ENSEMBLES FOR SUPERVISED CLASSIFICATION LEARNING

A DISSERTATION
SUBMITTED TO THE DEPARTMENT OF COMPUTER SCIENCE
AND THE COMMITTEE ON GRADUATE STUDIES
OF STANFORD UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

By
Ofer Matan
August 1996

I certify that I have read this dissertation and that in my opinion it is fully adequate, in scope and in quality, as a dissertation for the degree of Doctor of Philosophy.

David E. Rumelhart
(Principal Adviser)

I certify that I have read this dissertation and that in my opinion it is fully adequate, in scope and in quality, as a dissertation for the degree of Doctor of Philosophy.

Michael R. Genesereth

I certify that I have read this dissertation and that in my opinion it is fully adequate, in scope and in quality, as a dissertation for the degree of Doctor of Philosophy.

Nils J. Nilsson

Approved for the University Committee on Graduate Studies:

Abstract

This dissertation studies the use of multiple classifiers (ensembles or committees) in learning tasks. Both theoretical and practical aspects of combining classifiers are studied.

We consider two different goals: The first is to achieve better classification rates. We analyze both the representation ability of ensembles and algorithms that search for a solution in this representation space. Second, we consider the cost and time it takes to train classifiers. This leads us to consider systems that either learn with fewer labeled examples or learn while performing at their current ability.

First we analyze the representational ability of voting ensembles. A voting ensemble may perform either better or worse than each of its individual members. We give tight upper and lower bounds on the classification performance of a voting ensemble as a function of the classification performances of its individual members.

Boosting is a method of combining multiple "weak" classifiers to form a "strong" classifier. Several issues concerning boosting are studied in this thesis. We study SBA, a hierarchical boosting algorithm proposed by Schapire, in terms of its representation and its search. We show that if the weak learner has low representational complexity, SBA's search may fail to boost or may give a sub-optimal solution. We present a rejection boosting algorithm that trades-off exploration and exploitation: It requires fewer pattern labels at the expense of lower boosting ability.

Ensembles may be useful in gaining information. We study their use to minimize labeling costs of data and to enable improvements on performance over time. For that purpose a model for on-site learning is presented. The system learns by querying "hard" patterns while classifying "easy" ones. This model is related to query-based filtering methods, but takes into account that, in addition to labeling, filtering through the data has a cost.

The Query-By-Committee algorithm is used as a good approximator of the model space

for real-world domains. Results using this algorithm on a synthesized problem and a real-world OCR task using both a back-propagation network and a nearest neighbor classifier show that an on-site learner can perform as well as a classifier trained off-site, while achieving significant cost reduction.

Acknowledgments

In the past five years, I have learned much more than Computer Science and Artificial Intelligence. I been been exposed to the rich environment of people, ideas and programs that exist at Stanford. I am grateful to the members of the community who have contributed to my experience here.

This is a time in which the educational mission of institutions such as Stanford is being re-examined. This is coupled with and fueled by budgetary concerns and has lead to elimination of services and programs during my tenure here. The changes have been gradual, but there is danger that eventually, all that will remain are narrow professional degree programs. I truly hope that the leaders of this institution will ensure that future students will still have a broad educational experience.

Each member of my reading committee has been instrumental to seeing me through this period. Ever since I came to Stanford, David Rumelhart has always had time to talk and pace the Quad with me. I am thankful that he was willing to become my principal adviser when it became apparent that my dissertation research would be in machine learning. Mike Genesereth was my principal adviser for over 3 years. I have benefited from both his critical comments and support over the years. Nils Nilsson has been an enormous help and I am happy that he agreed to be part of my reading committee. His kindness, openness and useful advice have helped me on a few occasions when my head was not screwed on in the right direction.

I have shared an office and a good part of my time with two wonderful individuals: I have probably learned more from Kathleen Fisher and Illah Nourbakhsh than from anyone else at Stanford. Though located far away, Ronny Kohavi has been an inspiring colleague and an indispensable source of LaTeX knowledge.

I've benefited from discussions with Tom Cover, Yoav Freund, Jerry Friedman, Trevor Hastie, and Rob Schapire. Thanks also go to the various members of MUGS, (K)(no)Bots and the PDP Lab who have put up with me over the years.

Colleagues at what used to be AT&T Bell Laboratories in Holmdel have been supportive before and during my studies. Many thanks to Sara Solla who has helped me much more than she imagines. I also wish to thank Jan Ben, Yann le Cun, John Denker, Hans Peter Graf, Isabelle Guyon, Larry Jackel and Edi Sackinger.

Friends at KZSU radio and the dance division have opened up worlds that I was unaware of when I first arrived at Stanford, and have been instrumental in adding spice to my residence here.

As luck may have it, my closest Israeli friends live right here in the Bay Area. For the past five years, Doron, Gili, Danielle and Haggar have opened their home and their hearts to me. Rinat and Vered have done the same in the past two years.

Adee Matan has been toiling away on her own dissertation on the other side of the continent. Though we are thousands of miles apart I feel her presence and love as if she were nearby.

My parents, Shalva and Avi, have been loving, encouraging and patient from afar. This has been as much an effort on their part as it has been mine. I dedicate this work to them.

Stanford, August 1996.

Contents

List of Figures

Chapter 1

Introduction

This dissertation studies topics in the use of ensembles for machine classification systems. In this chapter we introduce the subject area of machine classification, machine learning and the use of ensembles. We give a brief overview of the research described in the dissertation and list the main contributions.

1.1 Machine Classification

```
clas-si-fi-ca-tion \,klas-(e-)fe-'ka^--shen\ n (1790)
1: the act or process of classifying
2a: systematic arrangement in groups or categories according to established
     criteria; specif: TAXONOMY
2b: CLASS, CATEGORY
```

-Merriam Webster's Collegiate Dictionary

One of the goals of Computer Science and Engineering is building computers that can perform tasks without human intervention. Many of these tasks can be viewed as classification problems. For example if one needs to decide whether to cross a street - the relevant classification problem would be "is it either a safe or a dangerous situation". Applications of classification encompass diverse areas such as biology, banking, and mobile robotics. For example, a digit classifier as shown in Figure 1.1 might be used to read the sum from a check and either debit or credit a bank account accordingly. Alternatively, it may be used to read a handwritten ZIP code from an envelope and direct it into a bin of mail going to the proper destination.

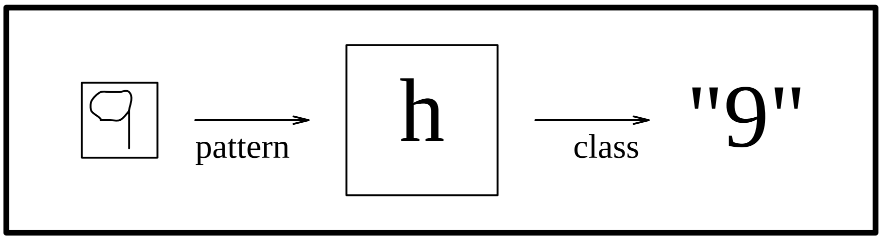

Figure 1.1: An example of a classifier. The classifier h receives a pattern and returns a class label. In this particular example he pattern is the image of a handwritten number.

A **machine classifier** is a system that receives a description of an object or situation and responds with an association of the object to one of a set of predefined class labels. Figure 1.1 portrays a schematic of a Machine Classifier that classifies handwritten digits. The classifier receives an image of a handwritten digit depicting the digit 9. In this case, the classifier returns the numeral 9 which is the correct class label. Had it responded with the numeral 3, it would have been in *error*. We would like to build classifiers that make no errors (100% accurate), but in practice this is often unachievable. The primary goal of the research in this dissertation is to design techniques to improve the accuracy of state-of-the-art classifiers.

1.2 Supervised Classification Learning

How are machine classifiers constructed? There are two main methodologies:

1. **Build it yourself**. This is the approach taken by *expert systems* (Buchanan and Smith, 1988): the knowledge of a human expert in the field is coded into rules (You, the reader, could be considered an expert in digit recognition. There are probably fewer experts able to recognize fraudulent \$100 bills and even fewer experts able to recognize certain types of fauna). The coded rules are used by a machine classifier to return the class label. For example a possible digit recognizing rule may be

   $$\textit{Two cavities in the ink pattern} \Rightarrow \text{“8”}$$

   This rule is a reasonable rule for recognizing the digit eight, but it is far from perfect. On one hand (no pun intended) some people write the digit 8 without closing the top

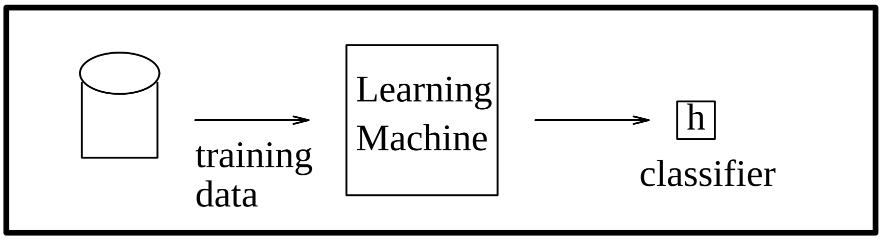

Figure 1.2: A learning machine. A supervised learning machine or learning algorithm takes a training set composed of labeled patterns and returns a description of a classifier

and on the other hand some write the digit 2 such that it has two cavities.

2. **Learn by example**. Humans are incredibly good at learning by example. This fact has not escaped researchers in AI and pattern recognition, who wish to enable computers to perform classification tasks as well as humans. Teaching by example may be the only way to build classifiers in domains where the rules are unknown or hard to express.

The work in this dissertation is chiefly concerned with the latter methodology. A supervised classification learning algorithm or **learning algorithm** or **learning machine** receives a set of patterns and their corresponding class labels. The algorithm returns a classifier termed **hypothesis** to be used in the classification task. A schematic of a learning machine is shown in Figure 1.2.

A learning algorithm returns the classifier in a language that is often called the **representation**. The choice of a good representation is crucial to the ability of the learning process to produce an accurate classifier. Examples of different representations are a linear separator, a neural network, and a decision tree.

1.3 Ensembles

A common theme to the work presented in this dissertation is the use of ensembles of classifiers. An **ensemble** is a collection of classifiers that are potentially different. The ensemble collaborates together to come to a decision as shown in Figure 1.3. We term the individual members of the ensemble **base-level** classifiers. The use of ensembles or committees has been studied practically from the advent of machine learning research (Nilsson, 1965) and

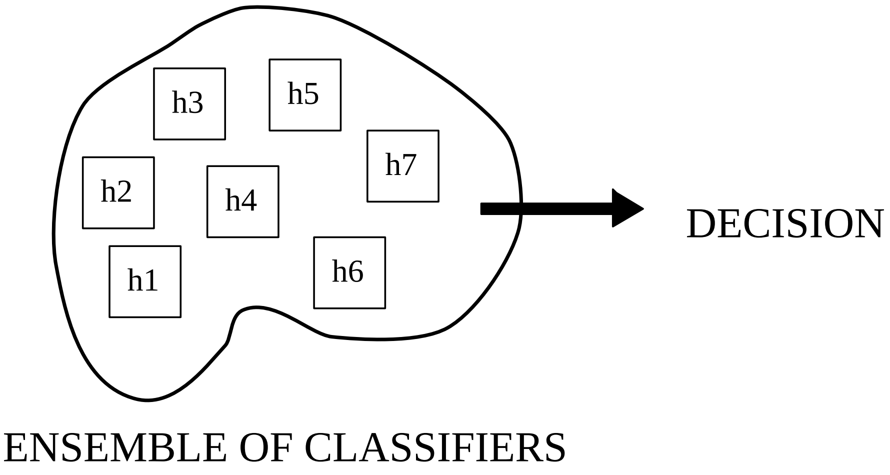

Figure 1.3: An ensemble of classifiers. In the following chapters we study models where the answers of individual classifiers in an ensemble are combined to come up with a decision. In Chapters 3 and 4, the decision is a class label. In Chapter 5, the decision is whether to request the class label of a given pattern.

has recently gained renewed interest. Not surprisingly, the topic has also been studied in the area of Social Science (Arrow, 1963; Black, 1958; Arrow and Raynaud, 1986).

This dissertation considers two uses for ensembles. In Chapters 3 and 4, ensembles are used to generate classifiers with higher classification accuracy than their individual members. In Chapter 5 the ensemble is used to minimize human-intervention in the learning process and to allow learning during the performance task.

It is useful to characterize an ensemble of classifiers along two characteristics:

1. How are the answers of the individual classifiers combined?

2. What are the building blocks of the ensemble? What is the representation used by the individual members and how are the individual members created or learned?

In this work, we consider ensembles in which each member has one (equal) vote. The equally weighted ensemble is attractive since it is:

(a) Simple to use.

(b) Simple to analyze.

Related work has considered more complex methods of combination such as weighted voting (Hastie and Tibshirani, 1990), gating (Jacobs et al., 1991) and hierarchical models (Jordan and Jacobs, 1994).

The various pieces of work in this dissertation differ on the uniformity among the building blocks: Chapter 3 makes no assumptions on the representations of the individual classifiers or on how they are created. Chapters 4 and 5 assume that all members belong to the same representation space and are created by the same learning algorithm. the following table compares the work presented in Chapters 3-5.

	same representation	same learning algorithm	same training data
Voting (Chap 3)	No assumption	No assumption	No assumption
Boosting (Chap 4)	Yes	Yes	No
QBC (Chap 5)	Yes	Yes	Yes (*)

(*) For certain cases we relax this to training data drawn from the same distribution.

1.4 Organization

Chapter 3 is concerned with constructing a new classifier from an ensemble of classifiers. The combined classifier makes a classification choice based on votes from all of its members and decides according to the majority. Previous work (Perrone and Cooper, 1993; Krogh and Vedelsby, 1995) has strengthened the perception that combining can, on average, only enhance the correctness of the classifier in comparison with any individual. We show that this is false and we present closed form, exact bounds on the accuracy of the majority vote ensemble as a function of the accuracies of its individual members. The analysis focuses on the representational power rather than the method used to create each of the classifiers. The analysis is relevant to machine learning algorithms since it exposes the conditions under which classifiers combine in the best possible fashion.

This last point leads to Chapter 4 which is concerned with meta-level, boosting algorithms that use base-level learning algorithms and try to combine the results to give better classifiers. The method used is manipulation of the training data passed to the base-level algorithm. We study a particular learning algorithm (Schapire, 1990) and present an analytical model of the qualities necessary for it to succeed. We also present a new boosting method that allows training with fewer training resources.

Chapter 5 introduces a model for learning during the performance task. The idea is to capitalize on the current competency of a classifier, perform the task on areas of certainty and continue training on areas of uncertainty. We propose an ensemble algorithm that uses the disagreement among an ensemble of classifiers to identify areas of uncertainty. We give experimental results showing the strengths and weaknesses of the algorithm on synthesized and real data.

We conclude in Chapter 6 with a review of the work presented and its place in a bigger context.

1.5 Principal Contributions

The following is a list of contributions made in this dissertation:

1. Proof that a majority-vote ensemble may have lower accuracy than all individual members.
2. Proof of tight upper and lower bounds on the accuracy of majority-vote ensembles.
3. Analysis of Schapire's boosting algorithm (Schapire, 1990) under conditions where there is no "weak" learning.
4. Introduction of a rejection boosting algorithm that uses fewer labels at a moderate loss in accuracy.
5. Introduction of the on-site learning model that allows learning during the performance task.
6. Experimentation with the use of an ensemble method suggested in (Seung, Opper and Sompolinsky, 1992). This method is used for on-site learning and for minimizing labeling costs.

1.6 Summary

The underlying motivation for the work in this dissertation is developing methodology for improving the accuracy of state-of-the-art machine classifiers. The focus of the research has not been any particular machine learning algorithm or representation. Rather, we have studied meta-level techniques that combine base-level classifiers into more complex ensemble classifiers. This makes the methods and analysis applicable to a variety of different learning algorithms and representations.

Chapter 2

Preliminaries

In this chapter we cover basic concepts and definitions we shall use later. We introduce some methodology and additional advanced concepts.

2.1 Basic Concepts

2.1.1 Instances, labels and features

The handwritten image in Figure 1.1 is termed the **input pattern** and is often referred to as either an **instance**, a **pattern**, or an **example**. The classifier outputs a **class label** or **label**.

An input pattern is usually composed of a set of features. In the above example, the features are the individual pixels in the image. Each feature indicates whether there is any ink in that particular area. Often machine applications apply an extra level of **preprocessing** or **feature extraction** that transforms the original representation into one more fitting the machine classification task (Duda and Hart, 1973). In the OCR example the pre-processed features may be width of strokes, number of strokes, average curvature and center of mass.

2.1.2 The target concept

For a given classification task, the **target concept** is a function from the instance domain to the "true" class label. The target function may not be known, but the underlying assumption is that it exists.

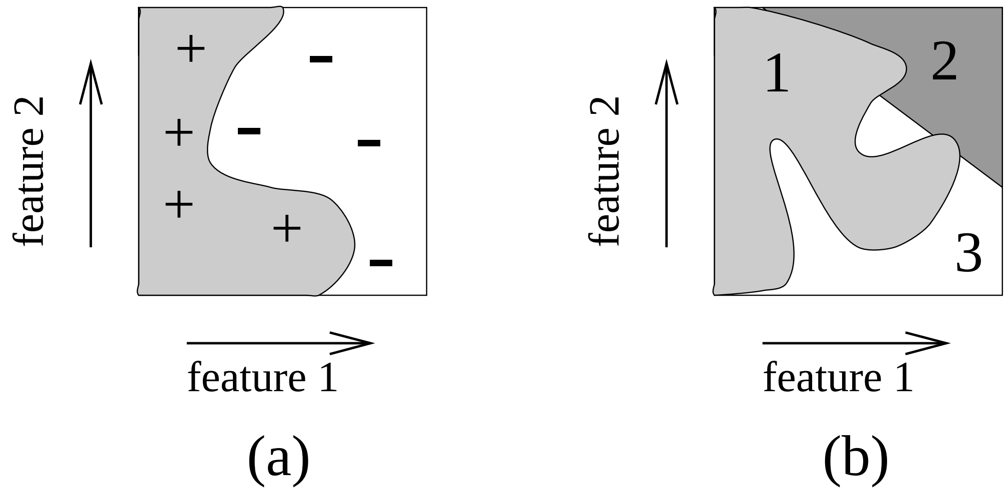

Figure 2.1: Target concepts. (a) A two-dimensional binary target concept. Positive instances are colored grey; negative instances are colored white. (b) A two dimensional ternary target concept.

For example in a handwritten-digit application the target function would associate an image with a 0-9 digit. In a gender classification problem there could be two classes (i.e. a binary classification problem): male or female.

In many domains the target concept is not a deterministic function, that is, there is a probability associated with each class label. In these cases we may define the target concept as the label we expect the classifier to return. We will address this issue further in section 2.4.1.

Throughout the thesis we demonstrate ideas graphically. We do this using binary target concepts over either a one-dimensional or a two-dimensional domain. While many ideas in this thesis will be demonstrated with binary target concepts, they hold for multi-class concepts as well. An example binary target concept is shown in Figure 2.1a and ternary target concept is shown in Figure 2.1b.

2.1.3 Correctness indicator functions

When a certain classifier assigns a label in agreement with the "true" label we say that the classifier is **correct**. Otherwise it is **incorrect** or is in **error**.

In some of the analyses to come, it will be beneficial to consider areas of the domain

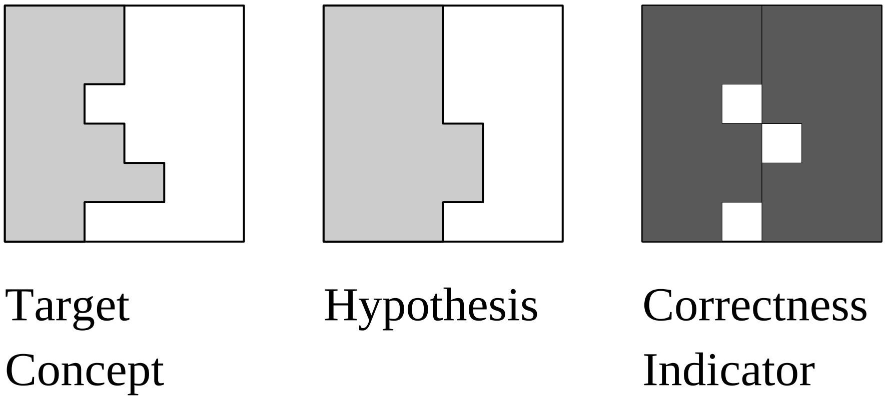

Figure 2.2: The correctness indicator function. The target concept is shown on the left. The classifications of a hypothesis or classifier is plotted in the center. This classifier is correct in most of the domain but is in error in three areas of the domain. This is demonstrated graphically by plotting the correctness indicator function for the hypothesis and the particular target concept.

where a given hypothesis is correct and where it is in error.

Given a hypothesis and a target concept, we define the **correctness indicator function** to be 1 where the classifier is correct (agrees with the target) and 0 when it is in error (disagrees with the target). The indicator function is more useful to deal with than the hypothesis itself. It enables us to concentrate on the correctness of a classifier and not on its particular output. Figure 2.2 plots an example of a target concept, a hypothesis and the corresponding correctness indicator function. The grey areas correspond to areas of correctness and the white to areas of errors. Assuming that all instances in the domain are equally probable, it is easy to read off the accuracy of the classifier as the fractional area of grey.

As we shall see in the following subsection, minimizing the areas that are incorrect is not the definitive criterion for evaluating a classifier.

2.1.4 Distributions, accuracy, fit and error

In any particular input domain, not all patterns are equally likely (see Figure 2.3) . Individual features in nature are often distributed according to the Gaussian distribution. In our postal example, more mail is sent locally than out of state. We therefore expect more

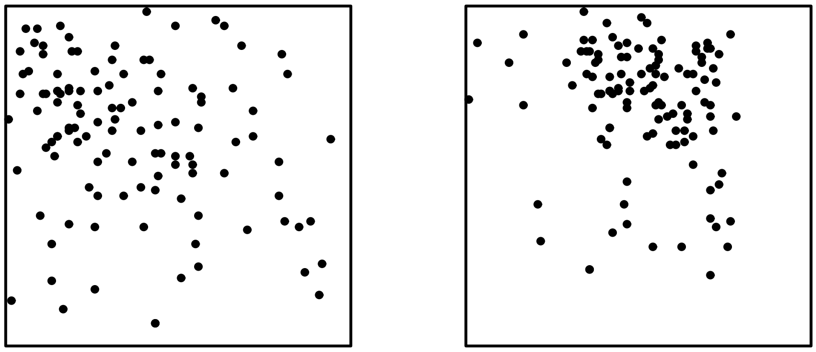

Figure 2.3: Not all distributions are equal. In this simple 2D domain, the distribution depicted on the left is skewed to the upper left quadrant , whereas the distribution on the right hand side is skewed toward the upper right quadrant.

"9"'s in California's mail stream.

The **accuracy** or **correct rate** of a classifier is the portion of patterns drawn from the distribution that are correct. The **error rate** is the portion of patterns that are in error. The accuracy is dependent on the distribution of patterns over the domain. For example our ZIP code reader's accuracy would decrease disproportionately if the classifier would have low recognition rates on the numeral "9". This point is demonstrated in Figure 2.4.

The lesson to learn is that machine classifiers should be built to be more accurate in domain areas where it matters. The fact that the classifier makes errors in cases that rarely occur is of less importance.[1]

We have just seen that **accuracy** differs from the **fit** which quantifies how close a hypothesis is to the target concept. Formally, the fit is the accuracy under a uniform distribution. We shall use this distinction in Chapter 4.

2.2 Building Machine Classifiers: Machine Learning

Chapters 4 and 5 are concerned with building machine classifiers using supervised learning techniques. In this section we present the fairly common steps of building a machine classifier

[1]In fact, this depends on the cost or utility of making an error, but we shall ignore this point for the moment.

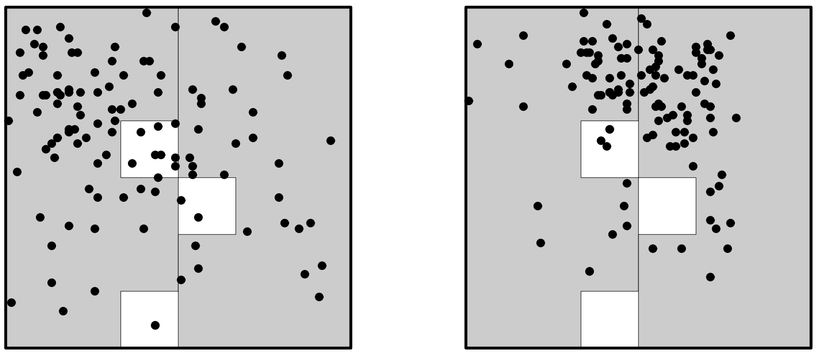

Figure 2.4: The accuracy depends on the pattern distribution. The same classifier can have different accuracies dependent on the distribution of patterns. The correctness indicator functions shown above are equivalent. However the classifier on the left will have lower accuracy since more instances "fall" into the white areas where it is incorrect.

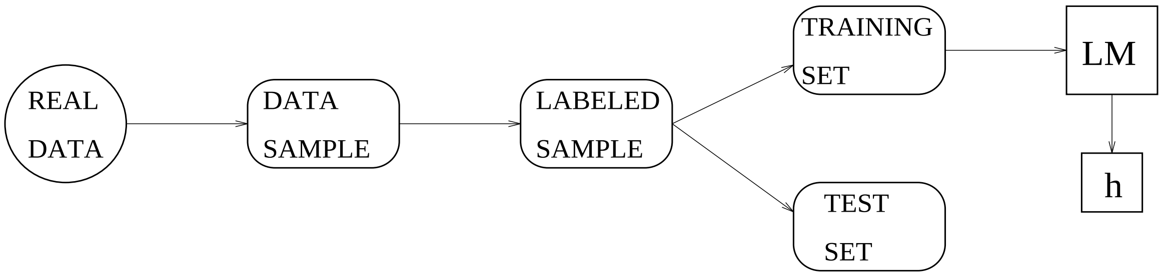

Figure 2.5: The methodology of building a classifier using machine learning. A corpus of data is sampled from the real world. The data is labeled with the "true" class label and divided to training and test data. A learning machine (LM) is applied to the training data and results in a classifier h. The classifier is tested using the test set. The accuracy on the test set is an estimate of the accuracy on the real world data (the "generalization" accuracy).

using a learning algorithm. Figure 2.2 depicts these steps. As an example, let us assume our application requires reading the dollar amount on bank checks.

1. Collect data from the real application distribution. For this purpose we may collect a week's worth of check images from the bank's processing center.

2. Label the digit images with the "true" class label. This is sometimes done automatically, but usually necessitates a human expert to look at the instance and name the correct class label. In our example, an operator would view each check and would type in the correct amount. Note that often this process is error-prone, leading to "noise" in the training and test patterns.

3. This corpora of labeled data is then usually divided into the following subsets:

   - A **training set**
   - A **test set**

   Much work has been done on the methods for dividing and using these sets. The reader is referred to (Weiss and Kulikowski, 1991; Langley, 1995; Kohavi, 1995) for details.

4. A model or models are generated by a learning algorithm using the training set. Examples of common learning algorithms are instance-based methods (Dasarathy, 1991), tree-induction methods (Breiman et al., 1984; Quinlan, 1993), or back-propagation (Rumelhart, Hinton and Williams, 1986). Often a **validation set** is held out of the training set for the purpose of choosing among the various models.

5. The accuracy of the hypothesis resulting from the learning stage is measured on the test set. This serves as an estimate of the accuracy of the hypothesis on the real application distribution (**generalization accuracy**). For various learning algorithms this estimate may be either optimistic or pessimistic (Kohavi, 1995).

2.3 Formal Definitions

In this section we introduce formal notation for some of the concepts we presented earlier. A *pattern* is denoted by $\vec{x}$ and is in some domain X which is typically a subset of R^n. The patterns are drawn according to a fixed distribution $\mathcal{D}$ over the domain.

The *target concept* for a binary classification problem is the function

$$C(\vec{x}) : X \rightarrow \{0,1\} \tag{2.1}$$

A *hypothesis*, $h(\vec{x})$, is also a function over the domain X:

$$h(\vec{x}) : X \rightarrow \{0,1\} \tag{2.2}$$

A given pattern $\vec{x} \in X$ is classified correctly by h if $h(\vec{x}) = C(\vec{x})$. The classifier is in error on pattern $\vec{x}$ if $h(\vec{x}) \neq C(\vec{x})$. The *correctness indicator function* for a given hypothesis equals 1 when the hypothesis is correct and 0 when it is in error:

$$\mathcal{I}_h(\vec{x}) = \begin{cases} 1 & h(\vec{x}) = C(\vec{x}) \\ 0 & h(\vec{x}) \neq C(\vec{x}) \end{cases} \tag{2.3}$$

The *accuracy* or *correct rate* of a classifier h for a given target concept and input distribution is the expected value of the correctness indicator function:

$$g_h = \int \mathcal{I}_h(\vec{x})\rho(\vec{x})d\vec{x} \tag{2.4}$$

where $\rho(\vec{x})$ is the probability density function associated with $\mathcal{D}$. The equivalent expression for a discrete distribution would be:

$$g_h = \sum_{\vec{x} \in X} \mathcal{I}_h(\vec{x})p(\vec{x}) \tag{2.5}$$

Where the probability density function $\rho(\vec{x})$ is replaced with a probability function $p(\vec{x})$.

The *error rate* is just one minus the accuracy:

$$\epsilon_h = 1 - g_h \tag{2.6}$$

2.4 Advanced Concepts

In this section we introduce some concepts that will be useful later in this dissertation.

2.4.1 Non-deterministic target functions

Often the target function is not deterministic. There are two main cases where this occurs:

- Various classes overlap. This is often the case in natural species where variation and mutation cause different species to have similar features. Note that often overlapping classes are only an illusion due to inspecting an insufficient set of features. That is, there exists a set of features over which there is a deterministic target function and only when a subset, or projection, of the features are inspected do the classes seem to overlap.

- There is noise in the process. When dealing with perception, by either human or machine, there is bound to be error in measurement often referred to as **noise** (Cover and Thomas, 1991). Even though the "true" underlying target function is deterministic, errors in measuring or transmitting either the feature values or the class label may cause the underlying function to *seem* non-deterministic.

In both cases it inherently impossible to achieve 100% accurate classifiers. The best accuracy possible is that of the Bayes optimal classifier (Duda and Hart, 1973) where one chooses the most likely class for each instance.

For most of this dissertation, analysis is done under the assumption that the target function is deterministic. The results carry through when there is non-determinism, but in these cases the best accuracy one can hope to achieve is that of the Bayes optimal classifier.

2.4.2 Confidence levels and rejection

The basic classification model presented in section 1.1 can be extended to include confidence levels. Instead of a class label, the classifier outputs a tuple consisting of the class label and a value representing the confidence in the classification. Classifiers of this type allow the system to reject some of the patterns with lower confidence scores. For example in Figure 2.6 an ambiguous digit is presented to the classifier. The classifier returns a class label with a low score.

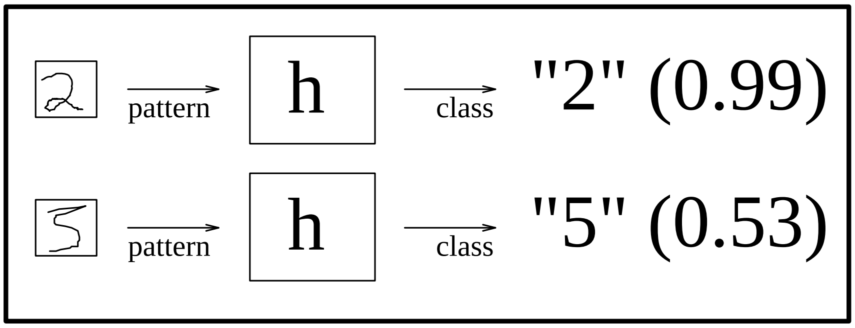

Figure 2.6: The classification model can be extended to include confidence levels. The classifier when presented with a pattern, returns a class label and a score representing its confidence in the correctness of its classification. The top pattern is clearly the numeral 2 and therefore the classifier return a high confidence (let us assume that the top score possible is 1.00). On the other hand, the bottom pattern could be either the numeral 3 or the numeral 5, the classifier decides it is most likely to be a "5", but gives this classification a low score. Depending on the application, this pattern may be rejected and/or passed for classification by a human operator.

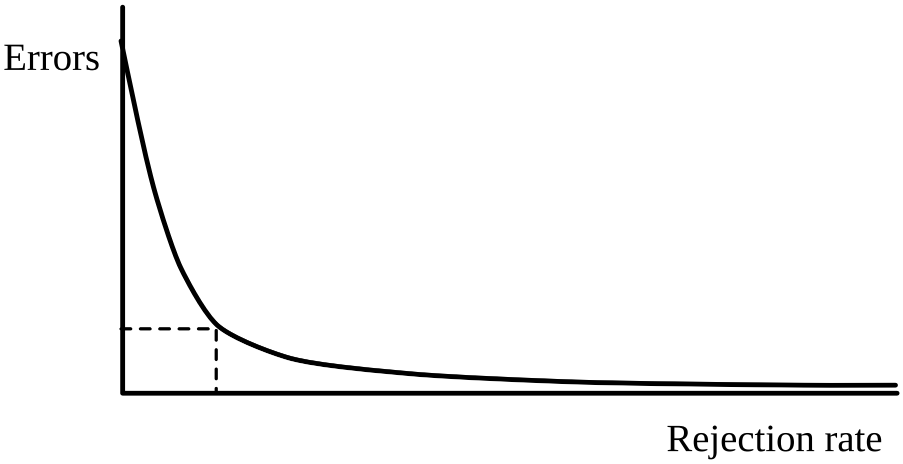

Figure 2.7: The typical form of error rate versus rejection rate. If the confidence score of a classifier is reliable, as one rejects more of the low confidence patterns, the number of errors among the accepted patterns will decrease. If we reject all patterns there are no errors, if no patterns are rejected, the error rate is just the raw error rate of Equation 2.6 often named the *raw error rate*.

Figure 2.7 shows the typical form of the error rate as a function of the rejection rate. If the error rate is computed as a fraction of the total number of patterns, then it is a monotonically decreasing function of the rejection rate.

If the confidence function is "well behaved" then its value correlates with the probability of error associated with the class label. Rejection of low confidence patterns will lead to a higher performance of the classifier on the patterns that are accepted. This is extremely important when designing a real-world system: the system can be tuned to a point on the error versus rejection graph so that one optimizes some payoff.

Let us take as an example a ZIP code reading system: classifying by human costs 20 cents, classifying by machine costs 5 cents. However, making an error ("Sorry but the letter was sent to Timbuktu") costs 50 cents. Therefore it may pay to be conservative and machine classify only high confidence patterns and pass low confidence ones to a human (Assuming the human has lower error rates than the machine). We explore this notion further in Chapter 5.

Some classifiers output confidence levels which may be estimates of the probability of the pattern belonging to the class label (El-Jaroudi and Makhoul, 1990; Matan et al., 1990). Therefore, the classification model can be expanded to return a vector where each element corresponds to the probability that the input pattern belongs to that class. In the case of digit recognition, the output would be a vector of length ten with each element corresponding to the probability that the input pattern is a certain digit 0-9.

2.4.3 Learning as a search problem

Consider the set of all possible hypotheses described by a particular representation. This is termed the **hypothesis space**. It may be either finite or infinite. A learning algorithm takes a training set and returns a hypothesis drawn from this hypothesis space. It is often useful to view the learning process as a search through the hypothesis space for the target function.

In real-world problems it is unknown whether the target concept can be expressed by the chosen representation, i.e. the target concept may not be in the hypothesis space. If the representation is unable to express the target concept we term it as **weak** (Kearns and Valiant, 1989) or lacking **capacity** (Vapnik and A., 1971; Vapnik, 1982). When a representation lacks capacity, the goal of the learner is to find, within the hypothesis space, the hypothesis that has the highest accuracy.

By choosing a specific representation an engineer building the learning machine has introduced a **representational bias**. In fact a bias is necessary in order to make an inductive "leap" (Mitchell, 1982). It is however possible that a hypothesis space includes the target concept, but the learning algorithm does not search that part of the space. This is often referred to as a **search bias**.

In this work we consider ensembles of classifiers. An ensemble of base-level classifiers defines a new meta-level hypothesis space. We analyze this space both in terms of its representation (Chapter 3) and in terms of search through it (Chapter 4).

Version spaces

In learning problems where the hypothesis space includes the target concept, one can define the **version space** (Mitchell, 1977) as the collection of hypotheses that are still consistent with the training data. Each additional training instance can potentially reduce the size of the version space.

In cases where the target function is not included in the hypothesis space (either due to noise or a weak representation), one can maintain a **model space** which is a Bayesian approach to the same problem: each hypothesis in the hypothesis space is assigned a probability that it is the target function. The version space is just a "hard" version of a model space.

There is similarity between learning a non-deterministic target function and learning with a weak representation: in both cases we cannot achieve 100% accuracy and often the behavior of learning algorithms is similar. The main difference is that one cannot overcome the inherent nature of non-determinism while one can enhance weak-representations to gain better classification performance.

2.4.4 Active learning

In the standard model of supervised learning as described in Section 2.2, the learning algorithm has no control over the training patterns. **Active learning** methods exert some control over the *way* they learn. Various examples of active learning are:

- Querying a teacher on a particular pattern (Angluin, 1988; Baum, 1991; Kinzel and Rujan, 1990; Sollich, 1994).

- Performing an action in the world to perceive a required training instance (Cohn, Ghahramani and Jordan, 1995). This bears relation to Experimental Design (Federov, 1972).
- Pruning the training set to a smaller set (Plutowski and White, 1991).
- Filtering a training set (Cohn, Atlas and Ladner, 1994; Lewis and Catlett, 1994).
- Using both labeled and unlabeled samples (Castelli and Cover, 1995).

The algorithms studied in Chapters 4 and 5 are forms of active learning.

Chapter 3

Majority Vote Ensembles

3.1 Introduction

In this chapter we study the representational power of voting ensembles of classifiers. An ensemble of classifiers is a set of potentially different classifiers. A voting ensemble classifies a test pattern by letting each member cast a single vote for the correct class and deciding according to a simple majority or special majority vote. This model is depicted in Figure 3.1.

We give upper and lower bounds on the classification accuracy of a voting ensemble as a function of the classification accuracies of its individual members.

In this work we are concerned with the representational power of voting classifiers: what are the best and worst combinations of individual classifiers, and what are the qualities of such combinations. Our results give insight, available to learning algorithms that search in this representation space. This work is of particular relevance to boosting algorithms that search the space of voting ensembles (Freund, 1992; Freund and Schapire, 1995). We consider boosting algorithms in Chapter 4.

Conventional wisdom is that "Two heads are better than one". This manifests itself in many walks of life and across different cultures: work committees, juries, parliamentary government, etc. The idea is that decisions made by a collection of individuals contain less risk. This idea has not been overlooked in the machine learning and pattern recognition communities.

Most practitioners have compared different classifiers on the same test data only to realize that although different methods may be close in their accuracy, they may differ on the particular errors they make. In an ideal situation a composite classifier would be able

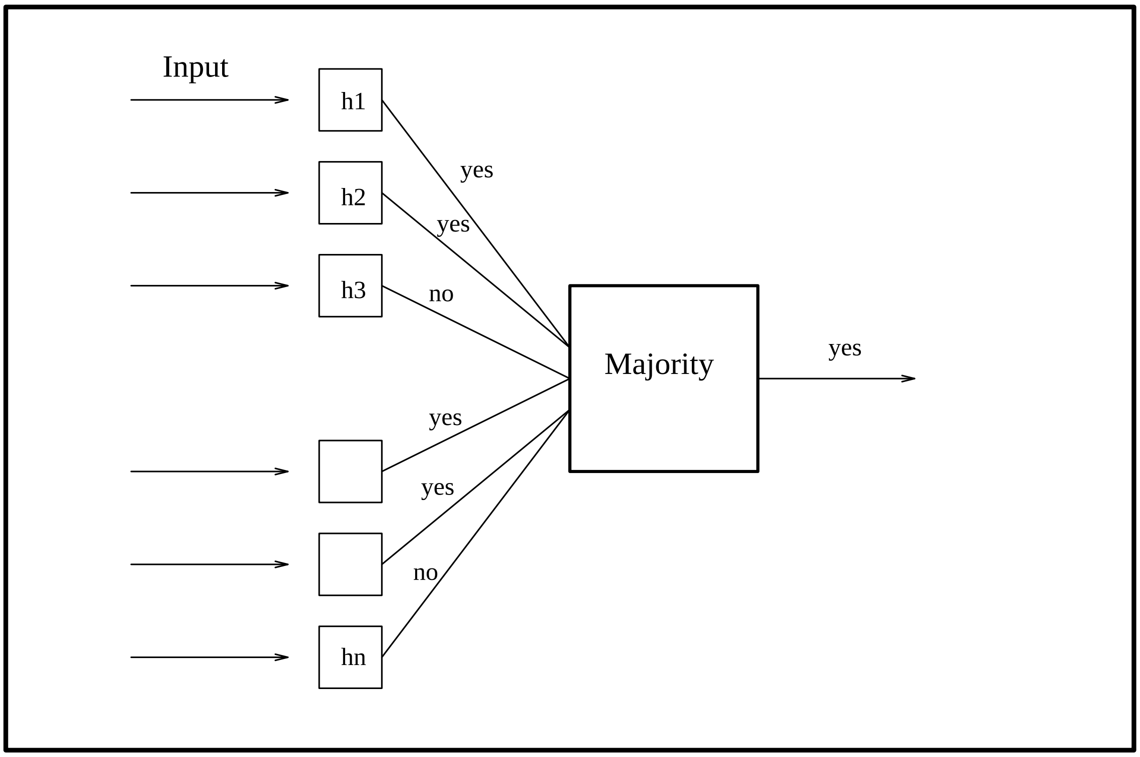

Figure 3.1: A majority-vote ensemble classifies by passing the input pattern to each of the individuals in the ensemble and choosing the class label according to a majority.

to outperform each individual classifier. A considerable amount of work has been done on combining classifiers. Perhaps the simplest method is majority voting. This method can often be successful. An interesting extreme use of this technique is (Ho, Hull and Srihari, 1990) where an ensemble of 41 different classifiers was used in a character recognition task.

3.2 Can Majority-Vote Ensembles Improve Accuracy?

What do we mean when we say that a majority-ensemble improves accuracy?

There are three possible formal definitions:

- Improve over the best case: the ensemble is at least as accurate as any individual in the ensemble.
- Improve over the average case: the ensemble is as least as accurate as the average individual classifier
- Improve over the worst: the ensemble is at least as accurate as the least accurate individual.

The first two definitions are reasonably strong. The final definition is quite weak. In the following example we show that even the weakest claim is false:

Example 1 (Generalized XOR)
Consider the target concept of a generalized XOR function as shown in Figure 3.2. Let us study a certain ensemble $\mathcal{E}_1$ of 3 classifiers shown below the target concept. The concept described by each classifier is shown. Each of the three ensemble members perform at a correctness rate of 50%. However, the concept described by the majority classifier, has a correct rate of only 25%, which is lower than any individual's accuracy. Figure 3.3 shows another ensemble, $\mathcal{E}_2$, in which each individual performs at a 50% correct rate. The concept described by this majority ensemble classifier achieves an accuracy rate of 75%.

Why do the two ensembles perform differently? Both ensembles are composed entirely of members performing at 50% correct. To answer this question we study the correctness indicator functions of the ensemble members. The indicator functions are plotted below their respective classifiers in Figures 3.2 and 3.3. Areas in which the classifier is correct are colored grey and areas in which it is in error are colored white.

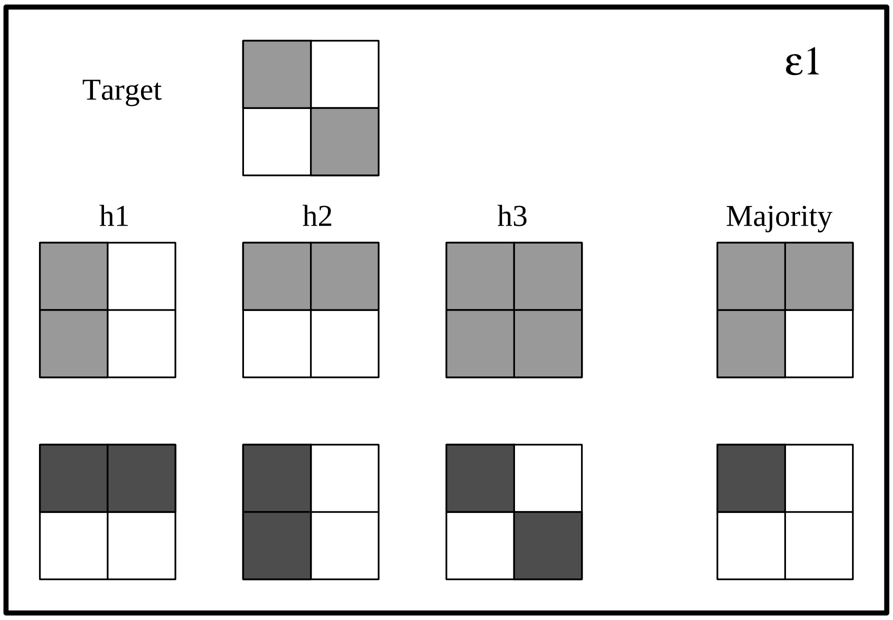

Figure 3.2: Ensemble $\mathcal{E}_1$. A majority ensemble composed of classifiers performing at 50% correct. $\mathcal{E}_1$'s combined accuracy is 25% which is worse than any individual in the ensemble. The bottom row plots the respective correctness indicator function.

In ensemble $\mathcal{E}_1$, the upper left quadrant is classified correctly by all ensemble members while all other quadrants are classified correctly by only one ensemble member. In both cases this is a "waste" of resources. First, the upper left quadrant needs only two ensemble members to classify correctly by majority. Second, the other correct answers of each ensemble member are not utilized due to participating in a minority. $\mathcal{E}_2$ which is an optimal combination "spends" only two ensemble members on each quadrant. Since each ensemble member is correct in only two quadrants (50% correct), a three member ensemble can classify 75% correctly by majority.

Ensembles $\mathcal{E}_1$ and $\mathcal{E}_2$ demonstrate the best and worst case combinations of three "50%" classifiers. This shall be demonstrated formally in the following sections where we present the upper and lower bounds on the accuracy of arbitrary majority-vote ensembles.

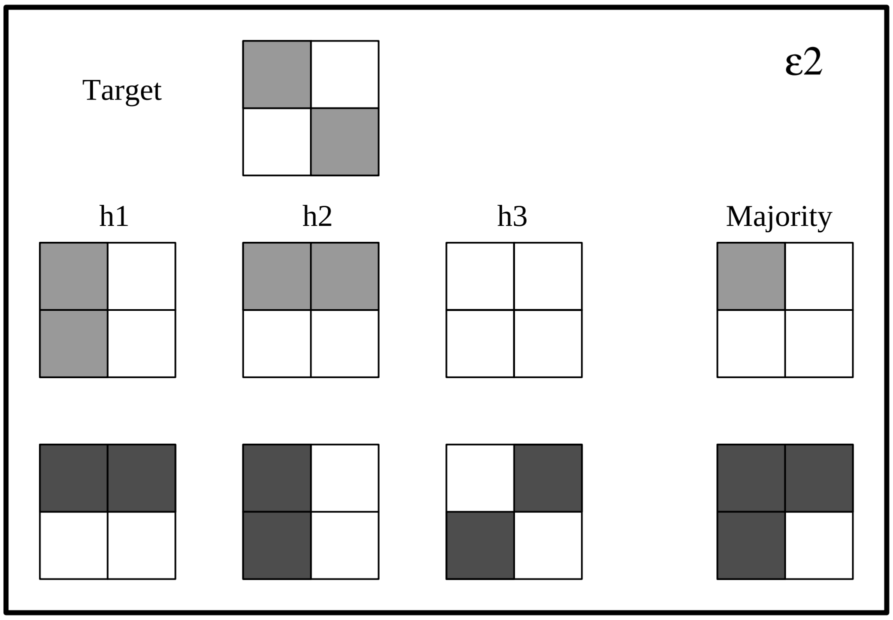

Figure 3.3: Ensemble $\mathcal{E}_2$. A majority ensemble composed of classifiers performing at 50% correct. $\mathcal{E}_2$'s combined accuracy is 75% which is better than any individual in the ensemble. The bottom row plots the respective correctness indicator function.

3.3 Definitions

Let the domain be a set X. The target concept is a boolean function $C : X \to \{0, 1\}$. Let an arbitrary classifier be denoted by $h : X \to \{0, 1\}$.

The *accuracy* of a classifier $h(x)$ is the expected value of the correctness indicator function:

$$g_h = \int_X \mathcal{I}_h(\vec{x})\rho(\vec{x})d\vec{x} \tag{3.1}$$

where $\rho(\vec{x})$ is the probability density function associated with distribution $\mathcal{D}$, a fixed and unknown distribution over the domain X.

Given an ensemble of n classifiers $\{h_1, h_2, ..., h_n\}$, where n is an odd integer, the majority classifier is defined as:

$$h(\vec{x}) = \begin{cases} 1 & \sum_1^n h_i(\vec{x}) \geq \frac{n}{2} \\ 0 & Otherwise \end{cases} \tag{3.2}$$

We generalize the notion of the majority classifier to a classifier with special majority by extending the range of the voting classifier to include an "unknown" response, $h : X \to \{0, 1, \mathtt{U}\}$. This is parallel to rejection we have seen in Section 2.4.2. For the purpose of this analysis we consider an "unknown" response as an error. Therefore, the accuracy measure in Equation 3.1 still holds.
A k-of-n special majority classifier for $k > \frac{n}{2}$ is defined as:

$$h_k^n(\vec{x}) = \begin{cases} 1 & \sum_1^n h_i(\vec{x}) \geq k \\ 0 & \sum_1^n h_i(\vec{x}) \leq n - k \\ \mathtt{U} & Otherwise \end{cases} \tag{3.3}$$

A majority classifier for odd n is just a k-of-n voting classifier with $k = \frac{n+1}{2}$.

In the following sections we ask the following question: *What is the best/worst possible accuracy of a majority-vote ensemble composed of n classifiers $h_1, h_2, \ldots, h_n$, with respective accuracies $g_1, g_2, \ldots, g_n$?*
We make the following notational convention: without loss of generality the classifiers are ordered in ascending order of their accuracy. *i.e.* $g_1 \leq g_2 \leq ... \leq g_{n-1} \leq g_n$.

3.4 Bounds as a Linear Programming Problem

The problem of giving the upper and lower bounds on the accuracy of a special-majority voting ensemble as a function of the individual members' accuracies can be formulated as a linear programming problem:

Given an ensemble $h_1, h_2, \ldots, h_n$, with respective accuracies $g_1, g_2, \ldots, g_n$ we can divide the domain into 2^n (not neccesarily contiguous) regions. Each region, R_β, corresponds to the collection of patterns on which a particular combination of classifiers is correct or incorrect. We label each region by a bit pattern , β representing which classifiers are correct and which are in error:

$$\beta \equiv \beta_1\beta_2....\beta_{n-1}\beta_n \tag{3.4}$$

For example in a 5-member committee, the region R_{10011} is composed of all patterns for which h_1, h_4, and h_5 are correct while h_2 and h_3 are in error. The portion of the distribution Ω_β in a certain region is equal to:

$$\Omega_\beta = \int_X [\mathcal{I}_1 = \beta_1] * [\mathcal{I}_2 = \beta_2] * \ldots * [\mathcal{I}_{n-1} = \beta_{n-1}] * [\mathcal{I}_n = \beta_n]\rho(\vec{x})d\vec{x} \tag{3.5}$$

The Ω's are constrained by the accuracies of the classifiers. In particular, the accuracy of classifier i is equal to:

$$g_i = \sum_{\{\beta|\beta_i=1\}} \Omega_\beta \tag{3.6}$$

In addition the Ω's cover the whole domain:

$$\sum_\beta \Omega_\beta = 1 \tag{3.7}$$

The k-of-n majority vote classifier is correct in regions R_β such that $(\sum \beta_i) > k$. Therefore the accuracy of the majority vote ensemble is:

$$g_k^n = \Omega_{maj} = \sum_{\{\sum \beta_i > k\}} \Omega_\beta \tag{3.8}$$

From the above, it is obvious that finding the upper or lower bound on the accuracy of a special majority-vote ensemble is a linear programming problem where we maximize/minimize the sum in Equation 3.8 subject to the constraints of equations 3.7 and

3.6.

Example 2 (3 member ensemble)
In the case of a 3-member ensemble the problem is formulated as follows:
maximize/minimize $c\Omega$ *subject to* $A\Omega = G$ *where*

$$A = \begin{pmatrix} 00011101 \\ 00100111 \\ 01001011 \\ 11111111 \end{pmatrix} \qquad G = \begin{pmatrix} g_1 \\ g_2 \\ g_3 \\ 1 \end{pmatrix} \tag{3.9}$$

$$\Omega = \begin{pmatrix} \Omega_{000} \\ \Omega_{001} \\ \Omega_{010} \\ \Omega_{100} \\ \Omega_{101} \\ \Omega_{110} \\ \Omega_{011} \\ \Omega_{111} \end{pmatrix} \qquad c = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 1 \\ 1 \end{pmatrix} \tag{3.10}$$

(3.11)

If we return to the example in the previous section where $g_1 = g_2 = g_3 = 0.5$ the solution to the minimization problem is shown in the following table:

		g_1	g_2	g_3
Ω_{000}	0			
Ω_{001}	0.25			0.25
Ω_{010}	0.25		0.25	
Ω_{100}	0.25	0.25		
Ω_{101}	0	0		0
Ω_{110}	0	0	0	
Ω_{011}	0		0	0
Ω_{111}	0.25	0.25	0.25	0.25
total	1	0.5	0.5	0.5

The first column is the solution for Ω which due to constraint 3.7 sums to one. The remaining three columns show the contributions of each classifier to each region and each sums to the classifier's individual accuracy. The ensemble's majority accuracy is the sum $\Omega_{maj} = \Omega_{101} + \Omega_{110} + \Omega_{011} + \Omega_{111}$ which in this case is only 0.25. Ensemble $\mathcal{E}_1$ is an embodiment of this solution.

The maximal solution embodied by $\mathcal{E}_2$ with $\Omega_{maj} = 0.75$ is shown below:

		g_1	g_2	g_3
Ω_{000}	0.25			
Ω_{001}	0			0
Ω_{010}	0		0	
Ω_{100}	0	0		
Ω_{101}	0.25	0.25		0.25
Ω_{110}	0.25	0.25	0.25	
Ω_{011}	0.25		0.25	0.25
Ω_{111}	0	0	0	0
total	1	0.5	0.5	0.5

From this example one may conclude that in order to maximize Ω_{maj}, the individual classifiers' resources should be "spent" primarily on Ω_{101}, Ω_{110} and Ω_{011}, only then on Ω_{111}, while avoiding spending on Ω_{001}, Ω_{010} and Ω_{100}.

One can solve this optimization problem for particular values of g_i using techniques such as simplex(Dantzig, 1963). An optimal solution in closed form will depend on the relationship between the various accuracies $g_1, \ldots, g_n$ and applying a technique like simplex is not practical. In the following section we present the closed form, upper and lower bounds on the classification accuracy of voting ensembles as a function of the individual accuracies of their members.

3.5 Accuracy Bounds for Voting Ensembles

3.5.1 Upper bound

Theorem 1 *Given an ensemble of n classifiers $\{h_1, h_2, ..., h_n\}$ with respective accuracies $g_1 \leq g_2 \leq ... \leq g_{n-1} \leq g_n$ under domain distribution $\mathcal{D}$, The accuracy g_k^n of an k-of-n special majority classifier under the same distribution is bounded from above by:*

$$\max(g_k^n) = \min(1, \Sigma(k), \Sigma(k-1) \ldots, \Sigma(2), \Sigma(1)) \tag{3.12}$$

where

$$\Sigma(m) \equiv \frac{1}{m} \sum_{i=1}^{n-k+m} g_i \qquad m = 1 \ldots k \tag{3.13}$$

Equation 3.12 can be reformulated to explicitly specify the conditions under which each of the terms is minimal:

$$\max(g_k^n) = \begin{cases} 1 & \Sigma(k) \geq 1 \\ \Sigma(k) & \Sigma(k-1) \geq g_n \\ \Sigma(k-1) & \Sigma(k-2) \geq g_{n-1} \\ . & \\ . & \\ \Sigma(k-j) & \Sigma(k-j-1) \geq g_{n-j} \\ . & \\ . & \\ \Sigma(2) & \Sigma(1) \geq g_{n-k+2} \\ \Sigma(1) & \Sigma(1) < g_{n-k+2} \end{cases} \tag{3.14}$$

The above expression should be read in an *if-then-else* fashion: g_k^n is bounded by the left-side expression corresponding to the topmost right-side inequality which is true. Equation A.15 in the appendix gives the precise disjoint conditions.

The full proof of this theorem is presented in Appendix A. The following is a sketch of the proof:

Proof (sketch)

The proof consists of the following steps.

1. Prove that g_k^n is bounded above by the right hand side of Equation 3.12. The main argument is a version of the pigeon-hole principle.

2. Enumerate the various cases of the bound. This done by proving that the right hand side of Equation 3.14 is equivalent to the upper bound of the previous step.

3. Show that there is a feasible solution at the bound (i.e. the bound is tight). This is done by double induction on the recursive form of Equation 3.14.

3.5.2 Lower bound

The lower bounds can be calculated from the upper bounds for binary concepts. The lower bound of the accuracy is computed by calculating upper bound on the inverse hypothesis (i.e. maximizing the error.)

Theorem 2 *Given an ensemble of n classifiers $\{h_1, h_2, ..., h_n\}$ with respective accuracies $g_1 \leq g_2 \leq ... \leq g_{n-1} \leq g_n$ under domain distribution $\mathcal{D}$, The accuracy g_k^n of an k-of-n special majority classifier under the same distribution is bounded from below by:*

$$\min(g_k^n) = \max(0, \xi(k), \xi(k-1)\ldots, \xi(2), \xi(1)) \tag{3.15}$$

where

$$\xi(m) \equiv \frac{1}{m} \sum_{i=k-m+1}^{n} g_i - \frac{n-k}{m} \qquad m = 1 \ldots k \tag{3.16}$$

This may be reformulated in the same manner as the upper bound to give the following:

$$g_k^n \geq \begin{cases} 0 & \frac{1}{k}(\sum_{i=1}^{n} g_i - (n-k)) \leq 0 \\ \frac{1}{k}(\sum_{i=1}^{n} g_i - (n-k)) & \frac{1}{k-1}(\sum_{i=1}^{n-1} g_i - (n-k)) \leq g_n \\ \frac{1}{k-1}(\sum_{i=1}^{n-1} g_i - (n-k)) & \frac{1}{k-2}(\sum_{i=1}^{n-2} g_i - (n-k) \leq g_{n-1}) \\ . & \\ . & \\ \frac{1}{2}(\sum_{i=1}^{n-k+2} g_i - (n-k)) & \sum_{i=1}^{n-k+1} g_i - (n-k) \leq g_{n-k+2} \\ \sum_{i=1}^{n-k+1} g_i - (n-k) & Otherwise \end{cases} \tag{3.17}$$

Proof

We define the inverse classifiers of each classifier of each ensemble member, and the inverse of the majority classifier:

$$\tilde{h}_i(\vec{x}) = 1 - h_i(\vec{x}) \tag{3.18}$$

$$\tilde{h}_k^n(\vec{x}) = 1 - h_k^n(\vec{x}) \tag{3.19}$$

The respective accuracies are:

$$\tilde{g}_i = 1 - g_i \tag{3.20}$$

$$\tilde{g}_k^n \quad = \quad 1 - g_k^n \tag{3.21}$$

$$\min(g_k^n) = \min(1 - \tilde{g}_k^n) = 1 - \max(\tilde{g}_k^n) \tag{3.22}$$

Note that $\tilde{g}_1 \geq \tilde{g}_2 \geq ... \geq \tilde{g}_{n-1} \geq \tilde{g}_n$. Therefore according to Theorem 1

$$\max(\tilde{g}_k^n) = \min(1, \tilde{\Sigma}(k), \tilde{\Sigma}(k-1) \ldots, \tilde{\Sigma}(2), \tilde{\Sigma}(1)) \tag{3.23}$$

Where

$$\tilde{\Sigma}(m) \quad \equiv \quad \frac{1}{m} \sum_{i=n}^{1+k-m} \tilde{g}_i = \frac{1}{m} \sum_{i=k-m+1}^{n} 1 - g_i \qquad m = 1 \ldots k \tag{3.24}$$

$$= \quad 1 + \frac{n-k}{m} - \frac{1}{m} \sum_{i=k-m+1}^{n} g_i \tag{3.25}$$

The proof if finalized by setting $\tilde{\Sigma}(m) \equiv 1 - \xi(m)$ and substituting Equations 3.24 into 3.23 and 3.22. ∎

3.6 Examples and Ramifications

Example 3 (A 3-member majority ensemble)
To better understand Theorems 1 and 2, we look at these statements for a simple 3-member majority-vote ensemble.

The accuracies of the three members are g_1, g_2, and g_3, where $g_1 \leq g_2 \leq g_3$. The correctness of the majority vote classifier is g_2^3. The upper bound on the correctness is:

$$\max(g_2^3) = \min(1, \Sigma(2), \Sigma(1)) \tag{3.26}$$

which is equivalent to:

$$\max(g_2^3) = \begin{cases} 1 & \frac{1}{2}(g_1 + g_2 + g_3) \geq 1 \\ \frac{1}{2}(g_1 + g_2 + g_3) & g_1 + g_2 \geq g_3 \\ g_1 + g_2 & Otherwise \end{cases} \tag{3.27}$$

The lower bound is:

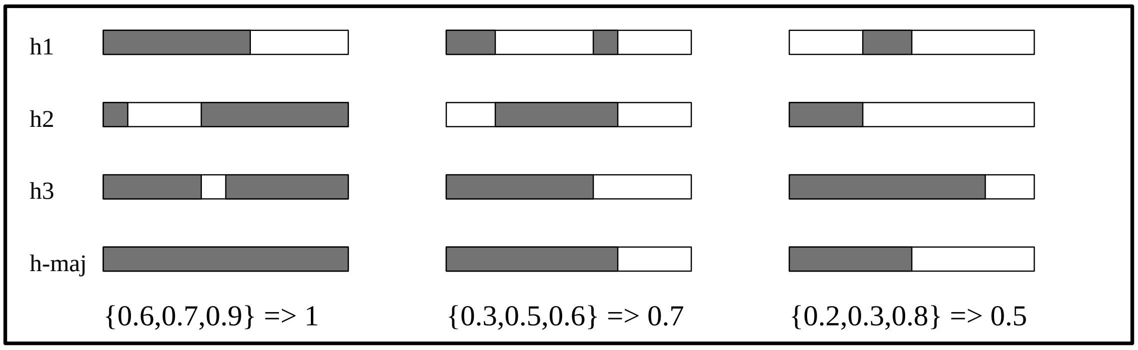

Figure 3.4: The upper bound on the accuracy for a 3-member ensemble. The three columns correspond to the three cases in Equation 3.27. Each column presents the correctness indicator functions of h_1,h_2 and h_3 to achieve a majority-vote ensemble with the highest accuracy possible with accuracies of g_1,g_2, and g_3. At the bottom of each column is a numeric example for each case. (1) The rightmost case occurs when the most accurate classifier has an accuracy greater than the sum of the other two. The upper bound in this case is just the sum of the two accuracies. (2) When the highest accuracy is smaller than the sum of the other two, it is possible to achieve an accuracy of half the sum of the accuracies (middle case) unless this value is greater than 1, which is the best one can achieve (leftmost case).

$$\min(g_2^3) = \max(0, \xi(2), \xi(1)) \tag{3.28}$$

which is equivalent to:

$$\min(g_2^3) = \begin{cases} 0 & g_1 + g_2 + g_3 \leq 1 \\ \frac{1}{2}(g_1 + g_2 + g_3 - 1) & g_2 + g_3 - 1 \leq g_1 \\ g_2 + g_3 - 1 & Otherwise \end{cases} \tag{3.29}$$

A graphical representation of the upper and lower bounds for a 3-member ensemble are shown in Figures 3.4 and 3.5. The solutions for Ω_β are given in Appendix B.

Example 4 (n-member majority ensemble where $\forall i\ g_i = g$)
An interesting case to analyze is an ensemble of n (where n is odd) classifiers of equal accuracy g. The upper and lower bounds on the majority vote classifier are:

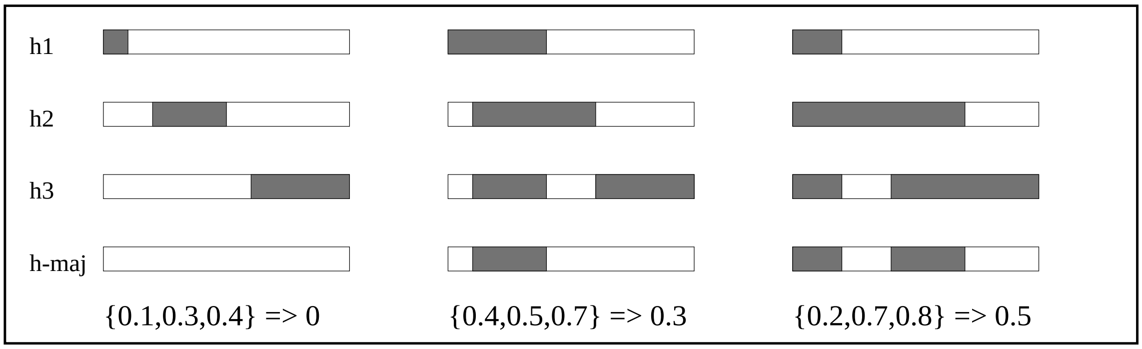

Figure 3.5: The lower bound on the accuracy for a 3-member ensemble. The three columns correspond to the three cases in Equation 3.29. Each column presents the correctness indicator functions of h_1,h_2 and h_3 to achieve h_2^3 with the lowest accuracy possible with accuracies of g_1,g_2, and g_3. At the bottom of each example are numeric examples for each case.

$$\max(g^n_{(n+1)/2}) = \begin{cases} 1 & \frac{2n}{n+1}g \geq 1 \\ \frac{2n}{n+1}g & Otherwise \end{cases} \tag{3.30}$$

$$\min(g^n_{(n+1)/2}) = \begin{cases} \frac{2n}{n+1}g - \frac{n-1}{n+1} & \frac{2n}{n-1}g \geq 1 \\ 0 & Otherwise \end{cases} \tag{3.31}$$

This example leads to some interesting results:

- A majority ensemble can improve the accuracy by at most a factor of two. One needs over 50% accuracy to achieve arbitrary accuracy (this is in accordance with the boosting results (Schapire, 1990; Freund, 1992; Freund and Schapire, 1995)).

- One can compute the lower bound on the number of ensemble members of accuracy g needed to achieve a given accuracy G :

$$n \geq \frac{G}{2g - G} \tag{3.32}$$

We return to this result in Chapter 4 when we discuss boosting algorithms.

Example 5 (Hierarchical structures)

Theorem 1 may be used to analyze complex structures such as the hierarchical structure

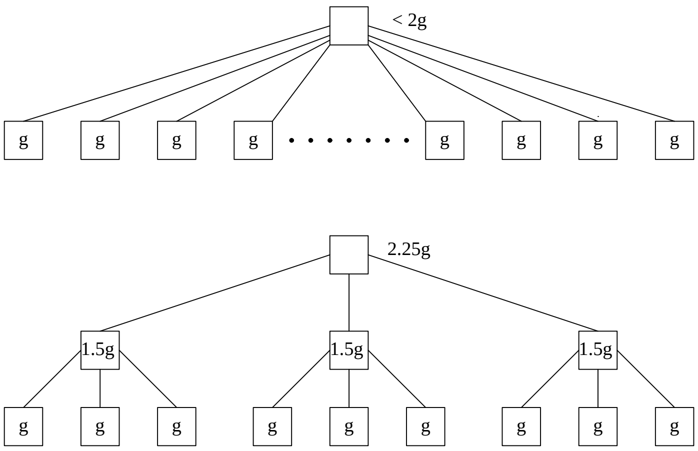

Figure 3.6: Hierarchical versus flat structures. Combining an arbitrary number of classifiers of accuracy g can give no more than accuracy $2g$ using a flat majority-vote classifier. However, using a hierarchical structure, 9 classifiers can be combined to achieve an accuracy of $2.25g$.

shown in Figure 3.6. In this structure each 3 classifiers combine to a 2-of-3 majority classifier. Each of the results of each of the 3 ensembles are combined again in a 2-of-3 majority vote. The well known fact that two level structures are more expressive follows from applying the bounds of Equation 3.14 twice. Combining only 9 classifiers with accuracy g one can construct classifiers that have accuracy of $2.25g$ which is larger than the bound for any number of classifiers combined in a flat majority-vote ensemble.

The reasoning is simple: consider the nine base-level classifiers in Figure 3.6. For a given pattern, it only requires four of the nine classifiers to be correct to cause the combined hierarchical structure to be correct: if the two left-most and two right-most classifiers are correct ([2,0,2]) then two of the first levels votes will be correct and therefore the second level classification will be correct. The flat majority vote classifier needs five of the nine to be correct.

Note however that additional expressiveness has a down side as well: the lower bound drops as well. Following our previous example, there is a configuration of five of nine base classifiers correct in the hierarchical ensemble, yet the final classification is incorrect ([3,1,1]).

3.7 Related Work

Combining estimators has been studied both theoretically and experimentally. In particular there has been work advocating the use of equally weighted majority vote. In this section we review some of this work.

3.7.1 Probabilistic analysis

In (Hansen and Salamon, 1990), the accuracy of an ensemble of classifiers is studied under a probabilistic modeling of the individual votes. The assumption is that each classifier arrives at the correct classification with probability proportional to its accuracy g. In addition each classifier is independent of the others. If all n classifiers have equal accuracy g, then the probability of of a k-of-n majority-vote classifier being correct is :

$$\sum_{k}^{n} \begin{pmatrix} n \\ k \end{pmatrix} g^{k}(1-g)^{n-k} \tag{3.33}$$

If we check this for a three member classifier this gives $3g^2(1-g)+g^3$ which falls between the lower bound of $g/2$ and the upper bound of $3g/2$.

3.7.2 Regression classifiers

Much of the work in combining ensembles has been done with combining regression classifiers. For example in (Perrone and Cooper, 1993; Krogh and Vedelsby, 1995) the model considered is one where the ensemble's individual results are averaged:

$$h_{ens}(\vec{x}) = \frac{1}{n}[h_1(\vec{x}) + h_2(\vec{x}) + \ldots + h_n(\vec{x})] \tag{3.34}$$

Classification by a regression classifier is done by comparing the value $h_n(x)$ to a threshold θ. For example, value below the threshold may be classified as class 0 and values above the threshold as class 1. below.

The most common performance measure used for regressors is the Mean Squared Error(MSE) measure:

$$MSE(h) \equiv \int_X (h(\vec{x}) - c(\vec{x}))^2 \rho(\vec{x}) d\vec{x} \tag{3.35}$$

It is well known that the MSE of an averaged ensemble is no larger than the average

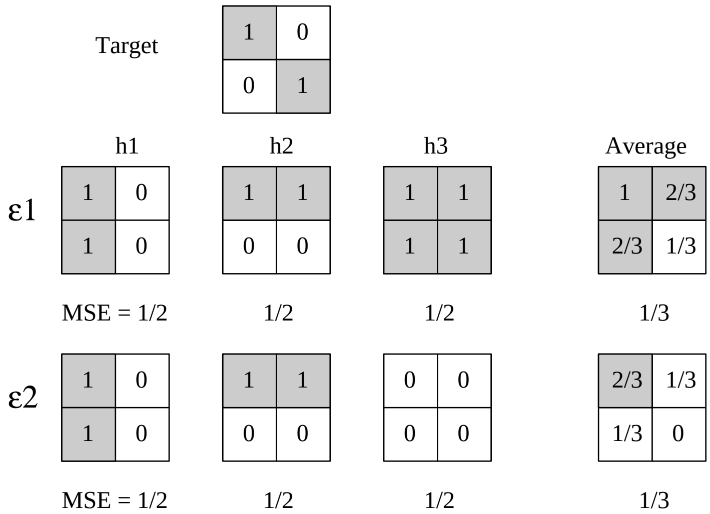

Figure 3.7: Combining $\mathcal{E}_1$ and $\mathcal{E}_2$ as regression classifiers.

individual MSE:

$$MSE(h_{ens}) \leq \overline{MSE(h_i)} \tag{3.36}$$

This states that whenever one uses averaging one can improve over the average performance of a single learner. This seems a powerful result which may mislead one to believe that combining regression classifiers alway give better results. In fact, if the relevant measure is the classification performance (zero-one loss error) combining regressors may lead to inferior performance. To demonstrate, we return to the generalized XOR example.

Figure 3.7 shows both $\mathcal{E}_1$ and $\mathcal{E}_2$ combined as regressors with the threshold $\theta = 0.5$. The classification behavior of averaging is equivalent to taking a majority vote. The MSE of the combined classifier is smaller than the average MSE. In fact the MSE for both is $\frac{1}{3}$. This demonstrates that the reduction in MSE for combining regressors does not necessarily bear on whether the combination will result in superior accuracy.

3.7.3 Other

We have studied combining classifiers by an equally weighted vote. Other more complex combinations have been suggested:

- Fixed non-equal weighted majority (Hastie and Tibshirani, 1990).

- Context dependent weighted majority. (Wolpert, 1992; Breiman, 1994; Jacobs et al., 1991)
- Many popular representations are composed of smaller building blocks. Therefore it is not hard to view neural networks, decision trees, nearest neighbor methods and other representations as an ensemble of simple classifiers.

3.8 Discussion and Future Work

We have given upper and lower bounds on the classification accuracy of majority-vote ensembles as a function of the accuracy of individual members. The proofs given are for binary classifiers, but hold for multi-class classifiers for the majority-vote rule ($k > \frac{n}{2}$). In multi-class tasks, it is possible that after tallying the ensemble votes, no class will achieve majority. In our analysis we have treated this case as a reject. Another possibility is the plurality-vote rule: choose the class with the most votes (Hansen and Salamon, 1990). The bounds in Theorems 3.12 and 3.15 do not apply for plurality.

The results of Theorems 3.12 and 3.15 are best and worst case scenarios and not necessarily typical: for example, the results state that there exists a majority ensemble of 90% classifiers such that the majority ensemble achieves 100% accuracy. Any practitioner knows that combining three 90% classifiers rarely changes the overall accuracy by much. This is due to the fact that the classifiers are too similar and agree on the majority of the domain.

More realistic bounds can be calculated by incorporating a measure of agreement, A, between the the members of the ensemble and applying the same analysis on the residuals after agreement. For example, consider a majority classifier composed of three $g = 90\%$ classifiers when they are unanimous in their vote on 91% of the domain. It is possible to show that the bounds on the majority classifier h_2^3 are $87\% \leq g_2^3 \leq 93\%$. Proving a closed form result for arbitrary size ensembles with arbitrary accuracies still remains to be achieved.

The bounds given in this chapter are of interest to learning algorithms that generate ensembles. The representational analysis gives insight into the way one should combine weak representations into strong representations. This will be discussed further in the following chapter.

Chapter 4

Boosting Algorithms

In the previous chapter we analyzed the theoretical representational ability of majority vote classifiers. In order to learn a majority-vote classifier, it is necessary to search within this representational space and provide a *particular ensemble*. Boosting algorithms are algorithms that build ensembles of base-level classifiers. The goal is to construct a classifier, based on the ensemble, that has higher accuracy than the base-level classifiers. In this chapter we discuss a number of different boosting algorithms and their use in practice. We give both relevant formal analysis and experimental results.

In particular, we study an algorithm suggested by Schapire (SBA)(Schapire, 1990). We consider the algorithm when true "weak" learning does not exist. That is, the boosting algorithm is no longer guaranteed to achieve arbitrary "strong" learning. SBA is analyzed in terms of its representation and its search. We show that the representation is a piecewise tiling of the domain and that if the weak learner has a low representational complexity ability, SBA's search may fail to boost or may give a sub-optimal solution.

We introduce a rejection boosting algorithm (RBA) (Matan and Solla, 1995) that trades-off exploration and exploitation. It requires fewer pattern labels at the expense of lower boosting ability. The motivation for such an algorithm is the large amount of labeled data that SBA uses. In cases where the classifier returns a confidence level (see section 2.4.2) we may use the rejection boosting algorithm to boost with fewer labeled patterns.

4.1 Introduction

Figure 4.1 gives a schematic view of a boosting procedure. In very loose terms, boosting algorithms make "strong" classifiers out of multiple "weak" classifiers. The boosting algorithm is a meta-level algorithm that uses a base-level "weak" learning algorithm as a black-box. The base level algorithm may be an arbitrary learning algorithm such as a perceptron-learning algorithm (Nilsson, 1965; Duda and Hart, 1973), a decision-tree learning algorithm (Breiman et al., 1984; Quinlan, 1993) or back-propagation (Rumelhart, Hinton and Williams, 1986).

A boosting algorithm is usually an iterative procedure. It filters through a training data source, manipulates the data and passes it onto the base-level algorithm with a possibly modified distribution. In that sense, boosting methods are active learning methods in that they skew the distribution of training patterns to their advantage. At each iteration the base level algorithm returns a classifier which is collected by the boosting algorithm. The distribution is modified in successive calls to the base-level algorithm, based on the performance of the classifiers that have been returned so far. The boosting algorithm returns some combination of classifiers that are collected as the new "boosted" classifier.

Boosting algorithms were introduced by researchers in Computational Learning Theory as tools to prove the equivalence of weak learning and strong learning (Schapire, 1990; Freund, 1992). Given an *arbitrary* target concept and an *arbitrary* distribution, a **weak learning** algorithm produces a binary classifier guaranteed to do better than 50% correct for the distribution it was trained on. This classifier is drawn from a pre-determined hypothesis space. A **strong learning** algorithm can achieve any level of accuracy (in the PAC sense (Valiant, 1984)).

The main theoretical property of the boosting algorithms developed in the theoretical community is the following: if given a target concept and a base-level learning algorithm which can produce a classifier that does better than chance on *any* training distribution, then a boosting algorithm can combine multiple classifiers generated by the base-level algorithm and create arbitrarily accurate classifier for that target concept. The theoretical idea is presented schematically in Figure 4.2.

In section 3.6 we calculated the upper bound on the accuracy of n classifiers of equal accuracy g. We can use equation 3.32 to put a lower bound on the number of weak classifiers with accuracy $g = \frac{1}{2} + \gamma$, necessary to achieve a strong accuracy of $G = 1 - \epsilon$:

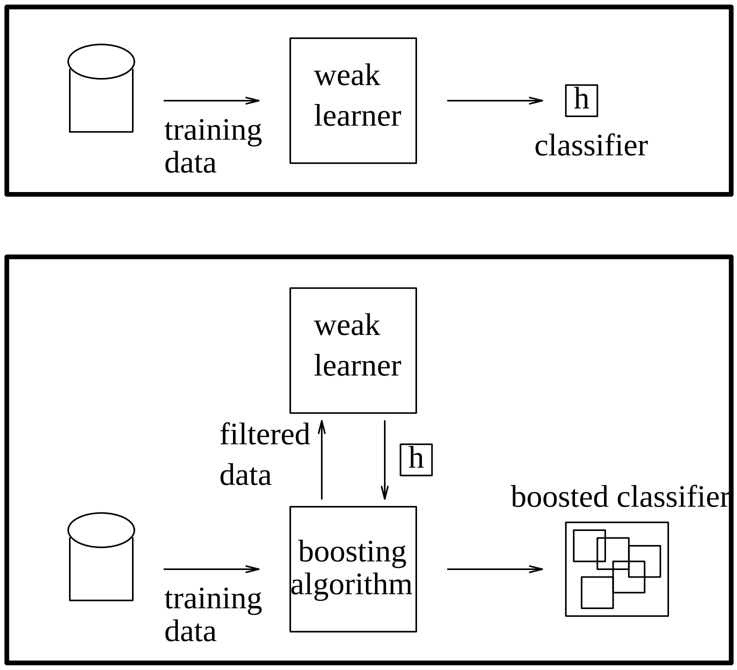

Figure 4.1: A boosting algorithm. The boosting algorithm uses a base-level weak learning algorithm as a black-box. The boosting uses the weak learner to generate classifiers from training data that is manipulated by the boosting algorithm. The boosting algorithm builds a new strong classifier using a combination of the classifiers returned by the weak learning algorithm.

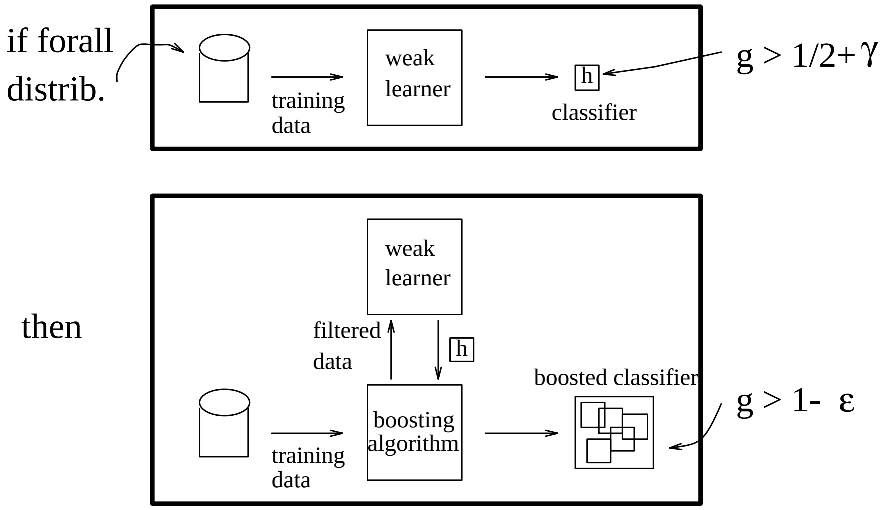

Figure 4.2: Theoretical qualities of boosting: if, for **any** training distribution, the weak learner is **guaranteed** to return a classifier that is better than 50% correct on the distribution it was trained on, then the boosting algorithm can return a classifier with an **arbitrary** level of accuracy.

$$n \geq \frac{1-\epsilon}{2\gamma+\epsilon} \tag{4.1}$$

In this chapter we analyze the first boosting algorithm proposed (Schapire, 1990). Schapire's Boosting Algorithm (SBA) is studied under conditions that exist in practice, where the "weakness" is due to the lack of effective capacity of the learning machine at hand. In this case the algorithm's theoretical properties do not necessarily hold, and performance improvement is not guaranteed. We characterize the conditions for success or failure in practice, as a function of the complexity of the base-level classifier. Our study is done in terms of SBA's *representation* and *search method.*

In the following section we describe SBA and outline a novel view of its representation. In section 4.3 we analyze a simple model to demonstrate possible failings of SBA in practice. In section 4.4 we present a rejection boosting algorithm as an alternative method of fitting the target concept that allows a trade-off between supervised labeling and boosting power. We close with a discussion.

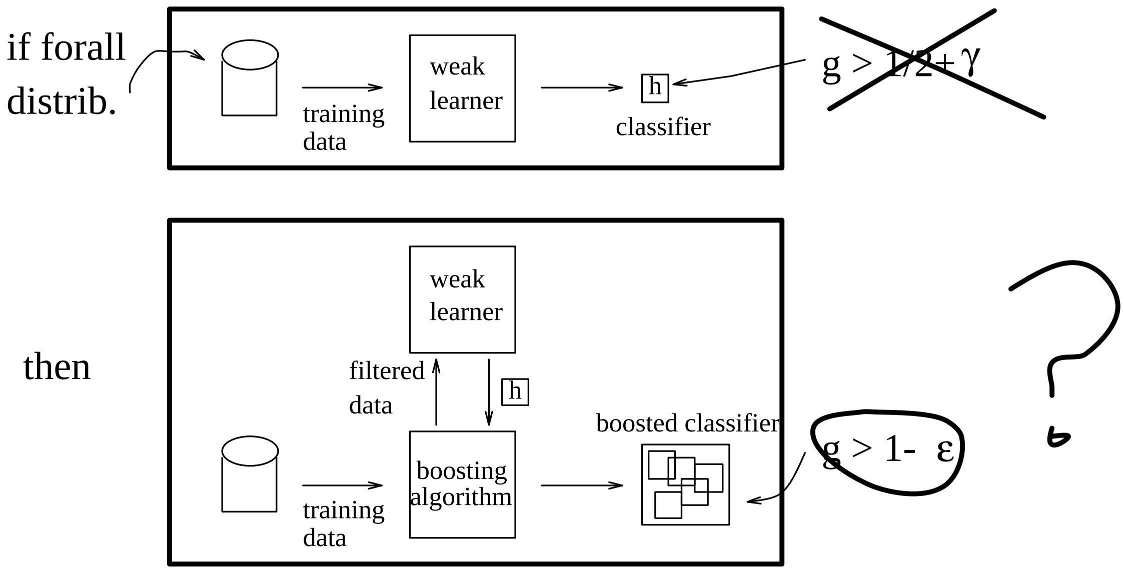

Figure 4.3: Boosting in practice. There are very few target concepts for which a weak learner is guaranteed, regardless of the distribution, to be better than chance. Therefore, a practical question is: will a boosting algorithm give a classifier that has superior accuracy and under what conditions?

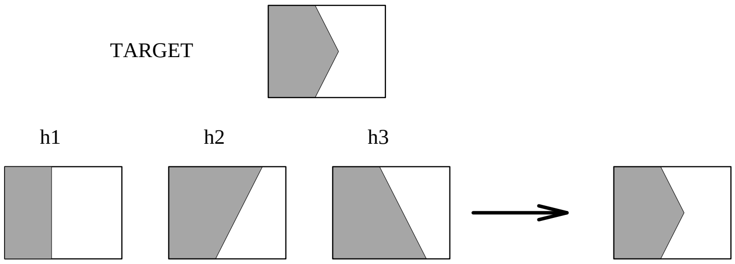

Figure 4.4: Boosting a weak representation. A linear separating classifier is too weak to represent the target concept shown above. However, a majority-vote ensemble of linear separators is able to represent this target. Most boosting algorithms build the ensemble incrementally. In this particular case three calls to the weak learner would return classifiers h_1, h_2, and h_3 in sequence.

Weakness in practice

For most practical applications there are no true weak learning algorithms. Usually one can distort the input distribution enough to cause the learning algorithm to produce a concept no better than 50% correct. Thus, in practice the boosting theorems do not hold, and performance improvement is no longer guaranteed.

In practical terms, a learning machine is "weak" either because the representation is not complex enough to capture the target concept or the search procedure (learning algorithm) fails to fine tune the parameters (i.e. use the full capacity of the representation). We lump these possibilities into one intuitive term and say that the learning machine lacks "effective capacity" (see section 2.4.3).

Boosting algorithms control the increase in capacity by (1) adding complexity to the representation and (2) controlling the search through the additional parameters by modifying the training distribution.

Figure 4.4 shows an example where the representation of the base-level classifier is too weak to represent the target concept and therefore any single classifier would have low accuracy, however an ensemble formed as a result of boosting is able to represent the target concept.

What is the purpose of boosting?

One promise of using boosting algorithms in practice is achieving a better fit to training data for difficult real-world problems. That is, boosting may prove to be a methodology for improving the accuracy of existing state-of-the-art learning machines (e.g. improving from 97% accuracy to 99% accuracy).

An alternative view on the utility of boosting algorithms has been that boosting with very simple "weak" learners may be a viable alternative to state-of-the-art learning algorithms such as C4.5 or back-propagation networks. In particular, the claim is that new boosting algorithms show some resilience to overfitting (Drucker and Cortes, 1995; Freund and Schapire, 1996).

In this work, we adopt the first view. Since many applications have practically unlimited training data, we believe the main challenge to applied machine learning is fitting better models to a vast amount of data. In these applications overfitting issues may not be of concern (the data may overwhelm any bias or priors). In any case, overfitting issues are orthogonal to the boosting algorithm's primary goal : achieving a good fit (efficient representation) in reasonable time (efficient search) while using a weak learner with limited effective capacity.

Boosting has potential for methods in which controlling the capacity is difficult. For example, the capacity of back-propagation networks seem to grow very slowly as additional units are added. In contrast, boosting does not appear to be useful for learning machines that have infinite potential capacity (*e.g.* $k-$nearest neighbors).

4.2 Schapire's Boosting Algorithm(SBA)

SBA uses a weak learning machine (LM) and a source for labeled examples from a distribution D in the following way[1]:

1. Run LM on the original distribution D resulting in classifier h_1.
2. Filter D through h_1 and generate distribution D_2, such that 50% of D_2 consists of h_1 errors and 50% corrects. Run LM on D_2 resulting in classifier h_2.

[1] In Schapire's original paper, the three step algorithm is run recursively to boost a weak learner to arbitrary strength.

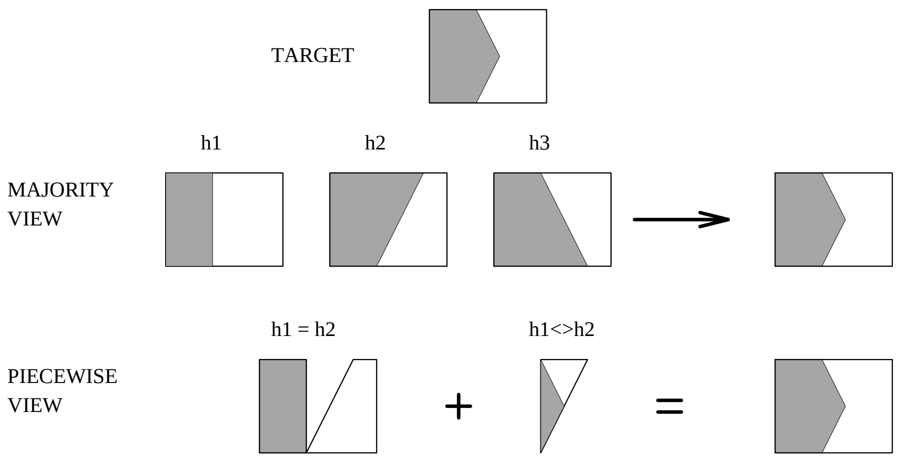

Figure 4.5: An alternative view of SBA. The SBA representation is commonly viewed as a majority of weak learners. An alternative view is of that of piecewise tiling of the domain. h_2 rejects the part of the domain on which it disagrees with h_1. The rejected portion is classified by h_3.

3. Filter D through h_1 and h_2 and generate distribution D_3 consisting of patterns upon which the two classifiers disagree. Run LM on D_3 resulting in h_3.

During the classification phase the classifier $h_{SBA}(\vec{x})$ is: Given pattern $\vec{x}$, if $h_1(\vec{x}) = h_2(\vec{x})$ return their agreed class. Otherwise return $h_3(\vec{x})$.

SBA's representation is a composite of the weak learner's representation. We view the composite hypothesis h_{SBA} as piecewise covering of the domain (see Figure 4.5). This is in contrast with the view of a majority of three hypotheses which is correct only in binary classification problems. h_2's role in this view is a rejector. The parts of the domain on which h_2 disagrees with h_1 are classified by h_3.

SBA's representation resembles other work in ensembles, yet the tiling nature makes it closer to decision trees and decision lists (Breiman et al., 1984; Quinlan, 1993; Rivest, 1987) than to additive mixture models (Jordan and Jacobs, 1994). The skewing of the distribution resembles work in Statistics where models are fitted to residuals. In relation to gating networks (Jacobs et al., 1991), SBA may be seen as using hard (rather than soft) gates.

SBA has been applied successfully to an OCR problem learned by back-propagation networks, yet has failed to boost in other architectures (Drucker, Schapire and Simard,

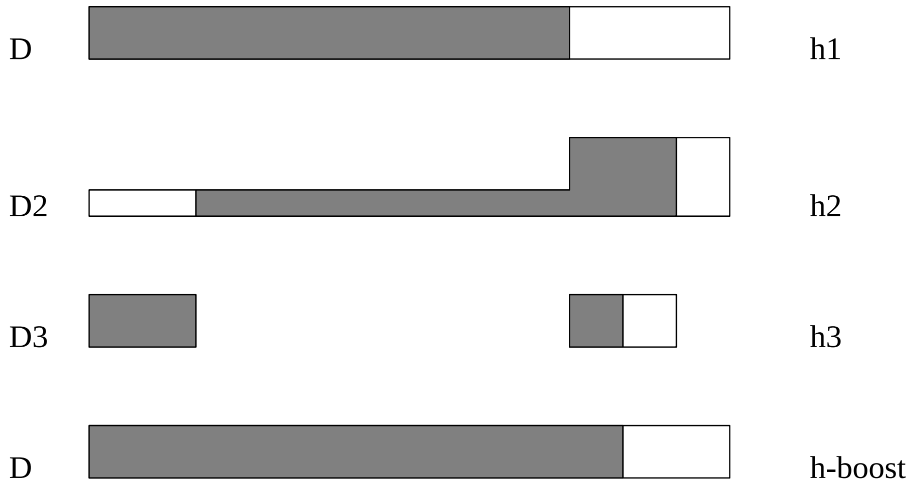

Figure 4.6: A pictorial description of Schapire's Boosting Algorithm. Each line depicts both the domain and a correctness indicator function for a classifier generated at a certain step. The whole domain is depicted as the points along the horizontal axis. The height at any point is proportional to the probability density function of the training distribution passed to the weak learner at that stage. We assume that the original distribution is uniform over the domain. The classifier h_1 only correctly classifies part of the domain and is trained on the original distribution D. The new training distribution D_2 is generated so that half of the patterns are h_1 errors and the other half are classified correctly by h_1. Training on the new distribution will produce h_2 that will classify some of the patterns h_1 was unable to classify (*the "hard" patterns*) at the expense of erring on others that h_1 classified correctly. In step 3, D_3 includes only patterns on which h_1 and h_2 disagree. Since the areas which are not part of D_3 are "don't care", h_3 will be able to use some of its capacity on "hard" patterns. The resulting voting ensemble h_{boost} has higher accuracy than, h_1, the original classifier the weak learner generated.

1993). When does applying a boosting algorithm to a specific learning machine lead to improved classification accuracy? In Section 4.3 we attempt to characterize the conditions for success or failure. We find that the important characteristic is the representational complexity of the hypothesis class with respect to the residual of the initial fit.

4.3 When Can SBA Increase Coverage?

The ability to fit a target concept by tiling depends on the complexity of the target and of the learner while the actual error depends on the distribution as well. A simple example that follows demonstrates this point. Figure 4.8 displays the zigzag concept approximated by a linear classifier. It is a poor match yet, depending on the pattern distribution, the error rate can be quite small. Boosting this weak learner will at best use the linear classifier multiple times, contributing only a small additional gain at each iteration of SBA.

In the remainder of this section we analyze a simple case demonstrating how the lack of complexity of the weak learner can cause no boosting or non-optimal boosting to occur when using SBA. A slightly different manifestation of this problem is presented graphically in Figure 4.7. In that example it is shown how an optimal initial fit can lead to inferior boosted accuracy.

The point of this section is to analyze SBA when using a fixed capacity base-level classifier. We show that the important parameter is not the initial accuracy of the base-level classifier, but the complexity of the base-level classifier (capacity) with respect to the target.

Definitions

We define a domain $D \subseteq R^N$. The goal is to learn a target function $C(\vec{x}) : D \to \{0,1\}$. The inputs $\vec{x}$ are distributed according to $\rho(\vec{x})$ where $\int_X \rho(\vec{x}) d\vec{x} = 1$. The uniform distribution over the domain D is $\rho_0(\vec{x}) = \frac{1}{V_D}$, where $V_D = \int_X d\vec{x}$.

Our weak classifiers are hypotheses $h(\vec{x})$ drawn from a family $\mathcal{H}$. For a given hypothesis $h(\vec{x})$ the corresponding correctness indicator function is:

$$\mathcal{I}_h(\vec{x}) = \begin{cases} 1 & h(\vec{x}) = C(\vec{x}) \\ 0 & \text{otherwise} \end{cases} \tag{4.2}$$

We define the **accuracy** as the correct rate with respect to the input distribution and

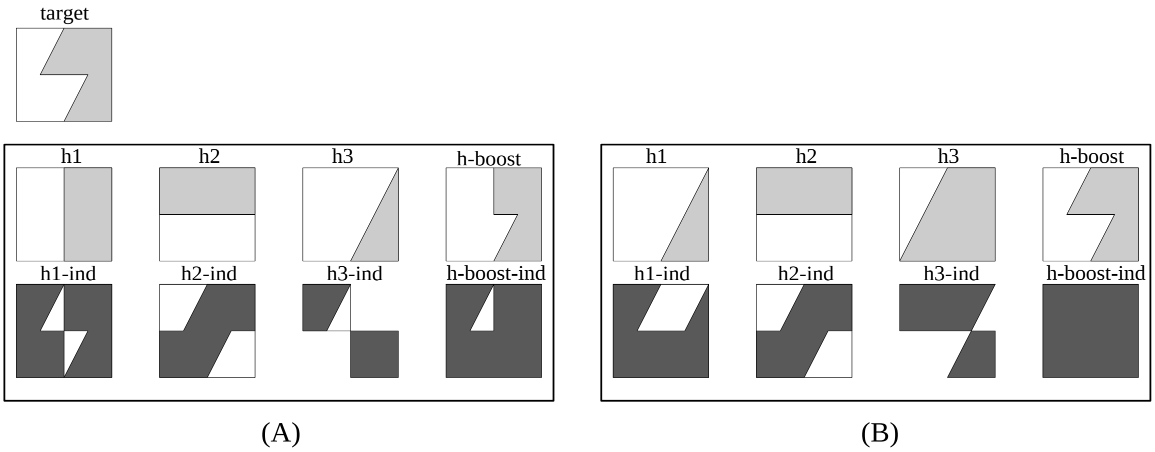

Figure 4.7: Results of applying SBA with a linear classifier as the weak learner in two different cases. The target concept is $f(\{X,Y\}) = \Theta(Y - 2X + 1 - \Theta(Y))$ where $\{X,Y\} \in [-1,1]^2$ and $\Theta(w) \equiv [w \geq 0]$ is a step function. This example is simple enough to run SBA analytically. The purpose of the examples is to demonstrate that the boosting algorithm's success may depend on the initial concept (h_1). In fact, a non-optimal fit for h_1 may give better overall results. Unless otherwise noted, the LM returns the best classifier it can fit for the distribution it is given. The top row gives the the three different concepts returned at each stage of the boosting algorithm and the final combined classifier. Directly below, the respective areas of the domain in error are shown. The error for h_3 is shown only for the part of the domain that is passed to the LM in step 3. (A): h_1's accuracy is 87.5%, this is the optimal accuracy for a linear classifier for this target concept. h_2's accuracy is 78.6% on D_2 and h_3's accuracy on D_3 is 87.5%. The combined boosted accuracy is 93.75%. (B): For h_1 we choose a classifier whose accuracy is less than optimal (75%). The corresponding h_2 has an accuracy of 75% on D_2 and h_3's accuracy D_3 is 100%. The combined boosted accuracy is 100%.

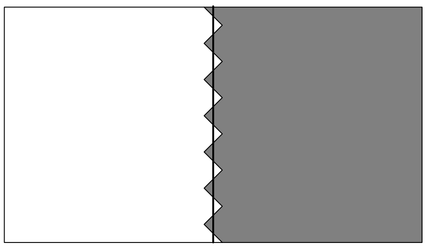

Figure 4.8: The zigzag concept may fit reasonably well by a linear separator, but boosting will cause only small increases in fit.

the **fit** as the correct rate with respect to the uniform distribution over the domain. The accuracy of a given hypothesis is given by :

$$g_h = \int_X \mathcal{I}_h(\vec{x})\rho(\vec{x})d\vec{x} \tag{4.3}$$

and the **error** is given by $1 - g_h$.

In this analysis we choose to consider families of hypotheses that have constant fit:

$$\forall h \quad \int_X \mathcal{I}_h(\vec{x})\rho_0(\vec{x})d\vec{x} = V_c \tag{4.4}$$

V_c is the relative fit of the family $\mathcal{H}$ with respect to the target $C(\vec{x})$. Intuitively, this is parallel to a representation with a fixed number of "memories". Although the constant fit restriction does not hold in practice, it captures the essence of fine tuning a weak classifier's decision boundary: gaining a better fit in one locale must come at the expense of the fit in another.

The hypothesis returned by SBA is a combination of three weak hypotheses: h_1, h_2, h_3. The combined correctness indicator is:

$$\mathcal{I}_T(x) = (1 - |\mathcal{I}_1(\vec{x}) - \mathcal{I}_2(\vec{x})|)\mathcal{I}_1(\vec{x}) + |\mathcal{I}_1(\vec{x}) - \mathcal{I}_2(\vec{x})|\mathcal{I}_3(\vec{x}) \tag{4.5}$$

We define the rejection function $R(\vec{x}) = |\mathcal{I}_1(\vec{x}) - \mathcal{I}_2(\vec{x})|$ and rewrite the rejected volume as:

$$\begin{aligned} V_R &= \int_X \rho_0(\vec{x})R(\vec{x})d\vec{x} && (4.6) \\ &= \underbrace{\int_X \rho_0(\vec{x})\mathcal{I}_1(\vec{x})(1 - \mathcal{I}_2(\vec{x}))d\vec{x}}_{\Delta_1} + \underbrace{\int_X \rho_0(\vec{x})(1 - \mathcal{I}_1(\vec{x}))\mathcal{I}_2(\vec{x})d\vec{x}}_{\Delta_2} && (4.7) \end{aligned}$$

The areas of $\mathcal{I}_1$ and $\mathcal{I}_2$'s differences are equal $\Delta_1 = \Delta_2 = \Delta$. One can view Δ as the amount of shift in $\mathcal{I}_1$ to get $\mathcal{I}_2$. The rejected area is therefore $V_R = 2\Delta$. One may now compute a bound on the combined fit V_T:

$$\begin{aligned} V_T &= \int_X \rho_0(\vec{x})\mathcal{I}_T(\vec{x})d\vec{x} && (4.8) \\ &= \int_X \rho_0(\vec{x})\mathcal{I}_1(\vec{x}) + R(\vec{x})(\mathcal{I}_3(\vec{x}) - \mathcal{I}_1(\vec{x}))d\vec{x} && (4.9) \end{aligned}$$

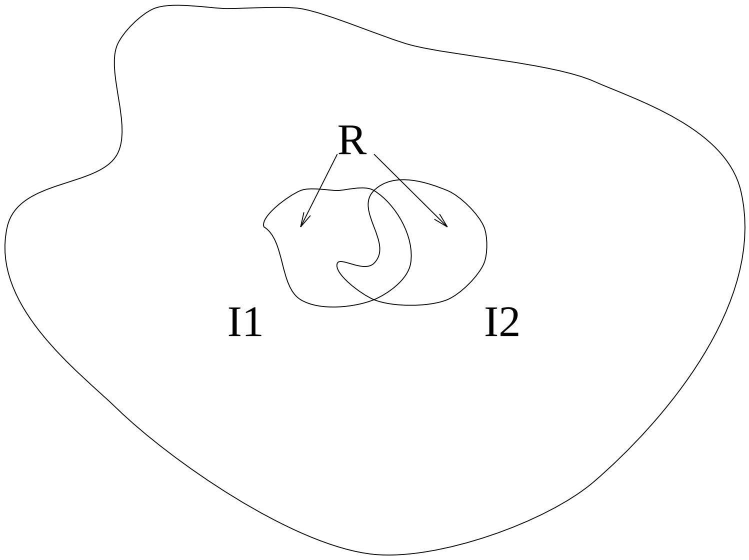

Figure 4.9: The rejection area. $\mathcal{I}_1$ is the area in domain that h_1 is correct. $\mathcal{I}_2$ is area where h_2 is correct. R is area where h_1 and h_2 disagree.

$$= V_c + \int_X \rho_0(\vec{x})(\mathcal{I}_3(\vec{x}) - \mathcal{I}_1(\vec{x}))\mathcal{I}_1(\vec{x})(1 - \mathcal{I}_2(\vec{x}))d\vec{x} \tag{4.10}$$

$$+ \int_X \rho_0(\vec{x})(\mathcal{I}_3(\vec{x}) - \mathcal{I}_1(\vec{x}))(1 - \mathcal{I}_1(\vec{x}))\mathcal{I}_2(\vec{x})d\vec{x} \tag{4.11}$$

$$= V_c - \Delta + \int_X \rho_0(\vec{x})\mathcal{I}_3(\vec{x})\mathcal{I}_1(\vec{x})(1 - \mathcal{I}_2(\vec{x}))d\vec{x} \tag{4.12}$$

$$+ \int_X \rho_0(\vec{x})\mathcal{I}_3(\vec{x})(1 - \mathcal{I}_1(\vec{x}))\mathcal{I}_2(\vec{x})d\vec{x} \tag{4.13}$$

The last two terms are bounded from above by both V_c and 2Δ. Therefore

$$V_T \leq V_c - \Delta + min(V_c, 2\Delta) \tag{4.14}$$

The optimal fit as a function of Δ is $V_T^{max} = \frac{3}{2}V_c$ that is achieved when $\Delta = \frac{V_c}{2}$. This is in accord with the results from section 3.5.

We now write expressions for the altered distributions generated by SBA. If the accuracy of h_1 is

$$g_1 = \int_X \rho(\vec{x})\mathcal{I}_1(\vec{x})d\vec{x} \tag{4.15}$$

then the distribution that h_2 is trained on is

$$\rho_2(\vec{x}) = \frac{1}{2}\rho(\vec{x})\left[\frac{\mathcal{I}_1(\vec{x})}{g_1} + \frac{1-\mathcal{I}_1(\vec{x})}{1-g_1}\right] \tag{4.16}$$

and the distribution h_3 is trained on is $\rho_3(\vec{x}) = \rho(\vec{x})R(\vec{x})$.

A tough case for SBA

We now look at a specific case that illuminates how SBA can fail to boost when the base-level classifier has relatively low capacity with respect to the target function. We choose a distribution of the form:

$$\rho(\vec{x}) = AG(\vec{x}) + B(1 - G(\vec{x})) \tag{4.17}$$

such that $A > B$, $\int_X \rho(\vec{x}) = 1$, G is a binary function, and $\int_X G(\vec{x})d\vec{x} = V_c$. We assume the weak learning algorithm returns h^* such that g_{h^*} is maximized under the input distribution. Since $\int \rho_0(\vec{x})\mathcal{I}_1(\vec{x})d\vec{x} = V_c$ we can use the equality

$$\int G(\vec{x})(1-\mathcal{I}_1(\vec{x}))d\vec{x} = \int (1-G(\vec{x}))\mathcal{I}_1(\vec{x})d\vec{x} \tag{4.18}$$

to write the accuracy of h_1 as

$$g_1 = (A-B)\int G(\vec{x})\mathcal{I}_1(\vec{x})d\vec{x} + B\int G(\vec{x})d\vec{x} \tag{4.19}$$

The second term is constant and since $A > B$ the maximal accuracy is achieved if $h_1(\vec{x}) = G(\vec{x})$ resulting in value $g_1 = AV_c$. This result is intuitive; since the capacity is limited, h_1 is chosen to fit areas with higher probability density. Now we can compute $\rho_2(\vec{x})$ and the corresponding accuracy of h_2:

$$\rho_2(\vec{x}) = \frac{1}{2}\left[\frac{G(\vec{x})}{V_c} + \frac{[1-G(\vec{x})]}{1-V_c}\right] \tag{4.20}$$

$$g_2 = \frac{1}{2}\left[\frac{1}{V_c} - \frac{1}{1-V_c}\right]\int G(\vec{x})\mathcal{I}_2(\vec{x})d\vec{x} + \frac{1}{2(1-V_c)}\int G(\vec{x})d\vec{x} \tag{4.21}$$

We look at the following cases [2] :

$V_c < 1/2$. The maximum accuracy occurs if $\mathcal{I}_2(\vec{x}) = g(\vec{x})$. Since $h_1(\vec{x}) = h_2(\vec{x})$, the rejection function is $R(\vec{x}) = 0$, and no boosting occurs.

[2] We ignore $V_c = 1/2$. where $\rho_2(\vec{x})$ is a uniform distribution making the choice of h_2 irrelevant to g_2.

$V_c > 1/2$. In this case the maximum accuracy occurs if $\mathcal{I}_2(\vec{x}) = 0$ when $G(\vec{x}) = 1$ (i.e $\Delta = V_c$). Since $V_c > 1/2$, Δ is bounded by $1 - V_c$. The resulting V_R is equal to $2(1 - V_c)$. In an argument similar to that needed to arrive at Equation 4.14, the total fit is equal to $V_T = 2V_c - 1 + \min(V_c, 2(1 - V_c))$, which for $V_c < 2/3$ is less than the optimal $\frac{3}{2}V_c$.
In summary the boosted fits for different values of V_c are:

$$V_T = \begin{cases} V_c & V_c < \frac{1}{2} \\ 3V_c - 1 & \frac{1}{2} < V_c \leq \frac{2}{3} \\ 1 & \frac{2}{3} < V_c < 1 \end{cases} \tag{4.22}$$

As we saw before, the optimal gain in fit is $\frac{V_c}{2}$. On this example SBA does optimally when the weak learner has reasonable fit. On the other hand SBA fails to achieve any gain if the weak learner is "too weak". In the mid range SBA achieves a fit that is less than the optimal value $\frac{3}{2}V_c$.
As a result the gain in accuracy is:

$$\Delta g = g_T - g_1 = \begin{cases} 0 & V_c < \frac{1}{2} \\ \frac{2V_c - 1}{1 - V_c}(1 - g_1) & \frac{1}{2} < V_c \leq \frac{2}{3} \\ 1 - g_1 & \frac{2}{3} < V_c < 1 \end{cases} \tag{4.23}$$

By setting the parameter A, the original accuracy can be made arbitrarily large. As described earlier in the zigzag example, the ability to boost the accuracy depends on the fit V_c and not on how large the base-level accuracy g_1 is.

In the low capacity region, SBA fails to boost at all. In the mid capacity region, SBA boosts to a degree, but less than optimal. In the high capacity region it achieves optimal boosting: full fit.

The above model was designed so that it would fail to boost for low capacity values . This is due to the sharp transition in the probability density at the precise capacity of the weak learner. The lesson learned still holds if the distribution is smooth: if the residual of the initial fit has high complexity with respect to the weak learner, SBA may fail to achieve significant boosting and will fall short of the optimal possible boost in accuracy

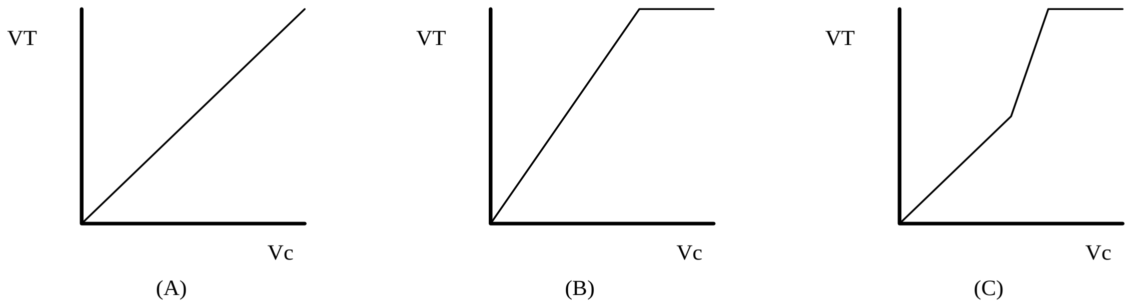

Figure 4.10: Comparison of V_T as a function of V_c: (A) No boosting is done. (B) Boosting using the optimal strategy to pick h_2. (i.e. $\Delta = V_c/2$) (C) boosting using SBA. One can see that for $V_c \geq 2/3$, SBA achieves 100% accuracy and for $V_c < 1/2$, SBA does not boost at all. In the mid range $1/2 < V_c < 2/3$, SBA is sub-optimal and does not chose the best h_2. *[Axes are between 0 and 1.]*

4.4 Boosting by Rejection

As observed earlier, SBA trains h_2 to become a rejector. Alternatively we suggest using a natural confidence criterion for this task. The following is a *rejection boosting algorithm (RBA)*:

1. Train a classifier on the original distribution D resulting in classifier h_1.
2. Filter D through h_1 and generate a distribution D_{2R} composed of patterns rejected according to a criterion $R(\vec{x}, h_1)$. Train a classifier on D_{2R} resulting in h_{2R}.

At classification time, patterns that pass the rejection criterion R, are classified according to h_1. Rejected patterns are classified by h_{2R}.

This scheme is attractive in that it (a) may use fewer patterns to train, and (b) only patterns that are rejected need to be labeled. This is of prime importance when labeling costs are high or the system is learning while on-site (Matan, 1995) (see Chapter 5).

Boosting in practice can fail due to the inability of the additional weak learners to perform well on the new distribution. If the weak learner has some natural confidence score, one may use this to generate a tunable rejection criterion that rejects a fraction β of the lowest scoring patterns. Under reasonable conditions, rejection causes the original error rate ϵ_{1R} to decrease to ϵ_{1R}. If the error rate of h_{2R} on D_{2R} is ϵ_2 then the error of the combined hypothesis is $(1-\beta)\epsilon_{1R} + \beta\epsilon_2$. From this we can compute the maximum error rate that h_{2R} may exhibit such that the combined hypothesis has a lower error rate than ϵ_1:

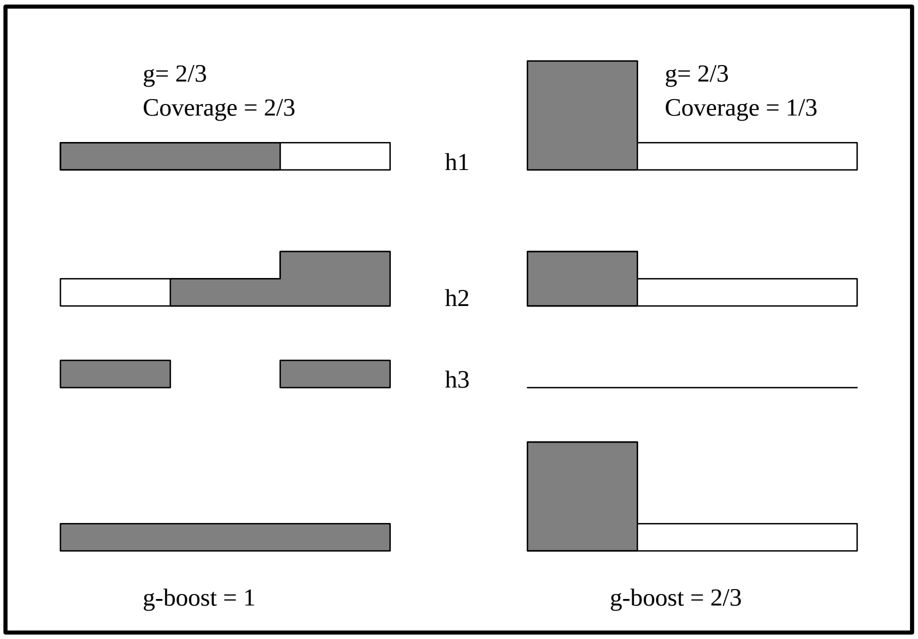

Figure 4.11: Boosting depends on fit. A pictorial depiction of the analysis in Section 4.3: in both the left and right examples the weak learner can produce a classifier with an accuracy of 2/3. In the left example this is due to a reasonable fit of the target function. In the right example h_1 has a poor fit, but has the same accuracy due to the fact that the part of the domain covered by h_1 has a high probability density. The figure demonstrates the fact that boosting with SBA is dependent on the fit (or capacity) of the weak representation and not on the accuracy. This is particularly extreme in the right example which shows the case where $\mathcal{D}_2$, the new training distribution for h, causes the generation of h_2 that is equivalent to h_1 and therefore no boosting occurs at all.

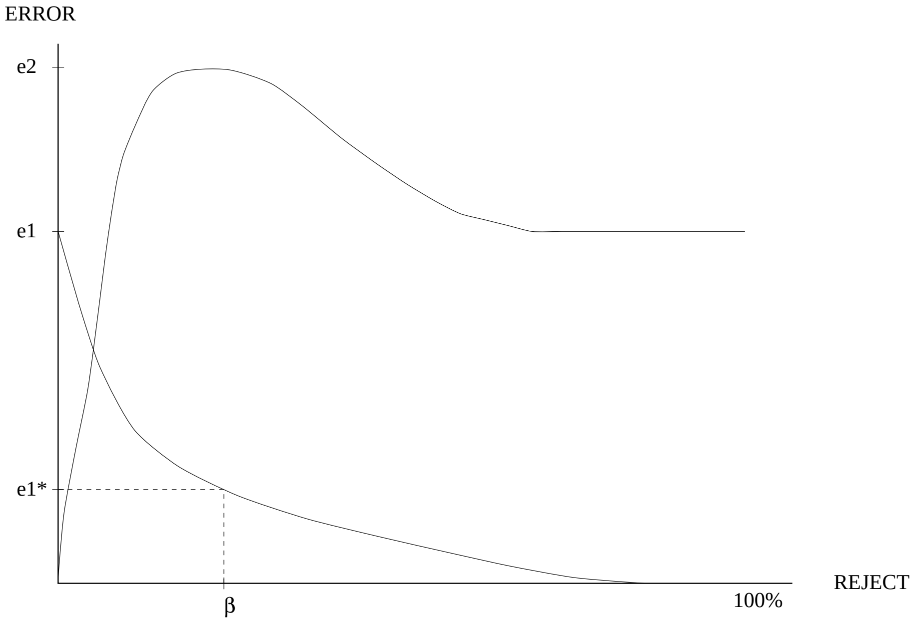

Figure 4.12: Schematics of the behavior of RBA as a function of β. The lower plot is the general form of the error rate ϵ_{1R} as a function of the rejection rate. The second plot shows the typical behavior of ϵ_2, the error rate of h_{2R}as a function of the rejection rate β. When β is close to zero - very few patterns are rejected and h_{2R} can easily learn them. As β grows the distribution D_{2R} has a higher distribution of h_1 errors which makes the task for h_{2R} harder and we may expect a higher error rate. As β approaches 100%, the distribution D_{2R} becomes similar to the original distribution, and we expect ϵ_2 to approach ϵ_1.

$$\epsilon_2^{max} = \frac{\epsilon_1 - (1-\beta)\epsilon_{1R}}{\beta} \tag{4.24}$$

If training h_{2R} shows a difficulty matching the needed accuracy we may adjust β accordingly. Alternatively we may adjust β to achieve maximum boosting for a fixed architecture. The best β can be found by some form of hill-climbing (Nilsson, 1980).

Experiments

We compared SBA and RBA with feed forward networks trained with back-propagation on the target concept shown in figure 4.14. It is a simple 2 dimensional 2-class concept that the weak learners cannot fit exactly. The weak learners had 2 fully connected hidden layers of the architecture 2x5x5x2.

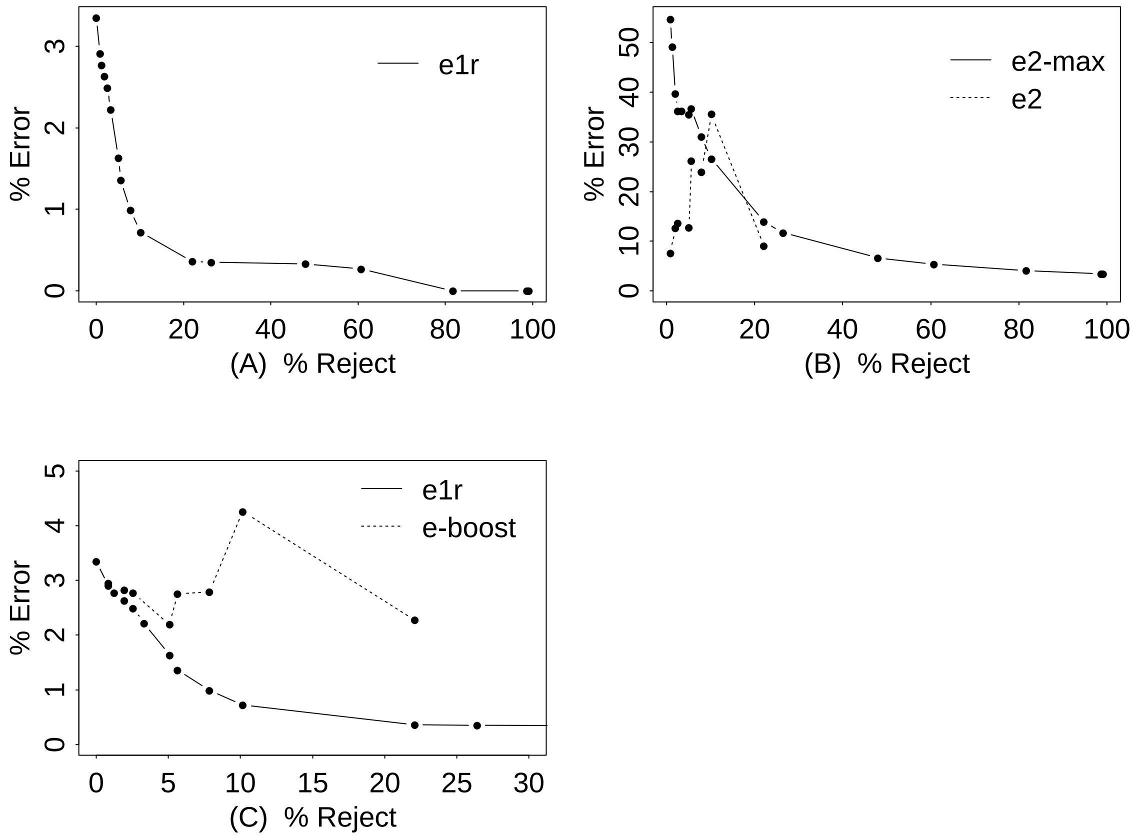

Figure 4.13: An example of the dependency of boosting on β. RBA give the ability to tune and possibly maximize the boosting. (A) Shows the empirical dependency of ϵ_{1R} on β. (B) Using Equation 4.24 one can calculate ϵ_2^{max}, the upper bound on the error of h_{2R} and compare it to the empirical value ϵ_2 for various values of β. One can see that at approximately $\beta = 15\%$ the error ϵ_2 surpasses this bound. (C) At the point that ϵ_2 surpassed ϵ_2^{max}, RBA actually increases the error. At all other points there is a reduction in the error. One can experimentally search through the values of β to find the value that gives the best error reduction.

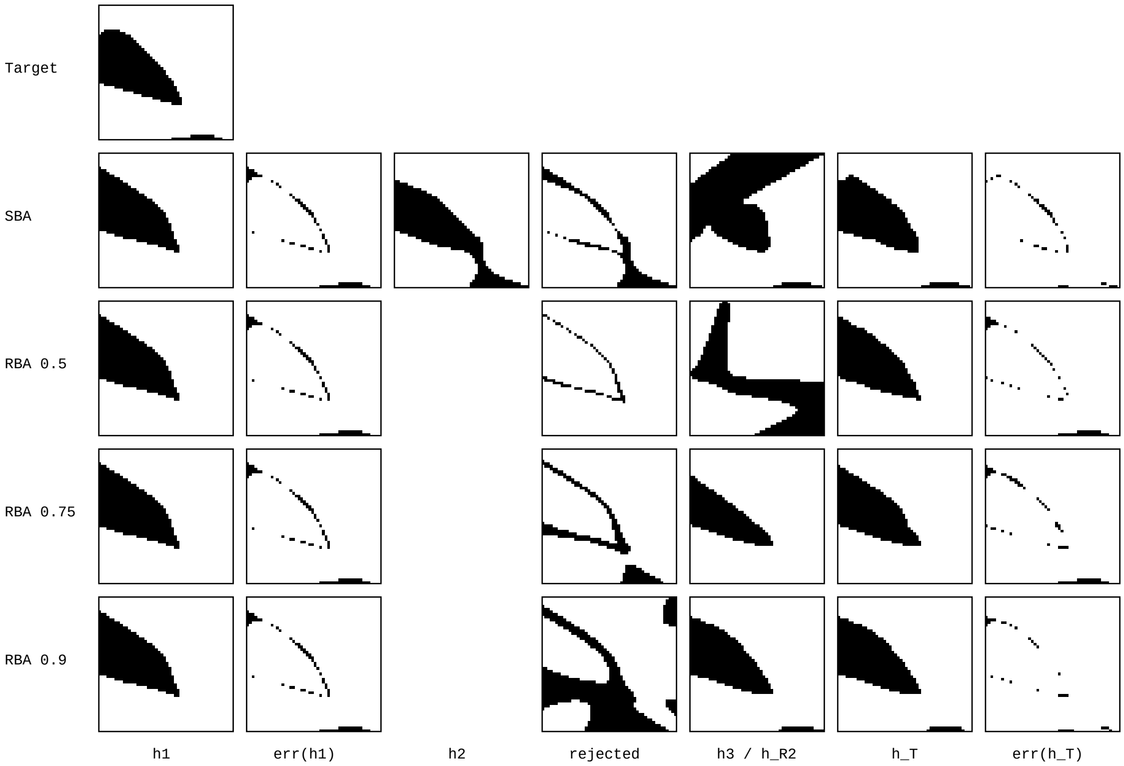

Figure 4.14: Comparison of run #1 of SBA and RBA on a target concept shown on top. The target concept is shown in the top left corner. The domain of the concept is $[-1, 1]^2$. Black areas are labeled $+$, white areas are labeled $-$. Samples are drawn uniformly from the domain. The second row presents SBA results and the following rows present RBA results for different rejection rates. The columns display the weak learner's concepts or their errors. For SBA the rejected plot shows the function $R(x) = |h_1 - h_2|$. Both SBA and RBA were trained on the same stream and shared the same initial weights - therefore h_1 is equivalent for all.

In the SBA experiments h_1, h_2 and h_3 were trained for $20K, 40K$ and $80K$ back-propagation iterations respectively. The training stream was randomly generated with a uniform distribution over the domain. Each instance was used only once. RBA's rejection function rejected patterns based on the difference between the two output units. The threshold was set using a validation set so that approximately 0.5, 0.75 and 0.9 of h_1's errors were rejected. h_1 and h_{2R} were trained for $20K$ and $80K$ iterations respectively. Each algorithm was tested on 25 independent data streams. The hypothesis h_1 was common to all algorithms and we used the same initial weights for h_3 and h_{2R}.

Figure 4.14 presents the resulting weak and boosted concepts for one individual run. In this case RBA compares favorably with SBA It also demonstrates the importance of a good rejection criterion. RBA 0.5 fails to reject the portion of the domain in the lower right corner and gives no chance to boost in that area. The following table summarizes the average number of patterns filtered and labeled (in thousands) for each algorithm and the mean and standard deviation of the accuracy of the final concept:

Alg.	patterns filtered			labeled	
	h_1	h_2	h_3 or h_{2R}		acc±std
No Boost	20	—	—	20	96.4 ± 0.59
SBA	20	647	820	667	98.2 ± 0.73
RBA 0.5	20	—	1340	100	97.5 ± 0.29
RBA 0.75	20	—	472	100	97.8 ± 0.73
RBA 0.9	20	—	273	100	97.6 ± 1.23

Pairwise comparison of SBA and RBA shows that SBA performs better than RBA on average. This is expected due to the fact that SBA has more information than RBA. What we have here is the well known tradeoff of "exploration vs. exploitation".

RBA reduces the resources needed to achieve a similar performance level: it labels and uses only patterns it rejects. It does not pass erroneous patterns on to h_{2R} if it is overconfident. In contrast, SBA checks every pattern to see if it correct or in error and has an opportunity to correct the performance in that part of the domain.

Note that as RBA rejects more patterns (such as RBA 0.9), boosting performance may degrade. This is since the rejected distribution approaches the original distribution, h_{2R} becomes closer to h_1 and the boosted hypothesis take less advantage of the capacity of h_1.

4.5 Related Work

We have studied the use of one step SBA in practice. One may ask the following questions :

1. Is the majority-vote ensemble representation a good one? How does it compare to other methods of combining classifiers?

2. Does SBA provide a good search control in this representation space?

We do not consider the first question, Chapter 3 dealt with the power of such a representation but does not present a relative comparison to other candidate representations.

This chapter has addressed the second question. The example in Figure 4.7 demonstrated the fact that the greedy search approach of SBA may lead to results that are sub-optimal. Furthermore, section 4.3 models the success and failure modes of the search using SBA. If the weak learner is complex enough with respect to the residual error of h_1, SBA will lead to substantial boosting. Otherwise is will fail to give substantial boosting. Although SBA has been used with some success in an OCR application using neural networks, it has failed to boost for other learning algorithms (Drucker, Schapire and Simard, 1993; Drucker et al., 1994).

A boosting algorithm suggested by Freund (Freund, 1992) builds a k-member majority-vote ensemble, where k is determined by the weak learner parameter γ. We do not describe this algorithm in detail, but note a few details. The weak classifiers are generated in sequence and the training distribution over the domain is a function of the number of current members in the ensemble that classify the pattern correctly. This distribution is centered slightly below the majority point.

No experimental studies of this algorithm have been done, apparently due to implementation difficulties. However, this method is consistent with the analysis of Chapter 3 concerning the optimal use of the resources of a majority vote ensemble: the training distribution places low weight on patterns that are either way above or way below the majority threshold. The most weight is placed on patterns close to the threshold with a slight bias to patterns just below threshold.

Freund and Schapire have recently proposed a new theoretical boosting algorithm (Freund and Schapire, 1995): the algorithm is similar in nature to the algorithm by Freund, however it differs in that there is no need to determine the ensemble size in advance. The representation is a *weighted* majority vote ensemble and the training distribution method is different. The weighted majority representation does not allow analysis using the methods introduced in this and in the preceding chapter. The adaptive boosting algorithm has spurred considerable interest and some successful experimentation (Freund and Schapire, 1996; Drucker and Cortes, 1995; Quinlan, 1996). The main focus of this experimentation has been using boosting with very weak representations as a competitor with other base-level algorithms as discussed in Section 4.1.

Other work in the machine learning and statistics communities resembles that of boosting. These include techniques such as bagging (Breiman, 1994) and stacking (Wolpert, 1992).

4.6 Summary

The motivation for using boosting algorithms in practice is the achieving of better fitting models. As in any learning system there are two components: representation and search. The main question concerning use of boosting algorithms in practice is whether their search methodology has any advantage over "conventional" learning algorithms that increase their capacity such as C4.5 (Quinlan, 1993) or cascade-correlation (Fahlman and Lebiere, 1990).

We have studied Schapire's Boosting Algorithm and argued that it may fail to boost efficiently under certain conditions. This limitation is due to the tiling nature of its representation. If the complexity of the weak hypothesis class is comparable to the residual of the initial fit, boosting will achieve superior accuracy. If the complexity is low, SBA may result in little or no accuracy improvement. Other boosting algorithms such as (Freund, 1992; Freund and Schapire, 1995) which have a mixture representation are not prone to such problems. While the new boosting methods have shown promising experimental results, it remains to be seen under what conditions their search is efficient.

We have suggested an alternative of boosting by rejection that uses fewer pattern labels. Combining a good rejection criterion with RBA can give comparable results to SBA at a fraction of the labeling cost.

Chapter 5

On-site Learning

This chapter introduces the on-site learning model. The system learns by querying "hard" patterns while classifying "easy" ones. This model is related to query-based filtering methods, but takes into account, that in addition to labeling, collecting data has a cost.

A few simple policies are introduced and analyzed for a simple problem. The Query-By-Committee algorithm (Seung, Opper and Sompolinsky, 1992) is an ensemble method that is suggested as a good approximator of the model space for real-world domains. Results using this algorithm on a synthesized problem and a real-world OCR task using both a back-propagation network and a nearest neighbor classifier show that an on-site learner can perform as well as a classifier trained off-site, while achieving significant cost reduction.

5.1 Introduction

We present and study a model for *on-site* learning. The system is composed of a classifier and a policy. When presented with a new pattern the policy may either (a) use the classifier's output to classify the pattern or (b) it may query an oracle for the correct label and modify the classifier. We wish to define a policy that will maximize expected utility.

The motivation for studying this model is that collecting and labeling data is costly. The information captured by a corpus as a function of its size flattens out quickly. For particular problems one can show that the expected information gain of a pattern's label diminishes exponentially (Seung, Opper and Sompolinsky, 1992). Work in query-based methods (Cohn, Atlas and Ladner, 1990; Baum, 1991; Seung, Opper and Sompolinsky, 1992) has dealt with the issue of minimizing labeling cost, but it ignores the cost of searching for and collecting the informative patterns. The on-site model is suggested on the reasonable assumption that patterns that are low on information with respect to the classifier are, in fact, patterns that can be classified with high-confidence. Hence it seems possible to build a system that learns "on the job".

The on-site model can be considered for any learning architecture. What is needed is a measure of the uncertainty in the hypothesis space. Theoretically, this is supplied by the version or model space. Practically, one needs to approximate these spaces. We propose using an ensemble method for this purpose.

In the following section we discuss the off-site model and its limitations. Section 5.3 introduces the on-site model. Section 5.4 presents some policies for the on-site model and studies an example. In Section 5.5 we describe the use of the Query-by-Committee algorithm for an on-site policy and present experimental results on a simulated on-site problem and a real-world OCR problem. We discuss questions to be addressed in the future in Section 5.6.

5.2 Off-site Learning

Off-site learning is the usual methodology of building a classifier. There are two stages:

- The training stage: a classifier is built using a learning algorithm and training data that is representative of the distribution one will encounter in the performance stage.
- The performance stage: the classifier receives patterns and is expected to classify them. No further adjustments to the classifier's parameters are made.

As discussed in section 2.4.2, a possible refinement of the performance stage is giving the classifier a choice between classifying or rejecting a given pattern. In this model the learning stage must tune the rejection policy in addition to the classifier's parameters. A common method of performing rejection involves the use of representations that not only output a class, but provide a confidence level for the classification as well. The idea is to reject low confidence decisions and thus increase the correctness of classifications. In real-world applications the rejected pattern may be passed on to a human for decision or to a more sophisticated classifier.

In systems with a rejection capability, we can incorporate a cost model associated with correct classifications, errors and queries and choose a rejection policy that maximizes the entire system's utility.

5.2.1 Advantages and limitations of off-site training

+ **Consistency.** Once the classifier is built in the laboratory one has a guaranteed accuracy (Assuming there is no concept drift). Certain applications may not be suited for a truly adaptive learner.

+ **Sufficient.** Payoff of additional learning is minimal if one builds a classifier with a good rejection mechanism.

– **Data collection.** Collecting data may be a costly and lengthy process. For good coverage of the domain it may be necessary to collect a large number of examples to achieve a sufficient sampling of the tails of the distribution.

– **Data labeling.** In addition to data collection, labeling training data is tedious and costly, giving rise to large costs for large corpora.

5.3 The On-site Model

5.3.1 Two types of query-based learning

There have been two approaches to reduce the potentially large cost of labeling training data:

Constructive Query Algorithms This paradigm tries to learn concepts without the use of a training set. There has been both theoretical (Angluin, 1988) and experimental study of this paradigm (Baum, 1991; Sollich, 1994; Cohn, Ghahramani and Jordan, 1995). The method picks informative patterns and requests their labels from an oracle. The limitation of this method is that it only considers optimally learning the concept without regard to the distribution of the data in the execution stage. That is it is concerned with *fit* rather than *accuracy* (see section 2.1.4).

Query Filtering In this paradigm the learning algorithm has an unlabeled training stream and it can request a label for selected examples. This method deals with a realistic distribution yet lacks the power to focus its attention on specific areas of uncertainty.

Query filtering is a compromise between constructive query learning and labeling all the training data. Its limitation is that the number of patterns examined between queries grows exponentially with time. This still leaves one with the task of filtering through large sets of data and discarding the uninformative patterns.

5.3.2 Learning on the job

To address the aforementioned exponential "waste" of patterns required to gain additional informative patterns, we propose on-site learning which is similar to query filtering yet differs in two ways:

1. The system classifies patterns that it does not query for a label.

2. The model may consider the utility of querying.

One can view the system as having a control policy that has access to the classifier. It can query an oracle or teacher, receive answers, and decide whether to adjust the classifier. The controller wants to maximize future gains, so in certain cases it may query at a loss for a future gain. The model description is as follows: the system receives a pattern. It responds with one of the following actions:

1. **Classify** The payoff is $+G$ if correct and $-M$ if incorrect.

2. **Query** A teacher will classify the pattern and the classifier may or may not adjust itself accordingly. Payoff is $-Q$ for the query and $+G$ for correct classification (assuming that the teacher is reliable and credit is given for repeating the teacher's response).[1]

It is important to note that if the system chooses to classify, it does not receive feedback as to whether it classified correctly or not.

One can view this as a dynamic system with a policy which is a function of the classifier and a pattern. The **Classify** action does not change the policy, but the **Query** action may change the policy and classifier.

[1]One might consider other types of queries with different costs: deferred queries, conditional queries etc. For the time being we consider only this one type with a fixed cost.

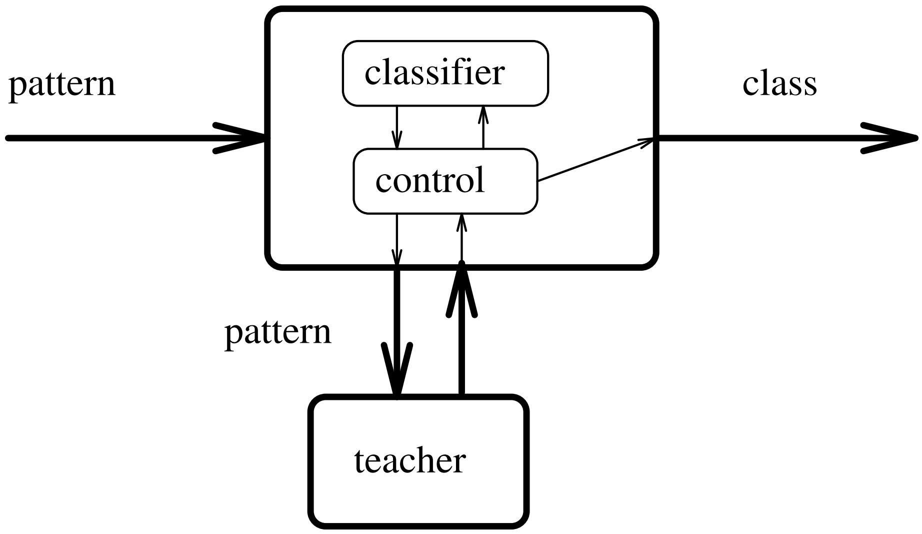

Figure 5.1: The on-site model.

5.4 When Does One Query?

In this section we consider a few policies for querying in a binary concept learning problem. Our first policy (A) ignores utility considerations and assumes that the concept is fully realizable by the representation chosen:

Query when the pattern is ambiguous w.r.t the version space.

This policy queries only patterns that are labeled differently by at least two concepts in the version space. It is guaranteed to make the minimum number of queries in the worst case, but may perform poorly on average. This policy is useful in that it supplies the intuition behind query-based learning: query when there is uncertainty. In practice one may not query even when one is uncertain if one take utility considerations into account.

If the target concept is not necessarily realizable or the inputs are corrupted by noise, we define a model space $\mathcal{H}$. Using the posterior probability distribution $P(h)$ over the model space and Bayes rule, we can assign probabilities to the pattern's label $P_+(\vec{x})$ and $P_-(\vec{x}) \equiv 1 - P_+(\vec{x})$. If we are to classify, the probability of correctness is $P_c(\vec{x}) = \max(P_+(\vec{x}), P_-(\vec{x}))$. At this point we can compare the utility of classification:

$$\mathcal{R}(\vec{x}) = P_c(\vec{x}) \times G - (1 - P_c(\vec{x})) \times M \tag{5.1}$$

to the utility of querying: $G - Q$. From this we compute policy (B) which queries only if

$$P_c(\vec{x}) \;<\; 1 - \frac{Q}{M+G} \;\equiv\; 1 - \eta \tag{5.2}$$

The utility of policy B is therefore

$$\mathcal{R}'(\vec{x}) = \max(G - Q, \mathcal{R}(\vec{x})) \tag{5.3}$$

Thus far we have not considered the future gain from a query. For that purpose one needs to take into consideration the distribution of examples $\rho(\vec{x})$. The expected utility of classification is therefore:

$$\overline{\mathcal{R}} = \int \rho(\vec{x})[P_c(\vec{x}) \times G - (1 - P_c(\vec{x}) \times M)]d\vec{x} \tag{5.4}$$

And the policy's expected utility is :

$$\overline{\mathcal{R}}' = \int \rho(\vec{x}) \max(G - Q, \mathcal{R}(\vec{x}))d\vec{x} \tag{5.5}$$

Now we can examine a myopic strategy of one-step lookahead. Since the model space changes as a result of a query, we subscript the appropriate functions with their respective model space. $\mathcal{H}$ denotes the original model space. $\mathcal{H}1$ and $\mathcal{H}2$ denote the two possible resulting model spaces after the outcome of the query is known. The expected two-step utility for classifying a pattern x is

$$\mathcal{R}_{\mathcal{H}}(\vec{x}) + \overline{\mathcal{R}}'_{\mathcal{H}} \tag{5.6}$$

while the expected two-step utility for querying is:

$$G - Q + P_{c,\mathcal{H}}(\vec{x})\overline{\mathcal{R}}'_{\mathcal{H}1} + (1 - P_{c,\mathcal{H}}(\vec{x}))\overline{\mathcal{R}}'_{\mathcal{H}2} \tag{5.7}$$

Policy (C) is obtained by solving for $P_c(\vec{x})$ such that querying gives a higher expected utility. Adjusting a policy for future gain can only increase the threshold in Equation 5.2. Thus the threshold for such a policy lies between $1 - \eta$ and 1 (i.e the thresholds of policies (B) and (A)). The solution in general is not simple and becomes more complex if we look further into the future.

5.4.1 An example: the high low game

As an example we study the one dimensional high-low game. The concept is a point h on the interval $[0, 1]$. Points less than h are negatively labeled and those greater than or equal to h, are labeled positively. The version space, $V \equiv [x^-_{max}, x^+_{min}]$ is between the largest negative example and the smallest positive example seen so far. P_c is 1 outside the version space and less than one inside. We assume that the pattern distribution is uniform over $[0, 1]$. The three policies from the previous section are:

A	$P_c < 1$	Query if inside the version space
B	$P_c < 1 - \eta$	Query according to Bayes' rejection criteria
C	$P_c < 1 - \eta + \|V\|\eta(1-\eta)/2$	Query according to one step lookahead

One may see that the query threshold increases as one takes into account more steps in the future. The average expected cost of classifying is

$$\langle c \rangle = \frac{(M+G)|V|}{4} \tag{5.8}$$

By comparing this to the expected cost after a query, we find that, for a payoff within n steps, one should query if

$$P_c < \frac{1}{2} + \frac{\sqrt{1 - \frac{8\eta - 2|V|}{n|V|}}}{2} \tag{5.9}$$

As expected, as one looks further into the future ($n \to \infty$), the policy approaches policy (A). Also, for fixed n the policy will eventually not query when the version space, V, becomes sufficiently small.

5.5 Implementing an On-site System Using Ensembles

In this section we describe some experiments done on synthesized and real-world data. Estimating P_c and the optimal policy is quite difficult in practice. We know that the best on-site policy lies somewhere between policy (A) and policy (B). The key problem is maintaining the model space. We propose using an ensemble of learners to maintain an estimate of the model space.

Ensembles of classifiers have been proposed for better generalization (Hansen and Salamon, 1990; Perrone and Cooper, 1993; Krogh and Vedelsby, 1995) and combining classifiers in an ensemble has been shown to converge to the Bayes estimate in perceptron learning (Opper and Haussler, 1991).

Our use of ensembles of learners is a modification of the **Query-By-Committee (QBC)** algorithm, proposed by (Seung, Opper and Sompolinsky, 1992; Freund et al., 1993) for approximating the version space in a query filtering problem. The algorithm maintains a $2k$-committee of concepts drawn at random from the version space. A pattern is queried only if it is classified positively by half the committee and negatively by the other half. Seung et al. show that for a perceptron trained with linearly separable data, QBC has a constant information gain per query. This leads to an exponential decay of the generalization error as a function of queries.

Voting in QBC is used to estimate the information of a pattern's label with respect to the current version space. The ensemble is used to estimate the class probabilities $\hat{P}_+ = \frac{N_+}{2k}$ and to decide accordingly whether to query or not. The goal in the model under which Seung et al. suggested QBC was to minimize the number of labelings, therefore they query only the most informative patterns, i.e., P_c is as close as possible to $\frac{1}{2}$. In addition they do not consider any cost associated with ignoring or rejecting a pattern. Thus, their asymptotic results hold even when they reject all patterns with $P_c > \frac{1}{2} + \epsilon$ for arbitrarily small ϵ. In the on-site model delaying the gain of extra knowledge will result in a penalty of a higher error rate. For this reason we wish to use QBC to estimate P_c and reject according to the cost model.

The following experiments were conducted to explore the potential of QBC as an applicable algorithm. QBC was originally studied under two unrealistic conditions: (1) The target concept is realizable by the learners and uncorrupted by noise, i.e., there exists a version space. (2) The committee members are chosen at random from the version space at each step (Gibbs sampling).

In the following subsections we show that despite the above preconditions, the non-"ideal" versions of QBC perform adequately. In particular, even the use of a two member committee can help focus on relevant patterns and reduce the total cost of labeling and making errors under a wide range of cost models. The two member committee gives a quantized estimate of P_c taking only the two values 1 and 0.5 ("certain","uncertain"), and

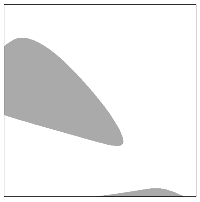

Figure 5.2: The target concept 252a. The domain of the concept is $[-1,1]^2$. Grey areas are labeled $+$, white areas are labeled $-$. Samples for training and testing were drawn uniformly from the domain.

enforcing a policy of type (A).[2]

5.5.1 A synthesized problem: 252a

We have chosen to demonstrate the use of QBC on a synthesized problem since it allows us to generate large streams of data. The classification task is shown in Figure 5.2. It is a reasonably simple asymmetric two dimensional function that one can visualize, yet it is not easy to fit with a moderate size architecture. It is composed of two regions, one of which is small with respect to the domain size, leaving it prone to a misleading local minima solution.

Feed forward networks

The architecture used was a 4 layer fully connected network (2x20x20x2) trained by back-propagation (BP). We compared three different systems all presented with the same stream of 200,000 patterns.

1. `Comm100`: A 2-member ensemble trained on the first 100 patterns.
2. `Comm1000`: A 2-member ensemble trained on the first 1000 patterns.
3. `Qbc100`: A 2-member committee trained using QBC. The initial committee members were set to be the networks in ensemble `Comm100`.

The results presented are averaged over ten independent runs (different streams and initial weights of the networks). `Comm100` and `Comm1000` were investigated under two different policies. The first policy always classifies. In this case the outputs of the two networks were added and the highest scoring class was the response. The second policy rejects patterns on

[2] Effectively the policy is somewhere between (A) and (B).

which the two members disagree. The rejected patterns are considered as queries. Results of off-site training of a single network are not presented but they were found to be similar yet slightly worse than ensembles (Hansen and Salamon, 1990).

The QBC network classified the patterns that the committee agreed upon and queried when there was disagreement. Each time an additional 20 new patterns had been queried, the committee was retrained on the complete set of queried patterns. This is similar to the method described in (Cohn, Atlas and Ladner, 1994), yet differs in that we use the current networks as the new initial points.

In this work we did not concern ourselves with the computational requirements of the methods. `Comm100` and `Comm1000` were trained for 500,000K BP iterations and cross-validated with 10,000 independent patterns to choose the best weights.[3] The QBC learning epochs were of 50K BP iterations with no validation criteria.

Figures 5.3a-f summarize the results. Given the number of errors and queries made by each system we can examine their cost under different cost models, i.e., different values of $\eta \equiv \frac{Q}{G+M}$:

$$\text{Cost} = \eta \times (\text{\# of queries }) + (\text{\# of errors}) \tag{5.10}$$

Costs are plotted in Figures 5.3d-f. One can see that, for any value of η, QBC will eventually outperform `Comm1000`.

`Comm100` and `Comm1000` exhibit fixed error and query rates. `Qbc100`'s errors rates and query rates decrease till they reach a constant. That is, at a certain point in time, additional queries do not decrease the error rate. This is due to a lack of capacity (Vapnik, Levin and Le Cun, 1994) of the committee learners for fitting the target concept. Adding more training points, however informative, does not help. A system detecting a constant rate of querying should either terminate querying or increase its capacity.

Nearest neighbor classifiers

QBC can easily be used with any incremental supervised learning algorithm that generates different concepts due to varying initial conditions. This can be extended to any learning procedure where the concepts do not depend on an initial state (e.g. nearest neighbors). For these algorithms, constructing different committee members is achieved by splitting the

[3] This could be considered "cheating', but our goal is to show that even "stupid" on-site strategies can outperform "smart" off-site strategies.

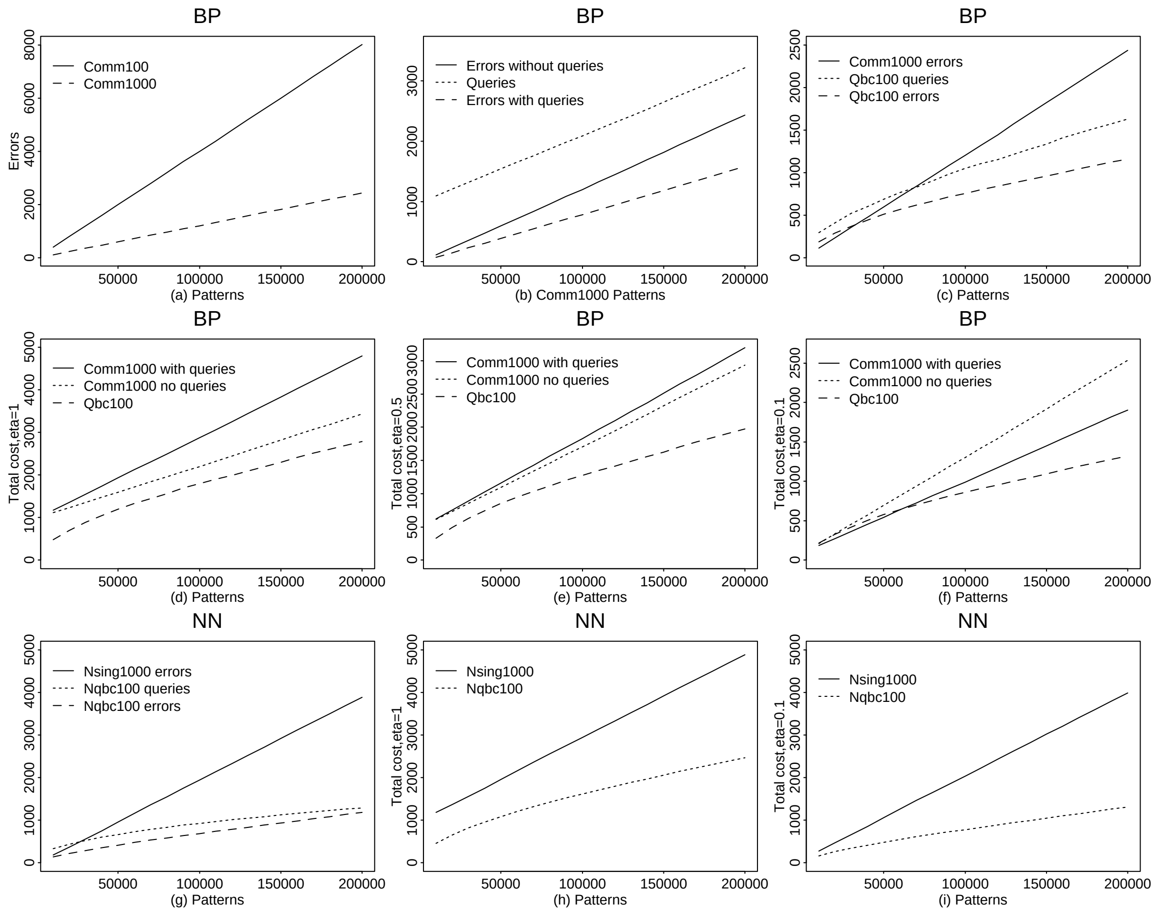

Figure 5.3: *[Note the changes in scale].* Back-propagation (BP) and nearest neighbor (NN) runs trained on the concept 252a. **(a)** Demonstration of the simple fact that training on 1000 random patterns (`Comm1000`) gives better results than training on 100 random patterns (`Comm100`). **(b)** `Comm1000` behavior with and without querying. **(c)** Raw results of BP runs. The error and query rates of `Qbc100` decrease till they reach a constant. Also `Qbc100`'s cumulative error drops below that of `Comm1000` before it has made 500 queries. **(d-f)** Comparing the two different policies of `Comm100` and that of `Qbc100` for different values of η: $\{1, 0.5, 0.1\}$. One can see that as the cost of a query decreases, querying becomes more attractive even for the off-site network. **(g)** Raw results of NN runs. `Nqbc100`'s query rate drops off very rapidly. **(h-i)** Total cost for NN with $\eta = \{1, 0.1\}$. The payoff for QBC seems more pronounced than in the feed forward networks.

training set between committee members and utilizing the randomness in the input stream. We have experimented with this modified version of QBC with nearest neighbor classifiers. We performed a comparison of the following systems:

1. `Nsing1000`: A single classifier consisting of the first 1000 patterns in the stream.

2. `Nqbc100`: A 2-member committee trained using QBC.

As in the previous section the results are averaged over 10 independent runs. For `Nqbc100` the first 100 patterns in the stream where divided between the two committee members. Following that, whenever there was disagreement, the queried pattern was added only to the classifier that had made an error (this is always possible since this is a two-class problem). The raw results and the total costs for different values of η are shown in Figures 5.3g-i.

It seems that QBC exhibits a more dramatic effect for nearest neighbors than it did for the BP networks. The query rate seems to decrease towards zero, yet the error rate reaches a constant as it did in the BP networks. The reason for this phenomenon is not lack of capacity, but erroneous local agreement between classifiers. To highlight this problem, imagine the highly unlikely case where the initial committee members receive the same 50 initial patterns. Under these conditions the committee members would always agree, never query and never decrease the initial error rate. Though this extreme case is unlikely to happen, it is quite likely that the two committees will agree erroneously in certain "pockets" of the distribution without ever shrinking them.

This limitation of QBC may occur for any learning algorithm that is not truly sampling the model space, but is conducting a beam search (Rich and Knight, 1991) in the space. Nearest neighbor classifiers seem more prone to encounter this problem due to the local nature of their representation. We conjecture that the severity of this problem may decrease if the committee size increases or we use k-nearest neighbors.

5.5.2 An OCR problem

We have experimented with QBC with two different classifiers and learning algorithms: a neural network trained by back-propagation and a nearest neighbor classifier. The data, on which we trained and tested, was a handwritten digit database described in (Le Cun et al., 1990). In both cases we used a two-member committee and compared its accuracy to a single learner trained on the same data. The results in both cases are not meant to exhibit state-of-the-art classification rates, but to show that the committee can achieve similar accuracy while requiring fewer resources. The results are summarized in the following table:

	% reject	% error	labels used
BP single	6	4.7	9709
BP QBC	6	4.2	1585
NN single	18.1	3.0	2000
NN QBC	18.1	1.8	577

We did not study the utility of the different methods since the data set is not large enough to exhibit long term effects. In addition the results are not meant to exhibit state-of-the-art classification rates, but to show that the QBC algorithm is able to "sift" through the input stream without losing generalization ability. It is interesting to note that the OCR runs displayed behavior similar to that of the synthetic data. In these runs we also observed the phenomenon of the query rate reaching a constant.

Feed forward networks

The architecture used was a simple 3 layer ($256 \times 30 \times 10$) fully connected feed-forward network. We trained the single network for 30 iterations over a training set of 9709 patterns for a total of 291K back-propagation iterations. The error rate on the test set was 7.7% (209 misclassified out of 2711).

We simulated a QBC scenario by passing through the training set multiple times (30 iterations were performed) and at each pass querying patterns upon which the two network disagreed. Patterns that had been queried in the past were trained upon whether there was agreement or not. 1585 patterns out of the 9709 were labeled and there were 42.5K back-propagation iterations. The accuracy on the test set was: 164 patterns (6%) were rejected and 102 (4.17%) were misclassified. If we reject the same number of patterns in the single network, the error rate drops to 4.7% (115 patterns).

The results show that using only 16% of the data we are able to achieve the same level of accuracy. Even though intermediate back-propagation networks are not necessarily good representatives of the model space it seems they may be sufficiently close.

Nearest neighbor classifiers

We used the same modified QBC algorithm described in the previous section where committee members are built by splitting the training set between committee members and utilizing the randomness of the input stream.

The single nearest neighbor classifier used 2000 training patterns. The error on the set was 5.5% (55 misclassified out of 1000).

The committee classifiers started with 10 random pattern each and alternately added a queried pattern to each of the classifiers. The total number of pattern used was 577 (including the initial 20). The error rate on non-queried patterns in the training set was 4.3%. The accuracy on the test set was 181 patterns rejected and out of the 819 pattern accepted, 18 patterns (1.8%) were misclassified. If we reject the 181 lowest scoring patterns using the single classifier, the misclassifications dropped to 25 (3%).

The results for the nearest neighbor classifier seem quite surprising. Only 30% of the data was used for comparable accuracy results. QBC may be a useful method for **editing** (Dasarathy, 1991): generating a small set of patterns that conveys the same information as the original larger set.

5.6 Discussion and Future Work

We have presented the on-site model and studied the design of a policy for querying during the performance stage. Query-By-Committee was tested as a possible method for approximating the model space in an on-site system and demonstrated encouraging results.

The on-site model may be seen as a combination of models studied in previous work. On one hand, *query filtering* focuses on minimizing the number of queries needed to reach a given error rate. On the other hand, the goal of *on-line* learning (Littlestone, 1988) is to minimize the errors made over time. On-site learning tries to minimize a combination of queries and errors with respect to a utility model.

We have studied the model under the assumption that the target concept is static. Additional work is needed to extend the methods to deal with concept drift. Even if a classifier is trained off-site, studying on-site learning is crucial if we wish the system to remain current and adapt to unforeseen cases. In addition, computational issues need to be addressed to efficiently incorporate queried patterns into the classifier's internal representation.

The two actions: *Classify* and *Query* allows the consideration of more complex learning models that are worth studying in the future:

- Combining the use of constructive queries with query filtering.
- Use of queries for accuracy validation and detection of concept drift.
- Use of the non-queried data to estimate the pattern distribution.

We have shown the QBC algorithm to be a useful method for on-site learning for both feed forward networks trained with back-propagation and nearest neighbor classifiers. The experiments also demonstrate some of QBC's limitations due to limited capacity of the learners, poor sampling of the model space and erroneous local consensus. Further study is needed to fix some of these problems and explore other candidate methods.

We conjecture that on-site systems need to keep information on the data distribution in addition to their current hypothesis, in order to distinguish between signal and noise. One method to implement such a system is to map the training set to detect uncertain regions. Alternatively one could use a classifier's "natural" confidence level (MacKay, 1992).

Chapter 6

Conclusion

In this chapter we summarize and comment on areas related to the work presented in the dissertation.

6.1 Small Datasets, Large Datasets and Filtering

The analysis done in this dissertation has taken the view that there exists a practically unlimited training source of patterns. Many applications generate enormous streams of data to tap into. Two examples from different areas are credit card transactions and radio telescopic measurements. For practical purposes these sources can be considered unlimited.

Learning from large corpora of data is not equivalent to learning from a small amount of data (see e.g. (Catlett, 1991)). In the past the machine learning and statistics communities focused on smaller data sets. Some of the reasons for focusing small training sets are:

1. More data does not exist. *e.g.* natural phenomena such as of the number of annual sun spots.

2. Computation limitation. Testing algorithms on small data sets is easy. The technology in the past limited the training set size one could train and test on.

3. Resources and effort. It take time and effort to collect a lot of data. Until recently, storage space was scarce and expensive.

The Irvine machine learning dataset repository (Merz and Murphy, 1996) holds many popular benchmark training data sets. The size of most of the datasets ranges between hundreds and several thousands of examples. Most of the data sets were constructed when models examined were relatively simple. As more complex representations were proposed and computing power enabled better search, researchers became aware of **overfitting** where the hypothesis accuracy on the training set is much better than its performance on the real data (Geman, Bienenstock and Doursat, 1992; Breiman et al., 1984). Much of the community's effort since then has been focused on model selection and error estimation in an effort to overcome the overfitting problem. While solving overfitting is of both intellectual and practical importance, we have chosen to concentrate on cases where the difficulties in learning are not due to lack of data.

It is well understood that in order to overcome overfitting it is necessary to structure the search by preferring certain models over others. From a Bayesian perspective this is equivalent to setting prior probabilities over the hypothesis space (Berger, 1985). In the realm of unlimited data, this is a non-issue since any initial biases or ill-formed priors are eventually overturned when trained on a large amount of data (Berger, 1985).

Often the following claim is made : "If we had more data, we could build a more accurate classifier". Does this this claim hold? To certain extent it does. For most applications increasing the number of training samples from hundreds to thousands makes a big difference. However can we expect improvement by increasing the sample set size from one hundred thousand examples to one million examples?

As mentioned in Chapter 5 the residual information of additional patterns diminishes rapidly. Focusing on this residual information (i.e. the "interesting" patterns) is akin to finding a needle in a haystack. The problem of improving the accuracy of one's classifier remains tough even when one has the correct representation. Much of the burden falls on the search through the model space. In addition, the quantity of data processed in very large databases makes training potentially a costly process. These facts amplify the necessity of filtering methods in problems with a lot of data. We have used filtering to address both the issues of increasing accuracy (Chapters 3 and 4) and minimizing training costs (Chapter 5).

6.2 Bias and Variance

Statisticians divide the average error of a learned hypothesis into two components *bias*[1] and *variance*.

[1]This *bias* should not to be confused with the representational bias or search bias we mentioned earlier.

Imagine that a learning algorithm is run multiple times on different training sets of equal size, but drawn from the same distribution. The **bias** measures how far off the average hypothesis is from the target concept. The **variance** measures how dissimilar the different hypotheses are. If an algorithm always returns the same hypothesis, all the error is due to bias. However, if on average, the hypotheses agree with the target concept, then all the error is due to variance.

We have already encountered phenomena related to bias and variance:

- "Weakness" is related to bias. A learning machine with limited capacity, either due to the representation or the search, will necessarily have bias.
- Overfitting is related to variance. Overfitting occurs when the model is under determined by the data (i.e. the capacity is sufficient) such that the learning machine fits the particularities in the training set. This will manifest itself as variance among different classifiers

Do equal vote ensembles reduce either bias or variance? The answer is that they can do either. Ensembles can reduce bias by increasing the effective capacity of the base level learning machine. We have studied this possibility in Chapters 3 and 4.

If the representation is under-determined by the data, voting or averaging ensembles can reduce variance (Breiman, 1994). If we assume the learning machine is unbiased, then the classifier resulting from averaging will have a smaller variance than any other individual in the ensemble and will still be unbiased[2]. Furthermore, an ensemble can serve as a gauge to the variance or uncertainty. We used this to our advantage for on-site learning in Chapter 5.

6.3 Theory and Practice

In this work I have made an effort to bridge theory and practice. There is often a gap between the two which is due to the fact that it is hard to analyze algorithms under conditions that exist in practice. Still there is a real need for furthering our understanding of machine learning if it is to become a principled engineering process and less of a black art.

I believe that one way to bridge the gap is by emulating theoretical work done in natural sciences: mathematical modeling of phenomena. While this type of analysis may not always

[2]This is due to a similar argument to that mentioned in Section 3.7.2

result in broad general theorems, it can give insight into the behavior of algorithms as a function of the problem specification.

In Chapters 4 and 5 we examined SBA and QBC. Both are algorithms proposed by the theoretical community. We have examined them analytically and experimentally and tried to understand their strengths and weaknesses in practice.

SBA has been applied with limited success (Drucker, Schapire and Simard, 1993; Drucker et al., 1994). Its main practical difficulties are: (a) It does not alway boost classification performance and (b) it requires a large amount of labeled training data (In (Drucker, Schapire and Simard, 1993) a distortion pattern generator was used to generate additional new patterns from the training set). In Chapter 4 we addressed both of these problems. We analyzed the relationship between the complexity of the base classifier and the efficiency of the boosting algorithm's search of the hypothesis space. In addition, we proposed a modified boosting algorithm that uses far fewer patterns. Boosting by rejection gives no theoretical guarantees, yet performs reasonably well in practice at a lower labeling cost.

In Chapter 5 we experimentally applied QBC to both real-world and synthesized problems. Only recently have others experimented with its use (Tirthankar and Hamey, 1995). We expanded on its proposed intent by suggesting its use for on-site learning.

In Chapter 3 we followed the opposite direction by looking at a practitioner's method and analyzing it theoretically. Majority-vote classifiers have been used in an ad-hoc fashion for some time. There are cases in which they succeed in improving accuracy and others in which they fail to provide any advantage. Our analysis results in exact bounds on the accuracy, yet more importantly, it shines light on the conditions in which ensembles will perform well. Unlike previous work that suggested making individual classifiers either independent or negatively correlated (Ali, 1996), our work shows that a balance is needed between agreement and disagreement to give the best boost in accuracy.

6.4 Summary

Ensembles in machine learning are currently in vogue. To a certain degree they have become popular as a result of helplessness in the community. Many of the popular techniques are not well understood and not much progress has been made in understanding the underlying processes. For example, there is still very little understanding of the back-propagation learning algorithm for feed-forward networks. The lack of understanding has lead to slow progress in improving state-of-the art accuracy rates.

Many learning architectures are indeed combinations of self-similar structures. For example one can view a 3-layer network as an ensemble of hidden-layer classifiers and argue that back-propagation is doing some form of boosting. One promise of research in the area of ensembles is that it may lead to better understanding of base-level representations and algorithms that learn these representations.

Appendix A

Accuracy of a Majority-Vote Classifier

We provide a proof of the upper bound on the accuracy of a special majority classifier (Theorem 1 on page 28).

Theorem *Given an ensemble of n classifiers $\{h_1, h_2, ..., h_n\}$ with respective accuracies $g_1 \leq g_2 \leq ... \leq g_{n-1} \leq g_n$ under domain distribution $\mathcal{D}$, The accuracy g_k^n of an k-of-n special majority classifier under the same distribution is bounded from above by:*

$$\max(g_k^n) = \min(1, \Sigma(k), \Sigma(k-1), \ldots, \Sigma(2), \Sigma(1)) \tag{A.1}$$

where

$$\Sigma(m) \equiv \frac{1}{m} \sum_{i=1}^{n-k+m} g_i \qquad m = 1 \ldots k \tag{A.2}$$

Proof: The proof consists of the following steps.

1. Prove that g_k^n is bounded above by the expression stated in the theorem.

2. Prove the reformulation A.15.

3. Show that there is a feasible solution at the bound (i.e. the bound is tight).

A.1 Definitions

We are given a set of n classifiers $h_1, \ldots, h_n$. The accuracy of each classifier may be written as

$$g_i = \int_X \mathcal{I}_i(\vec{x})\rho(\vec{x})d\vec{x} \tag{A.3}$$

Where $\mathcal{I}_i(x)$ is the correctness indicator function. The indicator function for the majority classifier can be shown to be:

$$\mathcal{I}_k^n(\vec{x}) = \begin{cases} 1 & \sum_1^n \mathcal{I}_i(\vec{x}) \geq k \\ 0 & \sum_1^n \mathcal{I}_i(\vec{x}) < k \end{cases} \tag{A.4}$$

We can write the following expression for g_k^n:

$$g_k^n = \int \mathcal{I}_k^n(\vec{x})\rho(\vec{x})d\vec{x} = \int_{\sum \mathcal{I}_i \geq k} \rho(\vec{x})d\vec{x} \tag{A.5}$$

In the following analysis we treat g_k^n as a mathematical entity. Although values of $k \leq \frac{n}{2}$ do not make practical sense for majority classifiers, both h_k^n and g_k^n are mathematically well defined according to equations 3.1 and 3.3. The semantics of such an entity is the fraction of the patterns in the input distribution on which at least k classifiers are correct.

A.2 Step 1: Upper bound

We find that g_k^n is bounded by a series of inequalities: the first is that g_k^n can be no larger than unity:

$$g_k^n \leq 1 \tag{A.6}$$

All patterns in a region for which the majority classifier is correct is classified correctly by at least k classifiers in the ensemble:

$$\sum_1^n g_i = \sum_1^n \int_X \mathcal{I}_i(\vec{x})\rho(\vec{x})d\vec{x} = \int_X \sum_1^n \mathcal{I}_i(\vec{x})\rho(\vec{x})d\vec{x} \tag{A.7}$$

$$\geq \int_{\sum \mathcal{I}_i \geq k} \sum_1^n \mathcal{I}_i(\vec{x})\rho(\vec{x})d\vec{x} \geq \int_{\sum \mathcal{I}_i \geq k} k\rho(\vec{x})d\vec{x} \geq kg_k^n \tag{A.8}$$

Therefore we get the additional bound:

$$g_k^n \leq \frac{1}{k}\sum_1^n g_i \tag{A.9}$$

If we remove any one classifier from the ensemble there are still at least $k-1$ classifiers that agree on g_k^n of the distribution. We use this fact in the following calculation. For arbitrary $1 \leq j \leq n$:

$$\begin{aligned}
\sum_{i\neq j} g_i &= \sum_{i\neq j}\int_X \mathcal{I}_i(\vec{x})\rho(\vec{x})d\vec{x} = \int_X \sum_{i\neq j}\mathcal{I}_i(\vec{x})\rho(\vec{x})d\vec{x} & \text{(A.10)}\\
&\geq \int_{\sum \mathcal{I}_i \geq k} \sum_{i\neq j}\mathcal{I}_i(\vec{x})\rho(\vec{x})d\vec{x} \geq \int_{\sum \mathcal{I}_i \geq k}(k-1)\rho(\vec{x})d\vec{x} \geq (k-1)g_k^n & \text{(A.11)}
\end{aligned}$$

In particular, this bound is strictest when we remove the classifier h_n, which has the highest accuracy g_n, which is larger or equal to all others (by our convention of ordering the classifiers). We get the following bound

$$g_k^n \leq \frac{1}{k-1}\sum_1^{n-1} g_i \tag{A.12}$$

Using the same reasoning as above we can remove additional classifiers to achieve the following bound for any m between 1 and k:

$$g_k^n \leq \frac{1}{m}\sum_{i=1}^{n-k+m} g_i \equiv \Sigma_k^n(m) \tag{A.13}$$

We will write $\Sigma(m)$ without the modifiers while it is clear from context.
Therefore g_k^n is bounded from above by the most restrictive of the above bounds:

$$g_k^n \leq \min(1, \Sigma(k), \Sigma(k-1), \ldots, \Sigma(2), \Sigma(1)) \tag{A.14}$$

A.3 Step 2: Reformulation

We rewrite Equation A.14 to explicitly state under which each bound is most restrictive.

Lemma 3 *Equation A.14 is equivalent to the following equation:*

$$\max(g_k^n) \leq \begin{cases} 1 & \Sigma(k) \geq 1 & \\ \Sigma(k) & \Sigma(k) < 1 & \& \quad \Sigma(k-1) \geq g_n \\ \Sigma(k-1) & \Sigma(k-1) < g_n & \& \quad \Sigma(k-2) \geq g_{n-1} \\ . & & \\ . & & \\ \Sigma(k-j) & \Sigma(k-j) < g_{n-j+1} & \& \quad \Sigma(k-j-1) \geq g_{n-j} \\ . & & \\ . & & \\ \Sigma(2) & \Sigma(2) < g_{n-k+3} & \& \quad \Sigma(1) \geq g_{n-k+2} \\ \Sigma(1) & \Sigma(1) < g_{n-k+2} & \end{cases} \tag{A.15}$$

The notation may seem complicated but it has a uniform structure. Each term of the form $\Sigma() < g_m$ compares the accuracy of the m-th classifier with an average accuracy of all classifiers with accuracy smaller than g_m (this average is the best accuracy an ensemble without classifier h_m can achieve).

We use a few propositions to prove the reformulation is equivalent. Lemma 3 will follow from Propositions 10 and 11 that are proven below.

Proposition 4

$$\Sigma(m) < g_{n-(k-m)+1} \Rightarrow \Sigma(m+1) < g_{n-(k-m)+2} \tag{A.16}$$

Proof

We start with the premise:

$$\sum_{1}^{n-(k-m)} g_i < m g_{n-(k-m)+1} \tag{A.17}$$

Add $g_{n-(k-m)+1}$ *to each side:*

$$\frac{1}{m+1} \sum_{1}^{n-(k-m)+1} g_i \; < \; g_{n-(k-m)+1} \; \leq \; g_{n-(k-m)+2} \tag{A.18}$$

Arriving at the conclusion

$$\Sigma(m+1) < g_{n-(k-m)+2} \tag{A.19}$$

▮

Corollary 5

$$\Sigma(m) \geq g_{n-(k-m)+1} \Rightarrow \Sigma(m-1) \geq g_{n-(k-m)} \tag{A.20}$$

Proof

Take the counter-positive of Proposition 4 and change the variable m to $m-1$. ▮

Proposition 6

$$\Sigma(m) \leq \Sigma(m-1) \Leftrightarrow \Sigma(m-1) \geq g_{n-(k-m)} \tag{A.21}$$

Proof

$$(m-1) \sum_{1}^{n-(k-m)} g_i \leq m \sum_{1}^{n-(k-m)-1} g_i \quad \Leftrightarrow \tag{A.22}$$

$$(m-1) g_{n-(k-m)} \leq \sum_{1}^{n-(k-m)-1} g_i \quad \Leftrightarrow \tag{A.23}$$

$$g_{n-(k-m)} \leq \Sigma(m-1) \tag{A.24}$$

▮

Corollary 7

$$\Sigma(m) \leq \Sigma(m+1) \Leftrightarrow \Sigma(m) < g_{n-(k-m)} \tag{A.25}$$

Proof

Take the counter-positive of Proposition 6 and change the variable m to $m+1$. ▮

Corollary 8

$$\Sigma(m-1) \geq g_{n-(k-m)} \;\Rightarrow\; (\forall j < m) \quad \Sigma(m) \leq \Sigma(j) \tag{A.26}$$

This corollary states that if the second condition on a given line in equation A.15 holds, then the corresponding value is smaller than all values on lines below.

Proof

By induction on j and application of Corollary 5 and Proposition 6. ▮

Corollary 9

$$\Sigma(m) < g_{n-(k-m)+1} \;\Rightarrow\; (\forall j > m) \quad \Sigma(m) < \Sigma(j) \tag{A.27}$$

This corollary states that if the first condition on a given line in equation A.15 holds, then the corresponding value is smaller than all values on lines above.

Proof

By induction on j and application of Proposition 4 and Corollary 7. ∎

Proposition 10

$$\Sigma(k) \geq 1 \;\Rightarrow\; \min(1, \Sigma(k), \Sigma(k-1), \ldots, \Sigma(2), \Sigma(1)) = 1 \tag{A.28}$$

Proof

$$1 \leq \frac{1}{k}\sum_{i=1}^{n} g_i \tag{A.29}$$

For any $0 \leq j < k$

$$k - j \leq \sum_{i=1}^{n} g_i - j \tag{A.30}$$

$$k - j \leq \sum_{i=1}^{n-j} g_i - \sum_{i=n-j+1}^{n} 1 - g_i \leq \sum_{i=1}^{n-j} g_i \tag{A.31}$$

$$1 \leq \Sigma(k-j) \tag{A.32}$$

∎

Proposition 11

$$\Sigma(m) < g_{n-(k-m)+1} \;\&\; \Sigma(m-1) \;\geq\; g_{n-(k-m)} \tag{A.33}$$

$$\Rightarrow\; \min(1, \Sigma(k), \Sigma(k-1), \ldots, \Sigma(2), \Sigma(1)) = \Sigma(m) \tag{A.34}$$

Proof

By Corollary 8 and the second precondition, $\Sigma(m)$ is smaller than all $\Sigma(j)$ for $j < m$. Similarly, by corollary 9 and the first precondition, $\Sigma(m)$ is smaller than all $\Sigma(j)$ for $j > m$ and smaller than 1. ∎

Lemma 3 follows from Propositions 10 and 11.

A.4 Step 3: Upper bound is tight

We now have an upper bound for g_k^n for any ensemble with accuracies $G = \{g_1, \ldots, g_n\}$. We now prove that the bound is tight by showing that there exists an ensemble with accuracies G where the bound is met. We show this by proving the following lemma which is a recursive form of the bound in Lemma 3:

Lemma 12 *There exists an ensemble of n classifiers $\{h_1, h_2, ..., h_n\}$ with respective accuracies $g_1 \leq g_2 \leq ... \leq g_{n-1} \leq g_n$ under domain distribution $\mathcal{D}$, such that the accuracy g_k^n of the k-of-n special majority classifier, for $n > k > 1$ is:*

$$\max(g_k^n) = \begin{cases} 1 & \Sigma(k) \geq 1 & (A) \\ \Sigma(k) & \Sigma(k) < 1 \ \& \ \Sigma(k-1) \geq g_n & (B) \\ \max(g_{k-1}^{n-1}) & \Sigma(k-1) < g_n & (C) \end{cases} \tag{A.35}$$

and for $n > k = 1$:

$$\max(g_1^n) \leq \begin{cases} 1 & \Sigma(k) \geq 1 \\ \Sigma(k) & \Sigma(k) < 1 \end{cases} \tag{A.36}$$

Proof

By induction on n and k. The induction hypothesis is used to prove case (B) in Equation A.35. We prove the base cases of g_1^n and g_{n-1}^n by using the linear program solutions Ω_β (see section 3.4):

Base case 1:

$$g_1^n \leq \begin{cases} 1 & \Sigma(k) \geq 1 \\ \Sigma(k) & \Sigma(k) < 1 \end{cases} \tag{A.37}$$

One needs only one classifier correct in the ensemble. If the sum is less than unity we set all g_i to be correct in different areas of the distribution. i.e. $\Omega_\beta = g_i$ for β such that $\beta_i = 1$ and $\beta_{j \neq i} = 0$. We apply Proposition 13 for the case when the sum is greater than one. ∎

Base case 2:

$$\max(g_{n-1}^n) = \begin{cases} 1 & \Sigma(n-1) \geq 1 & (a) \\ \Sigma(n-1) & \Sigma(n-1) < 1 \& \Sigma(n-2) \geq g_n & (b) \\ \max(g_{n-2}^{n-1}) & \Sigma(n-1) < g_n & (c) \end{cases} \tag{A.38}$$

Case 2a is similar to base case 1: the sum of the the errors $\Sigma\epsilon_i$ is less than unity therefore we can construct the h_i's so there are no overlapping errors and there are always

$n-1$ correct classifiers.

Case 2b is proven by giving the following solution for Ω_β:

$$\Omega_\beta = \begin{cases} \Sigma(n-1) - g_i & \beta_i = 0 \ \& \ \beta_{j\neq i} = 1 \\ 1 - \Sigma(n-1) & \beta = 000\ldots00 \\ 0 & otherwise \end{cases} \tag{A.39}$$

This solution satisfies the constraints

$$\sum \Omega_\beta = 1 \tag{A.40}$$

$$\forall i \quad \sum_{\{\beta|\beta_i=1\}} \Omega_\beta = g_i \tag{A.41}$$

From $\Sigma(n-2) \geq g_n$ it follows that the solution above is positive. The solution gives the required accuracy g^n_{n-1}:

$$g^n_{n-1} = \sum_{\{\sum \beta \geq n-1\}} \Omega_\beta = \Sigma(n-1) \tag{A.42}$$

Case 2c Follows from Propositions 14 and 15 shown below. ∎

Case A: This case follows from the following proposition, and the proof of Case B :

Proposition 13 *If accuracy g^n_k is achievable with classifiers of accuracies $g_1, \ldots, g_n$, then one can achieve at least the same accuracy g^n_k with classifiers of accuracy $\tilde{g}_1, \ldots, \tilde{g}_n$, where $\forall i \ \tilde{g}_i \geq g_i$.*

This proposition is quite intuitive. Improving the accuracy of any classifier can only raise the upper bound on the overall accuracy of the majority-vote classifier

Proof

By construction. Set classifier $\tilde{h}_i(\vec{x})$ to be equal to $h_i(\vec{x})$ wherever $h_i(\vec{x})$ is correct, set it to be equal to the target function, $C(\vec{x})$, in some arbitrary portion of size $\tilde{g}_i - g_i$ of the the domain and set it to $1 - C(\vec{x})$ elsewhere. As a result $\tilde{h^n_k}$ has a majority of at least k in every part of the domain that h^n_k has a k-majority. Therefore the accuracy $\tilde{g^n_k}$ is at least g^n_k. ∎

Case B: We prove by construction. We construct a k-of-n majority classifier by adding classifier h_n to a $(k-1)$-of-$(n-1)$ ensemble that achieves the upper bound $\max(g^{n-1}_{k-1})$ which is greater than g_n.

We set the concept h_n such that $\mathcal{I}_n$ is a subset of $\mathcal{I}_{k-1}^{n-1}$. That is

$$\mathcal{I}_n = 1 \quad \Rightarrow \mathcal{I}_{k-1}^{n-1} = 1 \tag{A.43}$$

This will divide each classifier h_i in the ensemble into two parts : $h\prime_i$ and $h\prime\prime_i$. The first agrees with h_n and the later disagrees. The double prime ensemble together with h_n is a k-of-n classifier with accuracy g_n. The single prime ensemble is a $(k-1)$-of-$(n-1)$ ensemble with accuracy:

$$\frac{1}{k-1}\sum_{1}^{k-1} g_i - g_n \tag{A.44}$$

This ensemble is composed of the "residual accuracy" of the original ensemble. Since there are at least $k-1$ correct classifiers in this residual ensemble we know that

$$\sum_{1}^{n-1} g\prime_i \geq \sum_{1}^{k-1} g_i - (k-1)g_n \tag{A.45}$$

This new ensemble must be in some order where some $g\prime_j$ is less or equal to $\Sigma\prime(k-2)$. Therefore according to the inductive hypothesis there exists a (k)-of-$(n-1)$ ensemble $\tilde{h}_1, \ldots\tilde{h}_{n-1}$ with accuracy

$$\frac{1}{k}\sum_{1}^{n-1} g\prime_i = \frac{1}{k}[\sum_{1}^{k-1} g_i - (k-1)g_n] \tag{A.46}$$

Setting the new classifiers to be $h\prime\prime_i + \tilde{h}_i$ we get a total accuracy of

$$g_k^n = \frac{1}{k}[\sum_{1}^{k-1} g_i - (k-1)g_n] - \frac{k-1}{k}g_n + g_n \tag{A.47}$$

$$= \frac{1}{k}\sum_{1}^{k-1} g_i + \frac{g_n}{k} = \Sigma(k) \tag{A.48}$$

Case C: We first prove the following two propositions:

Proposition 14

$$\max(g_k^n) \leq \max(g_{k-1}^{n-1}) \tag{A.49}$$

Proof

By construction. Take the best majority ensemble k-of-n. For each pattern where h_k^n *is* *correct there are k classifiers which are correct. If we remove h_n there are at least as many $k-1$ corrects as there were k-corrects in the larger ensemble (there may be more since there may have been $k-1$ corrects in the original ensemble).* ▮

Proposition 15

$$\Sigma(k-1) < g_n \quad \Rightarrow \quad \max(g_k^n) \geq \max(g_{k-1}^{n-1}) \tag{A.50}$$

Proof

$\max(g_{k-1}^{n-1})$ *is bounded above by* $\Sigma(k-1)$ *and since* $\Sigma(k-1) < g_n$*, we can assert that*

$$g_n > \max(g_{k-1}^{n-1}) \tag{A.51}$$

We now construct a k-of-n majority ensemble from the best $(k-1)$-of-$(n-1)$ ensemble by adding h_n that is correct on any pattern where there are $k-1$ corrects in the smaller ensemble. This is possible due to inequality A.51. We arbitrarily complete h_n to give accuracy g_n. The result is that $\max(g_k^n)$ *is at least* $\max(g_{k-1}^{n-1})$. ▮

It follows from the two propositions above, that if $\Sigma(k-1) < g_n$ then $\max(g_k^n) = \max(g_{k-1}^{n-1})$. ▮

Appendix B

Solution for the 2-of-3 Majority Classifier

The following are the solutions for Ω_β for a 2-of-3 majority classifier (see Section 3.6, Equations 3.26 and 3.28):

B.1 Upper bound solution

- Case $g_2^3 = 1$:

		h_1	h_2	h_3
Ω_{000}	0			
Ω_{001}	0			0
Ω_{010}	0		0	
Ω_{100}	0	0		
Ω_{101}	$1 - g_2$	$1 - g_2$		$1 - g_2$
Ω_{110}	$1 - g_3$	$1 - g_3$	$1 - g_3$	
Ω_{011}	$1 - g_1$		$1 - g_1$	$1 - g_1$
Ω_{111}	$\sum g - 2$	$\sum g - 2$	$\sum g - 2$	$\sum g - 2$
total	1	g_1	g_2	g_3

- Case $g_2^3 = \frac{1}{2}(g_1 + g_2 + g_3)$:

		h_1	h_2	h_3
Ω_{000}	$1 - \frac{1}{2}\sum g$			
Ω_{001}	0			0
Ω_{010}	0		0	
Ω_{100}	0	0		
Ω_{101}	$\frac{1}{2}(g_1 + g_3 - g_2)$	$\frac{1}{2}(g_1 + g_3 - g_2)$		$\frac{1}{2}(g_1 + g_3 - g_2)$
Ω_{110}	$\frac{1}{2}(g_1 + g_2 - g_3)$	$\frac{1}{2}(g_1 + g_2 - g_3)$	$\frac{1}{2}(g_1 + g_2 - g_3)$	
Ω_{011}	$\frac{1}{2}(g_2 + g_3 - g_1)$		$\frac{1}{2}(g_2 + g_3 - g_1)$	$\frac{1}{2}(g_2 + g_3 - g_1)$
Ω_{111}	0	0	0	0
total	1	g_1	g_2	g_3

- Case $g_2^3 = g_1 + g_2$:

		h_1	h_2	h_3
Ω_{000}	$1 - g_2 - g_1$			
Ω_{001}	$g_3 - g_2 - g_1$			$g_3 - g_2 - g_1$
Ω_{010}	0		0	
Ω_{100}	0	0		
Ω_{101}	g_1	g_1		g_1
Ω_{110}	0	0	0	
Ω_{011}	g_2		g_2	g_2
Ω_{111}	0	0	0	0
total	1	g_1	g_2	g_3

B.2 Lower bound solution

- Case $g_2^3 = 0$:

		h_1	h_2	h_3
Ω_{000}	$1 - \sum g$			
Ω_{001}	g_3			g_3
Ω_{010}	g_2		g_2	
Ω_{100}	g_1	g_1		
Ω_{101}	0	0		0
Ω_{110}	0	0	0	
Ω_{011}	0		0	0
Ω_{111}	0	0	0	0
total	1	g_1	g_2	g_3

- Case $g_2^3 = \frac{1}{2}(g_1 + g_2 + g_3 - 1)$:

		h_1	h_2	h_3
Ω_{000}	0			
Ω_{001}	$\frac{1}{2}(2g_3 - \sum g + 1)$			$\frac{1}{2}(2g_3 - \sum g + 1)$
Ω_{010}	$\frac{1}{2}(2g_2 - \sum g + 1)$		$\frac{1}{2}(2g_2 + \sum g + 1)$	
Ω_{100}	$\frac{1}{2}(2g_1 - \sum g + 1)$	$\frac{1}{2}(2g_1 - \sum g + 1)$		
Ω_{101}	0	0		0
Ω_{110}	0	0	0	
Ω_{011}	0		0	0
Ω_{111}	$\frac{1}{2}(\sum g - 1)$	$\frac{1}{2}(\sum g - 1)$	$\frac{1}{2}(\sum g - 1)$	$\frac{1}{2}(\sum g - 1)$
total	1	g_1	g_2	g_3

- Case $g_2^3 = g_2 + g_3 - 1$:

		h_1	h_2	h_3
Ω_{000}	0			
Ω_{001}	$1 - g_2$			$1 - g_2$
Ω_{010}	$1 - g_3$		$1 - g_3$	
Ω_{100}	0	0		
Ω_{101}	0	0		0
Ω_{110}	0	0	0	
Ω_{011}	$g_2 + g_3 - g_1 - 1$		$g_2 + g_3 - g_1 - 1$	$g_2 + g_3 - g_1 - 1$
Ω_{111}	g_1	g_1	g_1	g_1
total	1	g_1	g_2	g_3

References

Ali, K. (1996). Learning probabilistic relational concept descriptions. PhD Thesis, University of California, Irvine.

Ali, K. and Pazzani, M. (1995a). Error reduction through multiple descriptions. Technical Report UCI TR 95-39, University of California, Irvine.

Ali, K. and Pazzani, M. (1995b). On the link between error correlation and error reduction in decision tree ensembles. Technical Report UCI TR 95-38, University of California, Irvine.

Angluin, D. (1988). Queries and concept learning. *Machine Learning*, 2:319–342.

Arrow, K. (1963). *Social choice and individual values.* John Wiley & Sons, 2nd edition.

Arrow, K. and Raynaud, H. (1986). *Social choice and multicriterion decision-making.* MIT Press.

Baum, E. (1991). Neural net algorithms that learn in polynomial time from examples and queries. *IEEE Trans. Neural Networks*, 2:5–19.

Berger, J. (1985). *Statistical decision theory and Bayesian analysis.* Springer.

Black, D. (1958). *The theory of commitees and elections.* Cambridge University Press.

Breiman, L. (1994). Bagging predictors. Technical Report, Statistics Department, University of California at Berkeley.

Breiman, L., Friedman, J., Olshen, R., and Stone, C. (1984). *Classification and Regression Trees.* Wadsworth International Group.

Bridle, J. S. (1989). Probabilistic Interpretation of Feedforward Classification Network Outputs with Relationships to Statistical Pattern Recognition. In Fogelman-Soulie, F. and Hérault, J., editors, *Neuro-computing: algorithms, architectures and applications.* Springer-Verlag.

Buchanan, B. and Smith, R. (1988). Fundamentals of expert systems. *Annual Review of Computer Science.*

Castelli, V. and Cover, T. (1995). On the exponential value of labeled samples. *Pattern Recognition Letters*, 16(1):105–111.

Catlett, J. (1991). Megainduction: machine learning on very large databases. PhD Thesis, University of Sydney.

Cohn, D., Atlas, L., and Ladner, R. (1990). Training connectionist networks with queries and selective sampling. In Touretzky, editor, *Advances in Neural Information Processing Systems*, volume 2. Morgan Kaufman.

Cohn, D., Atlas, L., and Ladner, R. (1994). Improving generalization with active learning. *Machine Learning*, 15:201–221.

Cohn, D., Ghahramani, Z., and Jordan, M. (1995). Active learning with statistical models. In Tesaro, G., Touretzky, D., and Alspector, J., editors, *Advances in Neural Information Processing Systems*, volume 7. MIT press.

Cover, T. and Thomas, J. (1991). *Information Theory.* John Wiley & Sons.

Dantzig, G. B. (1963). *Linear programming and extensions.* Princeton University Press.

Dasarathy, B. V., editor (1991). *Nearest neighbor (NN) norms : NN pattern classification techniques.* IEEE Computer Society Press.

Drucker, H. and Cortes, C. (1995). Improving pattern classification performance using boosting techniques. (preprint).

Drucker, H., Cortes, C., Jackel, L. D., LeCun, Y., and Vapnik, V. (1994). Boosting and other ensemble methods. *Neural Computation*, 6:1289–1301.

Drucker, H., Schapire, R., and Simard, P. (1993). Boosting performance in neural networks. *International Journal of Pattern Recognition and Artificial Intelligence*, 4:705–719.

Duda, R. and Hart, P. (1973). *Pattern Classification And Scene Analysis.* Wiley and Son.

El-Jaroudi, A. and Makhoul, J. (1990). A new error criterion for posterior probability estimation with neural nets. In *IJCNN-90*, (San Diego, 1989).

Fahlman, S. and Lebiere, C. (1990). The cascade-correlation learning architecture. In Touretzky, editor, *Advances in Neural Information Processing Systems*, volume 2. Morgan Kaufman.

Federov, V. (1972). *Theory of optimal experiments.* Academic Press.

Freund, Y. (1992). An improved boosting algorithm and its implications on learning complexity. In *Proceedings of the 5th Workshop on Computational Learning theory*, pages 391–398. Morgan Kaufman.

Freund, Y. and Schapire, R. (1995). A decision-theoretic generalization of on-line learning and an application to boosting. In *EuroColt-95.*

Freund, Y. and Schapire, R. (1996). Experiments with a New Boosting Algorithm. In *Proceedings of the 13th International Conference on Machine Learning.*

Freund, Y., Seung, H., Shamir, E., and Tishby, N. (1993). Information, prediction, and query by committee. In Hanson, Cowan, and Giles, editors, *Advances in Neural Information Processing Systems*, volume 5. Morgan Kaufman.

Geman, S., Bienenstock, E., and Doursat, R. (1992). Neural networks and the bias/variance dilemma. *Neural Computation*, 4:1–48.

Ginsberg, M. (1993). *Essentials of Artificial Inteligence.* Morgan Kaufmann.

Hansen, L. K. and Salamon, P. (1990). Neural network ensembles. *IEEE Trans. Pattern Anal. Machine Intell.*, 12(10):993–10001.

Hastie, T. and Tibshirani, R. (1990). *Generalized Additive Models.* Chapman and Hall.

Hertz, J., Krogh, A., and Palmer, R. (1991). *Introduction to the theory of neural computation.* Addison Wesley.

Ho, T. K., Hull, J. J., and Srihari, S. N. (1990). Combination of structural classifiers. In *IAPR Workshop on Syntactic and Structural Pattern recognition*, pages 123–136.

Jacobs, R., Jordan, M., Nowlan, S., and Hinton, G. (1991). Adaptive mixtures of local experts. *Neural Computation*, 1.

Jordan, M. and Jacobs, R. (1994). Hierarchical mixtures of experts and the EM algorithm. *Neural Computation*, 6:181–214.

Kearns, M. and Valiant, L. (1989). Cryptographic limitations on learning Boolean formuale and finite automata. In *Proceedings of the 21st Annual ACM Symposium on the Theory of Computing.* ACM Press.

Kinzel, W. and Rujan, P. (1990). Improving a network generalization ability by selecting samples. *Europhysics Letters*, 13(5):473–477.

Kohavi, R. (1995). Wrappers for performance enhancement and oblivious decision graphs. PhD Thesis, Stanford University.

Krogh, A. and Vedelsby, J. (1995). Neural Network ensembles, Cross Validation and active learning. In Tesaro, G., Touretzky, D., and Alspector, J., editors, *Advances in Neural Information Processing Systems*, volume 7. MIT Press.

Kwok, S. and Carter, C. (1990). Multiple decision trees. In Schacter, R., Levitt, T., Kanal, L., and Lemmer, J., editors, *Uncertainty in Artificial Intelligence*, pages 327–335. Elsevier Science Publishers.

Langley, P. (1995). *Elements of machine learning*. Morgan Kaufmann.

Le Cun, Y., Matan, O., Boser, B., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W., Jackel, L. D., and Baird, H. S. (1990). Handwritten Zip Code Recognition with Multilayer Networks. In *Proceedings of the 10th International conference on Pattern Recognition*. IEEE Computer Society Press.

Lewis, D. and Catlett, J. (1994). Heterogeneous uncertainty sampling for supervised learning. In *Proceedings of the 7th Workshop on Computational Learning theory*, pages 148–156.

Littlestone, N. (1988). Learning quickly when irrelevant attributes abound: A new linear threshold algorithm. *Machine Learning*, 2:285–318.

MacKay, D. J. C. (1992). Information-based objective functions for active data selection. *Neural Computation*, 4:590–604.

Matan, O. (1995). On-Site Learning. Technical Report LOGIC-95-4, Stanford University.

Matan, O., Kiang, R. K., Stenard, C. E., Boser, B., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W., Jackel, L. D., and Le Cun, Y. (1990). Handwritten character recognition using neural network architectures. In *Proceedings of the 4th United States Postal Service Advanced Technology Conference*, volume 2, Washington D.C.

Matan, O. and Solla, S. (1995). Some notes on Schapire's boosting algorithm. Technical Report LOGIC-95-5, Stanford University.

Mazurov, V., Krivonogov, A., and Kazantsev, V. (1987). Solving of optimization and identification problems by the committee methods. *Pattern Recognition*, 20:371–378.

Merz, C. and Murphy, P. (1996). UCI Repository of machine learning databases. `http://www.ics.uci.edu/ mlearn/MLRepository.html`, University of California, Irvine.

Mitchell, T. (1977). Version spaces: A candidate elimination approach to rule learning. In *IJCAI 95*, pages 305–310.

Mitchell, T. (1982). Generalization as search. *Artificial Intelligence*, pages 203–226.

Nilsson, N. J. (1965). *Learning Machines*. McGraw-Hill.

Nilsson, N. J. (1980). *Principles of Artificial Intelligence*. Tioga, Palo Alto.

Opper, M. and Haussler, D. (1991). Generalization performance of bayes optimal classification Algorithm for learning a Perceptron. *Physical Review Letters*, 66(20):2677–2680.

Perrone, M. P. and Cooper, L. N. (1993). When networks disagree: ensemble methods for hybrid neural networks. In Mammone, R. J., editor, *Neural Networks for speech and Image Processing*. Chapman-Hall.

Plutowski, M. and White, H. (1991). Active selection of training examples for network learning in noiseless environments. Technical report cs91-180, University of California, San Diego.

Quinlan, J. (1993). *C4.5: Programs for machine learning*. Morgan Kaufmann.

Quinlan, J. (1996). Bagging, Boosting and C4.5. In *Proceedings of AAAI-96*.

Rich, E. and Knight, K. (1991). *Artificial Intelligence*. McGraw-Hill.

Rivest, R. L. (1987). Learning Decision Lists. *Machine Learning*, 2:229–246.

Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986). Learning internal representations by error propagation. In *Parallel distributed processing: Explorations in* *the microstructure of cognition*, volume I, pages 318–362. Bradford Books, Cambridge, MA.

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

Seung, H. S., Opper, M., and Sompolinsky, H. (1992). Query by committee. In *Proceedings of the annual Workshop on Computational Learning Theory*, pages 287–294.

Sollich, P. (1994). Query construction, entropy, and generalization in neural network models. *Phys. Rev. E*, 49:4637–4651.

Tirthankar, R. and Hamey, L. (1995). An algorithm for active data collection for learning. Technical report 95-173c, Macquarie University.

Valiant, L. G. (1984). A theory of the learnable. *Communications of the ACM*, 27:11:1134–1142.

Vapnik, V. (1982). *Estimation of dependencies based on empirical data.* Springer-Verlag.

Vapnik, V. and A., C. (1971). On the uniform convergence of relative frequencies of events to their probabilities. *Theory of Probability and its Applications*, 16(2):264–280.

Vapnik, V., Levin, E., and Le Cun, Y. (1994). Measuring the VC-dimension of a learning machine. *Neural Computation*, 6:851–876.

Weiss, S. and Kulikowski, C. (1991). *Computer systems that learn.* Morgan Kaufmann.

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