Infinite Ensemble Learning with Support Vector Machines

Thesis by

Hsuan-Tien Lin

In Partial Fulfillment of the Requirements
for the Degree of
Master of Science

California Institute of Technology
Pasadena, California

2005
(Submitted May 18, 2005)

Acknowledgements

I am very grateful to my advisor, Professor Yaser S. Abu-Mostafa, for his help, support, and guidance throughout my research and studies. It has been a privilege to work in the free and easy atmosphere that he provides for the Learning Systems Group.

I have enjoyed numerous discussions with Ling Li and Amrit Pratap, my fellow members of the group. I thank them for their valuable input and feedback. I am particularly indebted to Ling, who not only collaborates with me in some parts of this work, but also gives me lots of valuable suggestions in research and in life. I also appreciate Lucinda Acosta for her help in obtaining necessary resources for research.

I thank Kai-Min Chung for providing useful comments on an early publication of this work. I also want to address a special gratitude to Professor Chih-Jen Lin, who brought me into the fascinating area of Support Vector Machine five years ago.

Most importantly, I thank my family and friends for their endless love and support. I am thankful to my parents, who have always encouraged me and have believed in me. In addition, I want to convey my personal thanks to Yung-Han Yang, my girlfriend, who has shared every moment with me during days of our lives in the U.S.

This work is supported primarily by the Engineering Research Centers Program of the National Science Foundation under award EEC-9402726.

Abstract

Ensemble learning algorithms achieve better out-of-sample performance by averaging over the predictions of some base learners. Theoretically, the ensemble could include an infinite number of base learners. However, the construction of an infinite ensemble is a challenging task. Thus, traditional ensemble learning algorithms usually have to rely on a finite approximation or a sparse representation of the infinite ensemble. It is not clear whether the performance could be further improved by a learning algorithm that actually outputs an infinite ensemble.

In this thesis, we exploit the Support Vector Machine (SVM) to develop such learning algorithm. SVM is capable of handling infinite number of features through the use of kernels, and has a close connection to ensemble learning. These properties allow us to formulate an infinite ensemble learning framework by embedding the base learners into an SVM kernel. With the framework, we could derive several new kernels for SVM, and give novel interpretations to some existing kernels from an ensemble point-of-view.

We would construct several useful and concrete instances of the framework. Experimental results show that SVM could perform well even with kernels that embed very simple base learners. In addition, the framework provides a platform to fairly compare SVM with traditional ensemble learning algorithms. It is observed that SVM could usually achieve better performance than traditional ensemble learning algorithms, which provides us further understanding of both SVM and ensemble learning.

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

This thesis is about infinite ensemble learning, a promising paradigm in machine learning. In this chapter, we would introduce the learning problem, the ensemble learning paradigm, and the infinite ensemble learning paradigm. We would build up our scheme of notations, and show our motivations for this work.

1.1 The Learning Problem

1.1.1 Formulation

In this thesis, we would study the problem of learning from examples (Abu-Mostafa 1989). For such a *learning problem*, we are given a *training set* $Z = \{z_i : z_i = (x_i, y_i)\}_{i=1}^N$ which consists of the *training examples* z_i. We assume that the *training vectors* x_i are drawn independently from an unknown probability measure $P_{\mathcal{X}}(x)$ on $\mathcal{X} \subseteq \mathbb{R}^D$, and their *labels* y_i are computed from $y_i = f(x_i)$. Here $f : \mathcal{X} \to \mathcal{Y}$ is called the *target function*, and is also assumed to be unknown. With the given training set, we want to obtain a function $g^* : \mathcal{X} \to \mathcal{Y}$ as our inference of the target function. The function g^* is usually chosen from a collection $\mathcal{G}$ of candidate functions, called the *learning model*. Briefly speaking, the task of the learning problem is to use the information in the training set Z to find some $g^* \in \mathcal{G}$ that approximates f well.

For example, we may want to build a recognition system that transforms an image of a written digit to its intended meaning. This goal is usually called an inverse problem, but we can also formulate it as a learning problem. We first ask someone to write down N digits, and represent their images by the training vectors x_i. We then label the digits by $y_i \in \{0, 1, \cdots, 9\}$ according to their meanings. The target function f here encodes the process of our human-based recognition system. The task of this learning problem would be setting up an automatic recognition system (function) g^* that is almost as good as our own recognition system, even on the yet unseen images of written digits in the future.

Throughout this thesis, we would work on *binary classification problems*, in which $\mathcal{Y} = \{-1, +1\}$. We call a function of the form $\mathcal{X} \to \{-1, +1\}$ a *classifier*, and define the deviation between two classifiers g and f on a point $x \in \mathcal{X}$ to be $I[g(x) \neq f(x)]$.[1] The overall deviation, called the *out-of-sample error*, would be

$$\pi(g) = \int_{\mathcal{X}} I[f(x) \neq g(x)] \; dP_{\mathcal{X}}(x).$$

We say that g approximates f well, or *generalizes* well, if $\pi(g)$ is small.

We design the *learning algorithm* $\mathcal{A}$ to solve the learning problem. Generally, the algorithm takes the training set Z and the learning model $\mathcal{G}$, and outputs a *decision function* $g^* \in \mathcal{G}$ by minimizing a predefined error cost $e_Z(g)$. The full scenario of learning is illustrated in Figure 1.1.

To obtain a decision function that generalizes well, we ideally desires $e_Z(g) = \pi(g)$ for all $g \in \mathcal{G}$. However, both f and $p_{\mathcal{X}}$ are assumed to be unknown, and it is hence impossible to compute and minimize $\pi(g)$ directly. A substitute quantity that depends only on Z is called the *in-sample error*

$$\nu(g) = \sum_{i=1}^{N} I[y_i \neq g(x_i)] \cdot \frac{1}{N}\,.$$

Many learning algorithms consider $\nu(g)$ to be an important part of the error cost, because $\nu(g)$ is an unbiased estimate of $\pi(g)$. The general wish is that a classifier g with a small $\nu(g)$ would also have a small $\pi(g)$. Unfortunately, the wish does not

[1] $I[\cdot] = 1$ when the inner condition is true, and 0 otherwise.

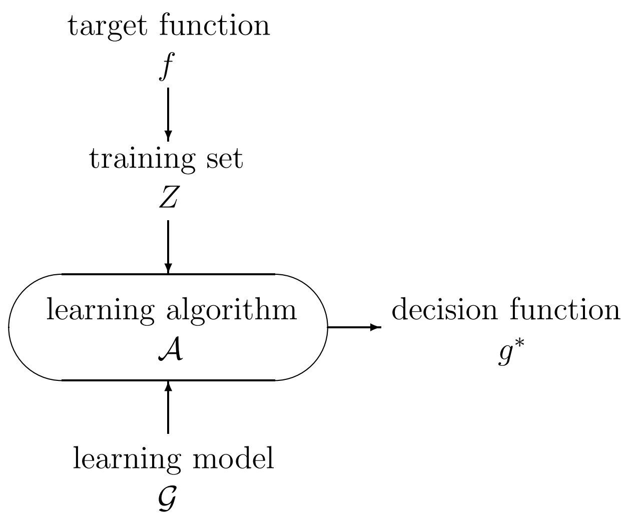


Figure 1.1: Illustration of the learning scenario

always come true. Although the in-sample error $\nu(g)$ is related to the out-of-sample error $\pi(g)$ (Abu-Mostafa et al. 2004), a small $\nu(g)$ does not guarantee a small $\pi(g)$ if the classifiers $g \in \mathcal{G}$ are considered altogether (Vapnik and Chervonenkis 1971). Next, we would show that the difference between $\pi(g)$ and $\nu(g)$ could indicate how well the decision function g^* generalizes.

1.1.2 Capacity of the Learning Model

It is known that the *capacity* of a learning model $\mathcal{G}$ plays an important role in the learning scenario (Cover 1965; Baum and Haussler 1989; Abu-Mostafa 1989; Vapnik 1998). The capacity of $\mathcal{G}$ denotes how the classifiers in $\mathcal{G}$ could classify different training sets (Cover 1965; Blumer et al. 1989). We say that $\mathcal{G}$ is more powerful, or more complex, if it has a larger capacity.

There are many approaches for measuring the capacity of a learning model (Cover 1965; Zhang 2002; Bousquet 2003; Vapnik 1998). One of the most important approaches is the *Vapnik-Chervonenkis dimension*:

Definition 1 *(Baum and Haussler 1989; Blumer et al. 1989) Consider the set of vectors* $X = \{x_i\}_{i=1}^N \in \mathcal{X}^N$. *We say that* X *is shattered by* $\mathcal{G}$ *if for all* $(y_1, y_2, \cdots, y_N) \in$

$\{-1,+1\}^N$, there exists $g \in \mathcal{G}$ such that $y_i = g(x_i)$ for $i = 1, 2, \cdots, N$.

The Vapnik-Chervonenkis dimension (V-C dimension) of a learning model $\mathcal{G}$, denoted $D_{\mathrm{VC}}(\mathcal{G})$, is the maximum number N for which there exists $X = \{x_i\}_{i=1}^N$ that can be shattered by $\mathcal{G}$. If there exists such X for all $N \in \mathbb{N}$, then $D_{\mathrm{VC}}(\mathcal{G}) = \infty$.

When $\mathcal{G}$ shatters a set of N training vectors $\{x_i\}_{i=1}^N$, we could find a classifier $g \in \mathcal{G}$ that achieves $\nu(g) = 0$ for any of the 2^N possible labelings. In other words, $\mathcal{G}$ is so powerful that no matter how those training labels are generated, we can always obtain zero in-sample error on this training set. The V-C dimension captures this classification power, or the capacity, by a single integer. Nevertheless, the integer is very informative in the learning theory (Baum and Haussler 1989; Abu-Mostafa 1989; Vapnik 1998). In particular, the difference between the out-of-sample and the in-sample error can typically be bounded by the V-C dimension.

Theorem 1 *(Vapnik and Chervonenkis 1971; Abu-Mostafa 1989; Vapnik 1998) For the binary classification problem, if the learning model $\mathcal{G}$ has $D_{\mathrm{VC}}(\mathcal{G}) < \infty$, then for any $\epsilon > 0$, there exists some $N_0 \in \mathbb{N}$ such that for any training set of size $N \geq N_0$,*

$$\Pr\left[\sup_{g \in \mathcal{G}} \left|\pi(g) - \nu(g)\right| > \epsilon\right] \leq 4\left((2N)^{D_{\mathrm{VC}}(\mathcal{G})} + 1\right) \exp\left(-\frac{N\epsilon^2}{8}\right) \tag{1.1}$$

Theorem 1 is a route to estimate the out-of-sample error from the in-sample error. The inequality (1.1) is independent of $p_{\mathcal{X}}$ and f. In addition, it is a worst-case bound for all $g \in \mathcal{G}$, and thus can also be applied to the decision function g^*. Next, we use this theorem to explain how the choice of a suitable $\mathcal{G}$ affects the generalization performance of g^*.

When we fix the training set and the learning algorithm, there are two extreme approaches for choosing $\mathcal{G}$. On the one hand, because the target function f is unknown, we may want to include many diverse classifiers in $\mathcal{G}$ to prepare for any possible f. In this situation, the V-C dimension of $\mathcal{G}$ would be large, because such a learning model is likely to shatter a larger set of training vectors. One of the main drawbacks here, called *overfitting*, is that the learning algorithm could possibly fit the training set Z without learning much about the underlying target function f. More specifically,

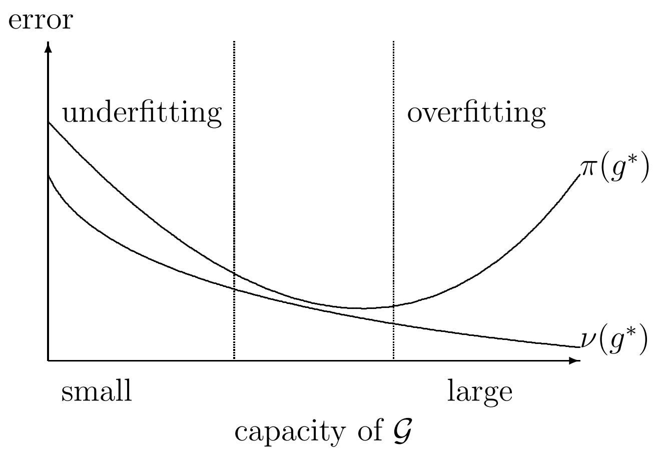


Figure 1.2: Overfitting and underfitting

even if we obtain a decision function g^* with a small $\nu(g^*)$, the function may still have a large $\pi(g^*)$.

On the other hand, we can use a learning model $\mathcal{G}$ with a very small capacity to avoid overfitting. Then, the bound in (1.1) would be smaller. However, if all $g \in \mathcal{G}$ are too simple, they could be very different from the target function f. In other words, both $\nu(g)$ and $\pi(g)$ would be large for all $g \in \mathcal{G}$. Hence, the learning algorithm could not output any g^* that has a small $\pi(g^*)$. This situation is called *underfitting*.

We illustrate the typical behavior of overfitting and underfitting in Figure 1.2. Successful learning algorithms usually handle these situations by working implicitly or explicitly with a reasonable learning model. One famous strategy, called *regularization*, implicitly shrinks the capacity of $\mathcal{G}$ by only considering some simpler subsets of $\mathcal{G}$, or by penalizing the more complex subsets of $\mathcal{G}$. Regularization helps to avoid overfitting, and is inherit in many learning algorithms, such as the Support Vector Machine that we would encounter later.

Another famous strategy, called *boosting*, is widely used in the ensemble learning paradigm. Boosting usually starts from a simple learning model, and combines multiple classifiers within the simple model to form a more complex classifier. The combination step boosts up the capacity of the actual learning model, and thus gives a remedy to underfitting. We would further explore the use of regularization and boosting in this thesis.

1.2 Ensemble Learning

1.2.1 Formulation

The *ensemble learning* paradigm denotes a large class of learning algorithms (Meir and Rätsch 2003). Instead of considering a powerful learning model $\mathcal{G}$, an ensemble learning algorithm $\mathcal{A}$ deals with a *base learning model* $\mathcal{H}$, which is usually simple. The classifiers $h \in \mathcal{H}$ are often called *hypotheses* or *base learners*. The algorithm then constructs a decision function g^* by [2]

$$\begin{gathered} g^*(x) = \operatorname{sign}\left(\sum_{t=1}^{T} w_t h_t(x)\right), \\ w_t \geq 0, t = 1, 2, \cdots, T. \end{gathered} \tag{1.2}$$

Any classifier g^* that could be expressed by (1.2) is called an *ensemble classifier*, and the w_t in (1.2) are called the *hypothesis weights*. Without lose of generality for possible ensemble classifiers, we usually normalize w by $\sum_{t=1}^{T} w_t$. Then the hypothesis weights would sum to 1.[3] For each ensemble classifier in (1.2), we can define its normalized version as

$$\begin{gathered} g^*(x) = \operatorname{sign}\left(\sum_{t=1}^{T} \tilde{w}_t h_t(x)\right), \\ \sum_{t=1}^{T} \tilde{w}_t = 1, \tilde{w}_t \geq 0, t = 1, 2, \cdots, T. \end{gathered} \tag{1.3}$$

We would denote the *normalized hypothesis weights* by $\tilde{w}$, while reserving w for possibly unnormalized ones. Note that $\tilde{w}$ and w can usually be used interchangeably, because scaling the hypothesis weights by a positive constant does not affect the

[2]sign(v) is 1 when v is nonnegative, -1 otherwise.

[3]We usually ignore the degenerate case where we obtain all zero hypothesis weights.

prediction after the sign$(\cdot)$ operation. By considering normalized classifiers, we can see that that $g^* \in \text{cvx}(\mathcal{H})$, where $\text{cvx}(\mathcal{H})$ means the convex hull of $\mathcal{H}$ in the function space. In other words, the ensemble learning algorithm $\mathcal{A}$ actually works on $\mathcal{G} = \text{cvx}(\mathcal{H})$.

Although $\mathcal{H}$ could be of infinite size in theory, traditional ensemble learning algorithms usually deal with a finite and predefined T in (1.2). We call these algorithms *finite ensemble learning*. Finite ensemble learning algorithms usually share another common feature: they choose each hypothesis h_t by calling another learning algorithm $A_{\mathcal{H}}$, called the *base learning algorithm*.

Another approach in ensemble learning is *infinite ensemble learning*, in which the size of $\{h_t\}$ is not finite. For infinite ensemble learning, the set $\{h_t\}$ could either be countable or uncountable. In the latter situation, a suitable integration would be used instead of the summation in (1.2).

Successful ensemble learning algorithms include Bayesian Committee Machines (Tresp 2000), Bootstrap Aggregating (Breiman 1996), Adaptive Boosting (Freund and Schapire 1997), and Adaptive Resampling and Combining (Breiman 1998). In a broad sense, Bayesian inference that averages the predictions over the posterior probability also belongs to the ensemble learning family (Vapnik 1998).

1.2.2 Why Ensemble Learning?

Ensemble learning algorithms are often favorable for having some or all of the following three properties: stability, accuracy, and efficiency (Meir and Rätsch 2003).

1. Stability:

   If a learning algorithm outputs a very different decision function g^* when there is a small variation in Z, we call the algorithm *unstable*. Unstable learning algorithms are often not desirable, because they are easily affected by noise, imprecise measurements, or even numerical errors in computing. Such algorithms also may not output some g^* that generalize well, because of the large variance of the possible outcomes (Bousquet and Elisseeff 2002).

The stability of ensemble learning algorithms is best illustrated by the *Bootstrap Aggregating* (Bagging) algorithm (Breiman 1996). Bagging generates T training sets $Z^{(t)}$ by bootstrapping Z, and applies the base learning algorithm $\mathcal{A}_{\mathcal{H}}$ on each $Z^{(t)}$ to obtain h_t. The majority vote of each $h_t(x)$ determines the prediction for some $x \in \mathcal{X}$. In other words, the normalized hypothesis weights $\tilde{w}_t$ are always set to $\frac{1}{T}$ in (1.3). Breiman (1996) shows that the Bagging algorithm $\mathcal{A}$ is stabler than the base learning algorithm $\mathcal{A}_{\mathcal{H}}$ because of the voting strategy. Thus, we can view the ensemble learning algorithm as an approach to make the base learning algorithm stabler.

2. Accuracy:

   The ensemble learning algorithm usually outputs a decision function g^* that has a smaller $\pi(g^*)$ than each individual $\pi(h_t)$. One simple explanation is that a voting approach like Bagging could eliminate uncorrelated errors made by each classifier h_t. A deeper justification comes from the Probably Approximately Correct (PAC) theory of learnability (Valiant 1984). In particular, when the size of the training set is large enough, even if each classifier h_t performs only slightly better than random guess to f, we would construct an ensemble classifier g^* that is very close to f (Kearns and Valiant 1994). The boosting strategy illustrated in Section 1.1 gets its name because of this theoretical result.

3. Efficiency:

   This property usually holds for finite ensemble learning. Although the basic learning model $\mathcal{H}$ is usually simple, the actual learning model $\mathcal{G} = \text{cvx}(\mathcal{H})$ could be large and complex. Thus, a learning algorithm that directly works on the learning model $\mathcal{G}$ may take a long running time. The ensemble learning algorithms, on the other hand, usually exploit the structure of $\mathcal{G}$ to divide the learning task into several small subproblems such that each of them could be efficiently solved by the base learning algorithm $\mathcal{A}_{\mathcal{H}}$. Because of the simplicity of $\mathcal{H}$, such divide-and-conquer approach could gain efficiency. For example, the Adaptive Boosting algorithm (see Section 2.2) could perform a complicated search in $\mathcal{G}$ with only some small number of calls to $\mathcal{A}_{\mathcal{H}}$.

1.3 Infinite Ensemble Learning

We have introduced the basics of ensemble learning. In this section, we would discuss the motivations and possible difficulties for infinite ensemble learning.

1.3.1 Why Infinite Ensemble Learning?

The most important reason for going from finite ensemble learning to infinite ensemble learning is to further increase the capacity of the learning model. Baum and Haussler (1989) show that ensemble classifiers in (1.2) with a finite predefined T is limited in power.

Theorem 2 *(Baum and Haussler 1989) For a base learning model $\mathcal{H}$ and a finite predefined $T \geq 2$, let*

$$\mathcal{G} = \{g \colon g \text{ can be represented by } (1.2) \text{ with } h_t \in \mathcal{H} \text{ for all } t = 1, 2, \cdots, T\}.$$

Then, $D_{\text{VC}}(\mathcal{G}) \leq 4T \log_2(eT)$.

Thus, choosing a suitable T for finite ensemble learning is as important as choosing a suitable $\mathcal{G}$ for a learning algorithm. On the one hand, the limit in capacity could make the algorithm less vulnerable to overfitting (Rosset et al. 2004). On the other hand, the limit raises a possibility of underfitting (Freund and Schapire 1997).

Although traditional ensemble learning algorithms usually work with a finite predefined T, many of their theoretical justifications are based on the asymptotic behavior when $T \to \infty$. In other words, these algorithms can be viewed as approximations to infinite ensemble learning. Some results show that it is beneficial to apply infinite ensemble learning paradigm (Demiriz et al. 2002), while others show that it is harmful to go closer to infinity by enlarging T (Rätsch et al. 2001). The controversial results suggest further research on infinite ensemble learning.

There are many successful learning algorithms that work well by combining infinite processes, transition probabilities, or features. For example, infinite Gaussian Mixture Model (Rasmussen 2000), infinite Hidden Markov Model (Beal et al. 2003), or Support Vector Machine (Vapnik 1998). They successfully demonstrate that it can be beneficial to consider infinite mixtures in the learning model. Thus, we want to study whether infinite ensemble learning, that is, an infinite mixture of hypotheses, could also work well. In particular, our motivation comes from an open problem of Vapnik (1998, page 704):

The challenge is to develop methods that will allow us to average over large (even infinite) numbers of decision rules using margin control. In other words, the problem is to develop efficient methods for constructing averaging hyperplanes in high dimensional space.

Here the "decision rules" are the hypotheses in $\mathcal{H}$, the "margin control" is for performing regularization, and the "averaging hyperplane" is the ensemble classifier. Briefly speaking, our task is to construct an ensemble classifier with an infinite number of hypotheses, while implicitly controlling the capacity of the learning model. Next, we will see the difficulties that arise with this task.

1.3.2 Dealing with Infinity

To perform infinite ensemble learning, we would want to check and output classifiers of the form

$$g(x) = \operatorname{sign}\left(\sum_{t=1}^{\infty} w_t h_t(x)\right),$$
$$w_t \geq 0, t = 1, \cdots, \infty,$$

or in the uncountable case,

$$\begin{aligned} g(x) &= \operatorname{sign}\left(\int_{\alpha \in \mathcal{C}} w_\alpha h_\alpha(x)\, d\alpha\right), \\ w_\alpha &\geq 0, \alpha \in \mathcal{C}. \end{aligned}$$

The countable and uncountable cases share similar difficulties. We would conquer both cases in this thesis. Here, we will only discuss the countable case for simplicity.

The first obstacle is to represent the classifiers. The representation is important both for the learning algorithm, and for doing further prediction with the decision function g^*. We cannot afford to save and process every pair of (w_t, h_t) because of the infinity. One approach is to save only the pairs with nonzero hypothesis weights, because the zero weights do not affect the predictions of any ensemble classifier g. We call this approach *sparse representation*.

An ensemble classifier that only has a small finite number of nonzero hypothesis weights is called a sparse ensemble classifier. The viability of sparse representation is based on the assumption that the learning algorithm only needs to handle sparse ensemble classifiers. Some algorithms could achieve this assumption by applying an error cost $e_Z(g)$ that favors sparse ensemble classifiers. An example of such design is the Linear Programming Boosting algorithm (Demiriz et al. 2002), which would be introduced in Section 2.2. The assumption on sparse ensemble classifiers is also crucial to many finite ensemble learning algorithms, because such property allows them to approximate the solutions well with a finite ensemble.

With the sparse representation approach, it looks as if infinite ensemble learning could be solved or approximated by finite ensemble learning. However, the sparsity assumption also means that the capacity of the classifiers is effectively limited by Theorem 2. In addition, it is not clear whether the error cost $e_Z(g)$ that introduces sparsity could output a decision function g^* that generalizes well. Thus, we choose to take a different route. We want to conquer the task of infinite ensemble learning without sparse representation, and see if our approach could perform better.

Another obstacle in infinite ensemble learning is the infinite number of constraints

$$w_t \geq 0, t = 1, 2, \cdots, \infty.$$

Learning algorithms usually try to minimize the error cost $e_Z(g)$, which becomes a harder task when more constraints needs to be satisfied and maintained simultaneously. The infinite number of constraints resides in the extreme of this difficulty, and is part of the challenge mentioned by Vapnik (1998).

1.4 Overview

This thesis exploits the Support Vector Machine (SVM) to tackle the difficulties in infinite ensemble learning. We would show the similarity between SVM and boosting-type ensemble learning algorithms, and formulate an infinite ensemble learning framework. based on SVM. The key of the framework is to embed infinite number of hypotheses into the kernel of SVM. Such framework does not require the assumption for sparse representation, and inherits the profound theoretical results of SVM.

We find that we can apply this framework to construct new kernels for SVM, and to interpret some existing kernels. In addition, we can use this framework to fairly compare SVM with other ensemble learning algorithms. Experimental results show that our SVM-based infinite ensemble learning algorithm has superior performance over popular ensemble learning algorithms.

This thesis is organized as follows. In Chapter 2, we discuss the connection between SVM and ensemble learning algorithms. Next in Chapter 3, we show our framework for embedding hypotheses into the SVM kernel, and explain how this framework converts SVM into an infinite ensemble learning algorithm. We then demonstrate the framework with some concrete instances in Chapter 4. Finally, we show the experimental results in Chapter 5, and conclude in Chapter 6.

Chapter 2

Connection between SVM and Ensemble Learning

In this chapter, we focus on connecting Support Vector Machine (SVM) to ensemble learning algorithms. The connection is our first step towards designing an infinite ensemble learning algorithm with SVM. We start by providing the formulations of SVM in Section 2.1, and show that SVM implements the concept of large-margin classifiers. Next in Section 2.2, we introduce some ensemble learning algorithms that also output large-margin classifiers. Then, we would further discuss the connection between SVM and those algorithms in Section 2.3.

2.1 Support Vector Machine

The *Support Vector Machine* (SVM) is a learning algorithm based on V-C type learning theory (see Theorem 1). We shall first introduce the basic idea of SVM: producing a hyperplane classifier with the largest minimum margin. Then, we would extend the basic idea to a more powerful and more robust SVM formulation.

2.1.1 Basic SVM Formulation

We start from a basic SVM formulation: *linear hard-margin* SVM (Vapnik 1998), which constructs a decision function [1]

$$g^*(x) = \mathrm{sign}\big(w^T x + b\big)$$

with the optimal solution (w, b) to the following problem:

$$\begin{aligned}
(P_1) \qquad \max_{w\in\mathbb{R}^D, b\in\mathbb{R}, \rho\in\mathbb{R}^N} \quad & \min_{1\le i\le N} \rho_i \\
\text{subject to} \quad & \rho_i = \frac{1}{\|w\|_2} y_i \left(w^T x_i + b\right), \; i = 1, 2, \cdots, N, \\
& \rho_i \ge 0, i = 1, 2, \cdots, N.
\end{aligned}$$

The classifier of the form $\mathrm{sign}\big(w^T x + b\big)$ is called a *hyperplane classifier*, which divides the space $\mathbb{R}^D$ with the hyperplane $w^T x + b = 0$. For a given hyperplane classifier, the value ρ_i is called the ℓ_2-*margin* of each training example $z_i = (x_i, y_i)$, and its magnitude represents the Euclidean distance of x_i to the hyperplane $w^T x + b = 0$. We illustrate the concept of margin in Figure 2.1. The constraint $\rho_i \ge 0$ means that the associated training example z_i is classified correctly using the classifier. Linear hard-margin SVM would output a hyperplane classifier that not only classifies all training examples correctly (called *separates* all training examples), but also has the largest minimum margin. In other words, the distance from any training vector to the hyperplane should be as large as possible.

The intuition for obtaining a large-margin classifier is to have less uncertainty near the decision boundary $w^T x + b = 0$, where $\mathrm{sign}(\cdot)$ switches sharply. A deeper theoretical justification is that the large-margin concept implicitly limits the capacity of the admissible learning model.

Theorem 3 *(Vapnik 1998) Let the vectors $x \in \mathcal{X} \subseteq \mathbb{R}^D$ belong to a sphere of ra-*

[1]Here u^T means the transpose of the vector u, and hence $u^T v$ is the inner product of the two vectors in $\mathbb{R}^D$.

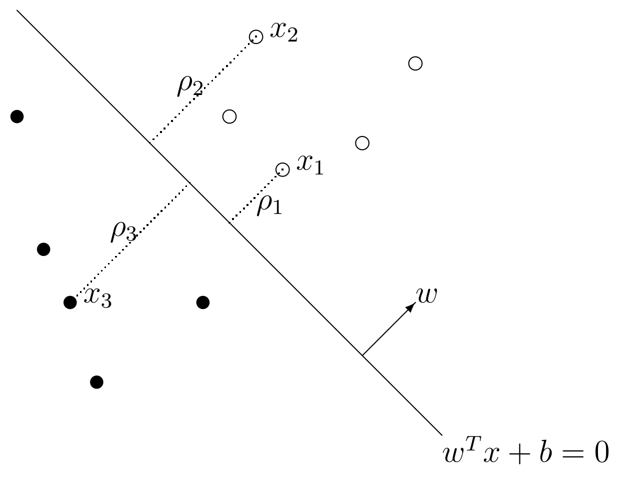


Figure 2.1: Illustration of the margin, where $y_i = 2 \cdot I[\text{circle is empty}] - 1$

dius R, and define the Γ-margin hyperplane

$$g_{w,b}(x) = \begin{cases} 1, & \text{if } \frac{1}{\|w\|_2} \cdot \left(w^T x + b\right) \geq \Gamma \\ -1, & \text{if } \frac{1}{\|w\|_2} \cdot \left(w^T x + b\right) \leq -\Gamma \end{cases}$$

Then, the learning model $\mathcal{G}$ that consists of all the Γ-margin hyperplanes has V-C dimension

$$D_{\text{VC}}(\mathcal{G}) \leq \min\left(\frac{R^2}{\Gamma^2}, D\right) + 1.$$

Thus, when choosing the hyperplane with the largest minimum margin, we could shrink and bound the capacity of the learning model. As mentioned in Section 1.1, this idea corresponds to the regularization strategy for designing learning algorithms.

Problem (P_1) contains the max-min operation and nonlinear constraints, which are complicated to solve. Nevertheless, the optimal solution (w, b) for a following simple quadratic problem is also optimal for (P_1).

$$\begin{array}{lll} (P_2) & \displaystyle\min_{w \in \mathbb{R}^D, b \in \mathbb{R}} & \frac{1}{2} w^T w \\ & \text{subject to} & y_i \left(w^T x_i + b\right) \geq 1. \end{array}$$

The quadratic problem (P_2) is convex and is easier to analyze. In the next section, we would construct more powerful SVM formulations based on this problem.

2.1.2 Nonlinear Soft-Margin SVM Formulation

Linear hard-margin SVM problem (P_2) has a drawback: what if the training examples cannot be perfectly separated with any hyperplane classifier? Figure 2.2(a) shows a training set of such situation. Then, the feasible region of (P_2) would be empty, and the algorithm could not output any decision function (Lin 2001).

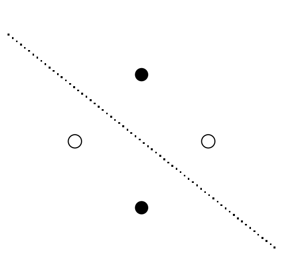
This situation happens because the learning model (set of hyperplane classifiers) is not powerful enough, or because the training examples contain noise. Nonlinear soft-margin SVM applies two techniques to deal with the situation. First, it uses the feature transformation to increase the capacity of the learning model. Second, it allows the hyperplane classifier to violate some of the the constraints of (P_2) (Schölkopf and Smola 2002).

Nonlinear SVM works in a feature space $\mathcal{F}$ rather than the original space $\mathcal{X} \subseteq \mathbb{R}^D$. The original vectors $x \in \mathcal{X}$ are transformed to the feature vectors $\phi_x \in \mathcal{F}$ by a *feature mapping*

$$\begin{aligned} \Phi\colon \quad & \mathcal{X} \to \mathcal{F} \\ & x \mapsto \phi_x \; . \end{aligned}$$

We assume that the feature space $\mathcal{F}$ is a Hilbert space defined with an inner product $\langle \cdot, \cdot \rangle$. Then, we can replace the $w^T x$ in linear SVM by $\langle w, \phi_x \rangle$. The resulting classifier would still be of hyperplane-type in $\mathcal{F}$, and hence Theorem 3 could be applied by considering ϕ_x instead of x. When Φ is a complicated feature transform, it is likely that we could separate the training examples in $\mathcal{F}$. For example, assume that

$$\begin{aligned} &\Phi\colon \mathbb{R}^2 \to \mathcal{F} = \mathbb{R}^5 \\ &\quad \Big((x)_1, (x)_2\Big) \mapsto \Big((x)_1, (x)_2, (x)_1^2, (x)_1(x)_2, (x)_2^2\Big), \end{aligned} \tag{2.1}$$


(a) No hyperplane classifier could separate the training examples

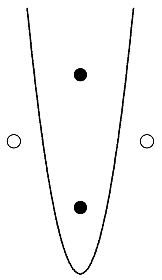

(b) A quadratic curve classifier separates the training examples

Figure 2.2: The power of the feature mapping in (2.1)

where $(x)_d$ is the d-th element of the vector x. Then, any classifier of the form

$$\operatorname{sign}\Big(\langle w, \Phi(x)\rangle + b\Big)$$

would be a quadratic curve in $\mathbb{R}^2$, and the set of such classifiers is more powerful than the set of hyperplane classifiers in $\mathbb{R}^2$, as illustrated by Figure 2.2.

With a suitable feature transform, *nonlinear soft-margin* SVM outputs the decision function

$$g^*(x) = \operatorname{sign}\Big(\langle w, \phi_x\rangle + b\Big) \tag{2.2}$$

with the optimal solution to

$$\begin{array}{lll} (P_3) & \displaystyle\min_{w\in\mathcal{F}, b\in\mathbb{R}, \xi\in\mathbb{R}^D} & \displaystyle\frac{1}{2}\langle w, w\rangle + C\sum_{i=1}^{N}\xi_i \\ & \text{subject to} & y_i\big(\langle w, \phi_{x_i}\rangle + b\big) \geq 1 - \xi_i, i = 1, 2, \cdots, N, \\ & & \xi_i \geq 0, i = 1, 2, \cdots, N. \end{array}$$

The value ξ_i is the violation that the hyperplane classifier makes on training examples $z_i = (x_i, y_i)$, and $C > 0$ is the parameter that controls the amount of the total violations allowed. When $C \to \infty$, we call (P_3) the problem of *nonlinear hard-margin* SVM, in which all ξ_i would be forced to zero.

Because the feature space $\mathcal{F}$ could be of infinite number of dimensions, SVM software usually solves the Lagrangian dual of (P_3) instead:

$$\begin{array}{lll}(P_4) & \displaystyle\min_{\lambda\in\mathbb{R}^N} & \displaystyle\frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N}\lambda_i\lambda_j y_i y_j \mathcal{K}(x_i,x_j) - \sum_{i=1}^{N}\lambda_i \\ & \text{subject to} & \displaystyle\sum_{i=1}^{N} y_i\lambda_i = 0, \\ & & 0 \le \lambda_i \le C, i = 1, \cdots, N.\end{array}$$

Here $\mathcal{K}$ is called the *kernel*, and is defined as

$$\mathcal{K}(x,x') \equiv \langle \phi_x, \phi_{x'} \rangle\,. \tag{2.3}$$

The duality between (P_3) and (P_4) holds for any Hilbert space $\mathcal{F}$ (Lin 2001). Through duality, the optimal (w,b) for (P_3) and the optimal λ for (P_4) are related by (Vapnik 1998; Schölkopf and Smola 2002)

$$w = \sum_{i=1}^{N} y_i\lambda_i\phi_{x_i} \tag{2.4}$$

and

$$\begin{array}{l} b \ge \max\left\{ -y_i\left(1 - \displaystyle\sum_{i=1}^{N}\lambda_i y_i \mathcal{K}(x_i,x_j)\right) : \begin{array}{r}\alpha_j > 0, y_j = -1 \\ \text{or} \quad \alpha_j < C, y_j = +1\end{array} \right\} \\ b \le \min\left\{ +y_i\left(1 - \displaystyle\sum_{i=1}^{N}\lambda_i y_i \mathcal{K}(x_i,x_j)\right) : \begin{array}{r}\alpha_j < C, y_j = -1 \\ \text{or} \quad \alpha_j > 0, y_j = +1\end{array} \right\}\end{array} \tag{2.5}$$

Then, the decision function becomes

$$g^*(x) = \operatorname{sign}\left(\sum_{i=1}^{N} y_i\lambda_i\mathcal{K}(x_i,x) + b\right) \tag{2.6}$$

An important observation is that both (P_4) and (2.6) do not require any computation of w explicitly. Hence, even if the feature space $\mathcal{F}$ has an infinite number of dimensions, we could solve (P_4) and obtain g^* with only the kernel $\mathcal{K}(x, x')$. The use of a kernel instead of directly computing the inner product in $\mathcal{F}$ is called the *kernel trick*, and is a key ingredient of SVM.

For the kernel trick to go through, the kernel $\mathcal{K}(x, x')$ should be easy to compute. Alternatively, we may wonder if we could start with an arbitrary kernel, and claim that it is an inner product $\langle \cdot, \cdot \rangle$ in some space $\mathcal{F}$. An important tool for this approach is the Mercer's condition. Next, we first define some important terms in Definition 2, and describe the Mercer's condition briefly in Theorem 4.

Definition 2 *For some N by N matrix K,*

1. *K is positive semi-definite (PSD) if $v^T K v \geq 0$ for all $v \in \mathbb{R}^N$.*
2. *K is positive definite (PD) if $v^T K v > 0$ for all $v \in \mathbb{R}^N$ such that some v_i is nonzero.*
3. *K is conditionally positive semi-definite (CPSD) if $v^T K v \geq 0$ for all $v \in \mathbb{R}^N$ such that $\sum_{i=1}^N v_i = 0$.*
4. *K is conditionally positive definite (CPD) if $v^T K v > 0$ for all $v \in \mathbb{R}^N$ such that $\sum_{i=1}^N v_i = 0$ and some v_i is nonzero.*

Theorem 4 *(Vapnik 1998; Schölkopf and Smola 2002) A symmetric function $\mathcal{K}(x, x')$ is a valid inner product in some space $\mathcal{F}$ if and only if for every N and $\{x_i\}_{i=1}^N \in \mathcal{X}^N$, the matrix K constructed by $K_{ij} = \mathcal{K}(x_i, x_j)$, called the Gram matrix of $\mathcal{K}$, is PSD.*

Several functions are known to satisfy Mercer's condition for $\mathcal{X} \subseteq \mathbb{R}^D$, including:

- Linear: $\mathcal{K}(x, x') = x^T x'$.
- Polynomial: $\mathcal{K}(x, x') = (x^T x' + r)^k, r \geq 0, k \in \mathbb{N}$.
- Gaussian: $\mathcal{K}(x, x') = \exp\left(-\gamma \left\|x - x'\right\|_2^2\right), \gamma > 0$.
- Exponential: $\mathcal{K}(x, x') = \exp\left(-\gamma \left\|x - x'\right\|_2\right), \gamma > 0$.

- Laplacian: $\mathcal{K}(x, x') = \exp\left(-\gamma \left\| x - x' \right\|_1\right), \gamma > 0$.

SVM with different kernels try to classify the examples with large-margin hyperplane classifiers in different Hilbert spaces. Some ensemble learning algorithms can also produce large-margin classifiers with a suitable definition of the "margin." Note that we deliberately use the same symbol w in (P_3) for the hyperplane classifiers and for the hypothesis weights in ensemble learning. In the next section, we shall see that this notation easily connects SVM to ensemble learning through the large-margin concept.

2.2 Ensemble Learning and Large-Margin Concept

We have introduced SVM and the large-margin concept. In this section we show two ensemble learning algorithms that also output large-margin classifiers. The first one is Adaptive Boosting, and the second one is Linear Programming Boosting.

2.2.1 Adaptive Boosting

Adaptive Boosting (AdaBoost) is one of the most popular algorithms for ensemble learning (Freund and Schapire 1996; Freund and Schapire 1997). For a given integer T, AdaBoost iteratively forms an ensemble classifier

$$g^*(x) = \operatorname{sign}\left(\sum_{t=1}^{T} w_t h_t(x)\right), \; w_t \geq 0, t = 1, 2, \cdots, T.$$

In each iteration t, there is a sample weight $U_t(i)$ on each training example z_i, and AdaBoost selects $h_t \in \mathcal{H}$ with the least weighted in-sample error:

$$h_t = \operatorname{argmin}_{h \in \mathcal{H}} \left(\sum_{i=1}^{N} I[y_i \neq h(x_i)] \cdot U_t(i)\right).$$

AdaBoost then assigns the unnormalized hypothesis weight w_t to h_t, and generates $U_{t+1}(\cdot)$ for the next iteration. The details of AdaBoost are shown in Algorithm 1.

Algorithm 1 has an interpretation as a gradient-based optimization technique (Mason et al. 2000). It obtains the hypotheses and weights by solving the following optimization problem:

$$
\begin{array}{llll}
(P_5) & \displaystyle\max_{w_t \in \mathbb{R}, h_t \in \mathcal{H}, \rho \in \mathbb{R}^N} & & \displaystyle -\sum_{i=1}^{N} \exp\left(- \|w\|_1 \rho_i\right) \\
& & \text{subject to} & \displaystyle \rho_i = \frac{1}{\|w\|_1} y_i \left(\sum_{t=1}^{\infty} w_t h_t(x_i) \right), \; i = 1, 2, \cdots, N, \\
& & & w_t \geq 0, \; t = 1, 2, \cdots, \infty.
\end{array}
$$

Although (P_5) has an infinite number of variables (w_t, h_t), AdaBoost approximates the optimal solution by the first T steps of the gradient-descent search. We could compare (P_5) with (P_1). First, they have a similar term ρ_i. However, for SVM, ρ_i is the ℓ_2-margin, while for AdaBoost, ρ_i is normalized by $\|w\|_1$, and is called the ℓ_1-*margin*. The objective function of SVM is

$$\min_{1 \leq i \leq N} \rho_i \, ,$$

while the objective function of AdaBoost is

$$\sum_{i=1}^{N} - \exp\left(- \|w\|_1 \rho_i\right).$$

For large ρ_i, the term $\exp\left(- \|w\|_1 \rho_i\right)$ would be close to zero and negligible. Thus, the two objective functions both focus only on small ρ_i. Note that for a fixed w, the term

$$- \exp\left(- \|w\|_1 \rho_i\right)$$

is an increasing function of ρ_i. Thus, both (P_5) and (P_1) want to maximize the smaller margins.

Under some assumptions, Rätsch et al. (2001, Lemma 5) show that when $T \to \infty$, AdaBoost indeed finds an ensemble classifier that has the largest minimum ℓ_1-margin.

In other words, AdaBoost asymptotically approximates an infinite ensemble classifier

$$g^*(x) = \text{sign}\left(\sum_{t=1}^{\infty} w_t h_t(x)\right),$$

such that (w, h) is an optimal solution for

$$\begin{aligned}
(P_6) \quad & \min_{w_t \in \mathbb{R}, h_t \in \mathcal{H}} && \|w\|_1 \\
& \text{subject to} && y_i \left(\sum_{t=1}^{\infty} w_t h_t(x_i)\right) \geq 1,\ i = 1, 2, \cdots, N. \\
& && w_t \geq 0,\ t = 1, 2, \cdots, \infty. \qquad (2.7)
\end{aligned}$$

Compare (P_6) with (P_3), we further see the similarity between SVM and AdaBoost. Note, however, that AdaBoost has additional constraints (2.7), which makes the problem harder to solve directly.

2.2.2 Linear Programming Boosting

Linear Programming Boosting (LPBoost) solves (P_6) exactly with linear programming. We will introduce *soft-margin LPBoost*, which constructs an infinite ensemble classifier

$$g^*(x) = \text{sign}\left(\sum_{t=1}^{\infty} w_t h_t(x)\right),$$

with the optimal solution to

$$\begin{aligned}
(P_7) \quad & \min_{w_t \in \mathbb{R}, h_t \in \mathcal{H}} && \sum_{t=1}^{\infty} w_t + C \sum_{i=1}^{N} \xi_i \\
& \text{subject to} && y_i \left(\sum_{t=1}^{\infty} w_t h_t(x_i)\right) \geq 1 - \xi_i, \\
& && \xi_i \geq 0,\ i = 1, 2, \cdots, N, \\
& && w_t \geq 0,\ t = 1, 2, \cdots, \infty.
\end{aligned}$$

Similar to the case of soft-margin SVM, the parameter C controls the amount of violations allowed. When $C \to \infty$, the soft-margin LPBoost approaches a *hard-margin* one, which solves (P_6) to obtain the decision function g^*.

LPBoost is an infinite ensemble learning algorithm. How does it handle infinite number of variables and constraints? First, there are many optimal solutions to (P_7), and some of them only have a finite number of nonzero hypothesis weights w_t. For example, consider two hypothesis h_{t_1} and h_{t_2} such that

$$h_{t_1}(x_i) = h_{t_2}(x_i) \text{ for } i = 1, 2, \cdots, N.$$

We say that h_{t_1} and h_{t_2} have the same *pattern*, or are *ambiguous*, on the training vectors $\{x_i\}_{i=1}^N$. Assume that (w, h) is an optimal solution for (P_7), and define

$$\hat{w}_t = \begin{cases} w_{t_1} + w_{t_2} & t = t_1 \\ 0 & t = t_2 \\ w_t & \text{otherwise.} \end{cases}$$

Then, we can see that $(\hat{w}, h)$ satisfies all the constraints of (P_7), and still has the same objective value as (w, h). Thus, $(\hat{w}, h)$ is also an optimal solution. We can repeat this process and get an optimal solution that has at most 2^N nonzero weights. LPBoost aims at finding this solution. Thus, it equivalently only needs to construct a finite ensemble classifier of at most 2^N hypothesis weights.

Even if LPBoost would need at most 2^N nonzero hypothesis weights w_t, the problem could still be intractable when N is large. However, minimizing the criteria $\|w\|_1$ often produces a sparse solution (Meir and Rätsch 2003; Rosset et al. 2004). Hence, LPBoost could start with all zero hypothesis weights, and iteratively considers one hypothesis h_t that should have non-zero weight w_t. Because (P_7) is a linear programming problem, such step can be performed efficiently with a simplex-type solver using the column generation technique (Nash and Sofer 1996). The detail of LPBoost is shown in Algorithm 2.

There are two drawbacks for LPBoost. First, solving the inner problem (P_9)

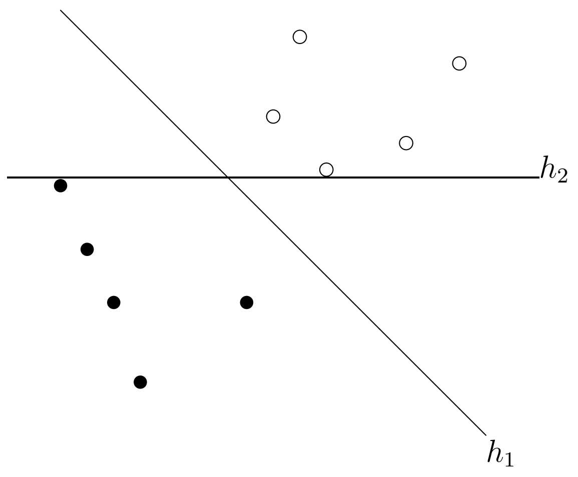


Figure 2.3: LPBoost can only choose one between h_1 and h_2

could be slow. As a consequence, the assumption on sparse representation becomes important, because the level of sparsity determines the number of inner optimization problems that LPBoost needs to solve. Second, for any specific pattern on the training vectors, LPBoost use one and only one h_t to represent it. The drawback here is that the single hypothesis h_t may not be the best compared to other ambiguous hypotheses. Figure 2.3 shows one such situation, in which the hypotheses are hyperplanes in two dimensional spaces. The classifier h_1 seems to be a stabler choice over h_2 for the pattern, but LPBoost might only select h_2 as the representative. This drawback contradicts the ensemble view for stability: we should average over the predictions of the ambiguous hypotheses to obtain a stabler classifier.

2.3 Connecting SVM to Ensemble Learning

In (P_7), we consider selecting an infinite number of hypotheses h_t from $\mathcal{H}$, and then assigning the hypothesis weights w_t to them. When $\mathcal{H}$ is of countably infinite size, an equivalent approach is to assume $\mathcal{H} = \{h_{(a)}\}_{a=1}^{\infty}$, and obtain a nonnegative weight $w_{(a)}$ for each hypothesis. We will use t when the hypotheses are iteratively selected, and a as a general enumeration. Consider the feature transform

$$\Phi(x) = (h_{(1)}(x), h_{(2)}(x), \cdots, h_{(a)}(x), \cdots). \tag{2.8}$$

From (P_3) and (P_7), we can clearly see the connection between SVM and LPBoost. The features in ϕ_x in SVM and the hypotheses $h_{(a)}(x)$ in LPBoost play similar roles. SVM and LPBoost both work on linear combinations of the features (hypothesis predictions), though SVM has an additional intercept term b. They both minimize the sum of a margin-control term and a violation term. However, SVM focuses on the ℓ_2-margin while LPBoost deals with the ℓ_1-margin. The later results in sparse representation of the ensemble classifier.

The connection between SVM and ensemble learning is widely known in literature. Freund and Schapire (1999) have shown the similarity of large-margin concept between SVM and AdaBoost. The connection has been used to develop new algorithms. For example, Rätsch et al. (2001) have tried to select the hypotheses by AdaBoost and obtain the hypothesis weights by solving an optimization problem similar to (P_3). Another work by Rätsch et al. (2002) has shown a new one-class estimation algorithm based on the connection. Recently, Rosset et al. (2004) have applied the similarity to compare SVM with boosting algorithms.

Although $\mathcal{H}$ can be of infinite size, previous results that utilize the connection between SVM and ensemble learning usually only consider a finite subset of $\mathcal{H}$. One of the reasons is that the constraints $w_{(a)} \geq 0$, which are required for ensemble learning, are hard to handle in SVM. If we have an infinite number of hypothesis weights, and directly add the nonnegative constraints to (P_3), Vapnik (1998) shows that the dual problem (P_4) would become

$$\begin{aligned}
(P_8) \quad \min_{\lambda \in \mathbb{R}^N, \zeta} \quad & \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N} \lambda_i \lambda_j y_i y_j \mathcal{K}(x_i, x_j) - \sum_{i=1}^{N} \lambda_i + \sum_{i=1}^{N} y_i \lambda_i \langle \phi_{x_i}, \zeta \rangle + \frac{1}{2}\langle \zeta, \zeta \rangle \\
\text{subject to} \quad & 0 \leq \lambda_i \leq C, i = 1, \cdots, N, \\
& \zeta_{(a)} \in \mathbb{R}, \zeta_{(a)} \geq 0, a = 1, 2, \cdots, \infty.
\end{aligned}$$

Because ζ is an unknown vector of infinite size, we cannot perform $\langle \cdot, \zeta \rangle$ with the kernel trick. In addition, we still have an infinite number of variables and constraints in (P_8), and we cannot solve such problem directly. We would deal with these difficulties in the next chapter. In addition, we would show how we could construct a kernel from (2.8).

Algorithm 1 AdaBoost

```
• Input:
    – The training set $Z = \{(x_1, y_1), ..., (x_N, y_N)\}$.
    – The number of iterations $T$.
    – The base learning model $\mathcal{H}$ and the base learning algorithm $\mathcal{A}_{\mathcal{H}}$.
• Procedure:
    – Initialize the sample weights $U_1(i) = 1/N$ for $i = 1, 2, \cdots, N$.
    – For $t = 1, \cdots, T$ do
        1. Call $\mathcal{A}_{\mathcal{H}}$ to obtain $h_t \in \mathcal{H}$ that achieves the minimum error on the
           weighted training set $(Z, U_t)$.
        2. Calculate the weighted error $\epsilon_t$ of $h_t$.
               $\epsilon_t = \sum_{i=1}^{N} I[h_t(x_i) \neq y_i] \cdot U_t(i),$
           Abort if $\epsilon_t = 0$ or $\epsilon_t \geq \frac{1}{2}$.
        3. Set
               $w_t = \log\left(\frac{1-\epsilon_t}{\epsilon_t}\right),$
           and update the sample weights by
               $U_{t+1}(i) = U_t(i) \exp\left(-w_t I[h_t(x_i) = y_i]\right),$ for $i = 1, 2, \cdots, N$.
        4. Normalize $U_{t+1}$ such that $\sum_{i=1}^{N} U_{t+1}(i) = 1$.
• Output:
    – The decision function $g^*(x) = \text{sign}\Big(\sum_{t=1}^{T} w_t h_t(x)\Big)$.
```

Algorithm 2 LPBoost

```
• Input:
    – The training set $Z = \{(x_1, y_1), ..., (x_N, y_N)\}$.
    – The soft-margin parameter $C$.
    – The base learning model $\mathcal{H}$ and the base learning algorithm $\mathcal{A}_{\mathcal{H}}$.
• Procedure:
    – Initialize the sample weights $U_1(i) = 1/N$ for $i = 1, 2, \cdots, N$.
    – Initialize $\beta_1 = 0$.
    – For $t = 1, \cdots, \infty$ do
        1. Call $\mathcal{A}_{\mathcal{H}}$ to obtain $h_t \in \mathcal{H}$ that achieves the minimum error on the
           weighted training set $(Z, U_t)$.
        2. Check if $h_t$ could really improve the optimal solution:
           If $\sum_{i=1}^{N} U_t(i) y_i h_t(x_i) \le \beta_t$, $T \leftarrow t - 1$, break.
        3. Update the sample weights $U_t$ and current optimality barrier $\beta_t$ by
           solving
```

$$\begin{aligned} (P_9) \quad & \min_{U_{t+1}, \beta_{t+1}} && \beta_{t+1} \\ & \text{subject to} && \sum_{i=1}^{N} U_{t+1}(i) y_i h_k(x_i) \le \beta_{t+1}, k = 1, \cdots, t, \qquad (2.9) \\ & && \sum_{i=1}^{N} U_{t+1}(i) = 1, \\ & && 0 \le U_{t+1}(i) \le C, i = 1, \cdots, N. \end{aligned}$$

```
        4. Update the weight vector $w$ with the Lagrange multipliers of (2.9).
• Output:
    – The decision function $g^*(x) = \text{sign}\Big(\sum_{t=1}^{T} w_t h_t(x)\Big)$.
```

Chapter 3

SVM-based Framework for Infinite Ensemble Learning

Our goal is to conquer the task of infinite ensemble learning without confining ourselves to the sparse representation. Traditional algorithms cannot be directly generalized to solve this problem, because they either iteratively include only a finite number of T hypotheses (AdaBoost), or assume sparse representation strongly (LPBoost).

The connection between SVM and ensemble learning shows another possible approach. We can form a kernel that embodies the predictions of all the hypotheses in $\mathcal{H}$. Then, the decision function (2.6) obtained from SVM with this kernel is a linear combination of those predictions (with an intercept term). However, there are still two main obstacles. One is to compute the kernel when $\mathcal{H}$ is of possibly uncountable size, and the other is to handle the nonnegative constraints on the weights to make (2.6) an ensemble classifier. In this chapter, we shall address the details of these obstacles, and then propose a thorough framework that exploits SVM for infinite ensemble learning.

3.1 Embedding Learning Model into the Kernel

In this section, we try to embed the hypotheses in the base learning model $\mathcal{H}$ into an SVM kernel. Our goal is infinite ensemble learning. Thus, although the embedding works when $\mathcal{H}$ is of either finite or infinite size, we would assume the infinite case.

We have shown in (2.8) that we could construct a feature mapping using the predictions of the hypotheses $h_{(a)} \in \mathcal{H}$. In Definition 3, we extend this idea to a more general form, and defines a kernel based on the feature mapping.

Definition 3 *Assume that $\mathcal{H} = \{h_\alpha \colon \alpha \in \mathcal{C}\}$, where $\mathcal{C}$ is a measure space with measure μ. The kernel that embodies $\mathcal{H}$ with a positive function $r \colon \mathcal{C} \to \mathbb{R}^+$ is defined as*

$$\mathcal{K}_{\mathcal{H},r}(x, x') = \int_{\mathcal{C}} \phi_x(\alpha)\phi_{x'}(\alpha)\, d\mu(\alpha), \tag{3.1}$$

where $\phi_x(\alpha) = r(\alpha)h_\alpha(x)$ is a measurable function over μ, and the embedding function $r(\alpha)$ is chosen such that the Lebesgue integral exists for all $x, x' \in \mathcal{X}$.

Here, the index α is called the *parameter* of the hypothesis h_α. Note that depending on the way that we parameterize $\mathcal{H}$, two hypotheses h_{α_1} and h_{α_2}, where $\alpha_1 \neq \alpha_2$, may have $h_{\alpha_1}(x) = h_{\alpha_2}(x)$ for all $x \in \mathcal{X}$. For example, we could parameterize the set of finite ensemble classifiers in (1.2) by (w, h). But an ensemble classifier with parameter (w, h) is equivalent to an ensemble classifier with parameter $(\tilde{w}, h)$, where $\tilde{w}_t$ are the associated normalized hypothesis weights. We would treat those hypotheses as different objects during parameterization, while bearing in mind that they represent the same function. That is, the learning model $\mathcal{H}$ is equivalently $\bigcup_{\alpha \in \mathcal{C}} \{h_\alpha\}$.

From now on, we shall denote $\mathcal{K}_{\mathcal{H},r}$ by $\mathcal{K}_{\mathcal{H}}$ when it is clear about the embedding function r from the context, or when the specific choice of r is irrelevant. If $\mathcal{C}$ is a closed interval $[L, R]$, we can easily observe that the right-hand-side of (3.1) is an inner product (Schölkopf and Smola 2002), and hence Definition 3 constructs a valid kernel. In the following theorem, we formalize such observation for a general $\mathcal{C}$.

Theorem 5 *Consider the kernel $\mathcal{K}_{\mathcal{H}} = \mathcal{K}_{\mathcal{H},r}$ in Definition 3.*

1. *The kernel is an inner product for ϕ_x and $\phi_{x'}$ in the Hilbert space $\mathcal{F} = \mathcal{L}_2(\mathcal{C}, d\mu)$, where $\mathcal{L}_2(\mathcal{C}, d\mu)$ is the set of functions $\varphi(\cdot) \colon \mathcal{C} \to \mathbb{R}$ that are square integrable over measure μ.*

2. *For a given set of training vectors $X = \{x_i\}_{i=1}^N \in \mathcal{X}^N$, the Gram matrix of $\mathcal{K}_{\mathcal{H}}$ is PSD.*

Proof. By Definition 3,

$$\mathcal{K}_{\mathcal{H},r}(x,x) \quad = \quad \int_{\mathcal{C}} \left(\phi_x(\alpha)\right)^2 \, d\mu(\alpha)$$

exists for all $x \in \mathcal{X}$. Thus, the functions $\phi_x(\alpha)$ belongs to $\mathcal{F} = \mathcal{L}_2(\mathcal{C}, d\mu)$. The results of Reed and Simon (1980, Section II.2, Example 6) show that $\mathcal{F}$ is a Hilbert space, in which the inner product between two functions φ and φ' is defined as

$$\langle \varphi, \varphi' \rangle = \int_{\mathcal{C}} \varphi(\alpha)\varphi'(\alpha)\, d\mu(\alpha).$$

Then, we can see that $\mathcal{K}_{\mathcal{H}}(x, x')$ is an inner product for ϕ_x and $\phi_{x'}$ in $\mathcal{F}$. The second part is just a consequence of the first part by Theorem 4. □

The technique for using an integral inner product between functions is known in SVM literature. For example, Schölkopf and Smola (2002, Section 13.4.2) explain that the Fisher kernel takes an integral inner product between two regularized functions. Our framework applies this technique to combine predictions of hypotheses, and thus could handle the situation even when the base learning model $\mathcal{H}$ is uncountable. When we apply $\mathcal{K}_{\mathcal{H}}$ to (P_4), the primal problem (P_3) becomes

$$\begin{array}{lll} (P_{10}) & \displaystyle\min_{w,b,\xi} & \displaystyle\frac{1}{2}\int_{\mathcal{C}} \left(w(\alpha)\right)^2 \, d\mu(\alpha) + C\sum_{i=1}^{N} \xi_i \\ & \text{s.t.} & \displaystyle y_i\left(\int_{\mathcal{C}} w(\alpha)r(\alpha)h_\alpha(x_i)\, d\mu(\alpha) + b\right) \geq 1 - \xi_i, \\ & & \xi_i \geq 0, i = 1, \cdots, N, \\ & & w \in \mathcal{L}_2(\mathcal{C}, d\mu), b \in \mathbb{R}, \xi \in \mathbb{R}^N. \end{array}$$

In particular, the decision function (2.6) obtained after solving (P_4) with $\mathcal{K}_{\mathcal{H}}$ is the same as the decision function obtained after solving (P_{10}):

$$g^*(x) = \operatorname{sign}\left(\int_{\mathcal{C}} w(\alpha)r(\alpha)h_\alpha(x)\, d\mu(\alpha) + b\right). \tag{3.2}$$

When $\mathcal{C}$ is uncountable, it is possible that each hypothesis $h_\alpha(x)$ only takes an infinitesimal weight $w(\alpha)r(\alpha)\,d\mu(\alpha)$ in $g^*(x)$. We shall discuss this situation further in Section 3.3. Note that (3.2) is not an ensemble classifier yet, because we do not have the constraints $w(\alpha) \geq 0$ for all possible $\alpha \in \mathcal{C}$, and we have an additional intercept term b.[1] In the next section, we focus on these issues, and explain that $g^*(x)$ is equivalent to an ensemble classifier under some reasonable assumptions.

3.2 Assuming Negation Completeness

To make (3.2) an ensemble classifier, we need to have $w(\alpha) \geq 0$ for all $\alpha \in \mathcal{C}$. Somehow these constraints are not easy to satisfy. We have shown in Section 2.3 that even when we only add countably infinite number of constraints to (P_3), we would introduce infinitely many variables and constraints in (P_8), which makes the later problem difficult to solve (Vapnik 1998).

One remedy is to assume that $\mathcal{H}$ is *negation complete*, that is,[2]

$$h \in \mathcal{H} \Leftrightarrow (-h) \in \mathcal{H}.$$

Then, every linear combination over $\mathcal{H}$ has an equivalent linear combination with only nonnegative hypothesis weights. Thus, we can drop the constraints during optimization, but still obtain a decision function that is equivalent to some ensemble classifier. For example, for $\mathcal{H} = \{h_1, h_2, (-h_1), (-h_2)\}$, if we have a decision function

$$\operatorname{sign}(3h_1(x) - 2h_2(x))\,,$$

it is equivalent to

$$\operatorname{sign}(3h_1(x) + 2(-h_2)(x))\,,$$

and the later is an ensemble classifier over $\mathcal{H}$.

[1]Actually, $w(\alpha)r(\alpha) \geq 0$. Somehow we assumed that $r(\alpha)$ is always positive.

[2]We use $(-h)$ to denote the function $(-h)(\cdot) = -(h(\cdot))$.

Algorithm 3 SVM-based framework for infinite ensemble learning

- Input:
  - The training set $Z = \{(x_1, y_1), \cdots, (x_N, y_N)\}$.
  - The soft-margin parameter C.
  - The base learning model $\mathcal{H}$ and the kernel $\mathcal{K}_{\mathcal{H}}$ given in Definition 3. The kernel is computed from $\mathcal{H}$, which is assumed to be an infinite, negation complete learning model that contains a constant classifier.
- Procedure:
  - Solve (P_4) with $\mathcal{K}_{\mathcal{H}}$ and obtain Lagrange multipliers λ_i.
  - Compute b from (2.5).
- Output:
  - The decision function $g^*(x) = \text{sign}\Big(\sum_{i=1}^N y_i \lambda_i \mathcal{K}(x_i, x) + b\Big)$, which is equivalent to some ensemble classifier over $\mathcal{H}$.

Note that negation completeness is usually a mild assumption for a reasonable learning model. Following this assumption, after solving (P_4) with $\mathcal{K}_{\mathcal{H}}$, the decision function (3.2) can be interpreted an ensemble classifier over $\mathcal{H}$ with an intercept term b. Note that b can be viewed as a hypothesis weight on a constant classifier c, which predicts $c(x) = 1$ for all $x \in \mathcal{X}$. In general, a reasonable learning model $\mathcal{H}$ contains c and $(-c)$, in order to handle, for example, the situation that all training labels are the same. We shall make the assumption that $\mathcal{H}$ contains both c and $(-c)$ from now on. Then, $g^*(\cdot)$ in (3.2) or (2.6) is indeed equivalent to an ensemble classifier.

We summarize our framework in Algorithm 3. The algorithm looks simple because most of the work could be done by existing SVM algorithms. The hard part is mostly in obtaining the kernel $\mathcal{K}_{\mathcal{H}}$. In the next section, we further show some properties of the framework, which would facilitate the demonstration of concrete instances in Chapter 4.

3.3 Properties of the Framework

In this section, we introduce two important properties of the SVM-based framework. First we show that the framework allows us to embed multiple base learning models together with a simple summation over the kernels. This property demonstrates not only an advantage of the framework, but also the simpicity of manipulating with kernels. Second, we show that the framework treats the ambiguous hypotheses fairly. We have mentioned in Section 2.2 that LPBoost could only select one hypothesis among the ambiguous ones, but our framework would average over the predictions of all of them. Thus, from the ensemble point-of-view, our framework would be stabler.

3.3.1 Embedding Multiple Learning Models

The SVM-based framework allows us to embed multiple base learning models altogether. Consider two base learning models $\mathcal{H}_1$ and $\mathcal{H}_2$, if we could embed them into kernels $\mathcal{K}_{\mathcal{H}_1}$ and $\mathcal{K}_{\mathcal{H}_2}$, respectively, then the kernel

$$\mathcal{K}(x, x') = \mathcal{K}_{\mathcal{H}_1}(x, x') + \mathcal{K}_{\mathcal{H}_2}(x, x')$$

embeds both of those learning models. In other words, if we use $\mathcal{K}(x, x')$ in Algorithm 3, and $\mathcal{H} = \mathcal{H}_1 \cup \mathcal{H}_2$ satisfies the required assumptions, we could obtain an ensemble classifier over $\mathcal{H}$.

When we want to consider multiple base learning models together, traditional ensemble learning algorithms usually require calling a base learning algorithm for each model. Such step is usually more time-consuming than just summing up the kernel evaluations. In fact, as shown in the next theorem, our framework could embed a countably infinite number of base learning models altogether, which can hardly be done by traditional ensemble learning algorithms.

Theorem 6 *Assume that for some $M \in \mathbb{N} \cup \{\infty\}$, we could define the kernels*

$\mathcal{K}_{\mathcal{H}_1}, \cdots, \mathcal{K}_{\mathcal{H}_M}$ with learning models $\mathcal{H}_1, \cdots, \mathcal{H}_M$, respectively. Then, let

$$\mathcal{K}(x, x') = \sum_{m=1}^{M} \mathcal{K}_{\mathcal{H}_m}(x, x').$$

If $\mathcal{K}(x, x')$ exists for all $x, x' \in \mathcal{X}$, and

$$\mathcal{H} = \bigcup_{m=1}^{M} \mathcal{H}_m \cup \{c, (-c)\}$$

is negation complete, Algorithm 3 using $\mathcal{K}(x, x')$ could output an ensemble classifier over $\mathcal{H}$.

Proof. From Theorem 5, each $\mathcal{K}_{\mathcal{H}_m}$ is an inner product in a Hilbert space $\mathcal{F}_m$. From the results of Reed and Simon (1980, Example 5), we can see that a countable direct sum over Hilbert spaces is still a Hilbert space. Let $\mathcal{F}$ be the countable direct sum of the spaces $\mathcal{F}_1, \cdots, \mathcal{F}_M$, we could define an inner product in $\mathcal{F}$ by summing the inner products in $\mathcal{F}_1, \cdots, \mathcal{F}_M$. The resulting inner product is $\mathcal{K}(x, x')$, and hence the decision function g^* from (3.2) represents a linear combination over the predictions of $\bigcup_{m=1}^{M} \mathcal{H}_m \cup \{c\}$. Under the assumption, we can see that g^* is equivalent to an ensemble classifier over $\mathcal{H}$. $\square$

Note that we do not intend to define a a kernel with $\mathcal{H}$ directly in Theorem 6, because doing so may require choosing suitable $\mathcal{C}$, μ, and r, which may not be an easy task for such a complex learning model. However, Theorem 6 allows us to construct kernels on the simpler learning models first, and use the combination of these kernels to obtain an ensemble classifier over the full union. We will further see the power of summing over kernels in Section 4.3.

3.3.2 Averaging Ambiguous Hypotheses

We have shown in Section 2.2 that LPBoost could only select one hypothesis among the ambiguous ones. In the next theorem, we show that our framework takes a different approach: if one of the ambiguous hypotheses is included with nonzero hypothesis weight in the ensemble, all the ambiguous hypotheses are also included.

Theorem 7 *For a given training set Z, and a kernel $\mathcal{K}_{\mathcal{H},r}$ defined from a given learning model $\mathcal{H}$, assume that $h_{\alpha_1} \in \mathcal{H}$ and $h_{\alpha_2} \in \mathcal{H}$ are ambiguous on the training vectors. Then, after solving (P_{10}),*

$$\frac{w(\alpha_1)}{r(\alpha_1)} = \frac{w(\alpha_2)}{r(\alpha_2)}.$$

In other words, if the hypothesis h_{α_1} has an infinitesimal, but nonzero, weight $w(\alpha_1)r(\alpha_1)d\mu(\alpha)$, then both hypotheses have nonzero weights.

Proof. Recall from (2.4) that for optimal solution (w, b, ξ) of (P_{10}) and optimal solution λ of (P_4),

$$w(\alpha) = \sum_{i=1}^{N} y_i \lambda_i \phi_{x_i}(\alpha).$$

Because of ambiguity, $h_{\alpha_1}(x_i) = h_{\alpha_2}(x_i)$ for $i = 1, 2, \cdots, N$. Then, by $\phi_{x_i}(\alpha) = r(\alpha)h_\alpha(x_i)$, we can see that

$$\frac{w(\alpha_1)}{r(\alpha_1)} = \frac{w(\alpha_2)}{r(\alpha_2)}.$$

$\square$

Let us take a deeper look at Theorem 7. For N training vectors, there are at most 2^N patterns to label them. Thus, we can divide the hypotheses in $\mathcal{H}$ to at most 2^N groups, each of which contains ambiguous hypotheses that produce the same pattern. LPBoost selects at most one hypothesis from each group to form the ensemble classifier, because of the restriction of sparse representation. On the other hand, our framework, which does not have such restriction, could average over the predictions of possibly infinite number of hypotheses within each group. Thus, even if each hypothesis only has an infinitesimal hypothesis weight, the average of their predictions, which is an integral, could be concrete. This shows how the infinitesimal hypothesis weights work.

We have introduced our SVM-based framework for infinite ensemble learning. At this point, the framework is still abstract. In the next chapter, we would demonstrate some concrete instances of the framework. In particular, we would show how we could parameterize some important learning models and embed their hypotheses into kernels.

Chapter 4

Concrete Instances of the Framework

In this chapter, we derive some concrete instances from the framework. In Section 4.1, we would start by introducing the stump kernel, which embodies an infinite number of decision stumps. The decision stump is one of the simplest base learning models that are applied to ensemble learning, and we would show that the stump kernel is simple yet powerful. Then, we would extend stump kernel to the perceptron kernel in Section 4.2. The perceptron is a very important learning model that is related to learning in neural network. We would show that our framework with the perceptron kernel equivalently constructs a neural network with an infinite number of neurons. In Section 4.3, we show an approach to construct kernels with hypotheses that are combined by logical operations. Interestingly, this technique allows us to give novel interpretations of some existing kernels from an ensemble point-of-view.

4.1 Stump Kernel

4.1.1 Formulation

The *decision stump* $s_{q,d,\alpha}\colon \mathcal{X} \to \{-1,+1\}$ is of the form

$$s_{q,d,\alpha}(x) = q \cdot \operatorname{sign}\big((x)_d - \alpha\big).$$

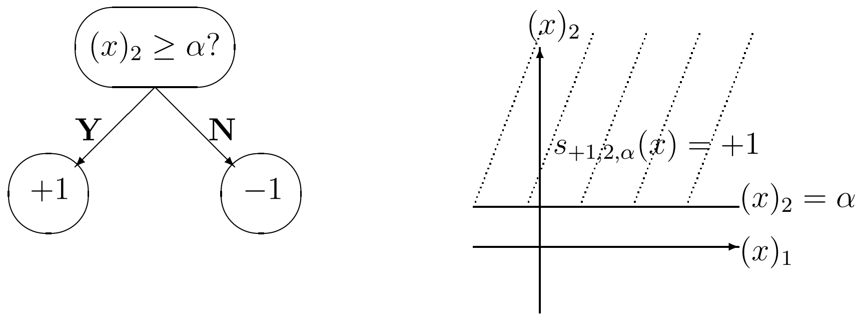


Figure 4.1: Illustration of the decision stump $s_{+1,2,\alpha}(x)$

The decision stump works on the d-th feature element of x, and classifies the vector x according to the direction $q \in \{-1, +1\}$ and the threshold α. In other words, the decision stump is a hyperplane classifier in which the associated hyperplane is perpendicular to the d-th axis. The operation of decision stumps is illustrated in Figure 4.1.

Although the set of decision stumps is a very simple learning model, ensemble learning algorithms with such base learning model can usually achieve reasonable performance. In addition, the associated base learning algorithm is efficient and easy to implement. Thus, the set of decision stumps is a popular base learning model for ensemble learning (Freund and Schapire 1996; Bauer and Kohavi 1999; Demiriz et al. 2002).

For constructing the stump kernel, we would consider the set of decision stumps

$$\mathcal{S} = \left\{ s_{q,d,\alpha_d} \colon q \in \{-1,+1\}, d \in \{1, \cdots, D\}, \alpha_d \in [L_d, R_d] \right\}.$$

In addition, we would assume that

$$\mathcal{X} \subseteq [L_1, R_1] \times [L_2, R_2] \times \cdots \times [L_D, R_D]. \tag{4.1}$$

In this case, we can easily see that $\mathcal{S}$ is negation complete, and contains $s_{+1,1,L_1}(\cdot)$ as a constant classifier. Thus, the stump kernel $\mathcal{K}_{\mathcal{S}}$, which would be defined below, can be applied to Algorithm 3 to obtain an infinite ensemble classifier.

Definition 4 *The stump kernel is $\mathcal{K}_{\mathcal{S}}$ with $r(q, d, \alpha_d) = \frac{1}{2}$ for all valid (q, d, α_d) in the definition of $\mathcal{S}$. The measure μ on parameters q and d is the counting measure, and $d\mu$ is uniform in the range of α_d. With* (4.1) *and Definition 3, we get*

$$\mathcal{K}_{\mathcal{S}}(x, x') = \Delta_{\mathcal{S}} - \sum_{d=1}^{D} \left| (x)_d - (x')_d \right|,$$

where $\Delta_{\mathcal{S}} = \frac{1}{2} \sum_{d=1}^{D} (R_d - L_d)$ is a constant.

The stump kernel, as its associated base learning model, is very simple to compute. However, it is very powerful, in the sense that SVM with the kernel is equivalently searching within a learning model of infinite capacity. The stump kernel also shows an interesting connection to radial basis functions, which is an important concept in learning theory. Next, we would further discuss these properties.

4.1.2 Power of the Stump Ensemble

In Theorem 3, we see that the set of hyperplane classifiers (which has $\Gamma \to 0$) in $\mathbb{R}^D$ has V-C dimension D. When we use nonlinear SVM, we work in a feature space $\mathcal{F}$ instead of $\mathbb{R}^D$. The general hope that the set of hyperplane classifiers in $\mathcal{F}$ would have a larger capacity than such classifiers in $\mathbb{R}^D$. This hope is indeed true for the stump kernel. Next, analyze the capacity of the hyperplane classifiers in the associated feature space of the stump kernel. With the negation completeness assumption, the capacity of those hyperplane classifiers would be the same as the capacity of the ensemble classifiers over $\mathcal{S}$. We start our analysis with the following lemma for one-dimensional training vectors.

Lemma 1 *Consider one-dimensional training vectors $\{x_i\}_{i=1}^{N}$, where $x_i \in \mathcal{X} \subseteq (L, R)$. Assume that $\mathcal{S}$ is defined with $q \in \{-1, +1\}$ and $\alpha_1 \in [L, R]$. If $x_i \neq x_j$ for all $i \neq j$, the Gram matrix of $\mathcal{K}_{\mathcal{S}}$ is PD.*

Proof. Assume that the Gram matrix is K. For each nonzero vector $v \in \mathbb{R}^N$, we want to test whether $v^T K v > 0$. Without loss of generality, consider

$$L < x_1 < \cdots < x_N < R.$$

Let $x'_i = x_i - L$. Then,

$$0 < x'_1 < \cdots < x'_N < R - L.$$

The matrix P with $P_{ij} \equiv \min(x'_i, x'_j)$ is PD because P is congruent to a diagonal matrix with positive diagonals $x'_1, (x'_2 - x'_1), \cdots, (x'_N - x'_{N-1})$. Similarly, for $x''_i = R - x_i$, the matrix Q with $Q_{ij} \equiv \min(x''_i, x''_j)$ is PD. Now we could write K in two different forms

$$\begin{aligned} & K_{ij} \\ = \; & \frac{1}{2}(-R + L) + \min(x'_i, x'_j) + \min(x''_i, x''_j) \quad (4.2) \\ = \; & \frac{1}{2}(x_1 - L) + \left(\frac{1}{2}(R - x_1) - |x_i - x_j| \right). \quad (4.3) \end{aligned}$$

When v is nonzero but $\sum_{i=1}^N v_i = 0$, we can evaluate $v^T K v$ in three parts with (4.2). The first part is 0. The second and the third parts are strictly positive when $v \neq 0$ because the matrices P and Q are PD. Hence, the sum is strictly positive.

When $\sum_{i=1}^N v_i \neq 0$, the first part of (4.2) is negative, and hence we cannot prove the PD-ness directly through this equation. Therefore, we apply (4.3) instead. From this equation, we can evaluate $v^T K v$ in two parts. The inner matrix of the second part, name it K', can be calculated from a stump kernel with the stumps in $[x_1, R]$. Thus, K' is PSD, and $v^T K' v \geq 0$. The first part is $\left(\sum_{i=1}^N v_i\right)^2$ multiplied by a positive constant. Hence, the sum of the two parts is still strictly positive. Therefore, K is PD. $\square$

With Lemma 1, we can extend the result to multi-dimensional vectors in the following theorem.

Theorem 8 *Consider training vectors* $\{x_i\}_{i=1}^N \in \mathcal{X}^N$ *and the stump kernel* $\mathcal{K}_\mathcal{S}$ *in Definition 4. If there exists a dimension* d *such that* $(x_i)_d \neq (x_j)_d$ *for all* $i \neq j$*, and*

$$\left[\min_{i=1,\cdots,N}(x_i)_d, \max_{i=1,\cdots,N}(x_i)_d\right] \subseteq (L_d, R_d)$$

then the Gram matrix of $\mathcal{K}_\mathcal{S}$ *is PD.*

Proof. The multi-dimensional stump kernel is the sum of several one-dimensional stump kernels, each of which produces a PSD Gram matrix by Theorem 5. If in one of dimensions, we can obtain a PD Gram matrix from Lemma 1, the sum of the matrices would be PD. □

The PD-ness of the Gram matrix is directly connected to the classification power of the hyperplane classifiers in the associated feature space. This can be formalized by the following theorem.

Theorem 9 *(Chang and Lin 2001b, Corollary 1) If the Gram matrix of the kernel on the training vectors is PD, the nonlinear hard-margin SVM with such kernel always has a feasible solution. That is, for every possible pattern of the training labels, the training examples can be separated by some hyperplane classifier in the associated feature space.*

In other words, if we can find a set of training vectors such that the Gram matrix is PD, we can shatter those training vectors with the set of hyperplane classifiers in $\mathcal{F}$. For the stump kernel, such set of training vectors exists for any positive integer N. Thus, the set of hyperplane classifiers in $\mathcal{F}$, or equivalently, the set of ensemble classifier over $\mathcal{S}$, has infinite capacity, as shown below.

Corollary 1 *The class of the infinite ensemble classifiers over* $\mathcal{S}$ *has an infinite V-C dimension.*

We make two remarks here. First, although the assumption of Theorem 8 is mild in practice, there are still training sets that do not have this property. An example is the XOR training set, which is illustrated in Figure 4.2. We can easily see

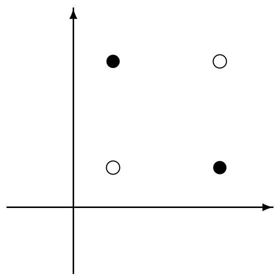


Figure 4.2: The XOR training set.

that every possible decision stump would have in-sample error $\frac{1}{2}$. Thus, AdaBoost and LPBoost would terminate with one stump in the ensemble. Similarly, the Gram matrix of stump kernel is only PSD but not PD, and nonlinear hard-margin SVM with the kernel cannot find any feasible solution for this problem. In other words, those training examples cannot be perfectly separated by an ensemble over the decision stumps.

Second, the unlimited capacity has to be used with suitable regularization in order to have good generalization performance. Although SVM implicitly regularizes the admissible learning model with the large-margin concept, the unlimited capacity may still drag the algorithm towards overfitting. This situation has been observed for SVM with the Gaussian kernel (Keerthi and Lin 2003). In this situation, soft-margin SVM with a suitable parameter selection usually performs better than hard-margin SVM, because trading accuracy with larger margin further regularizes the admissible learning model.

4.1.3 Stump Kernel and Radial Basis Function

Next, we show another property of the stump kernel, which facilitates its practical usage. In particular, we could drop the constant $\Delta_{\mathcal{S}}$ for the stump kernel $\mathcal{K}_{\mathcal{S}}$ without affecting the decision function obtained from Algorithm 3. That is, for finite number of training and testing examples, SVM could automatically compute the ranges $[L_d, R_d]$ and the constant $\Delta_{\mathcal{S}}$ for us.

Theorem 10 *Solving (P_4) with the simplified stump kernel $\tilde{\mathcal{K}}_{\mathcal{S}}(x, x') = -\left\|x - x'\right\|_1$ is the same as solving (P_4) with $\mathcal{K}_{\mathcal{S}}(x, x')$. That is, they could obtain equivalent decision functions* (2.6).

Proof. Berg et al. (1984) prove that the Gram matrix of $\tilde{\mathcal{K}}_{\mathcal{S}}(x, x') = -|x - x'|$ is CPSD for one-dimensional vectors x, x'. The Gram matrix of simplified stump kernel in $\mathbb{R}^D$ is the sum of the Gram matrix of several one-dimensional kernels, and hence would also be CPSD. In addition, for (P_4), a CPSD kernel $\tilde{\mathcal{K}}(x, x')$ works exactly the same as any PSD kernel of the form $\tilde{\mathcal{K}}(x, x') + \Delta$, where Δ is a constant, because of the linear constraint $\sum_{i=1}^{N} y_i \lambda_i = 0$ (Schölkopf and Smola 2002; Lin and Lin 2003). We have shown the value of $\Delta = \Delta_{\mathcal{S}}$ in Definition 4. Hence, the equivalence between $\tilde{\mathcal{K}}_{\mathcal{S}}$ and $\mathcal{K}_{\mathcal{S}}$ for (P_4) can be easily established. $\square$

Let us take a closer look at this result. $\tilde{K}_{\mathcal{S}}(x, x') = -\left\|x - x'\right\|_1$ is a *radial basis function* (RBF), just like the well-known Gaussian kernel. RBF is an important tool for classification, regression, interpolation, and many other learning tasks (Haykin 1999). Compared to the Gaussian kernel, the stump kernel computes the distance by one-norm, and did not use nonlinear transform of the distance. Nevertheless, the class of SVM classifiers with the stump kernel still has an infinite V-C dimension. We shall further compare them to some other types of RBF kernels in the end of this chapter.

4.1.4 Averaging Ambiguous Stumps

In Section 3.3, we have shown that our framework would average over the predictions of ambiguous hypotheses in the final ensemble. Next, we would use the stump kernel to further illustrate this property. For example, consider one-dimensional training vectors. We can see that all the decision stumps with $q = 1$ and thresholds strictly between two adjacent training vectors are ambiguous. In the next theorem, we extend this idea to multi-dimensional training vectors. We would explicitly show how these ambiguous stumps group together in the final ensemble classifier.

Theorem 11 *Define* $(\tilde{x})_{d,a}$ *as the a-th smallest value in* $\{(x_i)_d\}_{i=1}^N$*, where* x_i *are the input vectors in the training set, and* A_d *as the number of different* $(\tilde{x})_{d,a}$*. Let* $(\tilde{x})_{d,0} = L_d$*,* $(\tilde{x})_{d,(A_d+1)} = R_d$*, and*

$$\hat{s}_{q,d,a}(x) = \begin{cases} q & \text{for } (x)_d \geq (\tilde{x})_{d,t+1} \\ -q & \text{for } (x)_d \leq (\tilde{x})_{d,t} \\ q \cdot \frac{2(x)_d - (\tilde{x})_{d,a} - (\tilde{x})_{d,a+1}}{(\tilde{x})_{d,a+1} - (\tilde{x})_{d,a}} & \text{otherwise.} \end{cases}$$

Then,

$$K_{\mathcal{S}}(x, x') = \sum_{q \in \{-1,+1\}} \sum_{d=1}^{D} \sum_{a=0}^{A_d} \left(r(q,d,a)\right)^2 \hat{s}_{q,d,a}(x) \hat{s}_{q,d,a}(x'),$$

where $r(q,d,a) = \frac{1}{2}\sqrt{(\tilde{x})_{d,a+1} - (\tilde{x})_{d,a}}$.

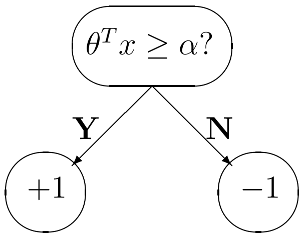
We can prove Theorem 11 by carefully writing down the equations. Note that the function $\hat{s}_{q,d,t}(\cdot)$ is not exactly a decision stump, but a smoother variant. Each $\hat{s}_{q,d,t}(\cdot)$ represents the group of ambiguous decision stumps in $((\tilde{x})_{d,t}, (\tilde{x})_{d,t+1})$. When the group is larger, $\hat{s}_{q,d,t}(\cdot)$ is smoother because it represents the average prediction over more decision stumps. Compared to LPBoost, which can use one discrete decision stump $s_{q,d,\alpha_d}(\cdot)$ for $\alpha_d \in \big((\tilde{x})_{d,t}, (\tilde{x})_{d,t+1}\big)$, our framework obtains a smoother stump by averaging ambiguous decision stumps. Even though each decision stump only has an infinitesimal hypothesis weight, the representative stump $\hat{s}_{q,d,t}(\cdot)$ could have a concrete weight in the ensemble. This result could give us further insights on how the infinitesimal hypothesis weights work.

4.2 Perceptron Kernel

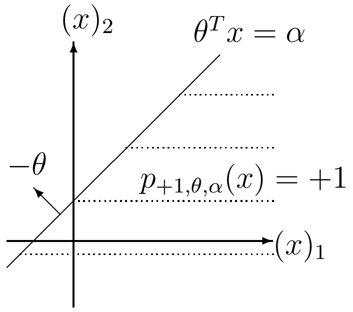
4.2.1 Formulation

In this section, we extend the stump kernel to the perceptron kernel, which embodies an infinite number of perceptrons. Each *perceptron* is of the form

$$p_{q,\theta,\alpha}(x) = q \cdot \text{sign}\big(\theta^T x - \alpha\big) .$$



(a) Decision Process



(b) Decision Boundary

Figure 4.3: Illustration of the perceptron $p_{+1,\theta,\alpha}(x)$

The perceptron defines a hyperplane classifier in $\mathbb{R}^D$, which is illustrated in Figure 4.3. We can see that the set of possible perceptrons in $\mathbb{R}^D$ includes the set of possible decision stumps. Hence, the perceptron model is usually considered to be more powerful than the decision stump model. The perceptron is a basic theoretical model for a neuron, and is very important for building neural networks (Haykin 1999). It is generally difficult to design a base learning algorithm for choosing a desired perceptron. Thus, the perceptron is hardly used in ensemble learning. Nevertheless, we are able to construct an ensemble classifier over infinite number of perceptrons with our framework.

We would consider the set of perceptrons

$$\mathcal{P} = \left\{ p_{q,\theta,\alpha} \colon q \in \{-1,+1\}, \theta \in \mathbb{R}^D, \|\theta\|_2 = 1, \alpha \in [-R, R] \right\}.$$

In addition, we would assume that

$$\mathcal{X} \subseteq \mathcal{B}(R),$$

where $\mathcal{B}(R)$ is a ball of radius R centered at the origin in $\mathbb{R}^D$. Then, we can see that the perceptron model $\mathcal{P}$ is negation complete, and contains a constant classifier $p_{+1,(1,0,\cdots,0),-R}(\cdot)$. Thus, we can apply the perceptron kernel $\mathcal{K}_{\mathcal{P}}$, which would be defined below, to Algorithm 3 and obtain an ensemble classifier over infinite number of perceptrons.

Definition 5 *The perceptron kernel is $\mathcal{K}_{\mathcal{P}}$ with $r(q,\theta,\alpha)=\kappa$, where κ is a constant to be defined below. The measure $\mu(q,\theta,\alpha)$ is the counting measure in the q direction, and $d\mu$ is uniform both on the surface $\|\theta\|_2 = 1$ and in $\alpha \in [-R,R]$. For $x, x' \in \mathcal{X} \subseteq \mathcal{B}(R)$,*

$$\mathcal{K}_{\mathcal{P}}(x,x') = \Delta_{\mathcal{P}} - \|x-x'\|_2\,,$$

where $\Delta_{\mathcal{P}}$ is a constant.

Proof. We have

$$\begin{aligned}
\mathcal{K}_{\mathcal{P}}(x,x') &= \kappa^2 \sum_{q\in\{-1,+1\}} \int_{\|\theta\|_2=1} \left(\int_{\alpha\in[-R,R]} p_{q,\theta,\alpha}(x) p_{q,\theta,\alpha}(x')\, d\mu_\alpha(\alpha) \right) d\mu_\theta(\theta) \\
&= 2\kappa^2 \int_{\|\theta\|_2=1} \left(\int_{\alpha\in[-R,R]} p_{+1,\theta,\alpha}(x) p_{+1,\theta,\alpha}(x')\, d\mu_\alpha(\alpha) \right) d\mu_\theta(\theta) \\
&= 2\kappa^2 \int_{\|\theta\|_2=1} \left(\int_{\alpha\in[-R,R]} s_{+1,1,\alpha}(\theta^T x) s_{+1,1,\alpha}(\theta^T x')\, d\mu_\alpha(\alpha) \right) d\mu_\theta(\theta) \\
&= 4\kappa^2 \int_{\|\theta\|_2=1} \left(\Delta_{\mathcal{S}} - \left|\theta^T x - \theta^T x'\right| \right) d\mu_\theta(\theta) \\
&= 4\kappa^2 \Delta_{\mathcal{S}} - 4\kappa^2 \cdot \left(\int_{\|\theta\|_2=1} \|\theta\|_2 \|x-x'\|_2 \left| \cos\bigl(\text{angle}(\theta, x-x')\bigr) \right| d\mu_\theta(\theta) \right) \\
&= 4\kappa^2 \Delta_{\mathcal{S}} - 4\kappa^2 \|x-x'\|_2 \cdot \left(\int_{\|\theta\|_2=1} \left| \cos\bigl(\text{angle}(\theta, (1,0,\cdots,0))\bigr) \right| d\mu_\theta(\theta) \right)
\end{aligned}$$

Here $\Delta_{\mathcal{S}}$ is the constant for a one-dimensional stump kernel with thresholds between $[-R,R]$. The operator angle$(\cdot,\cdot)$ is the angle between two vectors. Because θ is taken for all directions, we can use symmetry to obtain the last equality. Then, we can set

$$\kappa = \left(4 \int_{\|\theta\|_2=1} \left| \cos\bigl(\text{angle}(\theta, (1,0,\cdots,0))\bigr) \right| d\mu_\theta(\theta) \right)^{-\frac{1}{2}}$$

and

$$\Delta_{\mathcal{P}} = 4\kappa^2 \Delta_{\mathcal{S}}$$

to obtain the definition. □

With the perceptron kernel, we are able to construct an infinite ensemble classifier over perceptrons. Such classifier is a neural network with one hidden layer, infinite hidden nodes, and the hard-threshold activation functions (Haykin 1999). This kind of neural network could never be constructed with traditional neural network algorithms, both because of the infinity, and because the hard-threshold activation function is not smooth. Traditional ensemble learning algorithms also cannot construct such ensemble, because it is not easy to implement a base learning algorithm of perceptrons. This demonstrates another usefulness of our framework: obtaining novel classifiers that cannot be obtained by traditional algorithms.

The perceptron kernel shares many similar properties to the stump kernel. Next, we would further illustrate these properties in detail.

4.2.2 Properties of the Perceptron Kernel

Similar to the stump kernel, the SVM classifier with the perceptron kernel also has infinite V-C dimension. We start with a lemma that is well known in literature for interpolation with RBF functions.

Lemma 2 *(Micchelli 1986; Baxter 1991) When $N > 1$, for training vectors $\{x_i\}_{i=1}^{N} \in \mathcal{X}^N$ such that $x_i \neq x_j$ for all $i \neq j$, the Gram matrix of $\tilde{K}_{\mathcal{P}}(x, x') = -\|x - x'\|_2$ is CPD.*

Then, we could show a sufficient condition for the Gram matrix of the perceptron kernel to be PD.

Theorem 12 *Consider training vectors $\{x_i\}_{i=1}^{N} \in \mathcal{X}^N$, and the perceptron kernel $\mathcal{K}_{\mathcal{P}}$ in Definition 5. If $\mathcal{X} \subset \mathcal{B}(R)$ but $\mathcal{X} \neq \mathcal{B}(R)$, and $x_i \neq x_j$ for all $i \neq j$, then the Gram matrix of $\mathcal{K}_{\mathcal{P}}$ is PD.*

Proof. Assume that the Gram matrix is K, and $x_i \in \mathcal{B}(R')$, where $R' < R$. Then, we can evaluate K by two parts. Assume that $K = K' + (K - K')$, where K' can be computed from a perceptron kernel with only the perceptrons of thresholds $[-R', R']$. For all $x, x' \in \mathcal{X}$, we have

$$K(x,x') - K'(x,x') = \int_{\|\theta\|_2=1} \left(\int_{\alpha \in [-R,-R'] \cup [R',R]} \kappa^2 \, d\mu_\alpha(\alpha) \right) d\mu_\theta(\theta)$$

is a constant. We want to test whether $v^T K v > 0$ for all $v \neq 0$. Consider two cases. First, when $\sum_{i=1}^N v_i \neq 0$, the first part is nonnegative because K' is PSD by Mercer's condition, and the second part is strictly positive. Thus, $v^T K v > 0$.

When $\sum_{i=1}^N v_i = 0$ and we have some nonzero v_i, we must have $N > 1$. Then, the first part is strictly positive because K' is CPD by Lemma 2, and the second part is 0. Thus, we still have $v^T K v > 0$. That is, K is PD. $\square$

Therefore, we obtain the following corollary that illustrates the power of ensemble classifiers over the perceptron model $\mathcal{P}$.

Corollary 2 *The class of the infinite ensemble classifiers over $\mathcal{P}$ has an infinite V-C dimension.*

The perceptron kernel shares another similarity with the stump kernel. The constant $\Delta_\mathcal{P}$ could also be dropped when applying it to Algorithm 3. We formalize this by the following theorem.

Theorem 13 *Solving (P_4) with the simplified perceptron kernel $\tilde{\mathcal{K}}_\mathcal{P}(x,x') = -\|x - x'\|_2$ is the same as solving (P_4) with $\mathcal{K}_\mathcal{P}(x,x')$.*

Proof. The steps of the proof are exactly the same as the one for Theorem 10. $\square$

The stump kernel and the perceptron kernel both belong to the RBF kernel family. Are there any other kernels in the RBF family that can be explained from an ensemble point-of-view? In the next section, we extend the stump kernel and the perceptron kernel using combination of logic rules. Such extension allows us to construct kernels that embed an infinite number of decision trees. Interestingly, those kernels are directly related to the popular Laplacian kernel and exponential kernel.

4.3 Kernels that Embed Combined Hypotheses

So far we have considered hypotheses that return $\{-1, +1\}$. However, Definition 3 is not limited to such hypotheses. In the following, we show how to construct a kernel that embeds functions which output $\{0, 1\}$. Those functions would be called the *logic rules*. We would illustrate the connection between kernels that embed hypotheses and kernels that embed logic rules. Then, we shall work on combined logic rules as well as combined hypotheses, and would show how to derive kernels that embed combined hypotheses.

4.3.1 Logic Kernels

For a given hypothesis $h\colon \mathcal{X} \to \{-1, +1\}$, we define its associated logic rule $\tilde{h}$ to be

$$\tilde{h}(x) = 1 \Leftrightarrow h(x) = +1.$$

Arithmetically, we can easily derive the relationship between $h(\cdot)$ and $\tilde{h}(\cdot)$ as

$$\tilde{h}(x) = \frac{h(x) + 1}{2} \tag{4.4}$$

for all $x \in \mathcal{X}$.

Consider $\mathcal{H} = \{h_\alpha\colon \alpha \in \mathcal{C}\}$, and its associated set of logic rules $\tilde{\mathcal{H}} = \left\{\tilde{h}_\alpha\colon \alpha \in \mathcal{C}\right\}$. The following lemma reveals the relationship between $K_{\mathcal{H},r}$ and $K_{\tilde{\mathcal{H}},r}$.

Lemma 3 *We call a learning model $\mathcal{H} = \{h_\alpha\colon \alpha \in \mathcal{C}\}$ neutral to $\mathcal{X}$ with a given (r, μ) if for all $x \in \mathcal{X}$*

$$\int_{\alpha\in\mathcal{C}} h_\alpha(x) \left(r(\alpha)\right)^2 \, d\mu(\alpha) = 0.$$

Assume that for a neutral $\mathcal{H}$,

$$\Delta = \int_{\alpha\in\mathcal{C}} \left(r(\alpha)\right)^2 \mu(\alpha)$$

exists. Then,

$$\mathcal{K}_{\tilde{\mathcal{H}},r}(x,x') = \frac{1}{4}\mathcal{K}_{\mathcal{H},r}(x,x') + \frac{1}{4}\Delta.$$

for all $x, x' \in \mathcal{X}$.

Proof. We have

$$\begin{aligned}
K_{\tilde{\mathcal{H}},r}(x,x') &= \int_{\alpha\in\mathcal{C}} (r(\alpha))^2\, \tilde{h}_\alpha(x)\tilde{h}_\alpha(x')\, d\mu(\alpha) \\
&= \int_{\alpha\in\mathcal{C}} (r(\alpha))^2 \left(\frac{h_\alpha(x)+1}{2}\right)\left(\frac{h_\alpha(x')+1}{2}\right) d\mu(\alpha) \\
&= \int_{\alpha\in\mathcal{C}} (r(\alpha))^2 \left(\frac{h_\alpha(x)h_\alpha(x')+h_\alpha(x)+h_\alpha(x')+1}{4}\right) d\mu(\alpha) \\
&= \frac{1}{4}K_{\mathcal{H},r}(x,x') + \frac{1}{4}\Delta.
\end{aligned}$$

□

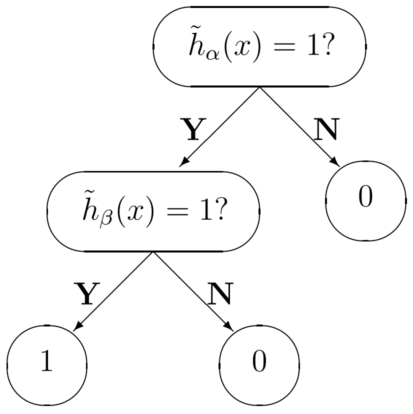
The kernel $K_{\tilde{\mathcal{H}}}$ would be called the *logic kernel* of $\mathcal{H}$. Note that for a negation complete learning model $\mathcal{H}$, neutrality is usually a mild assumption with a suitable parameterization and selection of (r, μ). The decision stump model and the perceptron model that we have used are both neutral with the (r, μ) in Definition 4 and Definition 5, respectively. This relationship allows us to work with hypotheses and logic rules interchangeably. Next, we shall see the benefit for working with logic rules.

4.3.2 Multiplication of Logic Kernels

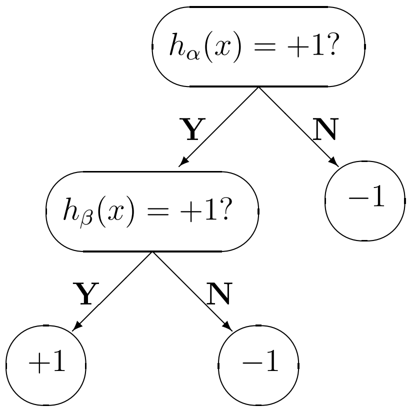
Consider two logic rules $\tilde{h}_\alpha(\cdot)$ and $\tilde{h}_\beta(\cdot)$. We can construct a logic rule $\tilde{h}_{\alpha,\beta}(\cdot)$ such that

$$\tilde{h}_{\alpha,\beta}(x) \;=\; \tilde{h}_\alpha(x) \wedge \tilde{h}_\beta(x).$$

for all $x \in \mathcal{X}$. This is called the *AND combination* of the logic rules. We illustrate the AND combination, both for the logic rules and for the hypotheses, in Figure 4.4. The leaf node in the tree structure represents the logic value TRUE if and only if



(a) AND combination of logic rules



(b) AND combination of hypotheses

Figure 4.4: Illustration of AND combination

both logic rules output TRUE. For example, when the first logic rule is associated with a decision stump of whether $(x)_1 \leq 2$, and the second logic rule is associated with a decision stump of whether $(x)_2 \geq 1$. The AND combination of these two rules would be whether $(x)_1 \leq 1$ and $(x)_2 \geq 1$. Note that for logic rules, the $\wedge$ operation can be performed simply with multiplication

$$\tilde{h}_{\alpha,\beta}(x) \quad = \quad \tilde{h}_\alpha(x)\tilde{h}_\beta(x).$$

This simple arithmetic property is the reason that we work on logic rules instead of hypotheses.

For two learning models $\mathcal{H}_1 = \{h_\alpha \colon \alpha \in \mathcal{C}_1\}$ and $\mathcal{H}_2 = \{h_\beta \colon \beta \in \mathcal{C}_2\}$, consider their associated set of logic rules $\tilde{\mathcal{H}}_1$ and $\tilde{\mathcal{H}}_2$, we can define the AND combination of them as

$$\begin{aligned} \tilde{\mathcal{H}} &= \text{AND}\Big(\tilde{\mathcal{H}}_1, \tilde{\mathcal{H}}_2\Big) \\ &= \Big\{\tilde{h}_{\alpha,\beta} \colon \tilde{h}_{\alpha,\beta}(x) = \tilde{h}_\alpha(x) \wedge \tilde{h}_\beta(x), \alpha \in \mathcal{C}_1, \beta \in \mathcal{C}_2\Big\}. \end{aligned}$$

Then we get the following definition for a kernel embedding the combined logic rules.

Definition 6 *For two sets of logic rules $\tilde{\mathcal{H}}_1$ and $\tilde{\mathcal{H}}_2$, let*

$$\tilde{\mathcal{H}} = \text{AND}\Big(\tilde{\mathcal{H}}_1, \tilde{\mathcal{H}}_2\Big).$$

Then, by constructing $r(\alpha, \beta) = r(\alpha)r(\beta)$ and μ as a natural extension of the original measures,

$$\mathcal{K}_{\tilde{\mathcal{H}}}(x, x') = \mathcal{K}_{\tilde{\mathcal{H}}_1}(x, x') \cdot \mathcal{K}_{\tilde{\mathcal{H}}_2}(x, x').$$

for all $x, x' \in \mathcal{X}$.

Proof. The reason is that each $\tilde{h}_{\alpha,\beta}$ is a simple multiplication of $\tilde{h}_\alpha$ and $\tilde{h}_\beta$. □

Multiplication of kernels is a common technique for constructing new kernels for SVM (Schölkopf and Smola 2002). The specific property of logic rules allows us to interpret multiplication of logic kernels as AND combination of logic rules. Then, through the relationship between logic rules and hypotheses, we could interpret the multiplication of kernels as combination of hypotheses. We formalize this idea by the following theorem.

Theorem 14 *For two learning models $\mathcal{H}_1 = \{h_\alpha \colon \alpha \in \mathcal{C}_1\}$ and $\mathcal{H}_2 = \{h_\beta \colon \beta \in \mathcal{C}_2\}$, consider their associated set of logic rules $\tilde{\mathcal{H}}_1$ and $\tilde{\mathcal{H}}_2$. Define*

$$\tilde{\mathcal{H}} = \text{AND}\Big(\tilde{\mathcal{H}}_1, \tilde{\mathcal{H}}_2\Big)$$

and $\mathcal{H}$ as the associated learning model for $\tilde{\mathcal{H}}$. Assume that for some choice of (r_1, μ_1), $\mathcal{K}_{\mathcal{H}_1}$ exists and $\mathcal{H}_1$ is neutral to $\mathcal{X} \subseteq \mathbb{R}^D$. Similarly assume such property for $\mathcal{H}_2$. In addition, assume that both

$$\Delta_1 = \int_{\alpha \in \mathcal{C}_1} (r_1(\alpha))^2 \, d\mu(\alpha)$$
$$\Delta_2 = \int_{\beta \in \mathcal{C}_2} (r_2(\beta))^2 \, d\mu(\beta)$$

exist. Then, there exists some $r(\alpha, \beta)$ *and* Δ *such that*

$$\mathcal{K}_{\mathcal{H},r}(x,x') = \frac{1}{4}\left(\mathcal{K}_{\mathcal{H}_1,r_1}(x,x') + \Delta_1\right) \cdot \left(\mathcal{K}_{\mathcal{H}_2,r_2}(x,x') + \Delta_2\right) - \Delta \tag{4.5}$$

for all $x, x' \in \mathcal{X}$*.*

Proof. Consider the operation on associated logic kernels, from Definition 6, we get

$$\frac{1}{4}K_{\mathcal{H},r}(x,x') + \frac{1}{4}\Delta \quad = \quad \left(\frac{1}{4}K_{\mathcal{H}_1,r_1}(x,x') + \frac{1}{4}\Delta_1\right) \cdot \left(\frac{1}{4}K_{\mathcal{H}_2,r_2}(x,x') + \frac{1}{4}\Delta_2\right)$$

with $r(\alpha, \beta) = r_1(\alpha)r_2(\beta)$. The equation is exactly the same as (4.5), and we could compute Δ from

$$\Delta = \int_{\alpha\in\mathcal{C}_1,\beta\in\mathcal{C}_2} \left(r_1(\alpha)r_2(\beta)\right)^2 \, d\mu(\alpha,\beta) = \Delta_1\Delta_2.$$

□

However, note the $\mathcal{H}$ may not be negation complete even if $\mathcal{H}_1$ and $\mathcal{H}_2$ are, because the AND combination "favors" the logic value FALSE. Thus, we would consider $\hat{\mathcal{H}}$, the negation complete closure of $\mathcal{H}$, when we want to apply the kernel to Algorithm 3.

Corollary 3 *Let*

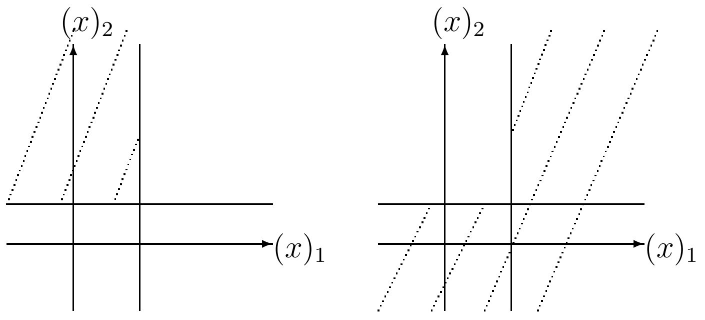
$$\hat{\mathcal{H}} = \left\{h_{q,\alpha,\beta} \colon h_{q,\alpha,\beta}(x) = q \cdot h_{\alpha,\beta}(x), q \in \{-1,+1\}, h_{\alpha,\beta} \in \mathcal{H}\right\},$$

where $\mathcal{H}$ *is constructed from Theorem 14. Then, let* $\hat{r}(q,\alpha,\beta) = \frac{1}{\sqrt{2}}r(\alpha,\beta)$ *for* $q \in \{-1,+1\}$*, we get*

$$\mathcal{K}_{\hat{\mathcal{H}},\hat{r}}(x,x') = \frac{1}{4}\left(\mathcal{K}_{\mathcal{H}_1,r_1}(x,x') + \Delta_1\right) \cdot \left(\mathcal{K}_{\mathcal{H}_2,r_2}(x,x') + \Delta_2\right) - \Delta.$$

In addition, $\hat{\mathcal{H}}$ *is neutral.*

Proof. The corollary is a simple extension of Theorem 14. The neutrality is because with the direction parameter q, every prediction has an unique negated prediction in $\hat{\mathcal{H}}$. □



(a) AND combination (b) NAND combination

Figure 4.5: Combining $s_{-1,1,2}(x)$ and $s_{+1,2,1}(x)$

The learning model $\hat{\mathcal{H}}$ contains the AND combinations of the hypotheses in $\mathcal{H}_1$ and $\mathcal{H}_2$, and the negation, which is called *NAND combination*, of them. For example, $\mathcal{H}_1 = \mathcal{H}_2 = \mathcal{S}$, which include the decision stumps of whether $(x)_1 \leq 2$ and whether $(x)_2 \geq 1$. The combined rule would contain whether $(x)_1 \leq 2$ and $(x)_2 \geq 1$, which represents some left-top corner of the two dimensional space, and whether $(x)_1 > 2$ or $(x)_2 < 1$, which represents the union of the other three corners. We illustrate this in Figure 4.5. That is, $\hat{\mathcal{H}}$ represents a decision region of at most two stump boundaries. Because $\hat{\mathcal{H}}$ is neutral, we could combine it with $\mathcal{S}$ again, and obtain a kernel that embeds decision regions of at most three stump boundaries. In the next section, we would show how we could extend this process to infinity.

4.3.3 Laplacian Kernel and Exponential Kernel

We have shown that we could combine two decision stumps to form a decision region with two stump boundaries, and embed the set of possible decision regions into a kernel. We now extend this procedure level by level. We first define the L-level stump region kernel $\mathcal{K}_{\mathcal{T}_L}$ as follows.

Definition 7 *The L-level stump region kernel $\mathcal{K}_{\mathcal{T}_L}$ is recursively defined with the* *following equations.*

$$\begin{aligned}\mathcal{K}_{\mathcal{T}_1} &= \mathcal{K}_{\mathcal{S}} \\ \Delta_1 &= \frac{1}{2}\sum_{d=1}^{D}(R_d - L_d) = \Delta_{\mathcal{S}}\end{aligned}$$

For $L \in \mathbb{N}$ and $L > 1$,

$$\begin{aligned}\mathcal{K}_{\mathcal{T}_L}(x,x') &= \frac{1}{4}\left(\mathcal{K}_{\mathcal{T}_{L-1}}(x,x') + \Delta_{L-1}\right)\left(\mathcal{K}_{\mathcal{T}_1}(x,x') + \Delta_1\right) - \Delta_L. \\ \Delta_L &= \Delta_{L-1}\Delta_1 = \Delta_1^L.\end{aligned}$$

Using Corollary 3, we see that an L-level stump region kernel embeds the decision region classifiers with at most L stump boundaries. Note that we can solve the recursion and get

$$\mathcal{K}_{\mathcal{T}_L}(x,x') = \left(\frac{1}{4}\right)^{L-1}\left(\mathcal{K}_{\mathcal{S}}(x,x') + \Delta_{\mathcal{S}}\right)^L - \Delta_{\mathcal{S}}^L.$$

Then, we could use Theorem 6 to construct a kernel $\mathcal{K}_{\mathcal{T}}$ that embodies all possible decision region classifiers with stump boundaries. Such kernel can be applied to Algorithm 3 to obtain an infinite ensemble classifier over those decision region classifiers.

Theorem 15 *The infinite stump tree kernel*

$$\mathcal{K}_{\mathcal{T}}(x,x') = 4\exp\left(\frac{1}{4}\mathcal{K}_{\mathcal{S}}(x,x') + \frac{1}{4}\Delta_{\mathcal{S}}\right) - \exp\left(\Delta_{\mathcal{S}}\right) - 3$$

can be applied to Algorithm 3 to obtain an ensemble classifier over decision region classifiers with any number of stump boundaries.

Proof. By Taylor's series expansion, we have

$$\begin{aligned}
\mathcal{K}_{\mathcal{T}}(x,x') &= 4\exp\left(\frac{1}{4}\mathcal{K}_{\mathcal{S}}(x,x')+\frac{1}{4}\Delta_{\mathcal{S}}\right)-\exp\left(\Delta_{\mathcal{S}}\right)-3\\
&= 4\sum_{L=0}^{\infty}\frac{1}{L!}\left(\frac{1}{4}\mathcal{K}_{\mathcal{S}}(x,x')+\frac{1}{4}\Delta_{\mathcal{S}}\right)^{L}-\sum_{L=0}^{\infty}\frac{1}{L!}\Delta_{\mathcal{S}}^{L}\left(\Delta_{\mathcal{S}}\right)-3\\
&= \sum_{L=1}^{\infty}\frac{1}{L!}\left(4\left(\frac{1}{4}\mathcal{K}_{\mathcal{S}}(x,x')+\frac{1}{4}\Delta_{\mathcal{S}}\right)^{L}-\Delta_{\mathcal{S}}^{L}\left(\Delta_{\mathcal{S}}\right)\right)\\
&= \sum_{L=1}^{\infty}\frac{1}{L!}\mathcal{K}_{\mathcal{T}_L}(x,x').
\end{aligned}$$

With Theorem 6, $K_{\mathcal{T}}$ embeds the learning models $\mathcal{T}_L$ for $L=1,\cdots,\infty$, where the $\frac{1}{L!}$ could be performed by scaling the embedding function r for $\mathcal{K}_{\mathcal{T}_L}$. □

Note that the NAND operator is universal. Thus, for any classifier that could be described by finite number of stump boundaries, there is an equivalent decision region classifier in some $\mathcal{T}_L$. An equivalent learning model to $\bigcup_{L=1}^{\infty}\mathcal{T}_L$ is the *decision tree* model, in which the classifiers are similar to the one in Figure 4.4, but with arbitrary tree structure. In other words, the kernel $\mathcal{K}_{\mathcal{T}}$ embodies an infinite number of decision trees. Decision trees are popular for ensemble learning, and some theoretical analysis of them are based on trees of infinite level (Breiman 2000). However, traditional algorithms can only deal with trees with finite levels (Breiman 1998; Dietterich 2000). On the other hand, our framework allows us to actually build an ensemble over decision trees with arbitrary levels.

The infinite stump tree kernel $\mathcal{K}_{\mathcal{T}}(x,x')$ is of the form

$$A_1\exp\left(-A_2\left\|x-x'\right\|_1\right)+A_3,$$

where A_1, A_2, A_3 are constants and A_1, A_2 are positive. The parameter A_2 is similar to the scaling parameter γ in the Laplacian kernel $\exp\left(-\gamma\left\|x-x'\right\|_1\right)$, and is mainly dependent to the embedding function r. Because of the linear constraint $\sum_{i=1}^{N}y_i\lambda_i=0$, adding any constant A_3 to the kernel does not affect the solution of (P_4) and the decision function (2.6). In addition, scaling the kernel with A_1 is equivalent to scaling the soft-margin parameter C in SVM. Thus, when C is ∞ or when we perform suitable parameter selection, A_1 does not affect the decision function obtained from (2.6), either. Therefore, the infinite tree kernel is equivalent to the Laplacian kernel. This reveals a novel interpretation of Laplacian kernel: SVM with the Laplacian kernel allows us to obtain an infinite ensemble classifier over decision trees of any level.

Similarly, we could use the trick in Theorem 15 to show that the exponential kernel $\exp\left(-\gamma \left\|x - x'\right\|_2\right)$ embeds infinite number of decision regions with perceptron (hyperplane) boundaries. Both Laplacian kernel and exponential kernel are RBF kernels, and have a PD Gram matrix when all training vectors x_i are distinct (Micchelli 1986; Baxter 1991). Next, we further discuss the application of these RBF kernels as well as the popular Gaussian kernel.

4.3.4 Discussion on RBF Kernels

We have shown that the stump kernel, the perceptron kernel, the Laplacian kernel, the exponential kernel, and the Gaussian kernel are all RBF kernels. Next, we compare two properties of these kernels, and discuss their use in applications of SVM.

First, we can group these kernels by the distance metrics they use. The stump kernel and the Laplacian kernel uses the one-norm distance between vectors, while the others uses the two-norm distance. An advantage of using the two-norm distance is that the distance is invariant to rotations. This is useful to several applications, including the Optical Character Recognition (OCR) problem, because images of written characters are (locally) rotation invariant under suitable vector representations. Some applications, however, may not desire rotation invariance. For example, when representing an image with color histograms, rotation invariance might mix up the information that are contained in each color component.

From the construction of the perceptron kernel (and the exponential kernel), we can see how the rotation invariance is obtained from an ensemble point-of-view. The transformation vectors θ in perceptrons represent the rotation, and rotation invariance comes from embedding all possible θ in the kernel.

Second, we can group kernels by whether they are linear to positive scaling. The stump kernel and the perceptron kernel are linear to positive scaling. That is, they satisfy $\mathcal{K}(\gamma x, \gamma x') = \gamma \mathcal{K}(x, x')$ for positive γ. Note that for SVM, a positive linear scaling to the kernel is equivalent to scaling the soft-margin parameter C. Thus, we do not need a specific scaling parameter γ in the stump kernel or the perceptron kernel; we only need to focus on the value of C. Compared to other kernels such as Gaussian, where the different combinations of (γ, C) must be considered during parameter selection (Keerthi and Lin 2003), SVM with the stump kernel or the perceptron kernel has an advantage of faster parameter selection. Nevertheless, from Corollary 1 and Corollary 2, they theoretically still have almost the same classification power as SVM with the Gaussian kernel. Thus, SVM applications that consider speed as an important factor may benefit from using the stump kernel or the perceptron kernel.

Chapter 5

Experiments

Next, we present experimental results of our framework. The experiments would demonstrate two important advantages of our framework. First, the framework allows a fair comparison between SVM and ensemble learning. We would compare SVM with traditional ensemble learning algorithms using decision stumps as the base learning model. The comparison illustrates the differences between our framework and traditional algorithms. In addition, we would show that infinite ensemble learning without sparse representation could achieve better performance.

Second, the framework is useful for constructing new kernels and interpreting existing kernels in SVM. We would show that the kernels derived in Chapter 4 have comparable performance to the popular Gaussian kernel, yet some of them could benefit from faster parameter selection.

5.1 Setup

We would use three artificial datasets: twonorm, threenorm, and ringnorm, which are described by Breiman (1998). We follow the procedure of Breiman (1999) to randomly generate a training set of size 300 and a test set of size 3000. The results are averaged over 100 different random runs.

In addition to artificial datasets, we would use nine real-world datasets: australian, breast, cleveland, diabetes, german, heart, ionosphere, sonar, and vote84. They are taken from the UCI Repository (Blake and Merz 1998). We clean each dataset by

Table 5.1: Details of the datasets

dataset	full name	number of examples	number of features
twonorm	Twonorm	-	20
threenorm	Threenorm	-	20
ringnorm	Ringnorm	-	20
australian	Australian credit approval	690	14
breast	Wisconsin breast cancer	683	10
cleveland	Cleveland heart disease	297	13
diabetes	Pima Indians diabetes	768	8
german	German credit	1000	24
heart	Statlog heart disease	270	13
ionosphere	Ionosphere	351	34
sonar	Sonar	208	60
vote84	Congressional voting records	435	16

removing the examples that contain missing feature elements. We then normalize each feature element to $[-1, 1]$. The details of the cleaned datasets are shown in Table 5.1. We randomly use 60% of the examples for training, and the rest for testing. The results are also averaged over 100 different random runs.

In Section 5.2, we would compare several ensemble learning algorithms that use decision stumps as the base learning model. The first one is called SVM-Stump, which is Algorithm 3 with the stump kernel and soft-margin SVM. The second one is AdaBoost-Stump, which is Algorithm 1 with the set of decision stumps as the base learning model. We follow a common implementation for the base learning algorithm $\mathcal{A}_{\mathcal{S}}$, which only picks the middle stumps in $\mathcal{S}$. In other words, the algorithm $\mathcal{A}_{\mathcal{S}}$ returns either a constant classifier or a decision stump with threshold α_d at some value (see Theorem 11 for notations)

$$\frac{(\tilde{x})_{d,a} + (\tilde{x})_{d,a+1}}{2}.$$

We also implement a variant of Algorithm 2 with the hard-margin setting. We call this algorithm LPBoost-Stump, which also only considers middle stumps, and solves (P_7) exactly with $C = \infty$.

For AdaBoost-Stump, we demonstrate the results using $T = 100$ and $T = 1000$. For SVM, we use the suggested procedure for soft-margin SVM (Hsu et al. 2003). We first conduct parameter selection with 5-fold cross validation of the training examples on $\log_2(C) \in \{-17, -15, \cdots, 3\}$, and use the best C value for actual training.

In Section 5.3, we would compare different RBF kernels for soft-margin SVM. We name them SVM-Stump, SVM-Perceptron, SVM-Laplace, SVM-Exponential, and SVM-Gauss. For SVM-Perceptron, we conduct parameter selection with the same setting as SVM-Stump. For SVM-Laplace, SVM-Exp, and SVM-Gauss, we conduct parameter selection on $\log_2(C) \in \{-5, -3, \cdots, 15\}$ and $\log_2(\gamma) \in \{-15, -13, \cdots, 3\}$. We use different candidate C values because the numerical ranges of the stump kernel and the Gaussian kernel could be quite different.

We apply LIBSVM 2.8 with all its default settings as our SVM solver (Chang and Lin 2001a). The AdaBoost algorithm comes from LEMGA (Li 2001), and the linear programming solver for LPBoost comes from GNU Linear Programming Kit.

5.2 Comparison of Ensemble Learning Algorithms

5.2.1 Experimental Results

Table 5.2 shows the comparison between our framework (SVM-Stump) and traditional ensemble learning algorithms. The results are shown with error bars computed by standard error. We use bold font to indicate those results that are as significant as the best one. We can see that SVM-Stump performs better than AdaBoost-Stump and LPBoost-Stump. In addition, LPBoost-stump performs much worse than AdaBoost-stump, which corresponds to past findings of Breiman (1999) and Li et al. (2003) that aggressively maximizing the ℓ_1-margin may not improve generalization performance. Next, we further analyze these observations.

Table 5.2: Test error (%) comparison of ensemble learning algorithms

dataset	SVM-stump C by cross validation	AdaBoost-stump $T = 100$	AdaBoost-stump $T = 1000$	LPBoost-stump $C = \infty$
twonorm	$\mathbf{2.86 \pm 0.04}$	5.06 ± 0.06	4.97 ± 0.06	5.54 ± 0.68
threenorm	$\mathbf{17.7 \pm 0.10}$	21.8 ± 0.09	22.9 ± 0.12	24.1 ± 0.14
ringnorm	$\mathbf{3.97 \pm 0.07}$	12.2 ± 0.13	9.95 ± 0.14	11.9 ± 0.15
australian	$\mathbf{14.5 \pm 0.21}$	$\mathbf{14.7 \pm 0.18}$	16.9 ± 0.18	19.8 ± 0.23
breast	$\mathbf{3.11 \pm 0.08}$	4.27 ± 0.11	4.51 ± 0.11	4.82 ± 0.12
cleveland	$\mathbf{17.6 \pm 0.21}$	19.7 ± 0.30	22.5 ± 0.35	24.4 ± 0.34
diabetes	$\mathbf{24.2 \pm 0.23}$	24.8 ± 0.22	27.0 ± 0.25	31.4 ± 0.23
german	$\mathbf{24.7 \pm 0.18}$	$\mathbf{25.0 \pm 0.18}$	26.9 ± 0.18	32.0 ± 0.22
heart	$\mathbf{16.4 \pm 0.27}$	19.9 ± 0.36	22.6 ± 0.39	24.3 ± 0.38
ionosphere	$\mathbf{8.13 \pm 0.17}$	11.0 ± 0.23	11.0 ± 0.25	11.6 ± 0.23
sonar	$\mathbf{16.6 \pm 0.42}$	19.0 ± 0.37	19.0 ± 0.35	19.7 ± 0.36
vote84	4.76 ± 0.14	$\mathbf{4.07 \pm 0.14}$	5.29 ± 0.15	5.88 ± 0.16

(those that are as significant as the best result are marked in bold)

5.2.2 Regularization and Sparsity

Table 5.3 adds hard-margin SVM into comparison for the artificial datasets. That is, we present the result for $C = \infty$ in SVM-Stump in the second column. First, we can see that the hard-margin SVM-Stump performs worse than the soft-margin one with parameter selection. We have illustrated this phenomena as a consequence of overfitting in Section 4.1. For LPBoost-Stump and AdaBoost-Stump, we have also mentioned that the hard-margin solution (LPBoost) performs worse than the regularized solution (AdaBoost). These results demonstrate the importance of further regularization rather than relying only on the large-margin concept.

The hard-margin SVM-Stump and hard-margin LPBoost-Stump only differ in the objective functions that they try to minimize. Thus, it is meaningful to compare their relative performance. From Table 5.3 we can see that hard-margin LPBoost-Stump performs worse than hard-margin SVM-Stump. Their regularized versions, soft-margin SVM-Stump and AdaBoost-Stump, also inherits this difference. One possible explanation is that the assumption for sparse representation degrades the performance of the ensemble classifier found. We further illustrate this by a sim-

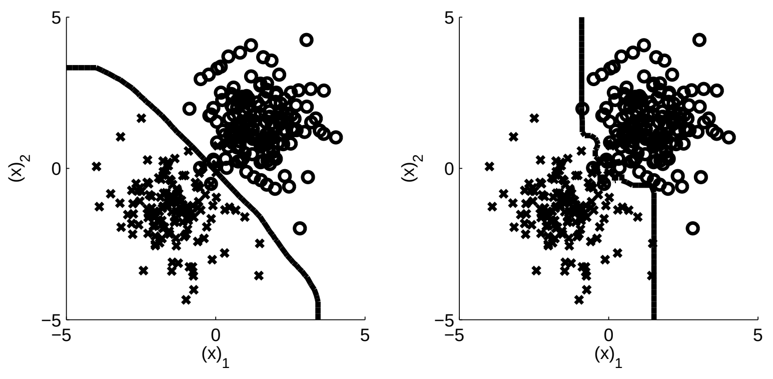


Figure 5.1: Decision boundaries of SVM-Stump (left) and AdaBoost-Stump (right) on a 2-D twonorm dataset

plified experiment. In Figure 5.1 we show the decision boundaries of soft-margin SVM-Stump and AdaBoost-Stump on a training set of size 300 generated from the two-dimensional version of the twonorm dataset. The Bayes optimal decision boundary is the line $(x)_1 + (x)_2 = 0$. We can see that SVM-Stump produces a decision boundary that is close to the optimal, but AdaBoost-Stump fails to do so. The smooth decision boundary of SVM-Stump means that many decision stumps are included the ensemble. On the other hand, the perpendicular decision boundary of AdaBoost-Stump indicates that the algorithm only considers a small finite number of decision stumps in the ensemble.

Although sparsity is often considered beneficial in learning paradigms like Occam's razor, sparse ensemble classifier does not always perform well. In our case, because the decision stumps are very simple, general dataset would require many of them to describe a suitable decision boundary. Thus, traditional ensemble learning algorithms, such as AdaBoost and LPBoost, may suffer from the assumption of sparse representation, regardless of the number iterations T when forming the ensemble. On the other hand, our framework, which does not have the sparse representation assumption, would gain an advantage in this situation.

Table 5.3: Test error (%) comparison on sparsity and regularization

dataset	SVM-stump C by cross validation	SVM-stump $C = \infty$	AdaBoost-stump $T = 1000$	LPBoost-stump $C = \infty$
twonorm	$\mathbf{2.86 \pm 0.04}$	4.02 ± 0.06	4.97 ± 0.06	5.54 ± 0.68
threenorm	$\mathbf{17.7 \pm 0.10}$	22.0 ± 0.13	22.9 ± 0.12	24.1 ± 0.14
ringnorm	$\mathbf{3.97 \pm 0.07}$	4.36 ± 0.06	9.95 ± 0.14	11.9 ± 0.15

(those that are as significant as the best result are marked in bold)

5.3 Comparison of RBF Kernels

Table 5.4 shows the test error comparison among the four RBF kernels derived in Chapter 4 the Gaussian kernel. This table confirms several discussions at the end of Section 4.3. First, the choice of which kernel is the best is application dependent. The distance metric used seems to play an important role. In most of the datasets, however, the performance of all of them are comparable. Even the simplest instance, SVM-Stump, could sometimes have superior performance.

When the performances of those kernels are comparable, the amount of training time spent becomes important. Recall that for parameter selection, we would solve 11 optimization problems (P_4) for SVM-Stump and SVM-Perceptron, but we solve 110 problems for SVM with other kernels. Table 5.5 shows the average amount of CPU time spent for parameter selection for each kernel. We compute the actual CPU time with the same machine for each SVM kernel, which is of dual 1.7 GHz Intel Xeon CPU running Fedora Linux Core 2. Note that the time for optimization is dependent on the condition number of the Gram matrix (which depends on the training set and the kernel), and the soft-margin parameter C in (P_4). Thus, it is difficult to choose a fair range of C for different kinds of kernels, and we should not compare the numbers in Table 5.5 quantitatively. However, qualitatively, we can see that SVM-Stump and SVM-Perceptron are indeed more efficient in parameter selection, which makes them favorable choices when the size of dataset is large or when time is an important concern in the application.

Table 5.4: Test error (%) comparison of RBF kernels

dataset	SVM-Stump	SVM-Perc.	SVM-Lapl.	SVM-Expo.	SVM-Gauss
twonorm	2.86 ± 0.04	**2.55 ± 0.03**	2.87 ± 0.04	**2.58 ± 0.04**	2.64 ± 0.05
threenorm	17.7 ± 0.10	**14.6 ± 0.08**	15.0 ± 0.11	14.0 ± 0.10	**14.6 ± 0.11**
ringnorm	3.97 ± 0.07	2.46 ± 0.04	2.25 ± 0.05	2.07 ± 0.04	**1.78 ± 0.04**
australian	**14.5 ± 0.21**	**14.5 ± 0.17**	**14.3 ± 0.18**	14.7 ± 0.16	14.7 ± 0.18
breast	**3.11 ± 0.08**	3.23 ± 0.08	**3.18 ± 0.08**	3.31 ± 0.09	3.53 ± 0.09
cleveland	**17.6 ± 0.21**	18.2 ± 0.31	18.2 ± 0.34	18.3 ± 0.30	18.0 ± 0.31
diabetes	24.2 ± 0.23	**23.6 ± 0.21**	24.0 ± 0.24	**23.3 ± 0.22**	**23.5 ± 0.19**
german	**24.7 ± 0.18**	**24.6 ± 0.19**	**24.9 ± 0.20**	**24.8 ± 0.19**	**24.5 ± 0.21**
heart	**16.4 ± 0.27**	17.6 ± 0.31	**16.8 ± 0.31**	18.0 ± 0.30	17.5 ± 0.31
ionosphere	8.13 ± 0.17	6.40 ± 0.20	6.48 ± 0.19	**5.49 ± 0.21**	6.54 ± 0.19
sonar	16.6 ± 0.42	15.6 ± 0.40	**14.7 ± 0.42**	**15.0 ± 0.37**	**15.5 ± 0.50**
vote84	4.76 ± 0.14	**4.43 ± 0.14**	**4.59 ± 0.15**	**4.36 ± 0.14**	**4.62 ± 0.14**

(those that are as significant as the best result are marked in bold)

Table 5.5: Parameter selection time (sec.) comparison of RBF kernels

dataset	SVM-Stump	SVM-Perc.	SVM-Lapl.	SVM-Expo.	SVM-Gauss
twonorm	1.34	1.44	19.5	17.9	23.1
threenorm	1.69	1.69	23.1	21.8	31.1
ringnorm	1.50	1.60	23.7	23.8	27.9
australian	2.04	2.01	32.0	31.0	43.3
breast	1.18	1.30	16.6	16.6	16.6
cleveland	0.95	0.96	8.53	7.91	10.0
diabetes	1.95	1.92	37.0	34.9	95.2
german	6.95	4.73	120	81.2	136
heart	0.90	0.88	7.42	6.86	8.57
ionosphere	1.21	1.27	13.1	12.1	12.6
sonar	1.14	1.16	9.06	8.86	9.30
vote84	1.02	1.10	12.4	11.5	13.8

Chapter 6

Conclusion

In this thesis, we have proposed a general framework to construct infinite ensemble classifiers with SVM. Our framework conquers the theoretical challenge of infinite ensemble learning without the assumption on sparse representation. It also inherits the profound theoretical and practical advantages of SVM. We have applied our framework to several different base learning models, which not only generates two new kernels for SVM, but also gives novel interpretations to two existing kernels from an ensemble point-of-view.

The framework also allows a fair comparison between SVM and ensemble learning algorithms. We have compared their performances empirically on both artificial and real-world datasets using the same base learning model. Experimental results show that our framework could achieve significantly better performance. We have analyzed the causes of the difference, and find that our framework benefits by suitable regularization with soft-margin SVM, and by dropping the assumption of sparse representation. The study provides more understanding for both SVM and ensemble learning algorithms.

The kernels that we have derived from the framework are useful for SVM. We find that those kernels could have comparable performance to the popular Gaussian kernel for SVM. The stump kernel and the perceptron kernel can further benefit from faster parameter selection, which makes both kernels favorable to the Gaussian kernel in the case of large datasets.

Further studies of the framework would include deriving kernels using some other base learning models. Note that all the kernels constructed in this thesis are all within the RBF family. We expect to interpret some other RBF kernels with techniques in Section 4.3. We also hope to see whether other popular kernels for SVM could be interpreted from an ensemble point-of-view. Another possible direction is to study how the parameterization (α) and embedding (γ) affect the geometry of the feature space, which may provide more understanding for designing SVM kernels.

Bibliography

Abu-Mostafa, Y. S. (1989). The Vapnik-Chervonenkis dimension: Information versus complexity in learning. *Neural Computation 1*, 312–317.

Abu-Mostafa, Y. S., X. Song, A. Nicholson, and M. Magdon-Ismail (2004). The bin model. Technical Report CaltechCSTR:2004.002, California Institute of Technology.

Bauer, E. and R. Kohavi (1999). An empirical comparison of voting classification algorithms: Bagging, boosting, and variants. *Machine Learning* (36), 105–142.

Baum, E. B. and D. Haussler (1989). What size net gives valid generalization. *Neural Computation 1*(1), 151–160.

Baxter, B. J. C. (1991). Conditionally positive functions and p-norm distance matrices. *Constructive Approximation 7*, 427–440.

Beal, M. J., Z. Ghahramani, and C. E. Rasmussen (2003). The infinite Hidden Markov Model. In T. Dietterich, S. Becker, and Z. Ghahramani (Eds.), *Advances in Neural Information Processing Systems*, Volume 14, pp. 577–584. MIT Press.

Berg, C., J. P. R. Christensen, and P. Ressel (1984). *Harmonic Analysis on Semigroups*. Springer-Verlag.

Blake, C. and C. Merz (1998). UCI repository of machine learning databases.

Blumer, A., A. EhrenFeucht, D. Haussler, and M. K. Warmuth (1989). Learnability and the Vapnik-Chervonenkis dimension. *Journal of the Association for Computing Machinery 36*(4), 929–965.

Bousquet, O. (2003). New approaches to statistical learning theory. *Annals of the Institute of Statistical Mathematics 55*(2), 371–389.

Bousquet, O. and A. Elisseeff (2002). Stability and generalization. *Journal of Machine Learning Research 2*, 499–526.

Breiman, L. (1996). Bagging predictors. *Machine Learning 24*(2), 123–140.

Breiman, L. (1998). Arcing classifiers. *Annals of Statistics 26*(3), 801–824.

Breiman, L. (1999). Prediction games and arcing algorithms. *Neural Computation* (11), 1493–1517.

Breiman, L. (2000). Some infinity theory for predictor ensembles. Technical report. Technical Report 577, Statistics Department, University of California, Berkeley.

Chang, C.-C. and C.-J. Lin (2001a). *LIBSVM: a library for support vector machines*. Software available at `http://www.csie.ntu.edu.tw/~cjlin/libsvm`.

Chang, C.-C. and C.-J. Lin (2001b). Training ν-support vector classifiers: Theory and algorithms. *Neural Computation 13*(9), 2119–2147.

Cover, T. M. (1965). Geometrical and statistical properties of systems of linear inequalities with applications in pattern recognition. *IEEE Transactions on Electronic Computers 14*, 326–334.

Demiriz, A., K. P. Bennett, and J. Shawe-Taylor (2002). Linear programming boosting via column generation. *Machine Learning 46*(1-3), 225–254.

Dietterich, T. G. (2000). An experimental comparison of three methods for constructing ensembles of decision trees: Bagging, boosting, and randomization. *Machine Learning* (40), 139–157.

Freund, Y. and R. E. Schapire (1996). Experiments with a new boosting algorithm. In *Machine Learning: Proceedings of the 13th International Conference.*

Freund, Y. and R. E. Schapire (1997). A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences 55*(1), 119–139.

Freund, Y. and R. E. Schapire (1999). A short introduction to boosting. *Journal of Japanese Society for Artificial Intelligence 14*(5), 771–780. English version downloadable in `http://boosting.org`.

Haykin, S. (1999). *Neural Networks: A Comprehensive Foundation* (Second Edition ed.). Prentice Hall.

Hsu, C.-W., C.-C. Chang, and C.-J. Lin (2003, July). A practical guide to support vector classification. Technical report, National Taiwan University.

Kearns, M. and L. Valiant (1994). Cryptographic limitations on learning Boolean formulae and finite automata. *Journal of the ACM 41*(1), 67–95.

Keerthi, S. S. and C.-J. Lin (2003). Asymptotic behaviors of support vector machines with gaussian kernel. *Neural Computation 15*, 1667–1689.

Li, L. (2001). *LEMGA: Learning Models and Generic Algorithms.* Software available at `http://www.work.caltech.edu/ling/lemga`.

Li, L., Y. S. Abu-Mostafa, and A. Pratap (2003). CGBoost: Conjugate gradient in function space. Technical Report CaltechCSTR:2003.007, California Institute of Technology.

Lin, C.-J. (2001). Formulations of support vector machines: a note from an optimization point of view. *Neural Computation 13*(2), 307–317.

Lin, H.-T. and C.-J. Lin (2003). A study on sigmoid kernels for SVM and the training of non-PSD kernels by SMO-type methods. Technical report, Dept. of CSIE, National Taiwan Univ.

Mason, L., J. Baxter, P. L. Bartlett, and M. Frean (2000). Functional gradient techniques for combining hypotheses. In A. J. Smola, P. J. Bartlett, B. Schökopf, and D. Schuurmans (Eds.), *Advances in Large Margin Classifiers.* MIT Press.

Meir, R. and G. Rätsch (2003). An introduction to boosting and leveraging. In S. Mendelson and A. J. Smola (Eds.), *Advanced Lectures on Machine Learning.* Springer-Verlag.

Micchelli, C. A. (1986). Interpolation of scattered data: Distance matrices and conditionally positive definite functions. *Constructive Approximation 2*, 11–22.

Nash, S. G. and A. Sofer (1996). *Linear and Nonlinear Programming.* McGraw-Hill.

Rasmussen, C. E. (2000). The infinite Gaussian Mixture Model. In S. A. Solla, T. K. Leen, and K. Müller (Eds.), *Advances in Neural Information Processing Systems*, Volume 12, pp. 443–560. MIT Press.

Rätsch, G., S. Mika, B. Schölkopf, and K. Müller (2002). Constructing boosting algorithms from SVMs: an application to one-class classification. *IEEE Transactions on Pattern Analysis and Machine Intelligence 24*(9), 1184–1199.

Rätsch, G., T. Onoda, and K. Müller (2001). Soft margins for AdaBoost. *Machine Learning 42*(3), 287–320.

Reed, M. and B. Simon (1980). *Functional Analysis* (Revised and Enlarged ed.). Methods of Modern Mathematical Physics. Academic Press.

Rosset, S., J. Zhu, and T. Hastie (2004). Boosting as a regularized path to a maximum margin classifier. *Journal of Machine Learning Research 5*, 941–973.

Schölkopf, B. and A. Smola (2002). *Learning with Kernels*. Cambridge, MA: MIT Press.

Tresp, V. (2000). A Bayesian Committee Machine. *Neural Computation 12*, 2719–2741.

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

Vapnik, V. N. (1998). *Statistical Learning Theory.* New York, NY: Wiley.

Vapnik, V. N. and A. Y. Chervonenkis (1971). On the uniform convergence of relative frequencies of events to their probabilities. *Theory Prob. Appl. 16.*

Zhang, T. (2002). Covering number bounds of certain regularized linear function classes. *Journal of Machine Learning Research 2*, 527–550.

Index