Active Learning with Semi-Supervised Support Vector Machines

by

Leila Chinaei

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Mathematics
in
Computer Science

Waterloo, Ontario, Canada, 2007

Declaration

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revising, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Leila Chinaei

Abstract

A significant problem in many machine learning tasks is that it is time consuming and costly to gather the necessary labeled data for training the learning algorithm to a reasonable level of performance. In reality, it is often the case that a small amount of labeled data is available and that more unlabeled data could be labeled on demand at a cost. If the labeled data is obtained by a process outside of the control of the learner, then the learner is passive. If the learner picks the data to be labeled, then this becomes active learning. This has the advantage that the learner can pick data to gain specific information that will speed up the learning process. Support Vector Machines (SVMs) have many properties that make them attractive to use as a learning algorithm for many real world applications including classification tasks. Some researchers have proposed algorithms for active learning with SVMs, i.e. algorithms for choosing the next unlabeled instance to get label for. Their approach is supervised in nature since they do not consider all unlabeled instances while looking for the next instance. In this thesis, we propose three new algorithms for applying active learning for SVMs in a semi-supervised setting which takes advantage of the presence of all unlabeled points.

The suggested approaches might, by reducing the number of experiments needed, yield considerable savings in costly classification problems in the cases when finding the training data for a classifier is expensive.

Acknowledgments

First, I would like to thank God for giving me the opportunity to enjoy all the experiences I have had during my time at the University of Waterloo. I thank my family and especially my siblings for their support and encouragement.

My great thanks go to my advisor, Pascal Poupart, for introducing me to many areas of research and allowing me to select the topic that interested me. I deeply thank him for his guidance, encouragement, and his precious time spent offering me his insight and valuable feedback. I would also like to acknowledge the valuable comments and suggestions from Shai Ben-David and Ali Ghodsi.

Last but not least, I would like to thank all of my professors, for the enjoyable classes they taught me, and my friends and colleagues, for making my time in Waterloo exciting and pleasant.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Artificial Intelligence has been receiving a lot of attention recently. A common problem in many applications is classification. For example, in diagnostic prediction of cancers, the task may involve analyzing tumor tissue types and genes related to cancers and classify them as "benign" or some other classes. Various machine learning methods have been investigated for analysis and classification of genes related to cancers [CW03, wC03]. While it is commonly assumed that labeled data is available, in practice, there is usually cost associated with obtaining this labeled data. As a result, it is often the case that only a small amount of labeled data is available. However, it may be possible that a large pool of unlabeled data is also freely available and that the learner can request the labels of some data points for some cost. This thesis considers the problem of *active learning* whereby the learner must decide which data instances to request labels for. Intuitively, the learner would like to get the labels of the data instances that are the most informative to reduce the uncertainty with respect to the concept that it is trying to learn. A key observation of this thesis is that in selecting the instances to be queried, one should take into account all the information already available, including the labeled instances as well as all the unlabeled instances.

This thesis focuses on active learning for Support Vector Machine (SVM) classifiers. SVMs provide one approach for efficiently training a linear separator. SVMs have many properties that make them attractive for many real world applications. For example, Yeang et al. [YRT+01] reported successful application of SVMs for multiclass cancer diagnosis. SVMs enjoy good empirical performance while being grounded in a well developed theory for generalization.

Traditionally, SVMs were applied in a *supervised learning* setting in which the dis-

criminant function is created by considering only labeled data. However, by exploiting unlabeled data, the learner may get additional information about the problem which can be used to improve the accuracy. This setting is referred as *semi-supervised learning*.

In previous work, Tong and Koller [TK00b] proposed an algorithm that applies active learning for SVMs to text classification. Their approach is supervised (Figure 1.1) in the sense that the decision regarding the next instance to be labeled in made only based on labeled data. In this thesis, we propose some semi-supervised algorithms which take into account the presence of all unlabeled instances when selecting the next unlabeled instance (Figure 1.2).

There are many classification problems where the number of classes varies. For the sake of simplicity, we consider only the two-class problems (also called binary classification). The data that we are working with is represented by features and the task is to predict the classes.

1.1 Research Contribution

The result of this thesis are two algorithms, called MinMax Distance and Pruning, which apply Active Learning with Semi-supervised Support Vector Machines (ALS^{3}VM). To our knowledge this work is the first to consider active learning for SVM in a semi-supervised setting. Previous work on active learning with SVMs is in a supervised setting which does not take advantage of unlabeled data [TK00b]. Our approaches take into account the presence of existing unlabeled instances and are able to pick better instances to be labeled. Empirically, we show that on some examples, ALS^{3}VM algorithms outperform supervised algorithms.

1.2 Thesis Overview

The structure of this thesis is as follows: Chapter 2 provides some background which is essential to understand the material in the rest of the thesis. Chapter 3 is a literature review on some applications of active learning. Chapter 4 describes semi-supervised SVMs and proposes two algorithms for active learning with semi-supervised SVMs. Chapter 5 empirically compares our work to some existing work on active learning with SVMs. Finally, Chapter 6 concludes this work and proposes some future work.

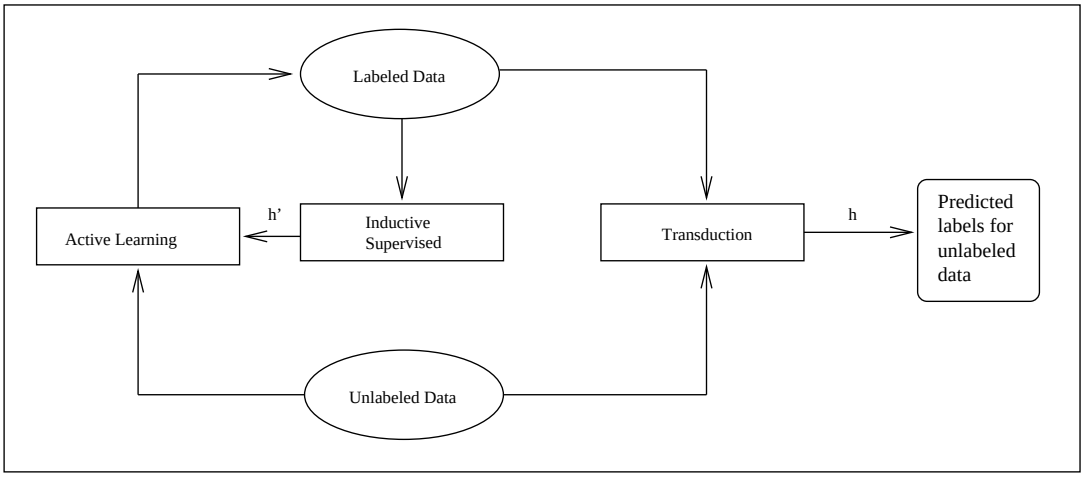

Figure 1.1: Active learning with Supervised Support Vector Machines.

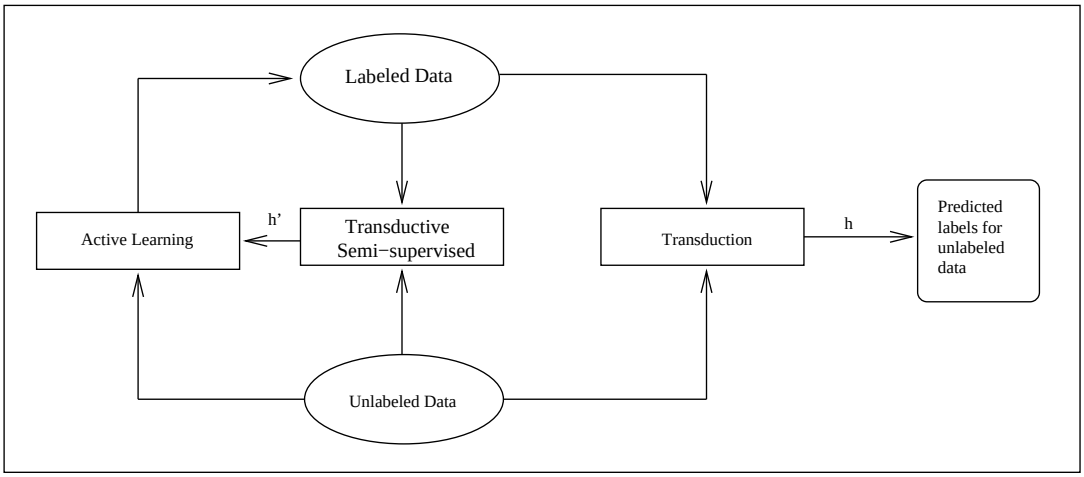

Figure 1.2: Active learning with Semi-Supervised Support Vector Machines.

Chapter 2

Background

In this chapter we review some background about machine learning in general, and about active learning, supervised and unsupervised support vector machines in particular.

2.1 Machine Learning

Machine learning is a broad subfield of artificial intelligence. It involves developing algorithms and techniques which allow computers to learn. This capacity to learn from experience, analytical observation, and other means, results in a system that can continuously improve itself and thereby offer increased efficiency. Machine learning has a wide range of applications such as medical diagnosis [LLZN05], bioinformatics [EZ04], detecting credit card fraud, classifying DNA sequences [FCD+00], speech and handwriting recognition [BHB02], object recognition in computer vision [Dra00], and robot locomotion.

The learner is provided with some data which is usually divided into a training set and a test set. The training set consists of a set of input points in some potentially multi-dimensional space. The goal here is to find a mapping from the input points to some labels corresponding to some categories of interest in some domain. The testing set is a set of examples which is used to assess the performance of the learner.

Learning is about generalization. There are two types of learning called *inductive learning* and *transductive learning* (Figures 2.1 and 2.2). In inductive learning, the task is to build a good classifier on the training set with the ability to generalize to any unseen data. Here the test set is unknown at the time of training. However, in transductive learning the learner knows the test set at the time of training and therefore only needs

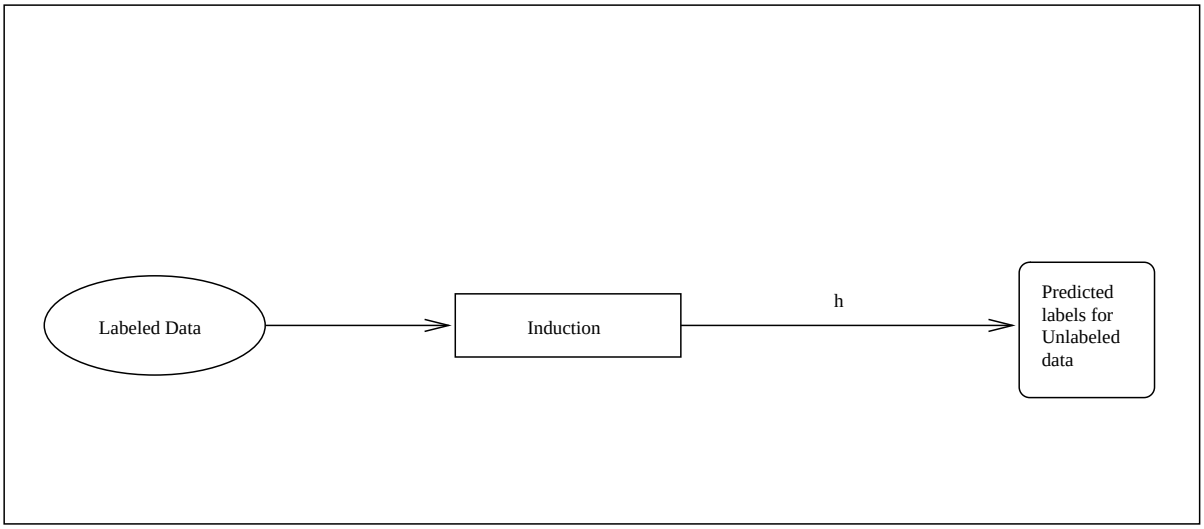

Figure 2.1: Inductive Learning.

to build a good classifier that generalizes to this known test set.

There are different techniques in machine learning for learning a concept from data. One general approach is *supervised learning* in which the training set consists of only labeled data. The goal is to learn a function which is able to generalize well on unseen data. The name supervised given to this approach is because desired labels are given to the learner. Another approach is *unsupervised learning* whereby some examples are presented to the system as observations without any label. There is no prior output. Unsupervised learning uses procedures that try to find the natural patterns of the data. There is no explicit teacher for the system to find the pattern of the model and the learner must discover by itself what to do. In supervised, learning all the training data is labeled and in unsupervised learning, none of the training data is labeled. However, in *semi-supervised learning*, both labeled and unlabeled data are used. In contrast to labeled data, which is often expensive and time consuming to gather, unlabeled data is usually easier to collect. So, in semi-supervised learning, it is common to have a small amount of labeled data with a large amount of unlabeled data. Therefore, semi-supervised learning tries to find a better classifier from both labeled and unlabeled data.

Traditionally, in machine learning, it is assumed that data is already given. However, it is time-consuming and costly to gather the necessary data for training a classifier to a reasonable level of performance. In reality, it is often the case that a small amount of labeled data is available and that more unlabeled data could be labeled on demand at a

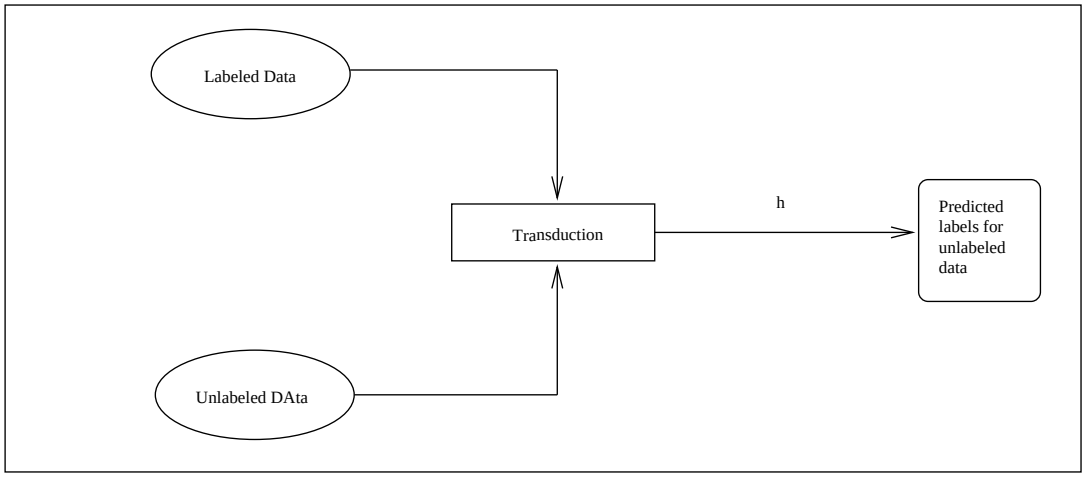

Figure 2.2: Transductive Learning.

cost. If the labeled data is obtained by a process outside of the control of the learner, then the learner is passive and therefore this is called *passive* learning. If the learner picks the data to be labeled, then this becomes *active* learning. This has the advantage that the learner can pick data to gain specific information that will speed up the learning process. For example, suppose a medical researcher wants to find the correlation between lung cancer and smoking, including second-hand smoke. The researcher may get a bunch of people to volunteer to be part of the study. There will likely be lots of smokers and non smokers, but very few non smokers exposed to second-hand smoke. This is where active learning could help the learner to bias his sample towards the subjects that he is interested in.

2.2 Active Learning

There are many real-world tasks that involve classification. Because the number of data points that need to be classified is often very large, we need an automated way to select data which is going to be labeled. The learner has a training set which is classified in advance, and uses it to learn a mapping that is used later on to automatically classify the unlabeled points. Since getting labels for data points is sometimes expensive, we would like to ask labels for some of the instances that give the most information. Asking for the label is called query. Also, as mentioned in the previous section, there are some real applications where the learner has the ability to select samples.

Given this ability, it is in interest of a the learner to select the training data points that give it the most information based on its current state of knowledge. Active learning can be used to reduce the amount of data needed for training by selecting those data points which may be deemed the most useful by some measure based on the current information.

Active learning is also known as *query learning* [CCS00] and *selective learning* [CAL94, TK00a]. A variant of active learning is *pool-based active learning*, which means the learner has access to a pool of instances, and can query among those examples in the pool. Sometimes the learner asks for a full instance where some of the attributes take on requested value. In medical researcher example, he can ask for people of certain age met.

In some applications the cost of queries may vary and the learner has a fixed budget that it can spend to identify the best model. In this case, the active learner tries to select the queries (within the budget) that reveal the most information. For example, suppose the medical researcher has allocated a certain amount of money to develop a system that diagnoses lung cancer by doing some tests with different known costs and unknown discriminative strength on various tissue. The question is how he should spend this money to discover the best lung cancer diagnostic procedure i.e. which tests should be run in which tissue. In other situations, there is no budget, however a desired level of accuracy should be achieved. In such situations, the goal is to minimize cost of queries necessary to find a good classifier. When the queries have the same cost, then the objective is to ask the least number of queries in order for the learner algorithm to get a reasonable level of performance.

The common question, using active learning, is how to choose the next instance or unlabeled point to query. Depending on the learning application, different techniques are applied.

2.3 Classification

In order to introduce support vector machine classifiers, we first introduce linear classifiers and learning in the feature space.

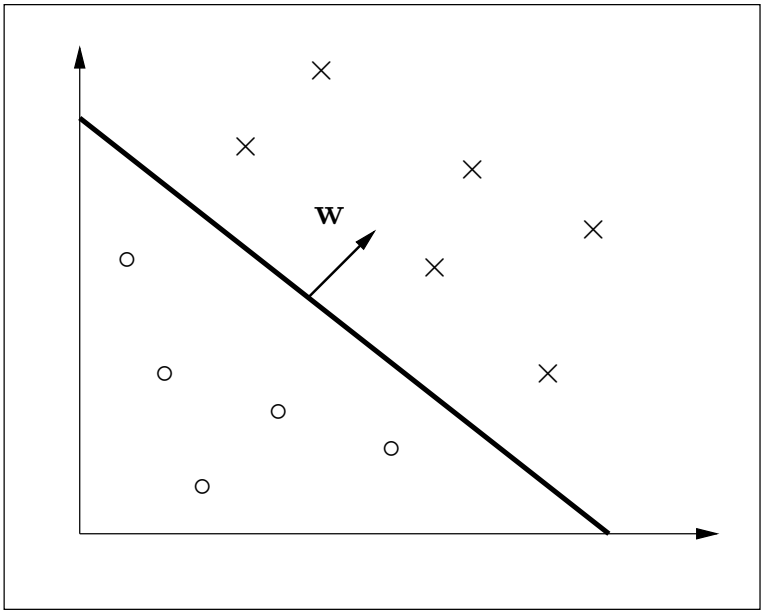

Figure 2.3: A separating hyperplane controlled by $(\mathbf{w}, b)$ for a two dimensional training set.

2.3.1 Linear Classification

Given a set of training data $S = \{\mathbf{x}_1, ..., \mathbf{x}_l\}$ with vector $\mathbf{x}_i$ from $\mathbf{X} \subseteq \Re^n$ and their labels $\{y_1, ..., y_l\}$, where $y_i \in \{-1, 1\}$, a binary classifier is a real-valued function $f : \mathbf{X} \subseteq \Re^n \longrightarrow \Re$ where $\mathbf{x} = (x_1, ...x_n)$ is assigned to the positive class if $f(\mathbf{x}) \geq 0$, and otherwise to the negative class. The objective is to create a classifier which assigns labels to a set of unlabeled points of interest, called a test set. When $f(\mathbf{x})$ is a linear function of $\mathbf{x} \in \mathbf{X}$, it can be written as:

$$f(\mathbf{x}) = \langle \mathbf{w}, \mathbf{x} \rangle + b = \sum_{i=1}^{l} w_i x_i + b \tag{2.1}$$

where $(\mathbf{w}, b) \in \Re^n \times \Re$ are parameters that control the function. Function f gives the possible values $-1,$ and 1 for each point in the sample space. This function is called a hypothesis.

Figure 2.3 shows a separating hyperplane defined by the equation $\langle \mathbf{w}, \mathbf{x} \rangle + b = 0$ of dimension $n - 1$, that splits the space into two half spaces which correspond to two distinct classes.

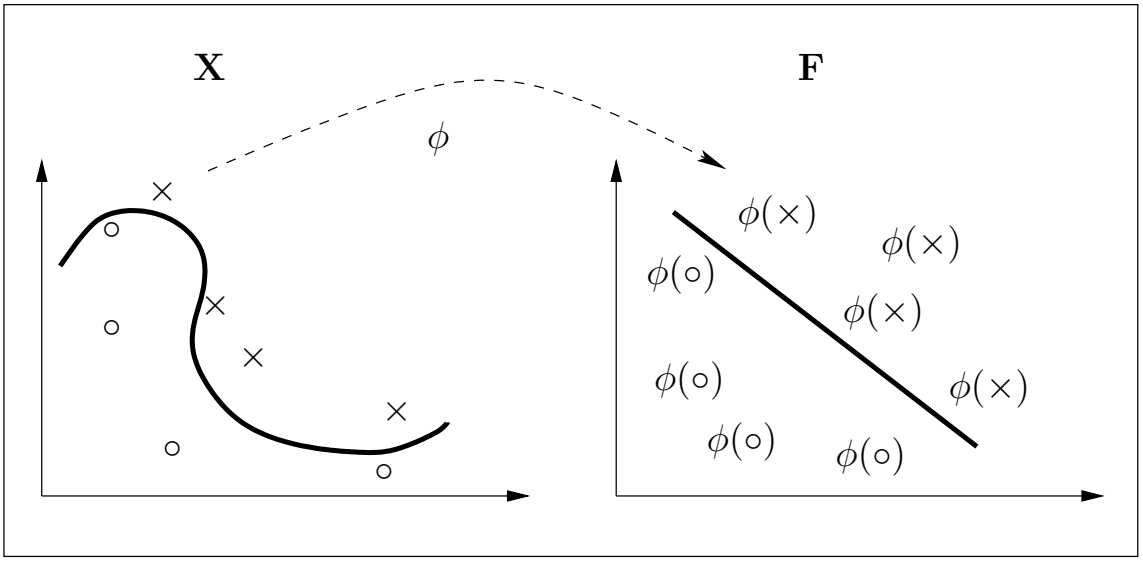

Figure 2.4: A feature map from input space to feature space.

2.3.2 Learning in Feature Space

Some real-world applications require more expressive hypothesis spaces than a linear separator, as linear separation of the data may not be possible. This leads us to non-linear separators. Non-linear separators may be viewed as linear separators in a different space. The complexity of learning the classifier function depends on the way it is represented, and so the difficulty of the learning task can vary accordingly. One strategy used in machine learning involves changing the representation of the data. This is equivalent to mapping the input data in space $\mathbf{X}$ to a new space $\mathbf{F}$:

$$\mathbf{x} = (x_1, ..., x_n) \longmapsto \Phi(\mathbf{x}) = (\Phi_1(\mathbf{x}), ..., \Phi_m(\mathbf{x})),$$

where $\Phi : \mathbf{X} \longrightarrow \mathbf{F}$ is a non-linear map from the input space to some space $\mathbf{F}$. The space $\mathbf{F}$ is called feature space since the quantities introduced to describe the data are called features. Figure 2.4 shows how a feature mapping can simplify the classification task.

Therefore, we can learn non-linear relations by using linear machines and by using the feature map Φ. Using Equation 2.1, we can write:

$$f(\mathbf{x}) = \langle \mathbf{w}, \Phi(\mathbf{x}) \rangle + b = \sum_{i=1}^{m} w_i \Phi_i(\mathbf{x}) + b \tag{2.2}$$

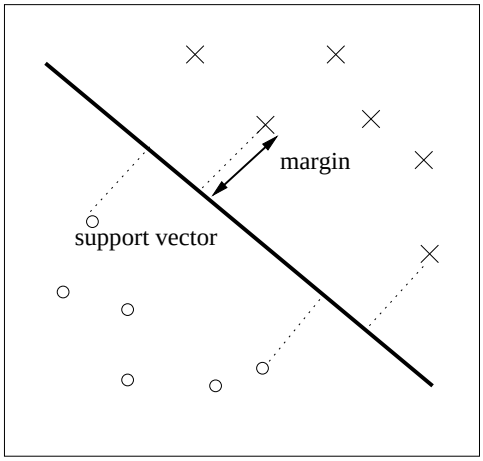

Figure 2.5: A linear support vector machine.

2.3.3 Supervised Support Vector Machines

Support Vector Machine (SVM) is a type of classifier first introduced by Vapnik [Vap82]. Consider SVM for binary classification, with a set of training data $S = \{\mathbf{x}_1, ..., \mathbf{x}_l\}$, where each $\mathbf{x}_i$ is some vector in $\mathbf{X} \subseteq \Re^n$, and their labels $\{y_1, ..., y_l\}$ where $y_i \in \{-1, 1\}$. The SVM classifier computes a hyperplane that separates the training data in two sets corresponding to the desired classes. Since there are often many hyperplane that separate the two classes, the SVM classifier picks the hyperplane that maximizes some distance measure to the points on either side. This distance is often referred as the margin. All points labeled -1 are on one side of the hyperplane and all points labeled 1 are on the other side. The instances of the training data that lie closest to the SVM are called support vectors. This classifier is called support vector machine since the solution only depends on the support vectors. Figure 2.5 shows a simple linear separator as well as the support vectors.

To understand the margin better, we define it in a formal way.

Definition 2.3.1 *Functional margin γ_i of an example $(\mathbf{x}_i, y_i)$ with respect to a hyperplane $(\mathbf{w}, b)$ is defined as:*

$$\gamma_i = y_i(\langle \mathbf{w}, \mathbf{x}_i \rangle + b).$$

A positive margin $\gamma_i > 0$ implies the correct classification of $(\mathbf{x}_i, y_i)$.

Among hyperplanes which separate the labeled points, the one which gives the largest margin intuitively gives the most confidence. SVMs find the maximal margin hyperplane,

however the functional margin as defined in 2.3.1 can be made arbitrary large by scaling $\mathbf{w}$ and b. Therefore, one can instead optimize the geometric margin γ_g, which corresponds to a normalized version of the functional margin:

$$\gamma_{g_i} = y_i(\langle \mathbf{w}, \mathbf{x}_i \rangle + b)/\|\mathbf{w}\|.$$

The geometric margin measures the Euclidean distances of the points from the decision boundary in the input space. Maximizing the geometric margin leads to the following optimization problem:

$$\begin{array}{ll} \max_{\mathbf{w},b} & \gamma_g \\ \text{s.t.} & y_i(\langle \mathbf{x}_i, \mathbf{w} \rangle + b)/\|\mathbf{w}\| \geq \gamma_g\,, \;\; i = 1, ..., l \end{array} \tag{2.3}$$

Instead of maximizing the margin we can minimize $\|\mathbf{w}\|_2$:

$$\begin{array}{ll} \min_{\mathbf{w},b} & \|\mathbf{w}\|_2 \\ \text{s.t.} & y_i(\mathbf{w} \cdot \mathbf{x}_i + b) \geq 1, \;\; i = 1, ..., l \end{array} \tag{2.4}$$

This follows from the fact that the geometric margin is equal to $\frac{1}{\|\mathbf{w}\|_2}$. We can derive this fact as follows. Consider again the function margin:

$$y_i(\langle \mathbf{w}, \mathbf{x}_i \rangle + b) = \gamma_f$$

By substituting two support vectors $\mathbf{x}^+$, $\mathbf{x}^-$, as some positive and negative points respectively, the following is obtained:

$$+1\left(\langle \mathbf{w}, \mathbf{x}^+ \rangle + b\right) = \gamma_f \tag{2.5}$$

$$-1\left(\langle \mathbf{w}, \mathbf{x}^- \rangle + b\right) = \gamma_f \tag{2.6}$$

Adding 2.5 and 2.6, and substituting γ_f with $\gamma_g\|\mathbf{w}\|_2$ results in the following:

$$2\gamma_g\|\mathbf{w}\|_2 = \left(\langle \mathbf{w}, \mathbf{x}^+ \rangle - \langle \mathbf{w}, \mathbf{x}^- \rangle\right)$$

Fixing γ_f equal to one implies:

$$\big(\langle \mathbf{w}, \mathbf{x}^+ \rangle - \langle \mathbf{w}, \mathbf{x}^- \rangle\big) = 2$$

Therefore:

$$\gamma_g = \frac{1}{\|\mathbf{w}\|_2}$$

2.3.4 Optimization in Support Vector Machines

SVMs can be modeled as different optimization problems with different corresponding techniques. One of the important properties of SVMs is that the generated optimization problems are often convex and therefore a global optimum can be guaranteed. Moreover, the solutions to the corresponding dual optimization problems are sparse, which allows more efficient algorithms.

In this section, we first review some optimization formulations of SVMs when the data instances are assumed to be linearly separable. Then, we discuss the case where some instances are misclassified which often makes the data not linearly separable.
Applying the definition of the inner product, we can rewrite the optimization problem 2.4 as follows:

$$\begin{array}{ll} \min_{\mathbf{w},b} & \frac{1}{2}\langle \mathbf{w}, \mathbf{w} \rangle \\ \text{s.t.} & y_i \left(\langle \mathbf{w}, \mathbf{x}_i \rangle + b \right) \geq 1, \;\; i = 1, ..., l \end{array} \tag{2.7}$$

In order to find the optimal solution of the above problem, we reformulate it by writing its dual representation which is easier to solve. The dual representation is also essential for the use of the kernel trick with SVMs. A strategy to find the dual of Problem 2.7 consists of finding its Lagrangian primal representation and then finding the dual of the Lagrangian. Bellow are the Lagrangian definition and the Kuhn-Tucker theorem which gives sufficient and necessary conditions for an optimum solution to the optimization problem.

Definition 2.3.2 *Given the optimization problem*

$$\begin{array}{ll} min_{\mathbf{X}} & f(\mathbf{X}) \\ s.t. & g_i(\mathbf{X}) \leq 0, \quad i = 1, ..., k \\ & h_j(\mathbf{X}) = 0, \quad j = 1, ..., m \end{array}$$

the Lagrangian primal is defined as :

$$\begin{array}{ll} L(X, \boldsymbol{\alpha}, \boldsymbol{\beta}) & = f(X) + \sum_{i=1}^{k} \alpha_i g_i(X) + \sum_{j=1}^{m} \beta_j h_j(X) \\ & = f(X) + \boldsymbol{\alpha}\mathbf{g}(\mathbf{X}) + \boldsymbol{\beta}\ \mathbf{h}(\mathbf{X}) \end{array}$$

where α_i and β_i are called the Lagrangian multipliers.

The Lagrangian is often used to move the constraints into the objective of an optimization problem. The Kuhn-Tucker theorem gives necessary and sufficient conditions expressed in terms of the Lagrangian for the optimal solution.

Theorem 2.3.3 (Kuhn-Tucker) *Given the optimization problem*

$$\begin{array}{lll} min_{\mathbf{X}} & f(\mathbf{X}), & \mathbf{X} \in \Omega \\ s.t. & g_i(\mathbf{X}) \leq 0, & i = 1, ..., k \\ & h_j(\mathbf{X}) = 0, & j = 1, ..., m \end{array}$$

with the convex domain $\Omega \subseteq \Re^n$, $f \in \mathbf{C}^1$ convex and g_i, h_j affine , $\mathbf{X}^*$ is an optimum point if and only if there exist $\boldsymbol{\alpha}^*, \boldsymbol{\beta}^*$ such that

$$\frac{\partial L(\mathbf{X}^*, \boldsymbol{\alpha}^*, \boldsymbol{\beta}^*)}{\partial \mathbf{w}} = 0,$$

$$\frac{\partial L(\mathbf{X}^*, \boldsymbol{\alpha}^*, \boldsymbol{\beta}^*)}{\partial \beta} = 0,$$

$$\begin{array}{ll} \alpha_i^* g_i(\mathbf{X}^*) = 0, & i = 1, ..., k \\ g_i(\mathbf{X}^*) \leq 0, & i = 1, ..., k \\ \alpha_i^* \geq 0, & i = 1, ..., k \end{array}$$

Now, applying the Lagrangian definition, the Lagrangian primal of Problem 2.7 is:

$$L(\mathbf{w}, b, \boldsymbol{\alpha}) = \frac{1}{2}\langle \mathbf{w}, \mathbf{w} \rangle - \sum_{i=1}^{l} \alpha_i [y_i \left(\langle \mathbf{w}_i, \mathbf{x}_i \rangle + b \right) - 1]$$

Note that variables $\mathbf{w}$ and b are the variables corresponding to $\mathbf{X}$ in the Lagrangian definition. Using the first equation of the Kuhn-Tucker theorem for variables $\mathbf{w}$, b of Problem 2.7, we can isolate $\mathbf{w}$ and compute it as follows:

$$\frac{\partial L(\mathbf{w}, b, \boldsymbol{\alpha})}{\partial \mathbf{w}} = 0 \Rightarrow \mathbf{w} = \sum_{i=1}^{l} y_i \alpha_i \mathbf{x}_i \tag{2.8}$$

$$\frac{\partial L(\mathbf{w}, b, \boldsymbol{\alpha})}{\partial b} = 0 \Rightarrow \sum_{i=1}^{l} y_i \alpha_i = 0 \tag{2.9}$$

where $\alpha_i \geq 0$ are the Lagrange multipliers associated with each point.
Equation 2.8 shows that the hypothesis $\mathbf{w}$ can be written as a linear combination of training points $\mathbf{x}_i$. By resubstituting the result in the Lagrangian primal, the following result is obtained :

$$\begin{aligned} L(\mathbf{w}, b, \boldsymbol{\alpha}) &= \frac{1}{2}\langle \mathbf{w}, \mathbf{w} \rangle - \sum_{i=1}^{l} \alpha_i [y_i \left(\langle \mathbf{w}_i, \mathbf{x}_i \rangle + b \right) - 1] \\ &= \frac{1}{2} \sum_{i,j=1}^{l} y_i y_j \alpha_i \alpha_j \langle \mathbf{x}_i, \mathbf{x}_j \rangle - \sum_{i,j=1}^{l} y_i y_j \alpha_i \alpha_j \langle \mathbf{x}_i, \mathbf{x}_j \rangle - b \sum_{i=1}^{l} \alpha_i y_i + \sum_{i=1}^{l} \alpha_i \\ &= \sum_{i=1}^{l} \alpha_i - \frac{1}{2} \sum_{i,j}^{l} y_i y_j \alpha_i \alpha_j \langle \mathbf{x}_i, \mathbf{x}_j \rangle \end{aligned}$$

Hence, instead of solving Problem 2.7, we can solve its dual problem which corresponds to maximizing α in the Lagrangian primal subject to necessary and sufficient conditions of the Kuhn-Tucker theorem. This leads to the following quadratic optimization problem [CST00]:

$$\begin{array}{ll} \max_{\boldsymbol{\alpha}} & f(\boldsymbol{\alpha}) = \sum_{i=1}^{l} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{l} y_i y_j \alpha_i \alpha_j \langle \mathbf{x}_i, \mathbf{x}_j \rangle \\ \text{s.t.} & \sum_{i=1}^{l} y_i \alpha_i = 0 \\ & \alpha_i \geq 0 \, , i = 1, ..., l. \end{array}$$

Finding the solution $\boldsymbol{\alpha}^*$ of the above problem and substituting in Equation 2.8 finds the parameter $\mathbf{w}$ of the maximal margin hyperplane with the geometric margin $\gamma = \frac{1}{\|\mathbf{w}^*\|_2}$:

$$\mathbf{w}^* = \sum_{i=1}^{l} y_i \alpha_i^* \mathbf{x}_i \tag{2.10}$$

Note that the value b_* of the hyperplane must be found from the primal problem since b does not appear in the dual problem. Considering the constraint of primal Problem 2.7:

$$y_i \left(\langle \mathbf{w}, \mathbf{x}_i \rangle + b \right) \geq 1 \, , i = 1, ..., l$$

and substituting $\mathbf{w}^*$ for both labels $\{1, -1\}$ gives:

$$\begin{aligned} b &\geq 1 - \langle \mathbf{w}^*, \mathbf{x}_i \rangle \quad \text{for } \; y_i = 1 \\ b &\leq -1 - \langle \mathbf{w}^*, \mathbf{x}_i \rangle \quad \text{for } \; y_i = -1 \end{aligned}$$

and the b which gives the largest margin from both classes is:

$$b^* = \frac{1}{2} [\max_{y_i = -1} \left(\langle \mathbf{w}^*, \mathbf{x}_i \rangle \right) - \min_{y_i = 1} \left(\langle \mathbf{w}^*, \mathbf{x}_i \rangle \right)]$$

By using the third equation in the Kuhn-Tucker theorem, known as Karush-Kuhn-Tucker(KKT) complementarity, we see that the following equation must hold between the optimal values $\boldsymbol{\alpha}^*, (\mathbf{w}^*, b^*)$:

$$\alpha_i^* [1 - y_i (\langle \mathbf{w}_i^*, \mathbf{x}_i \rangle + b^*)] = 0, \; i = 1, ..., l.$$

This expression implies that only for $\mathbf{x}_i$'s with functional margins equal to one are the corresponding α_i^*'s non-zero. Hence, all parameters α_i^* are zero, except for those that lie closest to the hyperplane, i.e. the support vectors.

Now we turn our attention to the case where a point $\mathbf{x}_j$ is misclassified. In general the classes are not separable. Assume the point x_j has label $+1$, but it is classified as -1, hence:

$$y_j (\mathbf{w}.\mathbf{x}_j + b) = -\varepsilon_j \; \text{ where } \; \varepsilon_j > 0$$

In order to satisfy the constraint $y_j (\mathbf{w} \cdot \mathbf{x}_j + b) \geq 1$, one needs to add the distance (ε_j in Figure 2.6) of that misclassified point from its true class to the left hand side of the constraint. Therefore, for each misclassified point x_i, a slack variable ε_i is used, and the following generalized optimal plane problem is obtained:

$$\begin{array}{ll} \min_{\mathbf{w},b,\varepsilon} & \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{l} f(\varepsilon_i) \\ \text{s.t.} & y_i(\mathbf{w}\cdot\phi(\mathbf{x}_i)+b) + \varepsilon_i \geq 1,\ \varepsilon_i \geq 0\ \ i = 1, ..., l \end{array} \tag{2.11}$$

where C is a fixed penalty parameter. Obviously, for points which are classified correctly, $\varepsilon_i = 0$.

There are different ways of penalizing for misclassified points. These different ways are encoded with various loss functions that correspond to the second term in the objective function of Problem 2.11. One option is a loss function of 0/1.

$$\delta(\varepsilon) = \begin{array}{ll} 1 & \text{for}\, \varepsilon > 0 \\ 0 & \text{otherwise} \end{array}$$

If a point is misclassified, it gets the penalty of 1, otherwise, the penalty is zero. In this case, the sum of penalties gives the number of misclassified points. Clearly, this loss function behaves the same with all misclassified points regardless of the distance of the misclassified point to its true class. Another loss function is the convex Hinge loss which gives a larger penalty if the misclassified point is far away from the true class.

Definition 2.3.4 *: The Hinge loss function (illustrated in Figure 2.7) is defined as:*

$$H_1(z) = \max(0, 1 - z)$$

2.3.5 Kernel Trick

As discussed in Section 2.3.2, we can learn non-linear separators by mapping the data (with a non-linear mapping) into some feature space and then learn a linear classifier in that space. Most of the time, this feature space has higher dimension than the original space, which suggests that finding a classifier will be computationally more expensive. However, by exploiting the fact that SVMs find a classifier based only on the inner products of the data points, it is sometimes possible to work in higher dimensions without paying any price computationally.

In this section, we describe the kernel trick, which allows us to find classifiers in the feature space with roughly the same amount of computation as in the original space when the inner products in the feature space can be quickly computed from the points coordinates in the original space.

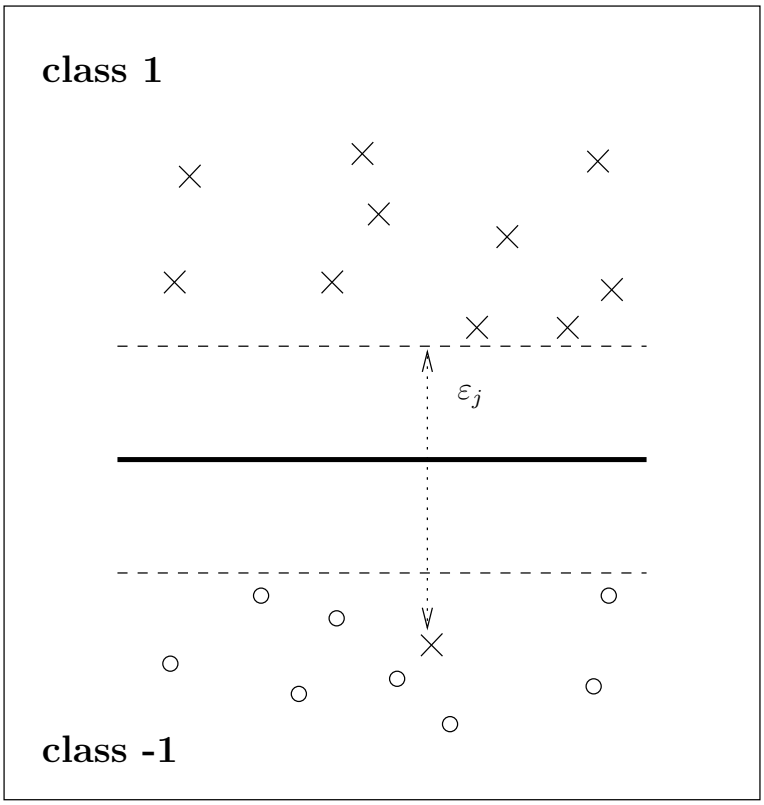

Figure 2.6: A misclassified point.

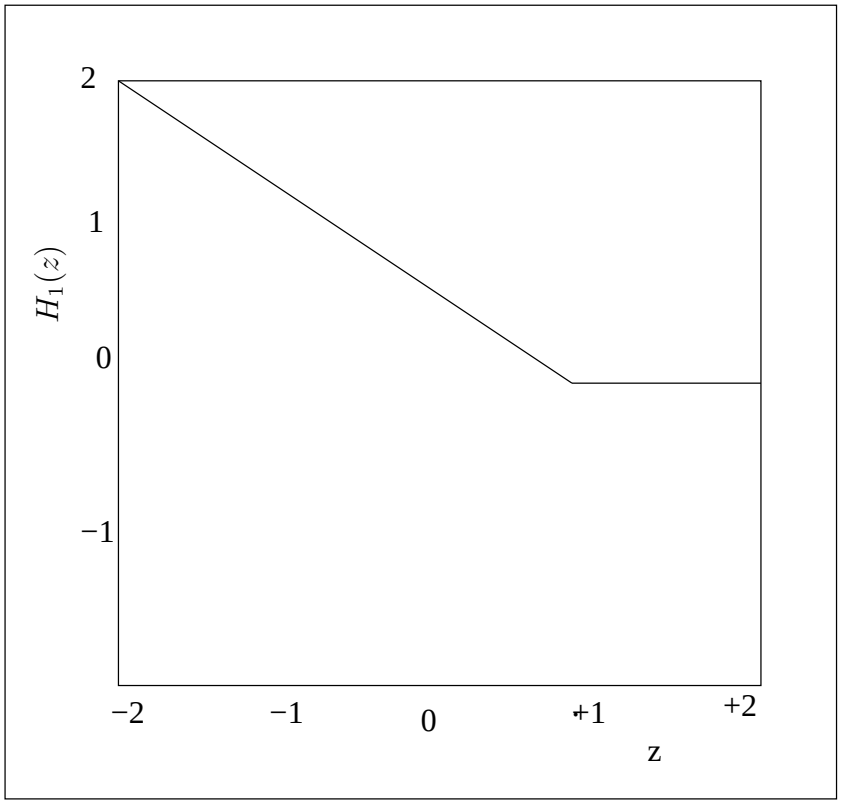

Figure 2.7: Hinge loss function.

Definition 2.3.5 *Given two points* $\mathbf{x}, \mathbf{z}$ *from the input space* $\mathbf{X} \subseteq \Re^n$*, the function* $\mathbf{K}$ *that returns the inner product between their images in the feature space* $\mathbf{F}$ *is known as the kernel function:*

$$\mathbf{K}(\mathbf{x}, \mathbf{z}) = \langle \Phi(\mathbf{x}), \Phi(\mathbf{z}) \rangle, \quad \forall \mathbf{x}, \mathbf{z} \in \mathbf{X}$$

where Φ *is a mapping from input space to the feature space.*

Here are some commonly used kernels:

- Identity function:

$$\mathbf{K}(\mathbf{x}, \mathbf{z}) = \langle \mathbf{x}, \mathbf{z} \rangle$$

- Radial Basis function kernel:

$$\mathbf{K}(\mathbf{x}, \mathbf{z}) = (e^{-\gamma(\mathbf{x}-\mathbf{z})\cdot(\mathbf{x}+\mathbf{z})})$$

- Polynomial kernel of degree p:

$$\mathbf{K}(\mathbf{x}, \mathbf{z}) = (\mathbf{x} \cdot \mathbf{z} + 1)^p$$

Following, we see the use of kernel function for SVMs. Using Equation 2.2 and 2.10, the hypothesis for the data instances can be written as following:

$$\begin{aligned} f(\mathbf{x}) &= \textstyle\sum_{j=1}^{m} \sum_{i=1}^{l} \alpha_i y_i \Phi_j(\mathbf{x_i}) \Phi_j(\mathbf{x}) + b \\ &= \textstyle\sum_{i=1}^{l} \alpha_i y_i \sum_{j=1}^{m} \Phi_j(\mathbf{x_i}) \Phi_j(\mathbf{x}) + b \\ &= \textstyle\sum_{i=1}^{l} \alpha_i y_i \langle \Phi(\mathbf{x_i}), \Phi(\mathbf{x}) \rangle + b \end{aligned}$$

Applying the kernel definition, it becomes:

$$f(\mathbf{x}) = \sum_{i=1}^{l} y_i \alpha_i \mathbf{K}(\mathbf{x}_i, \mathbf{x}) + b$$

The SVM task is to find the α_i that corresponds to the maximal margin hyperplane in the feature space. The choice of feature space has an impact on the complexity of inner product. A good choice can allow the inner product to be computed easily. In particular,

for some feature spaces, the inner product in feature space is a simple function (i.e., kernel function) of the points' coordinates in the original space.

2.3.6 Version Space

Mapping data in a higher space than the input space often makes data separable in the feature space. The set of all hyperplanes that separate the data in the induced feature space is referred as the version space. A hypothesis f is in the version space if $y_i f(\mathbf{x}_i) > 0$ for all $i = 1, ..., l$, where y_i is the corresponding true label for input $\mathbf{x}_i$.

An SVM computes the hyperplane in the version space with the largest margin. In the next section we discuss the generalization performance of SVMs, which gives us some insight to the question "Why is maximizing margin the right thing to do?"

2.3.7 Generalization With Support Vector Machines

One of the goals of learning is having a good generalization performance for the learning algorithm. There are different methods for estimating how well an algorithm will perform on future data. Following, we discuss the generalization error and predicted upper bound for two methods.

Usually in learning tasks, there is an assumption that training and testing data are generated from the same distribution D. Therefore, one intuition is to define the measure of the error, as the probability that a randomly chosen example is misclassified. Given h a hypothesis from the hypothesis space, the error can be defined as:

$$\mathrm{Er}_D(h) = \sum_{\mathbf{x},y} Pr(\mathbf{x}, y|D)\delta(h(\mathbf{x}) \neq y))$$

where

$$\delta(x) = \begin{array}{ll} 1 & \text{if } x \text{ is true} \\ 0 & \text{otherwise} \end{array}$$

Now, the aim is to put a bound on this error in terms of some quantities. PAC (probability approximately correct) results can be used to bound the number of training examples which a learner algorithm needs to get to a particular level of performance with some confidence. Considering the training set $S = \{(\mathbf{x}_1, y_1), ...(\mathbf{x}_l, y_l)\}$, a consistent hypothesis $h_s \in H$(the set of all hypotheses), the generalization error of h_s is:

$$\mathrm{Er}_D(h_s) \leq \varepsilon(l, H, \eta)$$

where η is the probability that the training set gives a hypothesis with an error larger than $\varepsilon(l, H, \eta)$. Vapnik and Chervonenkis derived the following bound for any consistent hypothesis and a hypothesis space with VC dimension $d \geq 1$ [CST00].

Theorem 2.3.6 (Vapnik and Chervonenkis) *Let H be a hypothesis space with VC dimension d. For any probability distribution D on $X \times \{-1, 1\}$, with probability $1-\eta$ over l random examples S, any hypothesis $h \in H$ that is consistent with S has error no more than*

$$Er_D(h_s) \leq \varepsilon(l, H, \eta) = 2/l \left(d \log(2el/d) + \log(2/\eta)\right)$$

provided $d \leq l$ and $l \geq 2/\varepsilon$.

While one would expect the amount of training data to increase with the size of the hypothesis space, instead, this bound shows that the amount of training data only needs to grow linearly with the VC dimension (for a fixed error). Hence, even for infinitely large hypothesis spaces, we do not necessarily need an infinite amount of data and we can avoid overfitting as long as the VC dimension is not too large.

A similar bound can be derived for inconsistent hypotheses [CST00]:

Theorem 2.3.7 *Let H be a hypothesis space having VC dimension d. For any probability distribution D on $X \times \{-1, 1\}$, with probability $1 - \eta$ over l random examples S, any hypothesis $h \in H$ that makes k errors on the training set S has error no more than*

$$Er_D(h_s) \leq \varepsilon(l, H, \eta, k) = 2k/l + 4/l \left(d \log(2el/d) + \log(4/\eta)\right)$$

provided $d \leq l$.

The first term indicates the empirical error (i.e., error on the training data) and the second term indicates the generalization error (i.e., expected error on unseen data). Again, for a fixed generalization error, the amount of data should grow linearly with the VC dimension of the hypothesis space. However in practice, the learner may not have any control over the amount of training data nor the hypothesis space, which means that the generalization error is constant and should only seek to minimize the empirical error when finding a good classifier (this is known as empirical risk minimization). In

situations where there is some freedom in the choice of the hypothesis space, the learner should seek to simultaneously minimize the empirical and generalization error (this is know as structural risk minimization).

In the case of SVMs, an alternative error bound can be derived based on the margin instead of the VC dimension [CST00]. This bound is inversely proportional to the margin, which means that generalization improves with larger margins. This explains why SVMs seek the classifier with maximal margin.

2.4 Active Learning for Support Vector Machines

SVMs are usually applied to a labeled data set which is already given, and assumed to be drawn randomly from some distribution. However, in active learning, the learner can query an oracle to obtain labels for some instances. We define a problem of active learning with five components; $\mathbf{X}, \mathbf{Y}, H, S, O$, where $\mathbf{X}$ is the domain of data, $\mathbf{Y}$ is a set of possible labels, H is the set of all mappings $h : \mathbf{X} \mapsto \mathbf{Y}$, $S = \{(\mathbf{x}_1, y_1), ..., (\mathbf{x}_l, y_l)\}$ is the training set, and O is the number of calls to some oracle which gives the desired label for the queried instance. The goal of learning is to find the best hypothesis h. Unlike traditional machine learning, in active learning the learner has the possibility of getting more labeled data by querying an oracle. After each query or a number of queries, the active learner returns the best classifier h, called hypothesis.

In pool-based active learning, there is a set $\mathbf{U}$ of unlabeled data and the learner selects the next instance to query from it. This brings us to the issue that given an unlabeled pool $\mathbf{U}$, it is not obvious how to choose the next instance to query. An intuition is to query the points which we are most uncertain about. Therefore, we need to find a measurement for each unlabeled instance that gives the gain in information.

Considering the notion of version space in active learning, after querying a point and revising its label, some of the hyperplanes from the version space which previously separate the data will no longer classify the newly queried point correctly. Such hyperplanes are no longer part of the version space, and therefore the version space will be reduced. Hence, the best point to query in each step is the one which gives the greatest expected reduction of the version space [TK00b, SOS92].

2.5 Semi-Supervised Support Vector Machines

There are many real-world applications in which unlabeled data is abundant and free to use, but labeled data is expensive and rare. If we use a training data set with a small amount of data, the learning classifier may face overfitting. However, we can take advantage of the presence of the unlabeled data by considering the information obtained from them. Given a training set of labeled and unlabeled data, semi-supervised support vector machines build a classifier for all available data.

Traditional work on inductive learning estimates a classifier based on some training data that generalizes to any input instances in $\mathbf{X}$. In transductive learning, there is a mixture of labeled and unlabeled data and the objective is to learn a classifier that generalize well to the set of unlabeled data. Demiriz and Bennett [BD98] showed that using transductive Semi-Supervised SVMs (S^3VMs), one can get a significant improvement by using the extra information from the unlabeled data. Their idea is to find the SVM classifier which correctly classifies the labeled data and has the widest margin based on the unlabeled data. Figure 2.8 shows the support vector machine classifier for inductive learning when training. The margin with respect to the test data is shown in Figure 2.9. Figure 2.10 shows transductive semi-supervised SVMs. Comparing Figure 2.9 and 2.10, we can see a significant improvement in the margin for semi-supervised SVMs over supervised SVMs.

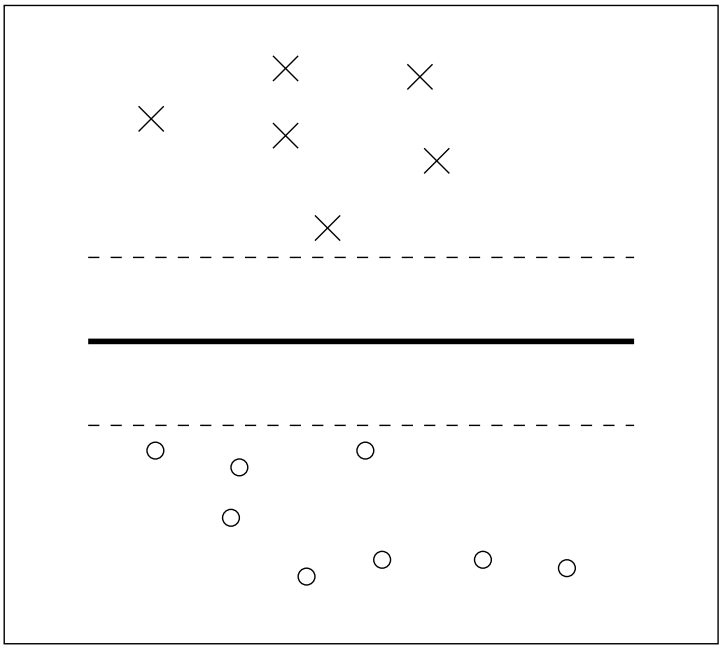

Figure 2.8: Inductive Learning, SVMs-Train

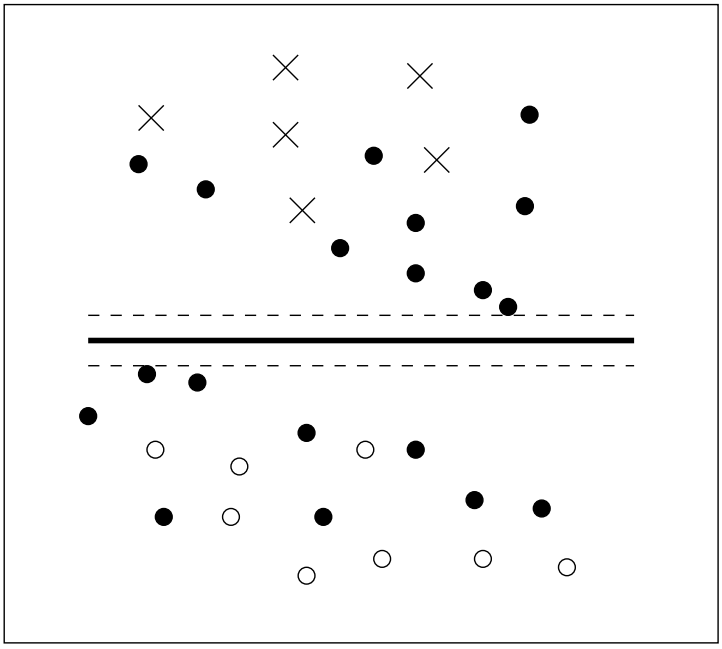

Figure 2.9: SVM Inductive Learning, SVMs-Test

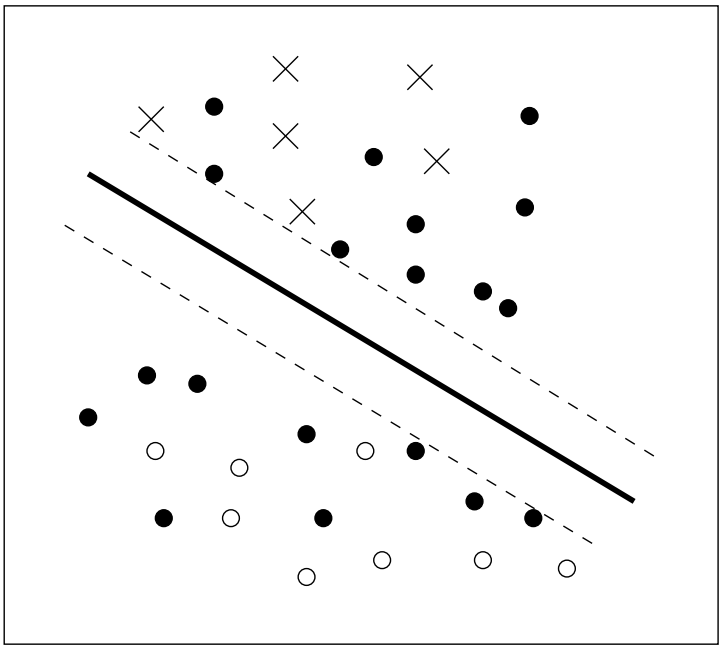

Figure 2.10: Transductive Learning, Semi-Supervised SVMs

Chapter 3

Related Work On Active Learning

There are various applications for active learning and depending on their properties, different models can be derived. The purpose of this chapter is to review some applications of active learning. Section 3.1 is about active learning in Bayesian networks. Section 3.2, shows how active learning helps in finding the best model from a fixed set of models for solving a problem. Section 3.3, describes the query by committee algorithm. Finally, an interesting application for active learning is *text classification*. We examine the use of Support Vector Machines for this problem in Section 3.4.

3.1 Bayesian Networks

Classical learning assumes that a learner is given a set of training data, from which it learns the model. However, in some situations the learner has the ability of being active to gain some specific information. In the following, we review the work of Tong and Koller [TK01, TK00a] which shows how active learning helps in learning Bayesian networks.

Bayesian networks, also known as Bayesian belief networks, are a form of probabilistic graphical model which is represented by an (acyclic) graph of nodes and some probabilities showing the dependency between the nodes. Let $\mathbf{X} = \{\mathbf{X_1}, ..., \mathbf{X_n}\}$ be a set of variables, where each variable $\mathbf{X_i}$ takes a value in some finite domain. A Bayesian network over $\mathbf{X}$ is defined with a pair $(\mathbf{G}, \theta_G)$ that represents a distribution over the joint space of $\mathbf{X}$, where $\mathbf{G}$ is a directed acyclic graph and θ_G is set of parameters which quantify the network. Nodes in the graph are some random variables in $\mathbf{X}$ and the arcs represent statistical dependence relations among the variables. For each variable a

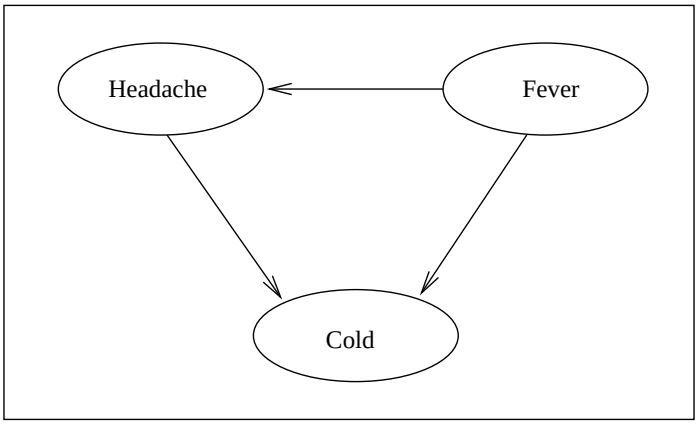

Figure 3.1: A simple Bayesian network

Table 3.1: Conditional distribution for headache: Pr($Headache|Fever$)

	Headache	
Fever	T	F
T	0.4	0.6
F	0.1	0.9

conditional probability distribution quantifies the statistical dependencies on its parents. These conditional distributions are parameterized by θ_G. The joint distribution of all the variables corresponds to the product of all those conditional distributions.

For example, we can use a Bayesian network to represent a joint distribution over the possible symptoms and diseases that a person may have. Figure 3.1 illustrates a simple Bayesian network with binary variables for fever, headache, and cold. Tables 3.1, 3.2, and 3.3 describe the conditional distribution of each variable. Here, θ would correspond to the numerical values in those tables.

Bayesian networks capture two kinds of information about a problem, structural and probabilistic. This implies two kinds of learning for Bayesian networks. Learning

Table 3.2: Conditional distribution for fever: Pr($Fever$)

Fever	
T	F
0.3	0.7

Table 3.3: Conditional distribution for cold: $\Pr(Cold|Headache, Fever)$

		Cold	
Headache	Fever	T	F
F	T	0.4	0.6
F	F	0.01	0.99
T	T	0.7	0.3
T	F	0.98	0.02

structure of a Bayesian network assuming that we know the value of each node and try to find the edges between nodes. The other problem is to estimate the values of the nodes when the structure of the network is known.

3.1.1 Learning Structure in Bayesian Networks

Tong and Koller [TK01] showed how to learn the structure of a Bayesian network using interventional active learning. Their idea is to intervene in the domain by setting the values of certain variables. In graphical models such as Bayesian networks, an intervention corresponds to cutting the edges coming into the node. Therefore, the authors assume some subset Q of variables are the query variables, and the learner can select a particular instantiation q, called the query. The result of the query is called the response, a randomly sampled instance x of all the non-query variables on condition $Q = q$. Given a prior density p, and a response x from a query q, one can use standard Bayesian updating of the parameters, and update only the Dirichlet distribution of updatable nodes. Therefore, not only they update the distribution based on the interventional data, but also they actively select the instance to query next. The authors used an active learner l, a function that selects a query based on its current distribution over the graph $\mathbf{G}$ and parameters θ_G. Each time, the learner takes the result of the query and uses it to update its distribution over the graph and its parameters. In order for the active learner l to decide on the next query, the authors defined a measure for the quality of the distribution over the graph and its parameters. This measurement evaluates the extent to which each instance can improve the quality of the distribution. To do that, they chose the edge entropy as the loss function and defined the expected posterior loss of the query as following:

$$E\mathbf{x}PLoss(P(\mathbf{G}, \theta_G|\mathbf{Q} := q)) = E_{x\sim P(\mathbf{X}|\mathbf{Q}:=q)}Loss(P(\mathbf{G}, \theta_G|\mathbf{Q} := q, \mathbf{x}))$$

Hence, the expected posterior loss for each candidate $\mathbf{Q} := q$ is evaluated, and the point that has the lowest is selected.

With an experiment on three networks, Cancer, Asia, Troubleshooter, Tong and Koller showed that using an active learning method outperforms both random sampling and uniform querying, where there is a setting for the query nodes from a uniform distribution.

3.1.2 Parameter Estimation in Bayesian Networks

In another work, Tong and Koller [TK00a], showed how to learn parameters of a Bayesian network when the structure of the network is known. This work is an application of selective active learning.

Given a Bayesian network structure G and a prior distribution $p(\theta)$ over the parameters of G, the goal is to estimate the parameters in the Bayesian networks. In classical learning, data instances are randomly sampled from some underlying distribution, but an active learner can requests certain types of instances. Assume that some subsets C of variables are controllable, meaning that the learner can select a subset of variables $Q \subset C$, and a particular instantiation q to Q. A myopic active learner l selects a query $Q = q$, takes the result x of the query, and updates its distribution to get the posterior.

There are two important questions. One is: given the instance x , how should the distribution be updated. Obviously, when getting the resulting instance x, the node Q itself and its parents cannot be updated. The nodes that are *updatable* are the nodes which are not ancestors of the node queried. The other question is: which instantiation of Q should be queried based on the model learned so far? The authors defined a measurement for the quality of the learned model which can evaluate how different instances would improve the quality of the learned model. This is based on *Bayesian point estimation* i.e a distribution p over all of the parameters of the model. If a model $\tilde{\theta}$ is chosen instead of the true model $\theta^\star$, then there is some loss $\text{Loss}(\tilde{\theta}\|\theta^\star)$. The goal is to minimize this loss. The true model $\theta^\star$ is not known but the posterior distribution represents the optimal belief about different possible values of $\theta^\star$. So, the risk of a $\tilde{\theta}$ with respect to p

is defined as follows:

$$E_{\Theta \sim p(\theta)}[Loss(\Theta \| \tilde{\theta})] = \int_{\theta} Loss(\theta \| \tilde{\theta}) p(\theta) d\theta$$

Now, the Bayesian point estimate is defined as the value of $\tilde{\theta}$ that minimizes this risk. Risk$p(\theta)$, the risk of a density p, is defined as the risk of the optimal $\tilde{\theta}$ with respect to p. An instance x will be chosen as the query instance if the risk of updating p with instance x is the lowest. The instance x is not known before query, but it is sampled from a distribution by the query. So, the expected posterior risk is:

$$E\mathbf{x}PRisk(p(\theta)|\mathbf{Q} = q) = E_{\theta \sim p(\theta)} E_{x \sim P_{\theta}(\mathbf{X}|\mathbf{Q}=q)} Risk(p(\theta|\mathbf{Q} = q, x))$$

where $(p(\theta|\mathbf{Q} = q, x))$ denotes the distribution $p^{'}(\theta)$.

After evaluating the expected posterior risk for each candidate $\mathbf{Q} = q$, the instance with the lowest expected posterior risk is selected.

Experimental results in different networks show that active learning provides a substantial improvement over learning by random sampling.

3.2 Active Model Selection

Another application of active learning is active model selection meaning how active learning helps to select one model from a set of possible models. This work differs from classical learning in that the learner does not start with a training sample, but has resources that can be used to get information to help identify the optimal model. The learner has a fixed budget of model probes to identify the model with the highest expected accuracy among a set of possible models. Each probe evaluates the specified model on a random indistinguishable instance.

As a simplified example of active model selection, consider the coins problem: suppose we are given n coins with unknown head probability, and after a known fixed number of flips, we want to find the coin with the highest head probability. In this example, the budget is the number of flips. The question is "Which coin is the winner(the coin with the highest head probability)?".

Madani et al., [MLR04] assigned priors over coins quality and by defining a measure of regret over choosing a coin. They identified a strategy (which coin to flip at a given

time point) with minimum regret. They showed that this problem is in PSPACE, but NP-hard under different coin costs. They considered some restriction such as unit cost and/or unit model distribution, and also two properties: namely, the magnitude of the mean particularly of each coin and the spread of its density.

Several algorithms/heuristics have been proposed:

Round-Robin: This algorithm flips the coin $i = (t - 1 \bmod n) + 1$ at time $t = 1, 2, ...,$

Random Algorithm: It picks a coin uniformly at random and flips it.

Constant-budget Algorithm: It considers a small constant k (independent of the number of coins and also the budget) as a smaller budget and finds the optimal strategy for budget k and flips the best coin according to this.

Greedy Algorithm: Constant-budget algorithm with $k = 1$.

Allocational Algorithm: This algorithm gives a number of flips to each coin and without considering when to flip the coin, it flips the coin as long as it has allocated a positive number of flips.

Interval Estimation: It flips the coin with the highest reasonably likely performance, as the top of the 95 percentage confidence interval, which is the sum of the current mean of the coin and a multiple $\gamma = 1.96$ of the standard deviation of the distribution $STD(\theta)$ over the coin's head probability. Therefore, at each time it flips the coin which gives the highest confidence interval.

Biased-Robin: is similar to the Round-Robin algorithm, but it keeps flipping the coin as long as it gives head.

The authors investigated the empirical performance of these algorithms on various problem instances, and showed that their simple Biased-Robin algorithm significantly outperforms other algorithms when the costs and priors are identical. However, it does not take into account either priors or the budget which are two important factors. When the remaining budget is low, the strategy chosen by Biased-Robin algorithm does not choose the optimal solution.

3.3 Selective Sampling Using Query by Committee Algorithm

The Query by Committee algorithm (QBC algorithm) is another method that does active learning. This method filters the most informative points to be queried from a random set of inputs. Freund et al. [FSST77] proved that if the QBC algorithm for learning the perceptron concept class achieves high information gain, then the generalization error decreases exponentially with the number of queries. Assume $\overrightarrow{\mathbf{X}} = \{x_1, x_2...\}$ a Euclidean space $\Re^d$ with a distribution D and some *concept class* C, which is a set of concepts $c : \mathbf{X} \rightarrow \{0, 1\}$. The version space generated by the sequence of the labeled examples $\langle \overrightarrow{\mathbf{X}}_{1...m}, c(\overrightarrow{\mathbf{X}}_{1...m}) \rangle$ is defined as the set of all concepts $c^{'} \in C$ such that $c^{'}(x_i) = c(x_i)$. Let V_i correspond to the first i labeled examples: $V_i = V(\langle \overrightarrow{\mathbf{X}}_{1...i}, c(\overrightarrow{\mathbf{X}}_{1...i}) \rangle)$. The author used $\langle \overrightarrow{\mathbf{X}}, c(\overrightarrow{\mathbf{X}}) \rangle = \{\langle x_1, c(x_1) \rangle, \langle x_2, c(x_2) \rangle, ...\}$ as a sequence of labeled examples which is generated by applying each concept c to each input x. The QBC algorithm proceeds in two iterations. It has two parameters $\epsilon > 0$ and $\delta > 0$ as the maximal tolerable prediction error and the desired reliability, respectively. The parameters are used to define the termination criterion for the algorithm. The algorithm has access to three oracles namely: *Gibbs, Sample,* and *Label* as inputs. By using the *Sample* oracle the algorithm gets unlabeled example $x \in \mathbf{X}$, and computes predictions by *Gibbs* oracle. A call to the *Label* oracle for an input x returns $c(x)$, the correct label of input x.

The algorithm starts by initializing a counter n, of calls to oracle *Label* to zero, and the version space V_0 to the complete concept class C. The goal is to output a prediction hypothesis $h : \mathbf{X} \rightarrow \{0, 1\}$. The expected error of the algorithm is defined as the probability of $h(x) \neq c(x)$. At the first step, the algorithm calls *Sample* in order to get an unlabeled example $x \in \mathbf{X}$ at random according to the distribution D. Then, by calling $Gibbs(V_n, x)$ twice (a committee of two members), it gets two predictions for the label of x. Now, if the predictions are equal it means the algorithm has confidence in the label, and so rejects that example and returns to the first step. However, in the case of different predictions, it calls *Label(x)* to get $c(x)$, increases n by 1, and sets V_n to be all of the concepts in V_{n-1} such that $c^{'}(x) = c(x)$. The algorithm terminates when the number of consecutive rejected examples is more than t_n, where

$$t_n = \frac{1}{\epsilon} ln \frac{\pi^2 (n+1)^2}{3\delta}$$

As we can see, t_n does not depend on the property of the concept class, however the performance of the algorithm does. Therefore, the algorithm can work without any prior information on the concept class. The authors showed that using the query by committee algorithm, they can query unlabeled instances with higher expected information gain than choosing examples randomly.

3.4 Application to Text Classification

Theoretical advantages and empirical success of SVMs in a lot of real-world learning tasks make them an attractive choice as a machine learning method for active learning. Applying active learning for SVMs has many applications in different fields such as image retrieval, text classification, hand written digit recognition and so on. Following we review support vector machines active learning with application to text classification.

Tong and Koller [TK00b], proposed an algorithm for pool-based active learning with support vector machines classifiers. They considered Transductive Support Vector Machines (TSVMs), where there is a pool of labeled and unlabeled instances and the goal is to find the best hyperplane considering all instances, including the unlabeled ones (Figure 2.8 and 2.10 show SVMs and transductive SVMs).

As many active learning tasks, the common question is: which unlabeled instance should be queried next? Each query reduces the size of the version space. Based on the label of the selected instance, the set of hyperplanes in the feature space will be restricted to those that classify that selected instance correctly. Now, the question is: how reduce the version space as quickly as possible?

Tong and Koller proposed to query the unlabeled instance which comes as close as possible to halving the current version space. Their idea involves a geometrical problem by considering a space called parameter space which is the dual of the feature space. (The existence of the parameter space is shown in [Vap98]). Every point in the feature space corresponds to a hyperplane in the parameter space and vice versa. Instead of evaluating the point to query next in the feature space, they consider the corresponding hyperplane to that point in the parameter space that cuts the current version space (a subset of parameter space) in two regions with sides as close as possible. Since it is not practical to compute the size of the new version space, they used some approximation. They proposed three methods called Simple Margin, MaxMin Margin, and Ratio Margin.

The idea in Simple Margin is to find the closest hyperplane to the center of the

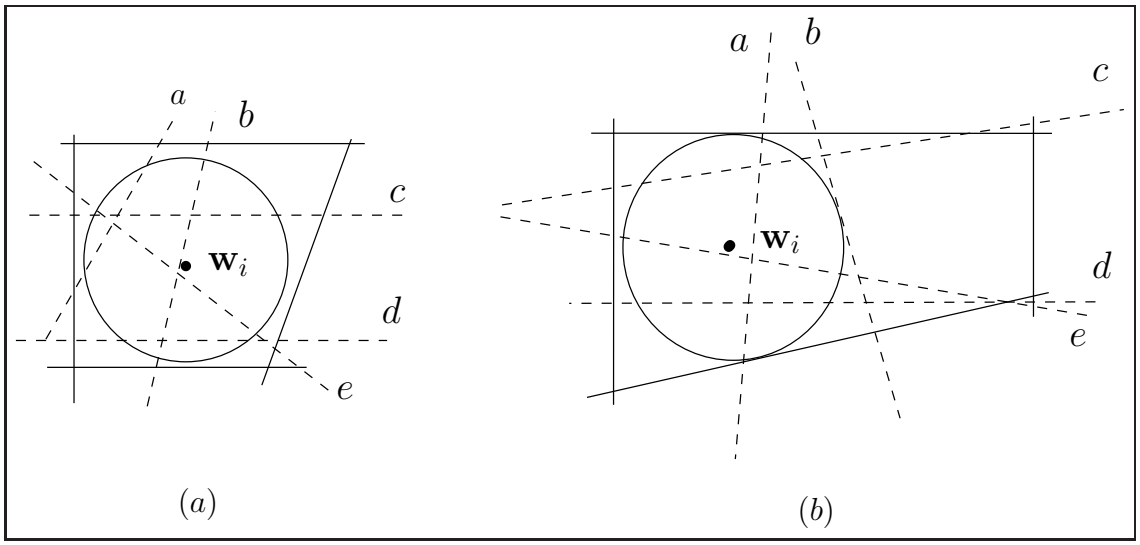

Figure 3.2: (a) Simple Margin will query b (b) Simple Margin will query a .

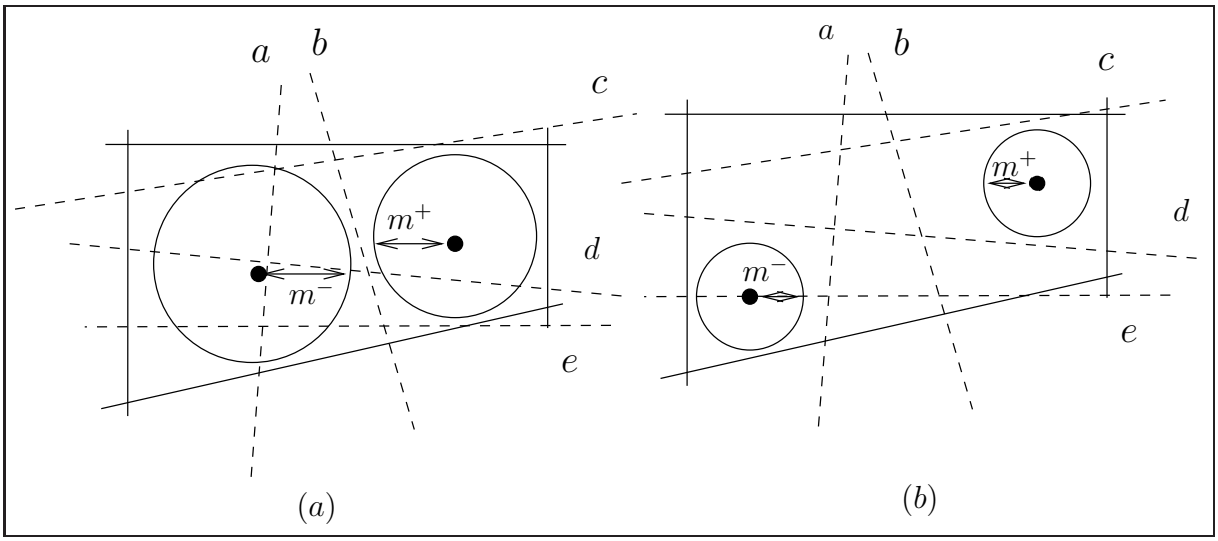

Figure 3.3: (a) MaxMin Margin will query b. The two SVMs with margin $m-$ and $m+$ for b are shown. (b) Ratio Margin will query e. The two SVMs with margin $m-$ and $m+$ for e are shown.

largest hypersphere which fits in the current version space (see Figure 3.2 [TK00b]) . The advantage is that it needs to find only one hypersphere for all hyperplanes, and therefore, it is computationally faster. However, this idea does not work well when the version space is thin and long or if it is not symmetric. Therefore, they proposed another method called MaxMin approximation. Every hyperplane divides the version space into two half spaces. The idea is to find the largest hyperspheres which fit into each of the two spaces. For each pair of hyperspheres the smaller one is chosen and finally among all chosen hyperspheres the largest hypersphere is returned (see Figure 3.3, (a) [TK00b]). Depending on the shape of the version space, the two hyperspheres which fit into the spaces divided by hyperplanes may be small. Under this situation the heuristic may not give a correct result. In this case, the ratio of the radii of the hyperspheres is considered. This method is called Ratio Margin (see Figure 3.3,(b) [TK00b]). These three heuristics find the unlabeled instance in feature space which approximately halves the current version space. In the inductive case, after performing some number of queries, the learner algorithm finds the SVM classifier which is trained on the labeled data. In the transductive case, the algorithm is trained on both labeled and unlabeled data.

Depending on the application, considering the cost of querying and computing time, either of Simple Margin or MaxMin Margin is suitable. Empirical evaluation for both real-word text classification domains: the Reuters-21578 data set and Newsgroups data set show that their work outperforms the random sample method which randomly chooses the next query point.

While reducing the version space, Tong and Koller considered the version space corresponding only to labeled instances. In the next chapter, we see how we can take advantage of the presence of all unlabeled instances.

Chapter 4

Semi-supervised Support Vector Machines

As in semi-supervised learning, the learner has access to both labeled and unlabeled data, the aim of semi-supervised SVMs is to take advantage of the presence of all unlabeled data, and find the best classifier with maximum margin based on all available data.

This chapter is divided into three sections. In the following two Sections, we review different approaches for semi-supervised support vector machines by formulating the problem as different optimization problems, and also by using different loss functions. Section 4.3 describes active learning with semi-supervised support vector machines, and proposes two new algorithms.

4.1 Optimization Approaches

As mentioned before, support vector machines can be formulated as some optimization problems. Writing the margin of separation in 1-norm, the problem of maximum margin can be formulated as a Robust Linear Programming (RLP) problem:

$$\begin{array}{ll} \min_{\mathbf{w},b} & \|\mathbf{w}\|_1 + C\sum_{i=1}^{l}\epsilon_i \\ \text{s.t.} & y_i(\mathbf{w}\cdot\mathbf{x}_i - b) + \epsilon_i \geq 1\,, \epsilon_i \geq 0 \quad i = 1, ..., l \end{array}$$

where $\|\mathbf{w}\|_1 = \sum_{j=1}^{n}|w_j|$ and ϵ_i is a slack term such that if the point is misclassified, then $\epsilon_i > 0$. Hence, we can write the problem as the following RLP:

$$\begin{array}{ll} \min_{\mathbf{w},b,\epsilon,s} & \sum_{j=1}^{n} s_j + C \sum_{i=1}^{l} \epsilon_i \\ \text{s.t.} & y_i(\mathbf{w} \cdot \mathbf{x}_i - b) + \epsilon_i \geq 1 \,, \epsilon_i \geq 0 \quad i = 1, ..., l \\ & -s_j \leq w_j \leq s_j \qquad\qquad\qquad\quad j = 1, ..., n \end{array}$$

Considering the margin in Euclidean distance leads to the 2-norm and the following quadratic programming optimization (QPO):

$$\begin{array}{ll} \min_{\mathbf{w},b,\varepsilon} & \frac{1}{2}\|\mathbf{w}\|^2 + C \sum_{i=1}^{l} \varepsilon_i \\ \text{s.t.} & y_i(\mathbf{w} \cdot \phi(\mathbf{x}_i) + b) + \varepsilon_i \geq 1 \,, \varepsilon_i \geq 0 \quad i = 1, ..., l \end{array} \tag{4.1}$$

Depending on the application, the quadratic problem can be very large and training an SVM requires a lot of time. There are some optimization approaches which make the problem easier to solve. For example Sequential Minimal Optimization (SMO) is one that breaks the problem into a series of smaller possible quadratic problems and tries to solve them [Pla99].

We now consider semi-supervised SVMs for problems with unlabeled data. One formulation for semi-supervised SVMs corresponds to mixed integer programming (MIP). Following we review Semi-Supervised Support Vector Machines (S^3VM) by Bennett and Demiriz [BD98].

Semi-Supervised Support Vector Machines

Bennett and Demiriz used some mathematical programming approaches to semi-supervised learning and constructed a classifier by using both a training set of labeled data and a working set of unlabeled data for some classification tasks. This work fits in transductive semi-supervised learning. The idea is to select the classifier on training data which has the most margin on unlabeled data. The authors assume linearly separable data and select the classifier for labeled data which has few unlabeled points in the margin. The classifier is penalized for each unlabeled point which is in the margin. Considering the optimization problem 4.1 and giving a penalty function for the misclassified unlabeled points, the following optimization problem is obtained:

$$\begin{array}{ll} \min_{\mathbf{w},b,\epsilon} & C \left(\sum_{i=1}^{l} \epsilon_i + \sum_{j=l+1}^{l+k} g(\mathbf{w} \cdot \mathbf{x}_j - b) \right) + \|\mathbf{w}\| \\ \text{s.t.} & y_i(\mathbf{w} \cdot \mathbf{x}_i - b) + \epsilon_i \geq 1 \,, \epsilon_i \geq 0 \quad i = 1, ..., l \end{array}$$

where the constant $C > 0$ is a penalty parameter and $g(\alpha)$ is a margin penalty function on unlabeled points. The authors defined the function g for hard margin (no unlabeled

points in the margin) as follows:

$$g_\infty(\alpha) = \begin{array}{ll} \infty & \text{for } -1 < \alpha < 1 \\ 0 & \text{otherwise} \end{array}$$

Depending on how the unlabeled points are penalized, the problem is formulated as a linear program, or as a quadratic program with more constraints. To formulate the semi-supervised SVM, the authors added two constraints which calculate the misclassification error for each unlabeled point. One constraint calculates the error for the unlabeled points which are supposed to be in class 1, but are in class -1 and another constraint for the reverse case. By adding an equilibrium constraint, one of the two misclassification errors for each unlabeled point will be forced to be zero. Hence, the following optimization problem is obtained:

$$\begin{array}{ll} \min_{\mathbf{w},b,\epsilon,\eta,z} & C\left(\sum_{i=1}^{l}\epsilon_i\right) + \|\mathbf{w}\| \\ \text{s.t.} & y_i(\mathbf{w}\cdot\mathbf{x}_i - b) + \epsilon_i \geq 1\,, \epsilon_i \geq 0 \quad i = 1,...,l \\ & +(\mathbf{w}\cdot\mathbf{x}_j - b) + \eta_j \geq 1\,, \eta_j \geq 0 \quad j = l+1,...,l+k \\ & -(\mathbf{w}\cdot\mathbf{x}_j - b) + z_j \geq 1\,, z_j \geq 0 \quad j = l+1,...,l+k \\ & \eta_j \cdot z_j = 0 \qquad\qquad\qquad\quad j = l+1,...,l+k \end{array}$$

The hard margin requirement can be very strong. So the problem can be relaxed by moving the last constraint to the objective and using it as a penalty function. Result is the following non-convex quadratic optimization problem:

$$\begin{array}{ll} \min_{\mathbf{w},b,\epsilon,\eta,z} & C\left(\sum_{i=1}^{l}\epsilon_i + \sum_{j=l+1}^{l+k}\eta_j\cdot z_j\right) + \|\mathbf{w}\| \\ \text{s.t.} & y_i(\mathbf{w}\cdot\mathbf{x}_i - b) + \epsilon_i \geq 1\,, \epsilon_i \geq 0 \quad i = 1,...,l \\ & +(\mathbf{w}\cdot\mathbf{x}_j - b) + \eta_j \geq 1\,, \eta_j \geq 0 \quad j = l+1,...,l+k \\ & -(\mathbf{w}\cdot\mathbf{x}_j - b) + z_j \geq 1\,, z_j \geq 0 \quad j = l+1,...,l+k \end{array}$$

In order to define the penalty function we consider two cases. First, if the unlabeled point x_j is outside of the margin, then either η_j or z_j is zero. Second, if the unlabeled point x_j is inside the margin, then $\eta_j = 1-\gamma$, and $z_j = \gamma + 1$, since $\gamma = w\cdot x_j - b$. Therefore, the penalty function is defined as:

$$g(\gamma) = \begin{array}{ll} 1-\gamma^2 & \text{for } -1 < \gamma < 1 \\ 0 & \text{otherwise} \end{array}$$

Another choice of the penalty function is the idea proposed by Vapnik [Vap98]. It calculates the minimum of the two misclassification errors, and assigns the class to the unlabeled point which results in the smallest error. This leads to the following semi-supervised support vector machine problem:

$$\begin{array}{ll} \min_{\mathbf{w},b,\epsilon,\eta,z} & C\left(\sum_{i=1}^{l}\epsilon_i + \sum_{j=l+1}^{l+k}\min\{\eta_j, z_j\}\right) + \|\mathbf{w}\| \\ \text{s.t.} & y_i(\mathbf{w}\cdot\mathbf{x}_i - b) + \epsilon_i \geq 1\,, \epsilon_i \geq 0 \quad i = 1, ..., l \\ & +(\mathbf{w}\cdot\mathbf{x}_j - b) + \eta_j \geq 1\,, \eta_j \geq 0 \;\; j = l+1, ..., l+k \\ & -(\mathbf{w}\cdot\mathbf{x}_j - b) + z_j \geq 1\,, z_j \geq 0 \;\; j = l+1, ..., l+k \end{array}$$

To solve the problem practically, the authors used some "Mixed-Integer Programming"(MIP) formulation. The idea is to add a decision variable $d_j = \{0, 1\}$ for each point in the working set. If the decision variable is one, it means that the corresponding point is in the class 1 and if it is zero, it means that the corresponding point is in the class -1. Adding the variable d_j, the following mixed-integer optimization problem is obtained.

$$\begin{array}{lll} \min_{\mathbf{w},b,\epsilon,\eta z,d} & C\left(\sum_{i=1}^{l}\epsilon_i + \sum_{i=l+1}^{l+k}(\eta j + z_j)\right) + \|\mathbf{w}\| & \\ \text{s.t.} & y_i(\mathbf{w}\cdot\mathbf{x}_i - b) + \epsilon_i \geq 1\,, \epsilon_i \geq 0 & i = 1, ..., l \\ & +(\mathbf{w}\cdot\mathbf{x}_j - b) + \eta_j + M(1 - d_j) \geq 1\,, \eta_j \geq 0 & j = l+1, ..., l+k \\ & -(\mathbf{w}\cdot\mathbf{x}_j - b) + z_j + Md_j \geq 1\,, z_j \geq 0 & j = l+1, ..., l+k \\ & d_j = \{0, 1\} & \end{array}$$

If the variable $d_i = 0$, then the constant M is chosen so large that $\eta_j = 0$ is feasible for any optimal $\mathbf{w}, b$. Similarly, $z_j = 0$ for the case $d_i = 1$. This problem can be solved by using commercial integer programming packages.

Bennett and Demiriz considered some problems with about 50 unlabeled data points, and CPLEX was sufficient to solve those problems. However, a lot of real world problems have a larger set of unlabeled data and the computational complexity is exponential in the size of the unlabeled set. Another optimization approach is semi-definite programming (SDP), which leads to a convex optimization problem with a polynomial complexity in the size of the unlabeled set [BC03, XWSS06, XNLS04].

4.2 Different Loss Functions

Given a training set $(\mathbf{x}_i, y_i)_{i=1...l}$ with $(\mathbf{x}_i, y_i) \in \Re^n \times \{-1, 1\}$, an SVM finds a decision function $f_\theta(\mathbf{x}) = \mathbf{w}.\Phi(\mathbf{x}) + b$, where $\Theta = (\mathbf{w}, b)$ are the parameters of the model, and $\Phi(.)$ is a feature map which is usually defined by a Mercer kernel. The standard SVM criterion uses the hinge loss to penalize the misclassified instances. Using the hinge loss function in the SVM optimization problem results in the following:

$$\min_{\mathbf{w}} \ \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{l} H_1(y_i f_\theta(\mathbf{x}_i)) \tag{4.2}$$

where $H_s(z) = \max(0, s - z)$, the subscript s indicates the position of the hinge point.

As we showed in Section 2.3.4, the solution $\mathbf{w}$ is a linear combination of the support vectors. Since the number of support vectors scales linearly with the number of points [Ste03], and also the complexity of SVMs grows fast with the number of support vectors, SVMs cannot deal with problems with a large number of examples. One idea is to reduce the number of support vectors. Collobert et al. [CSWB06] described two non-convex algorithms (Ramp Loss SVMs/TSVMs) for both SVMs and Transductive SVMs. Their algorithms improve scalability by using non-convexity over convexity. The idea is to prevent misclassified examples from being support vectors. So, instead of using the convex hinge loss, they optimized a non-convex loss function using the "Concave-Convex Procedure" (CCCP) [YR03], which is related to the "Difference of Convex"(DC) approach developed by Yuille and Rangarajan [Thi94]. Their idea is as follows:

Suppose that the Hinge loss function is differentiable for $z \in [1 - \epsilon, 1 + \epsilon]$, a small interval near the hinge point. By differentiating (4.2), the minimum $\mathbf{w}$ is:

$$\mathbf{w} = -C\sum_{l=1}^{l} y_i H_1'(y_i f_\theta(\mathbf{x}_i))\Phi(\mathbf{x}_i) \tag{4.3}$$

Training examples which are in the flat area of $z > 1 + \epsilon$ do not become support vectors since $H_1(z) = 0$, but those in the margin of $z < 1 - \epsilon$ will become support vectors since $H_1(z) = 1$ (see Figure(2.7)). Therefore, the Ramp loss (illustrated in Figure 4.1) is defined as a function which is flat for scores z smaller than a predefined value $s < 1$.

$$R_s(z) = H_1(z) - H_s(z)$$

Using the Ramp loss function R_s instead of H_1 in Equation 4.3 guarantees that examples

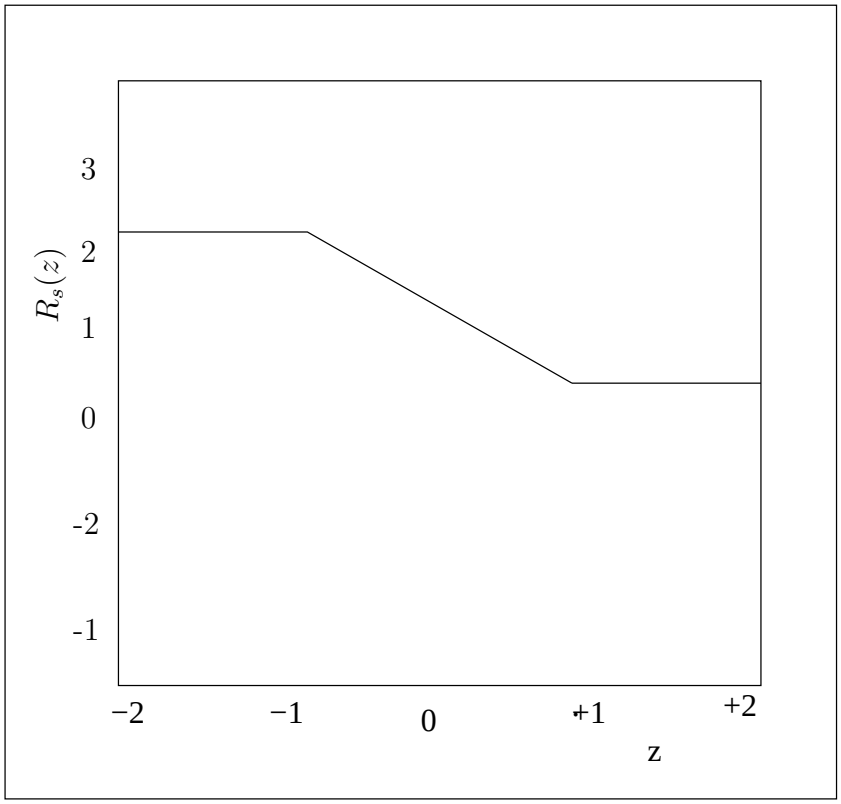

Figure 4.1: Ramp loss function.

with the score $z < s$ do not become support vectors anymore. This results in the following:

$$\min_{\mathbf{w}} \ \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{l} R_s(y_i f_\theta(\mathbf{x}_i)) \tag{4.4}$$

Following we see the CCCP procedure which allows to minimize a non-convex function.

Concave-Convex Procedure : Assume the loss function $J(\Theta)$ can be written as the sum of convex and concave parts. The CCCP procedure approximates the concave part by its tangent and minimizes the resulting convex function. This procedure has two properties. First, the cost $J(\Theta)$ decreases after each update of Θ. Second, the procedure converges (shown by Yuille and Rangarajan [YR03]). Hence, Equation 4.2 can be written as:

$$\frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{l} H_1(y_i f_\theta(\mathbf{x}_i)) - C\sum_{i=l}^{l} H_s(y_i f_\theta(\mathbf{x}_i)) \tag{4.5}$$

The Ramp Loss SVM algorithm is computed in three steps. First, by reformulating the first two terms of Problem 4.5 into its dual representation, it finds the variable α of the dual problem. Second, it computes the variable b from the primal problem (as discussed in Section 2.3.4). Finally, it applies the CCCP procedure for solving the last term in Problem 4.5.

TSVMs are similar to hinge loss SVMs, but the last term uses the Ramp loss for unlabeled examples. Given the unlabeled instances $\mathbf{x}_{l+1}, ..., \mathbf{x}_{l+u}$, the corresponding non-convex TSVMs can be written as:

$$\frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{l} H_1(y_i f_\theta(\mathbf{x}_i)) + C^*\sum_{i=l+1}^{l+u} T(f_\theta(\mathbf{x}_i)) \tag{4.6}$$

The algorithm for TSVMs is the same as SVMs where both possible labels are given to each unlabeled example.

The above formulations for semi-supervised SVMs assume that the unlabeled data clusters more or less according to the classes (this is referred as the *cluster assumption* [See02]). When the cluster assumption holds, semi-supervised SVMs can be very effective. However, if the cluster assumption does not hold, performance may be worse than supervised SVMs. In practice, the cluster assumption often holds to some extent, but not completely. Hence, this is why a parameter C^* is used to scale the importance of the

unlabeled points in the formulation of semi-supervised SVMs (see Equation 4.6). How to set C^* in a principled way is still an open problem. Since this is an issue orthogonal to active learning, we assume that somebody has already set C^*.

4.3 Active Learning with Semi-Supervised Support Vector Machines, ALS^{3}VM

Suppose we have a training set of labeled and unlabeled points, and the number of labeled points is too few for support vector algorithms to build a classifier with a reasonable level of performance. However, the learner has the ability to request the label for a certain number of unlabeled points by calling some *oracle*. The goal of active learning with semi-supervised SVMs is to query the unlabeled instance that reveals the most information while taking into account the information already provided by the pool of unlabeled points.

A common approach to picking unlabeled instances that reveal the most information is to look for the instances for which we are the most uncertain given our current belief about the best classification. With support vector machines, this belief corresponds to the hyperplane computed so far. The distance between each unlabeled instance and this hyperplane is often used as an indicator of the uncertainty for the class of each unlabeled instance. This is more less the idea underlying the heuristics proposed by Tong and Koller. While this is a good approach it is important to make sure that the best hyperplane possible is used.

In a supervised setting, this hyperplane is only based on the labeled data (Figure 1.1), but in a semi-supervised setting it can be based on both labeled and unlabeled data (Figure 1.2). When the cluster assumption holds, it has been demonstrated that better hyperplanes can be obtained by making use of unlabeled data [BD98]. Therefore, in this section, we propose two new algorithms called MinMax Distance (Algorithm 1), Pruning (Algorithm 2) that consider unlabeled data.

Algorithm 1 and Algorithm 2 are based on the definition of the version space and therefore we limit our discussion to hard margin SVMs, i.e both labeled and unlabeled data are linearly separable in the feature space. It is possible to modify any kernel so that the data will be linearly separable in the feature space [STC99].

4.3.1 MinMax Distance Algorithm

Let $S = \{(\mathbf{x}_1, y_1), ..., (\mathbf{x}_l, y_l)\}$ be the set of training instances, $U = \{\mathbf{x}_1, ...\mathbf{x}_u\}$ be a pool of unlabeled instances, where $\mathbf{x}_i$ is some vector in $\mathbf{X} \in \Re^d, y_i \in \{-1, +1\}$, l and u are the number of labeled and unlabeled instances, respectively. Given O, the number of calls to an oracle, the objective is to construct the best classifier for both labeled and unlabeled data by O calls to the oracle.

A Semi-supervised SVM (S^3VM) finds the hyperplane that separates the training data by finding the maximal margin based on both labeled and unlabeled data. However, giving the ability of being active to the learner allows it to pick instances to be labeled that will be more informative and may speed up the learning process and/or yield better accuracy.

As explained in Chapter 2, mapping data in the feature space makes data linearly separable. The set of all consistent hypothesis in the feature space is referred as version space. Obviously, unlabeled data instances will be classified differently depending on the hyperplane from the version space that the learner chooses. Figure 4.2 shows a feature space for a two-class problem with labeled and unlabeled instances.

As we reviewed in Section 3.4, in order to find the best unlabeled instance in the feature space, Tong and Koller [TK00b] looked for the hyperplane in the parameter space (the dual of the feature space) that halves the current version space as much as possible. Figure 4.3 illustrates a simple example of a version space with five support vectors and three unlabeled instances. The solid lines correspond to the support vectors of training data, and each dashed line corresponds to an unlabeled instance in the feature space. Obviously, based on the label of the selected unlabeled instance, the set of separating hyperplanes in the feature space will be restricted to those which classify that instance correctly. Therefore, the size of the version space will be reduced with each query to the oracle. One approach in active learning is to query an unlabeled instance that reduces the size of the version space as much as possible (Tong and Koller [TK00b], Seung et. al. [SOS92].) In order to mimic reduction by bisection, the best approach is to choose a hyperplane that approximately halves the version space. To achieve this, Tong and Koller proposed two heuristics, Simple Margin and MaxMin Margin. These two heuristics are supervised in nature since they are based on the information of version space which is based only on labeled instances.

Our approach is to consider SVM in a semi-supervised setting and to take advantage of the presence of all unlabeled instances. Considering both possible labels for each

unlabeled instance, each dashed line divides the version space into two regions. As a result the version space is divided in many sub-regions, each delimited by a combinations of dash-lines corresponding to some labeling of the unlabeled data that keeps the data linearly separable (see Figure 4.3). Semi-supervised SVMs consider each sub-region and return the solution of the optimization Problem 4.6 which is the $\mathbf{w}$ that correspond to the center of the largest hyperball that fits in any of those sub-regions. In Figure 4.3, the largest hyperball fits in region 3. While we could take the $\mathbf{w}$ returned by the optimization Problem 4.6 as the best current solution, this choice is based on the cluster assumption and therefore may not always be reasonable. Instead, we choose a $\mathbf{w}$ which is more stable without making any assumption. If we knew the labels of all instances, the version space would be reduced to one of the sub-regions in Figure 4.3 and the optimal $\mathbf{w}$ would be the center of the largest hyperball that fits in that sub-region. Since we do not know the labels, a conservative approach would be to pick the best $\mathbf{w}$ of the sub-region that is closest to all other sub-regions and to query instances closest to that $\mathbf{w}$.

Let W be the set of all $\mathbf{w}$'s that are solutions to Equation 4.4 for each sub-region. We define a notion of *regret* to estimate how far $\mathbf{w}$ is from the optimal $\mathbf{w}^*$ (had we known all labels). Intuitively, when picking a $\mathbf{w}$ we may regret its induced classification when it misclassifies some instances in comparison to the optimal classification given by $\mathbf{w}^*$. Ideally, we would measure this regret to be the proportion of data instances that are classified differently by $\mathbf{w}$ and $\mathbf{w}^*$. To simplify the computation of regret, we consider instead the Euclidean distance between $\mathbf{w}$ and $\mathbf{w}^*$ (we will return to regret based on misclassification later on).

Definition 4.3.1 *We define the regret of choosing* $\mathbf{w}$ *as the distance between* $\mathbf{w}$ *and* $\mathbf{w}^*$*:*

$$d_{\mathbf{w}} = \|\mathbf{w} - \mathbf{w}^*\|_2.$$

Since $\mathbf{w}^*$ is unknown, we consider the maximum possible regret with respect to all $\mathbf{w}$'s in W as an upper bound on regret.

Definition 4.3.2 *We define the maximum regret of choosing* $\mathbf{w}$ *as the maximum distance between* $\mathbf{w}$ *and any other solution* $\mathbf{w}' \in W$*:*

$$d_{\mathbf{w}} = \max_{\mathbf{w}' \in W} \|\mathbf{w} - \mathbf{w}'\|_2.$$

We pick the $\mathbf{w}$ that minimizes the maximum regret.

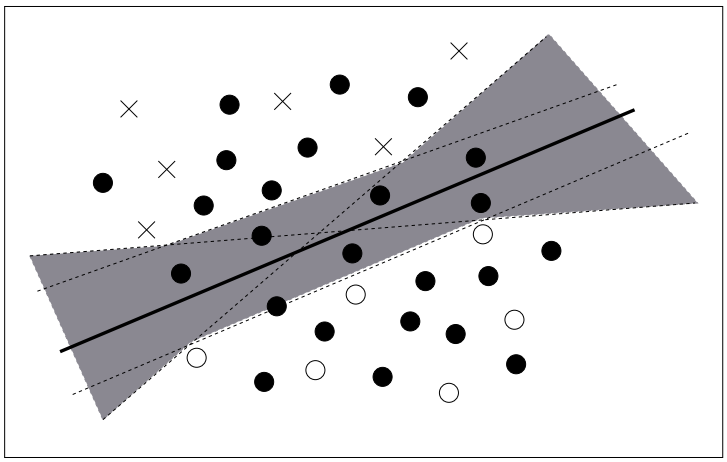

Figure 4.2: Feature space. Crosses and circles represent the two possible classes for training data and the solid circles represent unlabeled instances.

Definition 4.3.3 *We define the* $\mathbf{w}$ *with minimax regret to be the* $\mathbf{w}$ *that minimizes maximum regret.*

Since we are using the concept of minimax regret which arises from distances between $\mathbf{w}$'s, we call our method MinMax Distance algorithm. This algorithm queries the closest unlabeled instance to $\mathbf{w}$ with minimax regret, adds it and its true label to the training set, and updates the current hyperplane (the solution to the S^3VM algorithm). The closest unlabeled instances correspond to the support vectors. The pseudocode for the MinMax Distance algorithm is illustrated in Algorithm 1.

In real-world problems, there are a lot of unlabeled instances and so we end up with a large number of sub-regions. We are only interested in those unlabeled instances have corresponding dashed lines cutting through the version space. Assuming that the data is linearly separable, we can prune the forced unlabeled instances, i.e prune those which are forced to get either label $+1$ or -1 (Line 5 in Algorithm 1). To do that, we first create a *model*, a region from the constraints of labeled instances, and then for each unlabeled instance we check whether that instance is consistent with that model. Each labeled instance $\mathbf{x_i}$ has the following corresponding constraint:

$$y_i(\mathbf{w} \cdot \mathbf{x_i} + b) \geq 1$$

We create the model from the feasible region corresponding to the constraints of labeled instances as the following optimization problem:

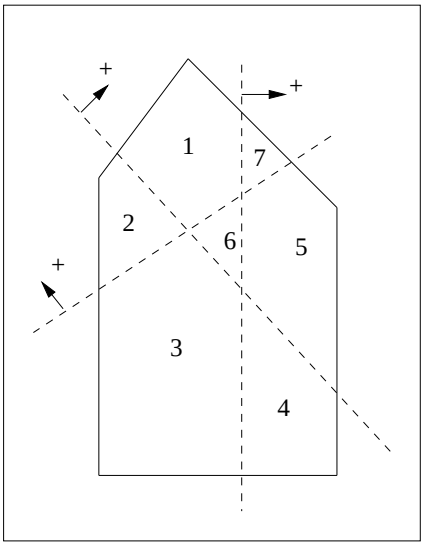

Figure 4.3: Version space for a two-class problem with five support vectors and three unlabeled instances.

$$\begin{aligned} \max_C \quad & C \\ \text{s.t.} \quad & y_i(\mathbf{w} \cdot \mathbf{x_i} + b) \geq C, \;\; i = 1, ..., l \end{aligned} \tag{4.7}$$

As long as C is greater than or equal to zero, then the model is feasible and the labeling is linearly separable.

Now, we are in the stage of pruning the forced unlabeled instances. For each unlabeled instance, we assign both possible labels +1 and -1 to it, create the corresponding constraints, and check if the new model is feasible. If the new model is not feasible with one of the two constraints, then that instance is forced to get the corresponding label of the other constraint.

There are still a lot of unlabeled instances in the feature space, and as a result there are many sub-regions in the version space. Therefore, we consider a subset of k unlabeled instances (Line 7 in Algorithm 1), where $1 \leq k \leq u$. Note that for each combination (subset of k unlabeled instances), not all possible labelings are feasible. So, for each combination, we look at the set of all feasible labelings (Line 9 in Algorithm 1.)

Algorithm 1 MinMax Distance algorithm

Input: labeled and unlabeled instances.
O: the number of calls to the oracle.
k: the number of unlabeled instances selected in each combination.
repeat
  Find $\bar{U}$, the set of all unlabeled instances which are not forced to get one of the possible labels.
  $W \leftarrow \emptyset$
  Let C_k be the set of all k-combinations from set $\bar{U}$.
  for each c element of C_k **do**
    Let F_c be the set of all feasible labelings of c.
    for each element of F_c **do**
      Let $\mathbf{w}$ be the solution of the optimization Problem 4.6
      $W \leftarrow W \cup \mathbf{w}$
    end for
  end for
  Find $\bar{\mathbf{w}} \in W$ that minimizes the maximum regret.
  Find the support vectors of $\bar{\mathbf{w}}$.
  Call oracle to get the true label instance/instances corresponding to support vectors.
  Let $P = \{p_1, ..., p_h\}$, where each p_i is a pair including the queried instance and its true label, and $1 \leq h \leq k$, where h correspond to the number of instances queried.
  Add set P to the training set.
  Remove set P from the set $\bar{U}$.
  $O \leftarrow O - h$
until $O < k$

Analysis for MinMax Distance Algorithm

Let l and u be the number of labeled and unlabeled instances, respectively, s the number of support vectors of training data, k number of unlabeled instances selected in each combination, and d the dimension of the space. In order to find the complexity of the algorithm, we analyze Line 5, and the nested loop in Algorithm 1.

In order to create set $\bar{U}$, at each time we assign both possible labels +1 and -1 to each element of set U, add the corresponding constraint to the optimization Problem 4.7, and check the feasibility of the problem. The complexity of optimization Problem 4.7 is $O(sd^2)$ (s number of support vectors of training set). Therefore, the complexity of checking feasibility for each unlabeled instance is $O((s+1)d^2) = O(sd^2)$. So, the complexity of Line 5 in Algorithm 1 is $O(usd^2)$ since the feasibility is checked for each unlabeled instance.

Let $\bar{u}$ be the number of elements of set $\bar{U}$. The number of iterations of the for loop in Line 8 is $C_k^{\bar{u}}$, the number of elements of set C_k i.e $\binom{\bar{u}}{k}$.

In order to find the number of iterations of the for loop in Line 10, we find the number of elements of set F_c. Consider the following Proposition [Sta07]:

Proposition 4.3.4 *Given a n-dimension space, the number of regions generated by m hyperplanes is:*

$$1 + m + \binom{m}{2} + ... + \binom{m}{n}$$

Now, consider F_c, the set of all feasible labelings of each combination. Since each combination has k elements (as some crossing hyperplanes in version space with dimension d), by applying Proposition 4.3.4, F_c has $1 + k + \binom{k}{2} + ... + \binom{k}{d}$ elements. Therefore, the complexity of the nested loop is:

$$\left(1 + k + \binom{k}{2} + ... + \binom{k}{d}\right) \times C_k^{\bar{u}}$$

or

$$\left(1 + k + \binom{k}{2} + ... + \binom{k}{d}\right) \times \binom{\bar{u}}{k}$$

The run time of the first part determines the complexity of the nested loop and so the complexity of Algorithm 1. If the number of hyperplanes is much larger than the dimension d, then the complexity of the first part is $O(k^d)$. So, if we consider the Euclidean space, then it is quadratic in k. If $k \leq d$, then the complexity is $O(2^k)$. This is exponential in k, but it is fine since, as discussed in Section 4.3.1, we have control over k, so we pick a small k. Therefore, the complexity of Algorithm 1 is: $O(2^k \bar{u}^k)$.

Different Approach for Selecting w

We now return to the definition of regret with respect to misclassification. Given some labeled and unlabeled instances, the objective is to find a classifier which classifies both labeled and unlabeled data as accurately as possible i.e. the one which gives the least number of misclassifications on unlabeled data.

Consider the two-class problem with $k = 3$, and the corresponding version space including seven sub-regions (Figure 4.3). Sub-region three has one different labeling with regions two, four, and six, two different labelings with regions one and five, and three different labelings with region seven. There is a similar argument for region seven. However, any of regions one, two, four, five, and six has at most two number of different labelings with other regions. It means, if the learner algorithm selects region three as the one that has optimum $\mathbf{w}$, but the true region is region seven, it ends up with the most number of misclassifications. However, if any of regions one, two, four, five, and six is selected, the number of misclassifications is smaller.

This leads to a different measurement for selecting $\mathbf{w}$ that takes into account the number of differences in labelings.

Definition 4.3.5 *We define the inconsistency of two sub-regions in version space as the number of points that are assigned different labels.*

Considering the two-class problem, regions three and seven have the maximum number of inconsistencies.

Definition 4.3.6 *Given $W = \{\mathbf{w}_1, ..., \mathbf{w}_m\}$ the centers of the m sub-regions in version space, the best $\mathbf{w}_i \in W$ is defined as the one whose region has the smallest maximum inconsistency over all regions.*

Therefore, we can modify Algorithm 1 to get another heuristic. In Line 15 of Algorithm 1, first the learner selects the $\mathbf{w}^* \in W$ based on Definition 4.3.6 i.e the one that

has smallest maximum inconsistency. Second, if there is more than one, then the learner selects $\mathbf{w}^* \in W$ based on Definition 4.3.3 i.e the one which minimizes the maximum distance.

4.3.2 Pruning Algorithm

Under the assumption of linear separability, we describe a different approach for finding the next unlabeled instance to query.

Since the data is assumed to be linearly separable, after adding one instance p_i (a pair of $\mathbf{x}_i$ and its true label) to the training set S, some instances of $\bar{U}$ will be forced to get one of the possible labels and could therefore be pruned from the set of all unlabeled instances. Therefore, our approach is to query the unlabeled instance which guarantees the most number of unlabeled instances to be pruned. Intuitively, the unlabeled instances that do not have a forced labeled are uncertain, so pruning as many as possible reduces the uncertainty as quickly as possible. More precisely, let $\mathbf{x}_i$ be an unlabeled instance from set $\bar{U}$. Also, let $\bar{U}_i \subseteq \bar{U}$ be the set of all instances in $\bar{U}$ which will be forced to get one of the possible labels after adding $\mathbf{x}_i$ to the S. Therefore, for each instance $\mathbf{x}_i \in \bar{U}$, we once assign it with the label +1 and let $\bar{u}_{i^+}$ be the number of instances in $\bar{U}$ which will be pruned. Similarly, we obtain the set $\bar{u}_{i^-}$ by assigning $\mathbf{x_i}$ the label -1. Obviously, the smaller of $\bar{u}_{i^+}$ and $\bar{u}_{i^-}$, denoted g_i, is the guaranteed number of instances to be pruned after querying $\mathbf{x}_i$. Hence, in order to pick the next unlabeled instance to query, we choose the instance $\mathbf{x}_i$ which has the largest g_i.

The Pruning algorithm works well when the number of unlabeled instances is large compared to the number of dimensions. This algorithm can be either a standalone approach, or can be combined with other approaches, when after each querying we can prune all the forced instances.

The Pruning algorithm is a greedy algorithm since it picks the unlabeled instance which prunes the most instances in each step. However, two best consecutive instances with most pruning may as a pair give worse results than some other pair that does not contain the instance that prunes the most instances as a pair. We will show an example in Section 5.2.2.

Algorithm 2 Pruning algorithm

Input: labeled and unlabeled instances.
O: the number of calls to the oracle.
repeat
  Find $\bar{U}$, the set of all unlabeled instances which are not forced to get one of the possible labels.
  for each $\mathbf{x}_i$ an element of $\bar{U}$ **do**
    Assign y_i as $+1$.
    Let $\bar{u}_{i^+}$ be number of unlabeled instances of $\bar{U}$ which will be forced to get one of the possible labels.
    Assign y_i as -1.
    Let $\bar{u}_{i^-}$ be the number of unlabeled instances of $\bar{U}$ which will be forced to get one of the possible labels.
    Let g_i be the smaller of $\bar{u}_{i^+}$ and $\bar{u}_{i^-}$, as the number of instances which are guaranteed to be pruned after querying $\mathbf{x}_i$.
  end for
  Call oracle to get the true label of the unlabeled instance $\mathbf{x}_i$ with the largest g_i.
  Add p_i (the pair including the queried instance and its true label) as well as the set $\bar{U}_i$ (set of force instances) with their respective labels to the training set.
  Remove $\mathbf{x}_i$ from the set $\bar{U}$.
until $O < 1$

Chapter 5

Experiments

5.1 Introduction

In Chapter 3, we reviewed some existing algorithms for performing active learning using SVMs (Tong and Koller [TK00b]). Their approach is supervised in nature, since when selecting the next unlabeled instance, they do not consider all unlabeled instances. In Chapter 4, we proposed two new semi-supervised algorithms to apply active learning with SVMs, which take into account the presence of all unlabeled instances. More precisely, all instances are active in the process of evaluating each single instance whether it will be queried next. In this chapter, we show some experimental results for some artificial and real problems.

In order to compare our work with the two heuristics (*Simple Margin*, and *MaxMin Margin*) of Tong and Koller, we reimplemented their algorithms which is described in [TK00b].

To find the solution of the optimization Problem 4.6 (Line 11 of MinMax Distance algorithm), we use the existing code, UniverSVM [CSWB06] available online at *http://www.kyb.mpg.de/bs/people/fabee/universvm.html* accessed January 2007. It implements the Concave-Convex procedure discussed in Section 4.2. For verifying linear separability of the data and for determining forced labels of data instances, we use $CPLEX$ version 10.0. This is embedded with the UniverSVM code using *ILOG concert technology*, which provides a set of objects to represent optimization problems and can be used in C++.

In the next section, we show some experimental results for some problems in different dimensions.

5.2 Problems

We compare the performance of the aforementioned algorithms on four test problems. In Section 5.2.1, two 2-dimensional problems are generated. For the first one, the hyperplane is picked at random and the data points are generated uniformly at random on each side of that hyperplane. For the second problem, a hyperplane is also generated at random, but the data points are generated randomly in clusters. In Section 5.2.2, a third problem in a 10-dimensional space is generated. Again hyperplane is picked at random, and data is generated randomly in clusters. In Section 5.2.3, a real handwritten digit recognition problem from the UCI database is used for testing. More details are given in each section on the characteristics of each problem.

We compare the result of five heuristics on those four problems. The algorithms MinMax Distance (Algorithm 1), Pruning (Algorithm 2), Simple Margin (Tong and Koller [TK00b]), MaxMin Margin (Tong and Koller [TK00b]), and Random (picking the next unlabeled instance at random) are designated with the labels MM Dis, Pruning, Smpl Mrg, MM Mrg, and Random, respectively. To measure the performance of each algorithm, we create some graphs which illustrate the accuracy (on the y coordinates) achieved after some number of queries (on the x coordinate) to the oracle.

The inputs for each algorithm consist of three different sets. The first set consists of the training set, which provides the learner with the initial set of labeled instances. The second set consists of the unlabeled instances, which are used for two purposes: the learner selects one of them for the next query to the oracle, and, in the case of the MinMax Distance and Pruning algorithms, these instances are also used for the semi-supervised active learning selection process. For all algorithms the unlabeled instances that were not yet queried are also used to compute the best classifier using the Universum algorithm [CSWB06]. The last set consists of testing instances: their labels are known and we use them to assess the accuracy of the algorithms after each query to the oracle.

For the experiments in Sections 5.2.1 and 5.2.2 the training set consists of 2 labeled instances and 48 unlabeled instances. The performance of each algorithms is assessed on 1000 instances. For each example, we generated 10 data sets and ran the algorithms until they exhausted the set of available unlabeled instances. We computed the accuracy reported in the graphs as the average accuracy achieved by each of the algorithms after a specified number of queries to the oracle.

5.2.1 Uniform distribution of random points in a 2-dimensional space

In our first experiment, data is randomly generated (uniformly) in a two-dimensional space in the $[-20, 20] \times [-20, 20]$ square. We also generate a hyperplane at random and assign labels accordingly. More precisely, the training set consists of 2 labeled instances and 48 unlabeled instances. The performance of each algorithm is assessed on 1000 test instances. We generated 10 data sets as described above and averaged the result of each algorithm on those data sets. We reported the average accuracy (percentage of correctly classified test instances) after a specified number of queries to the oracle in the graph illustrated in Figure 5.1.

As it is shown in Figure 5.1, Pruning and MinMax Distance algorithms achieve the highest accuracy with the lowest and second lowest number of queries, respectively. The Pruning algorithm gets to the highest accuracy (98 percentage) after only querying 8 instances. The MinMax Distance algorithm gets the same accuracy after querying 18 instances. Note that the differences between Simple Margin, MaxMin Margin, and Random are not statistically significant. Due to time constraints, we only ran each algorithm on ten data sets and for each data set, Random was run ten times. This explains why the curves oscillate. Note also that the classifiers were optimized by the CCCP procedure (described in 4.2) which may get stuck in local optima since semi-supervised SVMs correspond to non-convex optimization problems. We attribute the dips in the curves observed around 47 labeled instances to local optima issues.

There is a trade-off between using unlabeled data with the risk of getting stuck in local optima in semi-supervised learning versus ignoring unlabeled data and finding the global optima in supervised learning. Figure 5.2 reports the accuracy when the classifier was obtained by supervised learning after newly labeled instance. We can see that as the number of unlabeled instances decreases, it appears that obtaining the classifier by supervised learning is better. Perhaps semi-supervised learning would be better if we use another technique instead of CCCP such as Xu's convex relaxation [XNLS04].

In low-dimension problems there is a lot of pruning, and the Pruning algorithm performs quite well, as expected. The same phenomena may occur in higher dimensions, when the number of instances is much larger than the dimension.

Figure 5.3 illustrates the number of undetermined instances (instances with both labels possible) after a specified number of queries to the oracle in each step. For linearly

separable data sets, the highest accuracy is achieved when there are no undetermined instances remaining. As expected the Pruning algorithm outperforms the others. However, it is a greedy algorithm and therefore, there are problems for which it will not reduce the number of undetermined instances as fast other algorithms. This is illustrated in our second experiment.

We generated a second problem where a hyperplane is picked at random and the centers of 10 clusters are generated uniformly at random in the $[-20, 20] \times [-20, 20]$ square. For each cluster 100 test points and 5 training points are generated uniformly at random in a box of size 1×1. If the hyperplane cuts through the box, then the cluster is regenerated with a new center. Since there are 5 training points per cluster and 10 clusters, there is total of 50 training points. Two of those points are labeled (one for each class) and the remaining points are unlabeled. The test set consists of 1000 points with 100 points per cluster.

Figure 5.4 shows the number of undermined points after each query to the oracle. Those results are the average over 10 data sets. In this case the MinMax algorithm does better than the Pruning algorithm. As we can see, the sequence of instances chosen by the MinMax algorithm prunes all the points after six queries to the oracle, whereas the greedy Pruning procedure requires eleven queries to prune all the instances. We observe that the Pruning algorithm is greedy and therefore not optimal as discussed in Section 4.3.2.

5.2.2 Randomly generated clusters, in a 10-dimensional space

Our third experiment tests the performance of the algorithm in higher dimensions (10 dimensions). Here the Pruning algorithm is not expected to perform well when the number of training instances remains constant. The test problem used in this section is identical to the previous one, but in 10 dimensions.

A hyperplane is generated at random and the centers of 10 clusters are generated uniformly at random in the $[-20, 20]^{10}$ hypercube. For each cluster, 100 test points and 5 training points are generated uniformly at random in a box of size 1×1. If the hyperplane cuts through the box, then the cluster is regenerated with a new center. Since there are 5 training points per cluster and 10 clusters, there is total of 50 training points. Two of those points are labeled (one for each class) and the remaining points are unlabeled. The test set consists of 1000 points with 100 points per cluster.

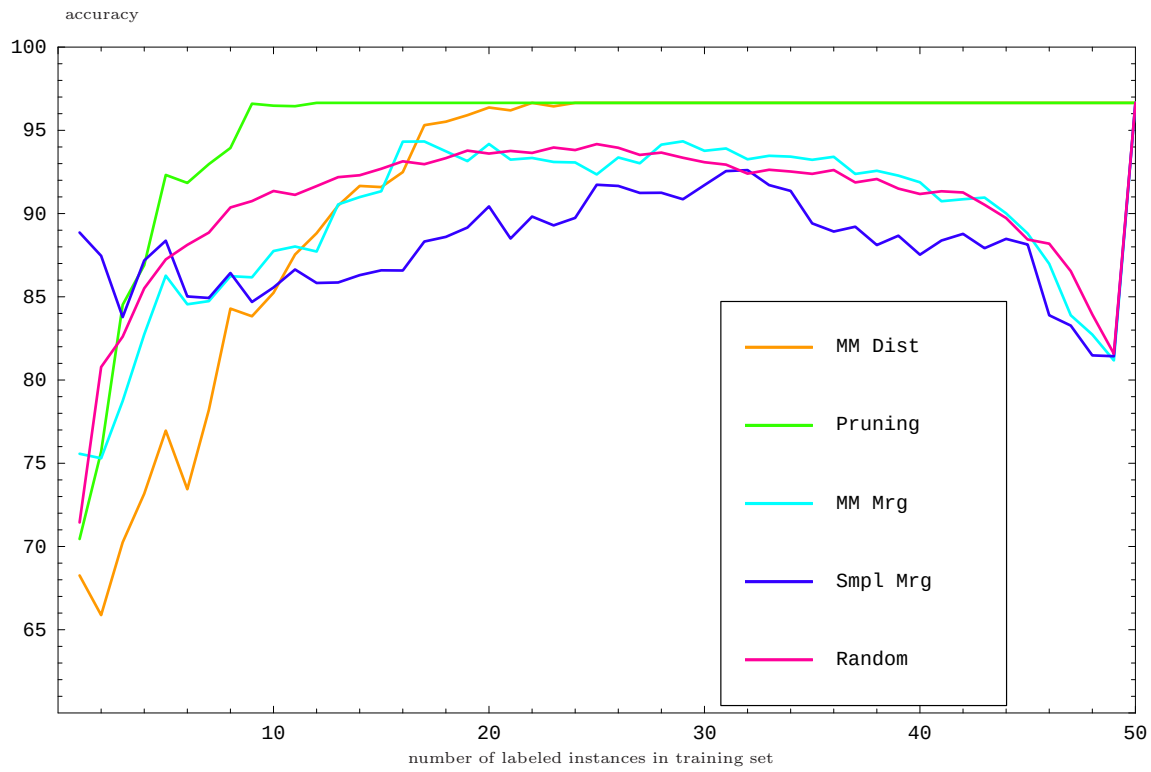

Figure 5.1: Test set accuracy of five heuristics for randomly generated instances in a 2-dimensional space. Order of performance: Pruning, MinMax Distance, [MaxMin Margin, Random, Simple Margin] (classifier found by semi-supervised learning).

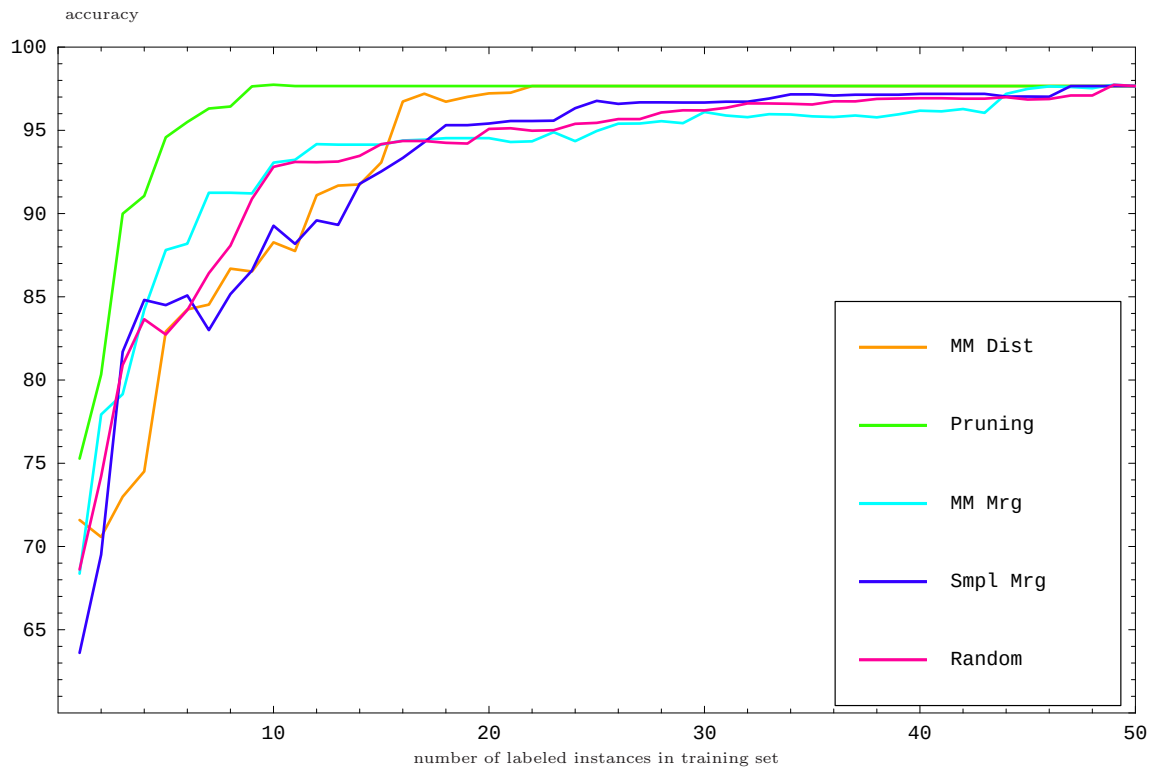

Figure 5.2: Test set accuracy of five heuristics for randomly generated instances in a 2-dimensional space. Order of performance: Pruning, MinMax Distance, [MaxMin Margin, Random, Simple Margin] (classifier found by supervised learning).

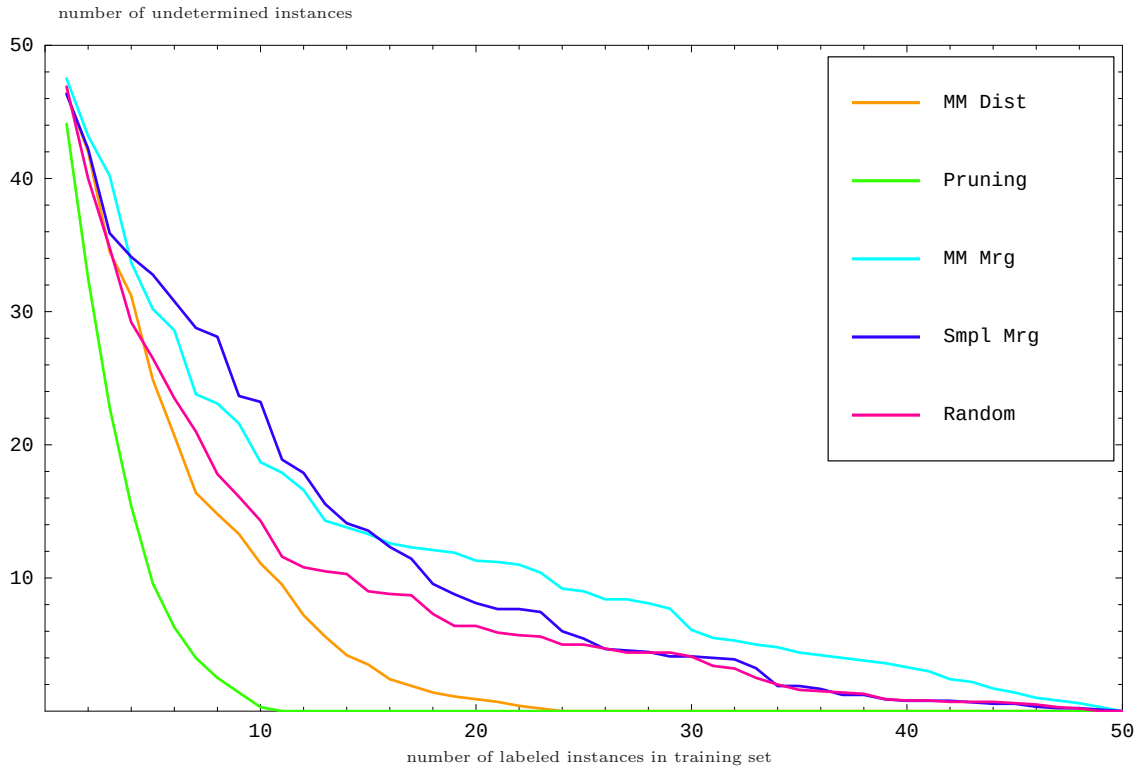

Figure 5.3: Number of undetermined instances after a specified number of queried instances in a 2-dimensional space (classifier found by semi-supervised learning).

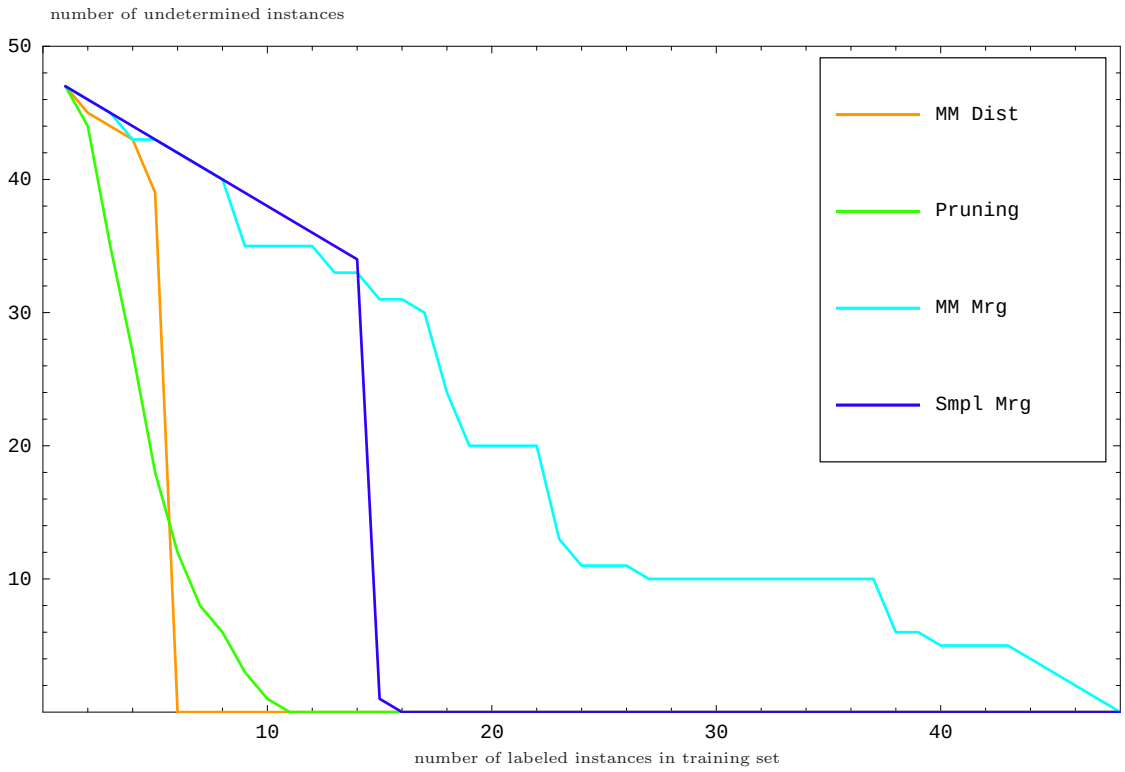

Figure 5.4: Number of undetermined instances after a specified number of queried instances for clusters in a 2-dimensional space.

Figures 5.5 and 5.6 show the accuracy (averaged over 10 data sets) of each algorithm after a specified number of queries to the oracle for classifiers found by semi-supervised and supervised SVMs, respectively. As the graphs show, the MinMax Distance and Random algorithms perform better than the other algorithms. Here because there are ten clusters, a good querying strategy is to request labels in different clusters until all clusters have been queried at least once. Random does well because points are queried in all the clusters uniformly. Pruning, MaxMin Margin, and Simple Margin do poorly because they query all the points in one cluster before moving to the next cluster. In fact this can be observed in the accuracy curves of those algorithms (Figure 5.6). Since there are five points per cluster, notice how the curves evolves in steps with flat parts of size five corresponding to each cluster.

In fact, after less than 10 calls to the oracle, the MinMax Distance algorithm has a remarkably higher accuracy than other algorithms.

Pruning, MaxMin Margin, Random algorithms all perform close to each other and the order of performance is not distinguishable. The Simple Margin algorithm initially performs poorly on such instances, but later its performance is similar to that of the latter three algorithms.

5.2.3 Optical Recognition of Handwritten Digits

Here, we show that the MinMax Distance algorithm has a performance comparable with the performance of both MaxMin Margin and Simple Margin algorithms on the real data set, Optical Recognition of Handwritten Digits, taken from the UCI machine learning repository.

The data set consists of 3823 instances with 10 classes and 64 attributes. In order to consider this data as binary classification, we consider the digits 1,..., 5 as positive class and the rest as negative class. We perform a k-crossvalidation by generating at random 10 different training sets, each made of 2 labeled instances (one per class) and 48 unlabeled instances. In each case the remaining data points form the test set. Each algorithm runs on the 10 different data sets until it exhausts the set of available unlabeled instances.

Figures 5.7 and 5.8 show the accuracy (averaged over the 10 data sets) of the MinMax Distance, MaxMin Margin, and Simple Margin algorithms with classifiers found by semi-supervised and supervised SVMs, respectively. All algorithms perform equally well.

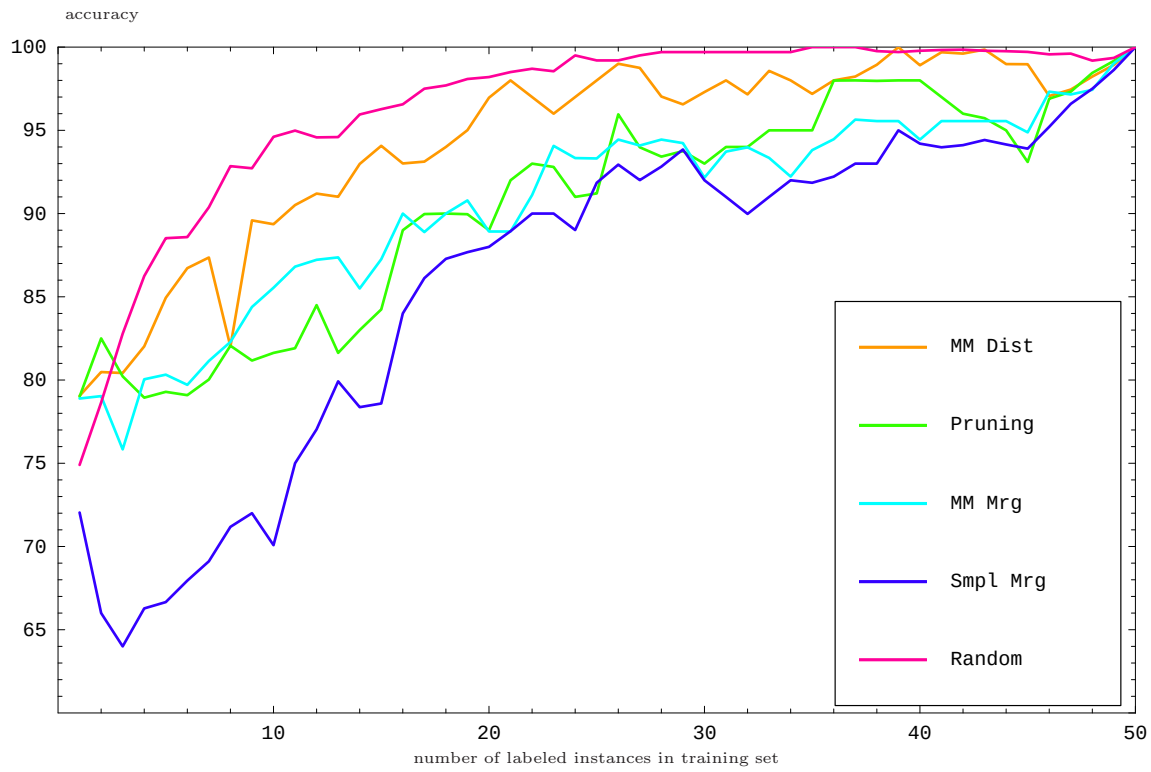

Figure 5.5: Test set accuracy of five heuristics for randomly generated clusters in a 10-dimensional space. Order of performance: [MinMax Distance, Random], [Pruning, MaxMin Margin, Simple Margin] (classifier found by semi-supervised learning).

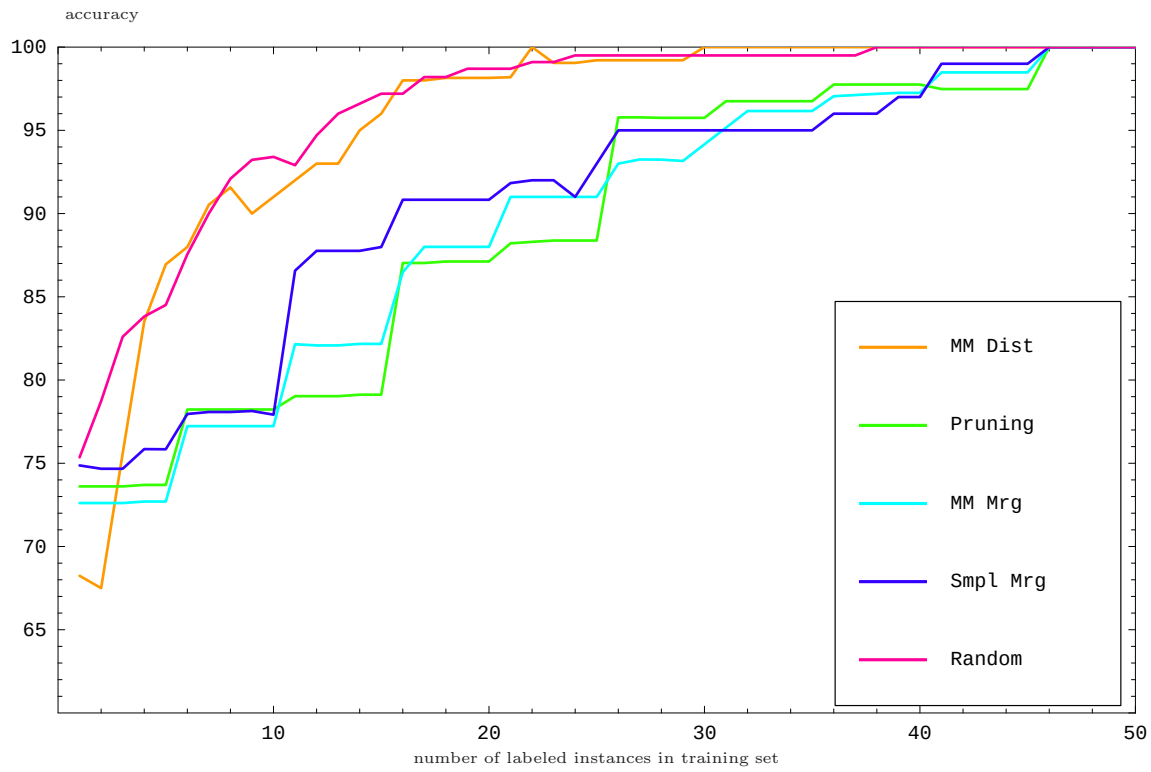

Figure 5.6: Test set accuracy of five heuristics for randomly generated clusters in a 10-dimensional space. Order of performance: [MinMax Distance, Random], [Pruning, MaxMin Margin, Simple Margin] (classifier found by supervised learning).

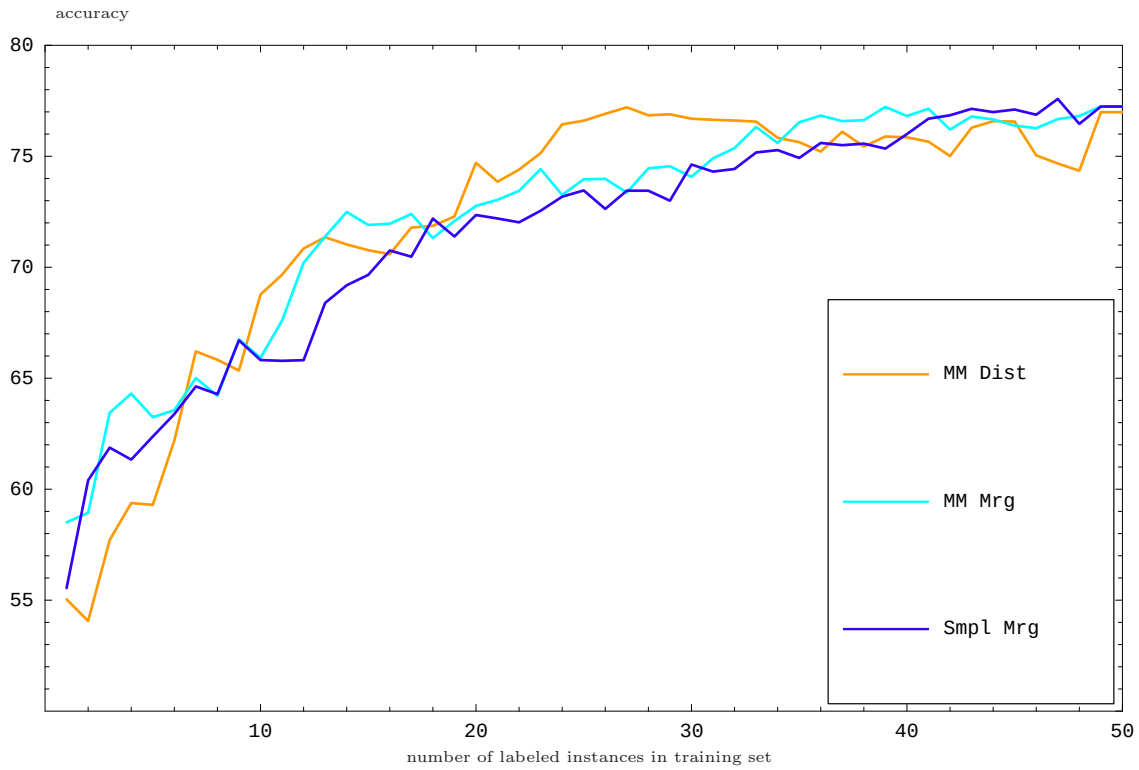

Figure 5.7: Test set accuracy of three heuristics on Optical Recognition of Handwritten Digits problem (classifier found by semi-supervised learning).

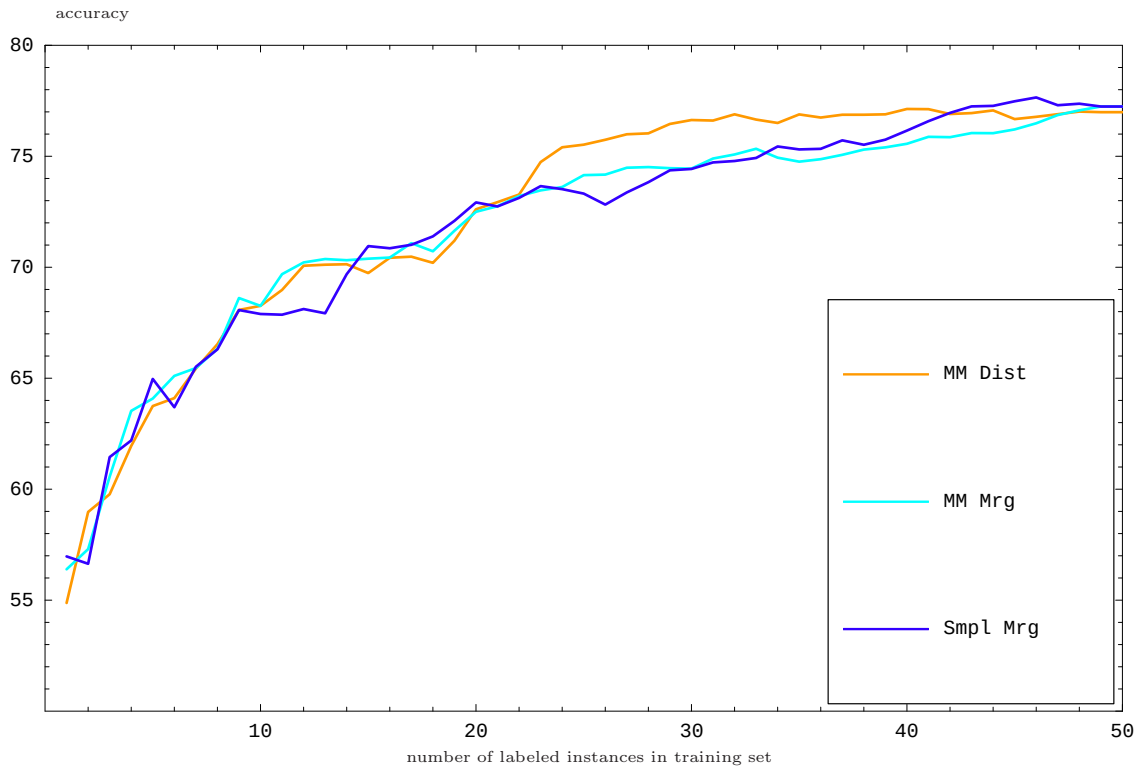

Figure 5.8: Test set accuracy of three heuristics on Optical Recognition of Handwritten Digits problem (classifier found by supervised learning).

Chapter 6

Conclusion

6.1 Summary

In this thesis, we studied active learning and reviewed some of its applications. More specifically, we reviewed two heuristic by Tong and Koller [TK00b] (Simple Margin, and MaxMin Margin for active learning with SVMs). In those approaches, the decision of selecting the next unlabeled instance to query is only based on the information of labeled data. In fact, unlabeled data has no influence on the decision of the instance to query. Therefore, the active learning process is in a supervised setting. We advanced active learning for SVMs by proposing a semi-supervised setting for selecting the next instance to query: all the unlabeled data was considered in assessing each instance for its suitability to be queried next. The result of our work are two semi-supervised algorithms, MinMax Distance and Pruning.

The idea of the MinMax Distance algorithm is as follows: we fined the hyperplane with minimax regret and query its unlabeled support vectors. At each step, the queried instances are added to the training data and removed from the unlabeled data. After some fixed number of queries, the algorithm returns a classifier by finding the hyperplane that maximizes the margin relative to both labeled and unlabeled data.

The Pruning algorithm works under the assumption of linear separability of data. At each step of the algorithm, the learner selects the unlabeled instance which guarantees the most pruning among the other unlabeled instances.

We showed with some experiments that our semi-supervised approaches performs better than or comparably to existing algorithms which are supervised in nature.

There are some possible directions of future research for active learning with SVMs

that we discuss in the following section.

6.2 Future Work

The MinMax Distance algorithm allows for values of k greater than 1 which result in up to k queries at a time. However, the complexity of the algorithm increases with larger k and this does not make sense be applied directly. Further research is necessarily to improve scalability with respect to k.

Since a common approach for active learning consists of querying the unlabeled instances that are most uncertain, it would be nice to have a distribution over the possible labels. The entropy of the distribution of each instance could be used as a measure of uncertainty. It turns out that SVMs have a Bayesian interpretation where it is possible to have a distribution over hyperplanes modeled by Gaussian processes [Sol02, TWW05]. We can use the distribution over hyperplanes to compute the expected probability of each class for the unlabeled instances. One approach could consist of picking the unlabeled instance with largest entropy.

When the cluster assumption holds, it is possible to get 100 percent accuracy with one labeled point per cluster. Hence, it would be desirable to devise an active learning algorithm that automatically recognizes clusters and queries one point per cluster. The number of queries could be as low as the number of clusters. The heuristics proposed in this thesis do not use the cluster assumption. One possibility is to query the point that leads to the most "different" hyperplane (compared to the current hyperplane). Here different could be measured by the number of points whose assigned label changes. Intuitively, this strategy should initially query points in different clusters since the assigned label to all the points in the cluster could change. In contrast if a second point is queried in a cluster, then only a subset of those points may change label.

While active learning can be used to select points that will speed up the learning process as much as possible, it can also be used to verify assumptions about the underlying distribution. For instance, the cluster assumption is commonly used in semi-supervised SVMs. However, the learner may not be certain about this assumption and therefore may refrain from using it. One could use the labels obtained by active learning to adjust the parameter C based on how well the cluster assumption holds in the labeled data acquired so far. More generally, it would be desirable to devise active learning strategies that help validate or even discover patterns in the underlying distribution that could be

used for semi-supervised learning.

Bibliography

[BC03] Tijl De Bie and Nello Cristianini. Convex methods for transduction. In *Neural Information Processing Systems*, 2003.

[BD98] Kristin P. Bennett and Ayhan Demiriz. Semi-Supervised Support Vector Machines. In *Neural Information Processing Systems*, pages 368–374, 1998.

[BHB02] Claus Bahlmann, Bernard Haasdonk, and Hans Burkhardt. On-line handwriting recognition with support vector machines. In *Eighth International Workshop on Frontiers in Handwriting Recognition (IWFHR'02)*, 2002.

[CAL94] David A. Cohn, Les E. Atlas, and Richard E. Ladner. Improving generalization with active learning. In *Machine Learning, 15(2)*, pages 201–221, 1994.

[CCS00] Colin Campbell, Nello Cristianini, and Alex J. Smola. Query learning with large margin classifiers. In *Proceedings of the Seventeenth International Conference on Machine Learning*, pages 111–118, 2000.

[CST00] Nello Cristianini and John Shawe-Taylor. *An Introduction to Support Vector Machines and other Kernel-based learning methods*. 2000.

[CSWB06] Ronan Collobert, Fabian H. Sinz, Jason Weston, and Leon Bottou. Trading convexity for scalability. In *the 23rd International Conference on Machine Learning*, pages 201–208, 2006.

[CW03] Sung-Bae Cho and Hong-Hee Won. Machine learning in DNA microarray analysis for cancer classification. In *Australasian Plant Breeding Conference*, pages 189–198, 2003.

[Dra00] Bruce A. Draper. Image-based feedback for learning object recognition strategies. In *Vision, Modeling, and Visualization*, pages 55–56, 2000.

[EZ04] Jae-Hong Eom and Byoung-Tak Zhang. Pubminer: Machine learning-based text mining system for biomedical information mining. In *Artificial Intelligence: Methodology, Systems, Applications*, pages 216–225, 2004.

[FCD+00] Terrence S. Furey, Nello Cristianini, Nigel Duffy, David W. Bednarski, Michel Schummer, and David Haussler. Support vector machine classification and validation of cancer tissue samples using microarray expression data. *Bioinformatics*, 16(10):906–914, 2000.

[FSST77] Yoav Freund, H.Sebastian Seung, Eli Shamir, and Naftali Tishby. Selective sampling using the query by committee algorithm. *Machine Learning*, pages 133–168, 1977.

[LLZN05] Huiying Li, Dechang Li, Changhai Zhang, and Shubin Nie. An application of machine learning in the criterion updating of diagnosis cancer. In *Neural Networks and Brain*, pages 187–190, 2005.

[MLR04] Omid Madani, Daniel. J. Lizotte, and Russell.Greiner. Active Model Selection. In *Uncertainty in Artificial Intelligence, Banff, Canada*, 2004.

[Pla99] John C. Platt. *Fast Training of Support Vector Machines using Sequential Minimal Optimization*. MIT Press, Cambridge, MA, USA, 1999.

[See02] Matthias Seeger. Learning with labeled and unlabeled data. Technical report, University of Edinburough, 2002.

[Sol02] Peter Sollich. Bayesian methods for support vector machines: Evidence and predictive class probabilities. *Machine Learning*, 46(1-3):21–52, 2002.

[SOS92] H. S. Seung, M. Opper, and H. Sompolinsky. Query by committee. In *Proceedings of the Fifth Workshop on Computational Learning Theory*, pages 287–294, 1992.

[Sta07] Richard P. Stanley. An introduction to hyperplane arrangments. 2007.

[STC99] John Shawe-Taylor and Nello Cristianini. Further results on the margin distribution. In *Conference on learning Theory*, pages 278–285, 1999.

[Ste03] Ingo Steinwart. Sparseness of support vector machines. *Journal of Machine Learning Research*, 2003.

[Thi94] H. A. Le Thi. *Analyse numérique des algorithmes de l'optimisation*. PhD thesis, INSA Rouen, 1994.

[TK00a] Simon Tong and Daphne Koller. Active Learning for Parameter Estimation in Bayesian Networks. In *Neural Information Processing Systems*, pages 647–653, 2000.

[TK00b] Simon Tong and Daphne Koller. Support vector machines active learning with application to text classification. In *17th International Conference on Machine Learning*, pages 999–1006, 2000.

[TK01] Simon Tong and Daphne Koller. Active Learning for Structure in Bayesian Networks. In *International Join Conferences on Artificial Intelligence*, pages 863–869, 2001.

[TWW05] Qing Tao, Gao-Wei Wu, and Fei-Yue Wang. Posterior probability support vector machines for unlabeled data. In *Institute of Electrical and Electronics Engineers*, 2005.

[Vap82] Vladimir Vapnik. In *Estimation of Dependences Based on Empirical Data*, Springer-Verlag, Berlin, 1982.

[Vap98] Vladimir. Vapnik. *Statistical Learning Theory*. Wiley Chichester GB, 1998.

[wC03] Xue wen Chen. Gene selection for cancer classification using bootstrapped genetic algorithms and support vector machines. In *Computer Society Bioinformatics Conference*, pages 504–505, 2003.

[XNLS04] Linli Xu, James Neufeld, Bryce Larson, and Dale Schuurmans. Maximum margin clustering. In *Neural Information Processing Systems*, 2004.

[XWSS06] Linli Xu, Dana F. Wilkinson, Finnegan Southey, and Dale Schuurmans. Discriminative unsupervised learning of structured predictors. In *International Conference on Machine Learning*, pages 1057–1064, 2006.

[YR03] A. L. Yuille and Anand Ranagarajan. The concave-convex procedure (CCCP). In *Advances in Neural Information Processing Systems 14*, pages 915–936, Cambridge MA: MIT Press, 2003.

[YRT+01] Chen-Hsiang Yeang, Sridhar Ramaswamy, Pablo Tamayo, Sayan Mukherjee, Ryan M. Rifkin, Michael Angelo, Michael Reich, Eric S. Lander, Jill P. Mesirov, and Todd R. Golub. Molecular classification of multiple tumor types. In *Intelligent Systems for Molecular Biology, (Supplement of Bioinformatics)*, pages 316–322, 2001.