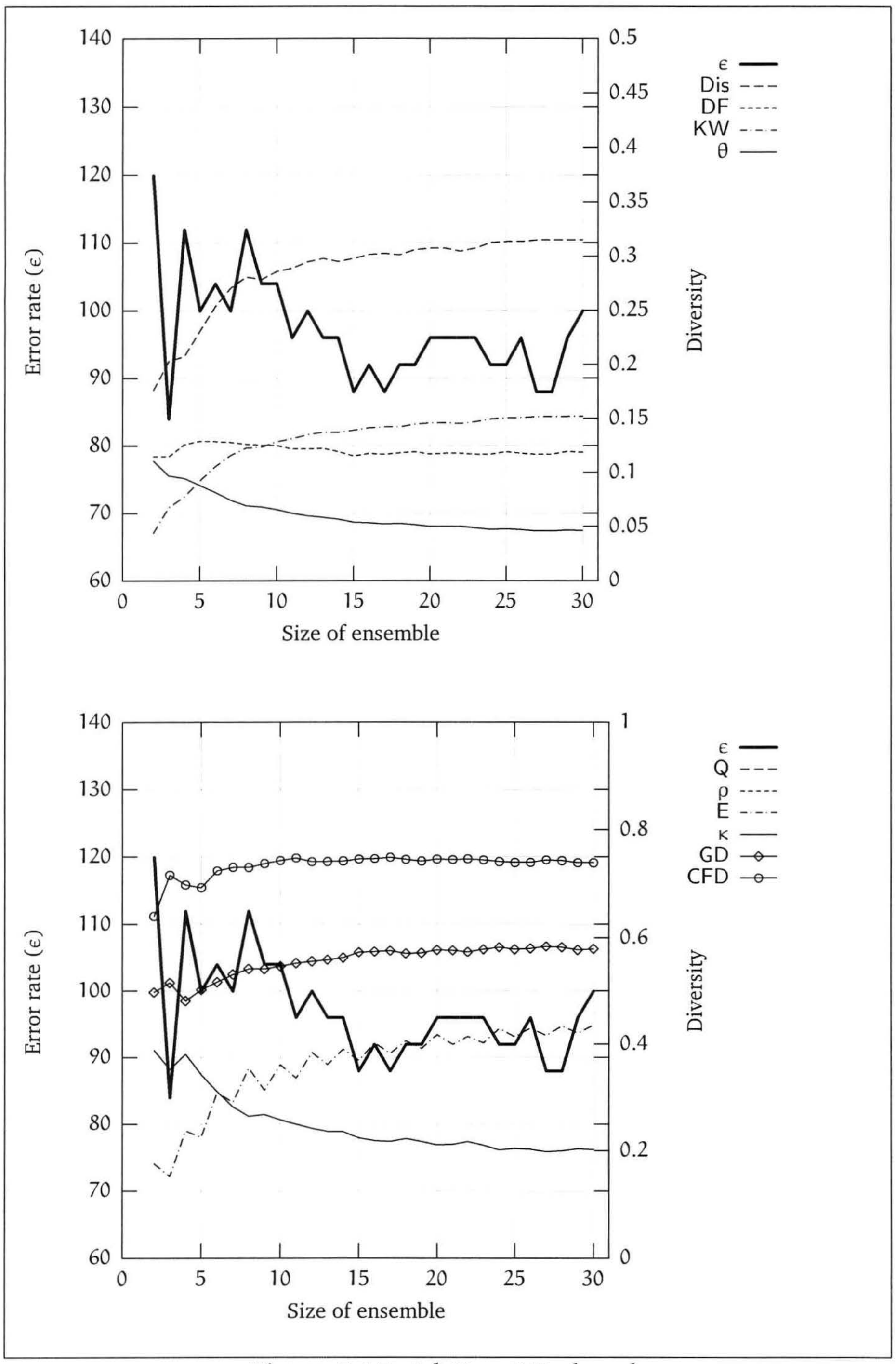


Figure 5.15: AdaBoostM1: lymph

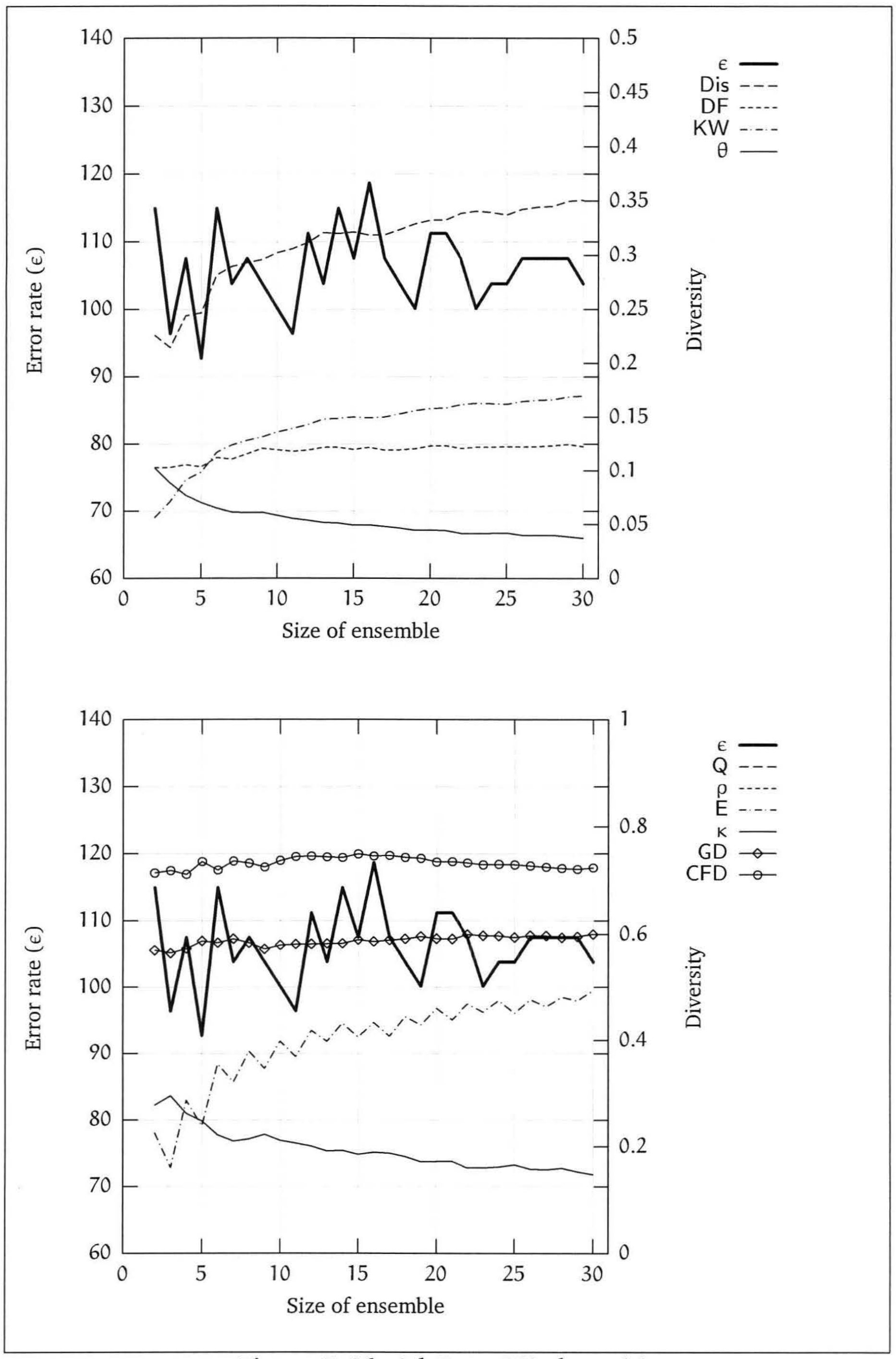


Figure 5.16: AdaBoostM1: hepatitis

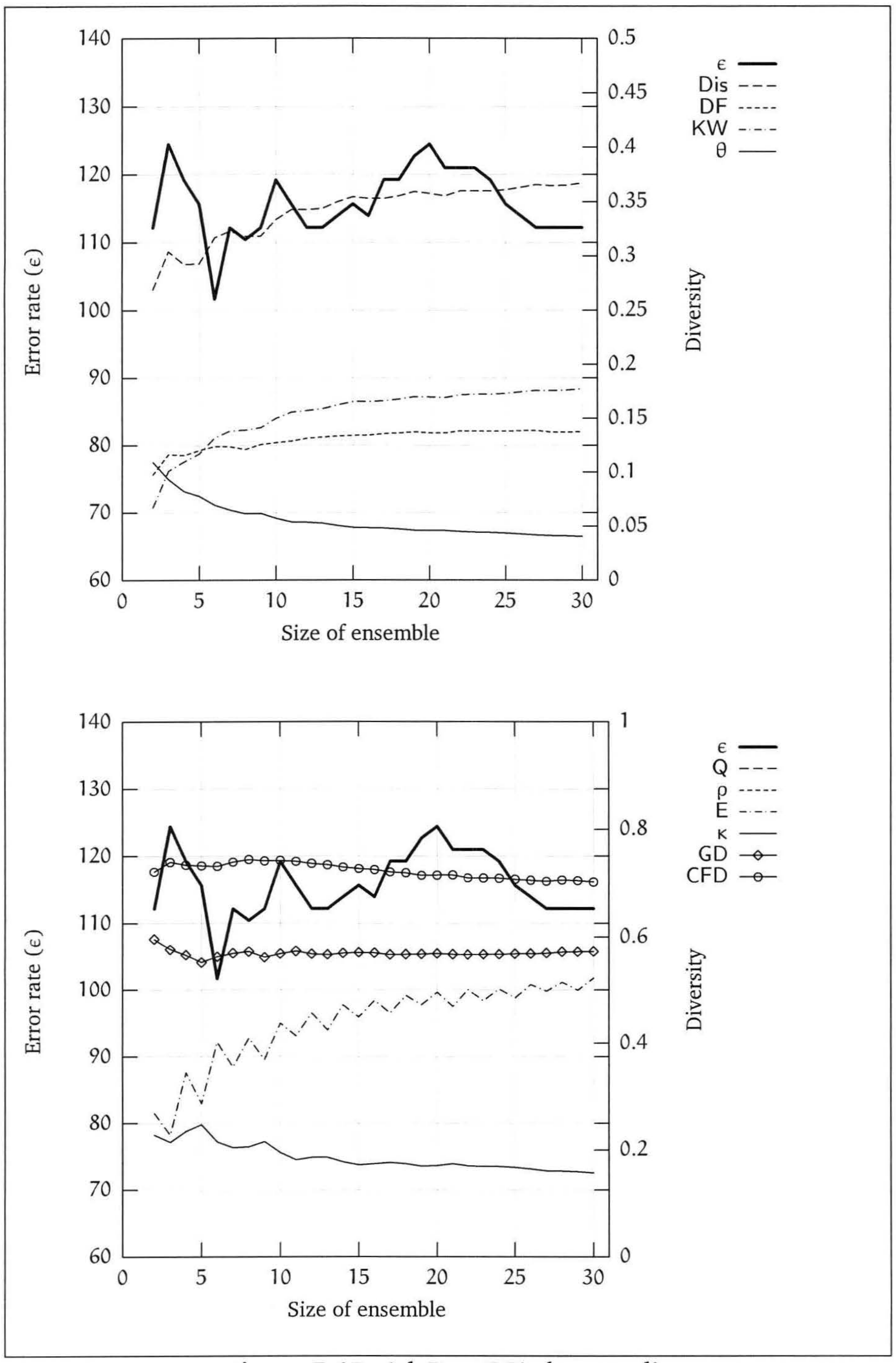


Figure 5.17: AdaBoostM1: horse-colic

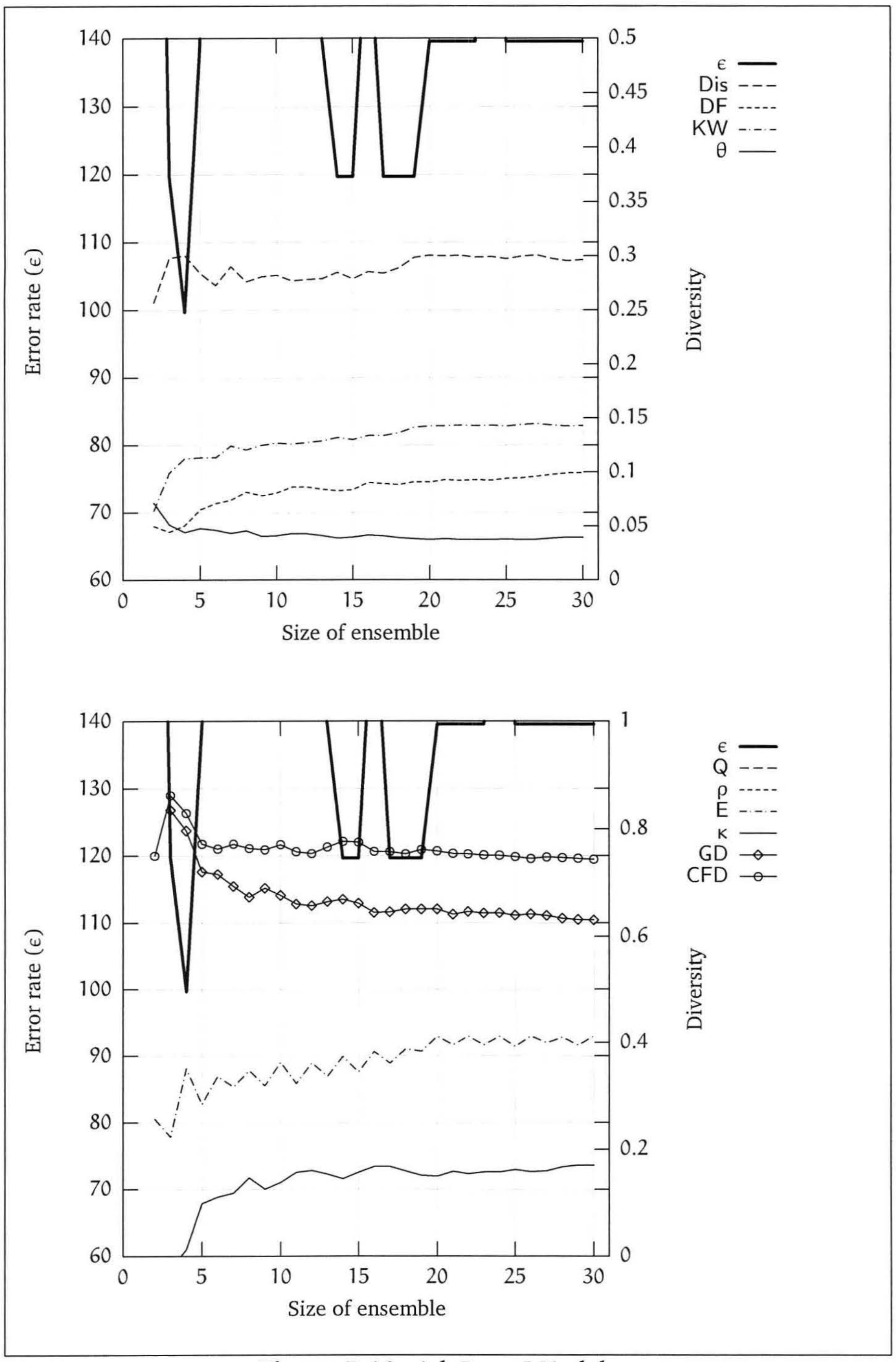


Figure 5.18: AdaBoostM1: labor

5.6 Summary

In this chapter I described in detail the process of boosting classifiers and building ensembles with the boosted classifiers. I presented the history of boosting, explained four different variants and discussed what weak learners to use and how big the ensembles should be. I explained the success of boosting by referring to the bias-variance decomposition.

In section 5.5 I presented my empirical results. I asked the following questions:

1. How does the accuracy of a single neural network compare to that of a boosting ensemble? The boosting ensemble was more accurate with 5 of the 10 data sets.

2. What is the effect of ensemble size on ensemble accuracy? Boosting kept on increasing on the single network accuracy even after the ensemble size grew bigger than 15 networks.

3. What is the effect of ensemble size on ensemble diversity? I found that the boosting process created ensembles with high diversity and that the diversity scores kept on increasing with larger ensembles.

4. How does the accuracy of boosted ensembles compare to bagged ensembles? When boosting did better than bagging—it did so with aplomb. When boosting did worse—it also was with a margin.

5. Does boosting generated more *diverse* ensembles than bagging? Yes!

6. Can I determine a relationship between the ensemble accuracy and diversity?

There is one question that had a disappointing answer: I cannot find any evidence of a relationship between ensemble diversity and accuracy. In fact—with some of the data sets boosting did not improve on the single neural network, but still had high diversity scores.

Chapter 6

Conclusion

This thesis explored in detail the concept of neural network ensembles. I presented a list of diversity measures (chapter 2) that may be used to score ensembles based on the diversity of their members. I have looked at the underlying theory and mechanisms that are employed by ensembling methods to generate and combine members in chapter 3. In chapter 4 and 5 I explored bagging and boosting. I discussed how they worked and attempted to explain their success by way of the bias-variance decomposition and the fundamental reasons (chapter 1).

I did an empirical evaluation of bagging and boosting with neural networks and found that:

- A bagging ensemble almost always outperformed the single neural network.

- A boosting ensemble can greatly outperform both the single neural network and a bagging ensemble. However, for some data sets the boosting ensemble showed zero improvements and in some cases even did worse than a single network. Boosting's performance is at least partly dependent on the data set it is training on, while bagging was less affected. This may be explained by boosting's tendency to overfit in the presence of noise.

- Much of the performance improvements came with ensembles of less than 15 members. Boosting continued to improve after bagging's improvements levelled off.

- Bagging is appropriate for most problems, but when suitable, boosting will produce larger improvements.

- Bagging did not build very diverse ensembles, as measures by our diversity measures. Also, the diversity scores did not change much as the ensembles grew bigger.

- In comparison, boosting generated ensembles with much higher diversity scores and the diversity scores did change as the boosting ensembles grew bigger.

- I could not empirically show that ensemble accuracy is related to diversity.

I was disappointed to be unable to find a clear link between ensemble accuracy and diversity, but was happy to find that boosting generated more diverse ensembles than bagging. We should maybe change our focus from trying to link diversity to accuracy and rather use the diversity measures as part of the training process.

Neural network ensembles are a successful tool in the machine learning world and there are exciting links between the different methods and ensemble diversity.

Appendix A

List of symbols

Symbol	Description
C	classifier
$\mathbb{E} = \{C_1, C_2, \ldots, C_n\}$	ensemble of n classifiers
Θ	input space with a attributes / features
l	class label
$\mathbb{L} = \{l_1, l_2, \ldots, l_m\}$	set of m possible class labels
$\vec{x} = \langle x_1, x_2, \ldots, x_a \rangle \in \Theta$	instance / example with a attributes
$\vec{z} = \langle z_1, z_2, \ldots, z_a \rangle \in \Theta$	training example
$\mathbb{Z} = \{\vec{z}_1, \vec{z}_2, \ldots, \vec{z}_N\}$	labelled training set of N examples

Appendix B

Data sets

This appendix includes a short description of every data set that I have used. All data sets except for "horse-colic" are from the UCI repository.

B.1 Soybean

Full title Large soybean database

Source R.S. Michalski and R.L. Chilausky "Learning by Being Told and Learning from Examples: An Experimental Comparison of the Two Methods of Knowledge Acquisition in the Context of Developing an Expert System for Soybean Disease Diagnosis", International Journal of Policy Analysis and Information Systems, Vol. 4, No. 2, 1980.

Description Soybean disease data set.

Number of instances 683

Number of attributes 19 (all discrete)

B.2 Breast-cancer

Full title Breast cancer data

Source Matjaz Zwitter & Milan Soklic (physicians), Institute of Oncology, University Medical Center, Ljubljana, Yugoslavia.

Description This is one of three domains provided by the Oncology Institute that has repeatedly appeared in the machine learning literature. This data set includes 201 instances of one class ("no-recurrence-events") and 85 instances of another class ("recurrence-events").

Number of instances 286

Number of attributes 9 (all discrete)

B.3 Iris

Full title Iris plants database

Source Created by R.A. Fisher; donated by Michael Marshall.

Description This is perhaps the best known database to be found in the pattern recognition literature. The data set contains 3 classes of 50 instances each, where each class refers to a type of iris plant. One class is linearly separable from the other two; the latter are not linearly separable from each other. The predicted attribute is the class of iris plant.

Number of instances 150

Number of attributes 4 (real)

B.4 Balance-scale

Full title Balance Scale Weight & Distance Database

Source Generated to model psychological experiments reported by Siegler, R. S. (1976). Three Aspects of Cognitive Development. Cognitive Psychology, 8, 481-520. Donated by Tim Hume.

Description This data set was generated to model psychological experimental results. Each example is classified as having the balance scale tip to the right, tip to the left, or be balanced. The attributes are the left weight, the left distance, the right weight, and the right distance.

Number of instances 625

Number of attributes 4 (real)

B.5 Heart-c

Full title Heart disease database (Cleveland)

Source V.A. Medical Center, Long Beach and Cleveland Clinic Foundation. Robert Detrano, M.D., Ph.D. Donated by David W. Aha.

Description This database contains 76 attributes, but all published experiments refer to using a subset of 14 of them. The "goal" field refers to the presence of heart disease in the patient.

Number of instances 303

Number of attributes 13 (7 discrete, 6 real)

B.6 Heart-h

Full title Heart disease database (Hungary)

Source Hungarian Institute of Cardiology, Budapest, Andras Janosi, M.D. Donated by David W. Aha.

Description This database contains 76 attributes, but all published experiments refer to using a subset of 14 of them. The "goal" field refers to the presence of heart disease in the patient.

Number of instances 294

Number of attributes 13 (7 discrete, 6 real)

B.7 Lymph

Full title Lymphography Domain

Source Institute of Oncology, University Medical Center, Ljubljana, Yugoslavia.

Description This is one of three domains provided by the Oncology Institute that has repeatedly appeared in the machine learning literature.

Number of instances 148

Number of attributes 19 (16 discrete, 3 real)

B.8 Mushroom

Full title Mushroom database

Source Mushroom records drawn from The Audubon Society Field Guide to North American Mushrooms (1981). G. H. Lincoff (Pres.), New York: Alfred A. Knopf. Donated by Jeff Schlimmer.

Description This data set includes descriptions of hypothetical samples corresponding to 23 species of gilled mushrooms in the Agaricus and Lepiota Family (pp. 500-525). Each species is identified as definitely edible, definitely poisonous, or of unknown edibility and not recommended. This latter class was combined with the poisonous one. The Guide clearly states that there is no simple rule for determining the edibility of a mushroom; no rule like "leaflets three, let it be" for Poisonous Oak and Ivy.

Number of instances 8124

Number of attributes 22 (all discrete)

B.9 Hepatitis

Full title Hepatitis domain

Source Donated by G. Gong (Carnegie-Mellon University) via Bojan Cestnik (Jozef Stefan Institute, Ljubljana Yugoslavia).

Description This data set has 19 attributes about 155 patients—each instance labelled as either "live" or "die".

Number of instances 155

Number of attributes 19 (16 discrete, 9 real)

B.10 Horse-colic

Full title Horse colic database

Source Created by Mary McLeish and Matt Cecile, Department of Computer Science, University of Guelph, Guelph, Ontario, Canada N1G 2W1. Donated by Will Taylor.

Number of instances 368

Number of attributes 22 (15 discrete, 7 real)

B.11 Labor

Full title Final settlements in labour negotiations in Canadian industry.

Source Collective Bargaining Review, monthly publication, Labour Canada, Industrial Relations Information Service, Ottawa, Ontario, Canada, (819) 997-3117. Donated by Stan Matwin, Computer Science Department, University of Ottawa, 34 Somerset East, K1N 9B4.

Description The data set includes all collective agreements reached in the business and personal services sector for locals with at least 500 members (teachers, nurses, university staff, police, etc.) in Canada in 1987 and the first quarter of 1988.

Number of instances 57

Number of attributes 16 (8 discrete, 8 real)

Bibliography

Auda, G. and Kamel, M., 1998. CMNN: Cooperative Modular Neural Networks. *Neurocomputing* 20:189–207.

Breiman, L., 1996a. Bagging predictors. *Machine learning* 24:123–140.

Breiman, L., 1996b. Bias, Variance and Arcing Classifiers. Technical report 460, Statistics Department, University of California.

Dietterich, T., 1999. An Experimental Comparison of Three Methods for Constructing Ensembles of Decision Trees: Bagging, Boosting and Randomization. *Machine Learning* pp. 1–22.

Dietterich, T., 2000. Ensemble Methods in Machine Learning. In *Lecture Notes in Computer Science*, eds. J. Kittler and F. Roli, vol. 1857, pp. 1–15. Springer.

Dietterich, T. and Bakiri, G., 1995. Solving multiclass learning problems via error-correcting output codes. *Journal of Artificial Intelligence Research* 2:263–286.

Drucker, H., 1999. *Combining Artificial Neural Nets*, chap. Boosting Using Neural Networks, pp. 51–78. Springer-Verlag.

Duin, R., 2002. The Combining Classifier: to Train or Not to Train? In *Proceedings of the 16th International Conference on Pattern Recognition*, vol. 2, pp. 765–770.

Freund, Y., 1990. Boosting a weak learning algorithm by majority. In *Proceedings of the Third Workshop on Computational Learning Theory*, pp. 202–216.

Freund, Y. and Schapire, R. E., 1996. Experiments with a new boosting algorithm. In *Machine Learning: Proceedings of the Thirteenth International Conference*, pp. 148–156.

Freund, Y. and Schapire, R. E., 1998. Discussion of the paper Arcing Classifiers by Leo Breiman. *The Annals of Statistics* 26 (3):824–832.

Friedman, J., Hastie, T., and Tibshirani, R., 1998. Additive Logistic Regression: a Statistical View of Boosting. Technical report, Department of Statistics, Stanford University.

Giacinto, G. and Roli, F., 2000. Design of effective neural network ensembles for image classification processes. *Image Vision and Computing Journal* .

Granger, C. W. and Ramanathan, R., 1984. Improved Methods of Combining Forecasts. *Journal of Forecasting* 3:197–204.

Hashem, S., Schmeiser, B., and Yih, Y., 1994. Optimal Linear Combinations of Neural Networks: An Overview. In *IEEE International Conference on Neural Networks*. Orlando, Florida.

Kohavi, R. and Wolpert, D. H., 1996. Bias Plus Variance Decomposition for Zero-One Loss Functions. In *Machine Learning: Proceedings of the Thirteenth International Conference*, ed. L. Saitta, pp. 275–283. Morgan Kaufmann.

Krzanowski, W. and Partridge, D., 1997. Software Diversity: Practical Statistics for its Measurement and Exploitation. *Information and Software Technology* 39:707–717.

Kuncheva, L. I. and Whitaker, C. J., 2003. Measures of Diversity in Classifier Ensembles and Their Relationship with the Ensemble Accuracy. *Machine Learning* 51:181–207.

Littlestone, N. and Warmuth, M. K., 1992. The Weighted Majority Algorithm. Technical report UCSC-CRL-91-28, Baskin Center for Computer Engineering & Information Sciences.

Meir, R. and Rätsch, G., 2002. An Introduction to Boosting and Leveraging. *Advanced Lectures on Machine Learning* pp. 118–183.

Milidiú, R. L., Machado, R. J., and Renteriá, R. P., 1999. Time Series Forecasting through Wavelets Transformation and a Mixture of Expert Models. *Neurocomputing* 28, no. 1–3:145–156.

Mitchell, T. M., 1996. *Machine Learning*. McGraw-Hill.

Opitz, D. and Maclin, R., 1999. Popular Ensemble Methods: An Empirical Study. *Journal of Artificial Intelligence* 11:169–198.

Opitz, D. W. and Shavlik, J. W., 1996. Generating Accurate and Diverse Members of a Neural-Network Ensemble. In *Advances in Neural Information Processing Systems*, eds. D. S. Touretzky, M. C. Mozer, and M. E. Hasselmo, vol. 8, pp. 535–541. The MIT Press.

Schapire, R. E., 1990. The strength of weak learnability. *Machine Learning* 5 (2):197–227.

Schapire, R. E., 1999. A Brief Introduction to Boosting. In *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence*, pp. 1401–1406.

Skalak, D., 1996. The sources of increased accuracy for two proposed boosting algorithms. In *Proceedings of the American Association for Artificial Intelligence, AAAI 1996, Integrating Multiple Learned Models Workshop*.

Witten, I. H. and Frank, E., 1999. *Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations*. Morgan Kaufmann Publishers.

Wolpert, D. H., 1992. Stacked generalization. *Neural Networks* 5:241–259.

Yule, G., 1900. On the association of attributes in statistics. *Philosophical Transactions of the Royal Society of London* Series A 194:257–319.

Zhou, Z.-H., Wu, J., and Tang, W., 2002. Ensembling Neural Networks: Many Could Be Better Than All. *Artificial Intelligence* pp. 239–263.

Index