Prototype Support Vector Machines: Supervised Classification in Complex Datasets

by

April Tuesday Shen

Andrea Danyluk, Advisor

A thesis submitted in partial fulfillment
of the requirements for the
Degree of Bachelor of Arts with Honors
in Computer Science

Williams College
Williamstown, Massachusetts

May 17, 2013

Abstract

Real-world machine learning datasets may be highly complex. Data of a single class may be distributed irregularly throughout the feature space and measures of distance as a proxy for similarity can be unreliable. Classification learning algorithms for such datasets typically require model selection, which in practice is often an ad-hoc and time-consuming process that depends on assumptions about the structure of data. To avoid this, I introduce the ensemble of prototype support vector machines (PSVMs). This algorithm trains an ensemble of linear SVMs that are tuned to different regions of the feature space and thus are able to separate the space arbitrarily, reducing the need to decide what model to use for each dataset. I also present experimental results demonstrating the efficacy of PSVMs in both noiseless and noisy datasets.

Acknowledgements

There are a number of people I'd like to thank for helping this thesis come to be. First, I'm grateful for the students and faculty of the computer science department, past and present, who have taught me so much throughout the years and made the lab such a wonderful place to work and play. Special thanks go to Mary Bailey for technical and emotional support, Morgan McGuire for being a caring and careful second reader, and Antal Spector-Zabusky for writing the custom algorithm environment for LaTeX that I used. Thanks also to my non-computer scientist friends and family, who have tolerated not only my year-long disappearance from society, but also many long-winded and probably confusing explanations of my work.

Last but certainly not least, I owe my deepest thanks to Andrea Danyluk. Her insightful questions and ideas have been critical to any success this work has achieved, and her comments on my writing have helped make my prose clear, correct, and concise. Most importantly, her good humor and friendship have kept me happy and sane throughout a challenging but incredibly rewarding year of research.

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

The goal of classification is to accurately predict class labels for a set of data. In machine learning, this is accomplished via algorithms that learn classification models for particular types of class distributions from sets of labeled training data. However, in real-world datasets, class distributions may be highly complex, and they are not generally known before learning takes place. Hence a data mining practitioner must choose an algorithm and its associated model without prior knowledge about the class distributions of the dataset in question, which often requires testing multiple models to find one that works well (Chatfield, 1995). The process of model selection, which is already arbitrary and time-consuming, becomes even more problematic for datasets with the most difficult class distributions, in which standard learning algorithms tend to over- or underfit.

This thesis introduces the ensemble of prototype support vector machines. This algorithm performs classification in datasets with complex class distributions without the need for model selection. Central to this approach is the use of multiple linear models positioned according to the actual distribution of the training data, which provides the flexibility needed to accurately model class boundaries of potentially high complexity. Experimental results demonstrate that this algorithm on average performs as well as or better than other classifiers in a number of different datasets.

1.1 Motivation

In supervised machine learning, algorithms learn a classifier for a set of training data whose true class labels have been provided, with the hope that this classifier will accurately predict class labels for novel unlabeled data. The data is represented as points in some feature space, where each component represents some attribute of the input data. Hence we can think of the classifier as a model or function that partitions the feature space into different regions corresponding to the data points of different classes.

Many learning algorithms and their corresponding models are highly accurate fits to certain types of class distributions. For example, many datasets are linearly separable, so choosing an algorithm that learns the equation of a hyperplane, such as a perceptron or a support vector machine (SVM) (Cortes and Vapnik, 1995), is sufficient to model that data accurately. A practical task that generally has linearly separable data is text classification, to which linear SVMs have been successfully applied (Joachims, 1998). Similarly, data that can be separated by polynomials or other functions can be

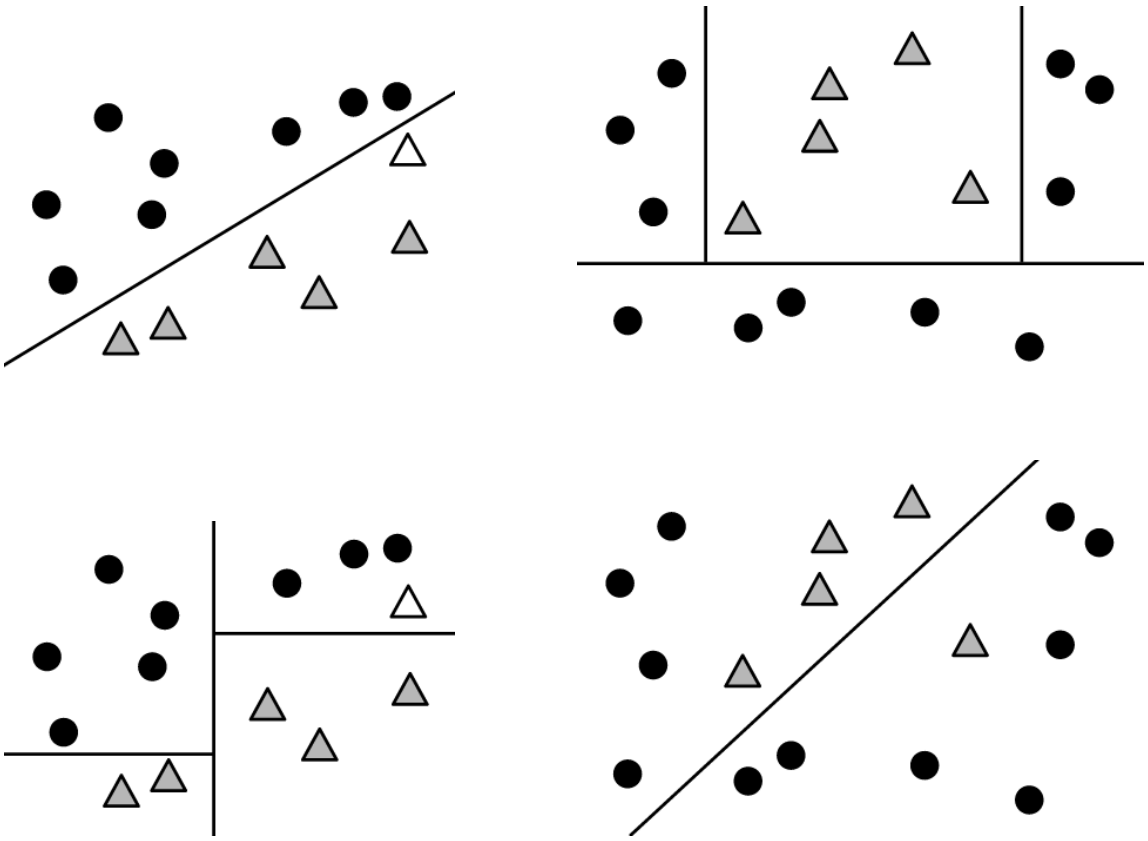


Figure 1.1: Some illustrations of how model selection can be problematic. Each column depicts the same dataset. The top row shows two models, namely a linear separator and axis-aligned rectangles, that accurately capture the class distributions, while the bottom row shows that the same two models applied to the "wrong" dataset do not fit the data as well. In particular, axis-aligned rectangles learned in the left dataset do not generalize to the white test instance of the triangle class. On the other hand, a linear classifier in the right dataset is not capable of separating the classes in even the training set.

modeled by nonlinear SVMs with the appropriate kernel function; see Chapter 3 for details.

More complicated class distributions, such as ones where multiple noncontiguous regions are mapped to the same class, can also be modeled given the correct choice of algorithm. For example, C4.5 is a common decision tree learning algorithm that learns axis-aligned rectangles in different parts of the feature space (Quinlan, 1993). C4.5 has been successfully used to diagnose errors in telephone networks, where data tends to occur in small clusters in the feature space (Danyluk and Provost, 1993). Combinations of multiple base classifiers allow for even more possibilities for modeling complicated distributions.

However, regardless of how aptly a given model captures a particular class distribution, choosing an inappropriate model can still result in poor classification performance. Figure 1.1 displays some examples of this, showing how linear separators and axis-aligned rectangles can be very accurate in some datasets, but perform poorly when applied to datasets for which they are not suitable. While visualizing the data, as we do here, would help with the model-selection problem, the feature spaces of most datasets are generally of higher dimension than two or three. In addition, we must assume that class distributions in real-world datasets can be arbitrarily complicated; there may be

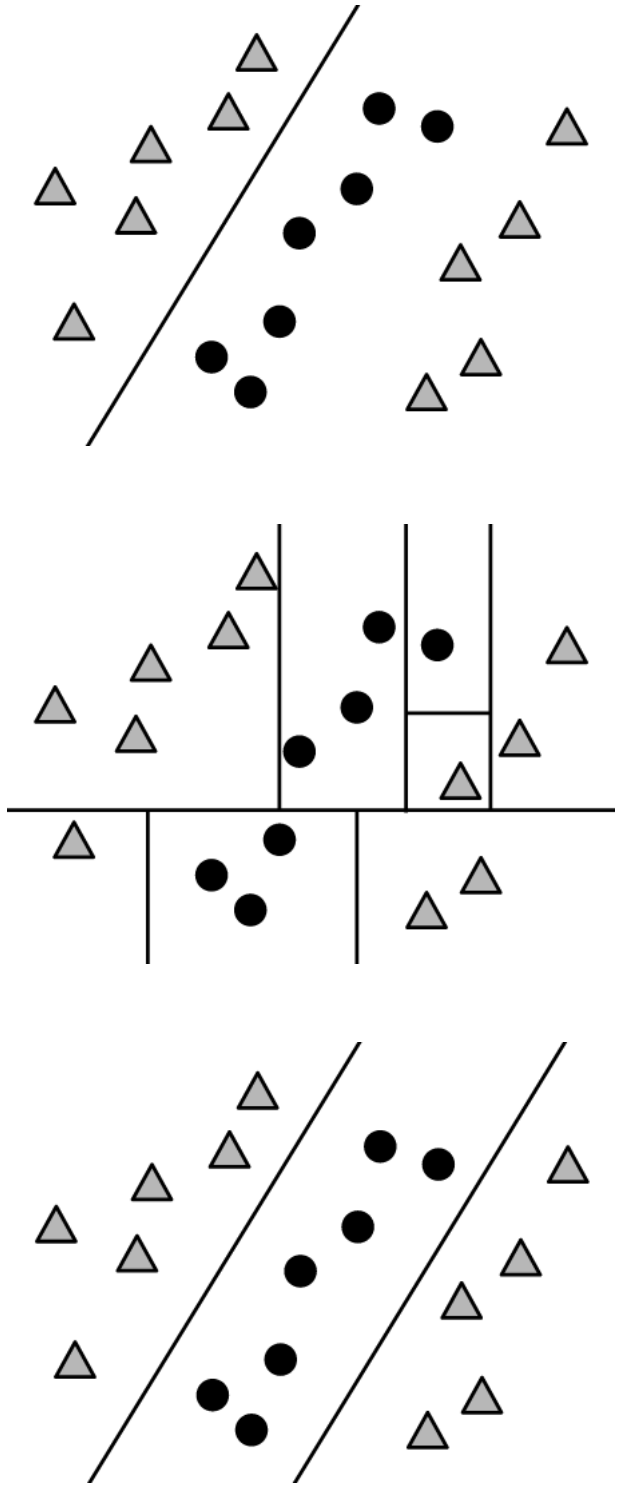


Figure 1.2: Model selection is even more difficult in datasets with complex class distributions. For example, here neither the linear classifier nor the axis-aligned rectangles are particularly suitable models of the class boundaries; the former underfits and the latter appears to overfit.

The true class distribution seems to be best captured by a pair of lines, as indicated in the bottom diagram; but unless one knows ahead of time to try this model, it may never be discovered. Note that while it is helpful to illustrate these issues with two-dimensional depictions, in reality few datasets are two-dimensional, and hence a data mining practitioner is unable to visualize datasets with possibly highly complicated distributions, and hence is unlikely to have any intuition about what models to try.

no *a priori* reasons to assume any kind of mathematically elegant structure to the data. In these extremely complex datasets, many common classifiers, including linear separators and decision trees, make assumptions that are too restrictive to allow them to capture the data adequately.

It is important to note that every learning algorithm has some inductive bias that limits the set of possible models it explores; the space of all possible hypotheses is simply too vast to exhaustively search. Furthermore, there are almost always multiple models that fit the training data, and so any learner needs a way to choose among them. Thus it is unreasonable to expect an algorithm to be completely agnostic towards the structure of the data in question. The point is that choices about what learning algorithm to use, and hence what model to learn, must happen prior to training and generally without knowledge of what data distributions look like. In many cases, this forces the practitioner to simply train and test multiple algorithms in order to discover which one performs the best, a time-consuming and ad-hoc process.

The issue is not simply resolved by choosing the most general or flexible class of models possible (with the reasoning that, for instance, linear functions are a subset of polynomial functions). There are two main problems that influence model choice in opposing ways. One is underfitting, which happens when the model is too simple and does not capture the full complexity of the data distribution. The other is overfitting, which happens when the model is too complex and fits noise in the training data; such a model does not generalize well to novel instances. Figure 1.2 shows some examples of these issues. The need to balance complexity of the model with accurate approximation of the true class distribution makes the problem of model selection much more subtle than it might seem at first glance.

Domain-specific knowledge about the data can conceivably help guide the model-selection process. However, it is unrealistic to expect a domain expert to have any intuition about a huge many-dimensional dataset, even if he has some understanding about the systems that generated it. Furthermore, while human intuition is a good guide, one of the benefits of machine learning is the ability to explore potentially fruitful hypotheses about data that human judgement would not necessarily consider. One tenet of machine learning is that large quantities of data contain information about underlying processes or properties, and that it is possible to create algorithms to extract that information. Eliminating model selection is just one way of trying to extract even more information from the data itself without human guidance.

1.2 Contributions

In this thesis, I introduce the ensemble of prototype support vector machines (PSVMs) as a classification learning algorithm addressing the problem of model selection in complex datasets. The PSVM algorithm learns a collection of linear classifiers tuned to different regions of the space in order to separate classes with arbitrarily complicated distributions. This algorithm is based on the exemplar SVM (ESVM) approach (Malisiewicz et al., 2011), which trains a separate linear separator specific to each instance in the training set. The PSVM algorithm trains an initial ensemble of ESVMs, but then iteratively improves boundaries to allow classifiers to capture groups of similar instances. Hence these new classifiers are tuned to more generalized prototypes instead of specific exemplars.

Although developing a cognitively plausible model is not a goal of this research, ideas from cogni-

tive science research have influenced my work, particularly exemplar- and prototype-based theories of concept representation (Kruschke, 2006). In particular, one could view the PSVM algorithm as an implementation of the cognitive model called the chorus of prototypes (Edelman and Shahbazi, 2012).

In addition to the standard PSVM algorithm, I introduce a variant that uses sampling of the training set to focus on the most difficult regions of the feature space. This modification is inspired by a similar sampling process in AdaBoost (Schapire, 1999), which shares with PSVMs the goal of improving accuracy in difficult datasets by combining multiple classifiers. Finally, I present empirical evidence that PSVMs are capable of high classification accuracy in a variety of noiseless and noisy datasets with different class distributions.

1.3 Organization

The rest of this thesis is organized as follows. Chapter 2 discusses several existing machine learning techniques for handling difficult class distributions, with a focus on dimensionality reduction and ensemble methods. The next chapter introduces support vector machines, as well as two ensemble-learning algorithms that use linear SVMs as their base classifiers. Chapter 4 describes both variations of the prototype SVM algorithm itself, while Chapter 5 details experiments comparing PSVMs with other classification algorithms on a number of different datasets with and without noise. The final chapter summarizes contributions and discusses future work.

Chapter 2

Related Work

There are many ways to attack the problem of complex class distributions. One common approach is to reduce or reformulate the feature space, since class boundaries may only seem complex when data is viewed in a particular space. This can involve finding which features or combinations of features are most relevant, an objective known as dimensionality reduction. Much of this work is done in the context of unsupervised learning or exploratory data analysis, where class labels for the training set are not provided to the learner and the explicit goal is to reveal interesting structure in the data.

Another approach to handling complex class distributions is to leave the feature space as-is, but learn classifiers from different subsets of examples in that space. Generally multiple such classifiers must be used in tandem to ensure sufficient coverage of the space, so these learning algorithms for supervised classification are known as ensemble methods. Of particular interest to us is AdaBoost, which builds a series of models that increasingly focus on the difficult to capture regions of the feature space.

After describing some important examples of dimensionality reduction techniques and ensemble methods, I close this chapter with a brief discussion of other approaches to dealing with difficult class distributions.

2.1 Dimensionality Reduction

In machine learning, many dimensionality reduction techniques fall under the category of either feature selection or feature extraction. Feature selection seeks to choose the attributes of the data that are most important for classification. While this is sometimes domain-specific by necessity, there has been substantial work in feature selection algorithms, in both supervised (Guyon and Elisseeff, 2003) and unsupervised (Ferreira and Figueiredo, 2011) learning contexts. In this section, I discuss subspace clustering as an interesting example of the latter.

In contrast, feature extraction involves combining features to get a new feature space. Classical techniques such as principal component analysis (PCA) and multidimensional scaling (MDS) fall under this category, as does constructive induction (Callan and Utgoff, 1991). The feature combinations can be nonlinear, as in the case of nonlinear SVMs, which I discuss in Chapter 3. Below, I discuss manifold learning as a recent and relevant example of nonlinear feature extraction.

2.1.1 Subspace Clustering

Subspace clustering performs feature selection in the unsupervised setting by seeking clusters of data in potentially very different subspaces of the complete feature space. This essentially selects features for each cluster independently. Though subspace clustering is a form of unsupervised learning, its goals of finding patterns in different regions of highly complex datasets are similar to the goals of my prototype SVM algorithm in the supervised learning context.

There are a number of algorithms for performing subspace clustering, which can be categorized as either bottom-up or top-down (Parsons et al., 2004). Bottom-up approaches such as CLIQUE (Agrawal et al., 1998) seek dimensions of the feature space with dense regions of data and use those dimensions to build up subspaces. This idea is based on the downward closure property of density, which states that dense regions in the full space (i.e., clusters) will also be dense when projected onto lower-dimensional subspaces; hence the candidate dimensions can be narrowed down to those with dense regions. In contrast, top-down approaches such as PROCLUS (Aggarwal et al., 1999) begin with an approximation of clusters in the full space, then weight the importance of each dimension based on evaluation of that clustering and re-cluster to iteratively refine the clustering with fewer dimensions.

A related but not identical approach to clustering in extremely high-dimensional spaces is multiple non-redundant clusterings (Cui et al., 2010). Here the goal is to discover several distinct clusterings, possibly in different feature subspaces, that can reveal different latent structures of the data. This is clearly distinct from subspace clustering, where each individual cluster can appear in a different subspace. However, it highlights the fact that class distributions and boundaries can change depending on the space in which an algorithm examines them.

2.1.2 Manifold Learning

Subspace clustering makes the assumption that only a linear combination of features could be relevant to classification, an assumption that nonlinear feature extraction can avoid. Manifold learning assumes only that data occurs on some linear or nonlinear manifold, and its goal is specifically to learn the manifold structure of a given set of input data.

Two important algorithms for manifold learning are Isomap and locally linear embedding (LLE), which both exploit the fact that local regions of manifolds resemble Euclidean space. Isomap estimates distances within the manifold by finding shortest paths in a nearest-neighbor graph of the training data, and then embeds the data into Euclidean space in a way that preserves the manifold's intrinsic geometry (Tenenbaum et al., 2000). LLE learns the geometry of the manifold by finding locally linear fits for the training data (Roweis and Saul, 2000).

While these techniques were developed to help discover and visualize interesting structure in high-dimensional data, they have also been extended from data exploration tasks to classification. One way to do this is to learn a complete manifold for a set of data and then learn a classifier within that space (Chang et al., 2003). Another way is to assume that data from different classes lie on different manifolds, and then classify by finding the manifold to which a new data instance is closest (Xiao et al., 2011).

Like subspace-based approaches, manifold learning must make certain assumptions about the distribution of data in the feature space. While the assumption that data of a certain type lie in a

certain kind of geometric structure is mathematically elegant and useful for classifying much data, there are no *a priori* reasons why this would be the case for many or even most types of datasets. For instance, it is not clear that natural language sentences must lie on a smooth manifold structure, or that it would be illuminating or intuitive to classify them as such. This kind of domain-specific knowledge influences the decision to use a particular algorithm with its particular biases, which makes a general-purpose algorithm that eliminates some of these assumptions appealing.

2.2 Ensemble Methods

Ensemble methods are of interest to us because they provide the possibility of capturing diverse regions of the feature space with localized models. But beyond that benefit, they are the subject of extensive research in the machine learning community as a way to build a more powerful classifier, since an ensemble of multiple base classifiers can often outperform each individual classifier. In this section, I discuss a number of ensemble methods, particularly AdaBoost; in the next chapter I will discuss ensemble-based algorithms that use support vector machines, which are especially important to this thesis.

2.2.1 General Ensemble Methods

There are two main issues to address when it comes to generating ensembles: how to train each base classifier, and how to combine their results for the final prediction. Most algorithms combine via some sort of weighted voting, and so the majority of past research on ensemble methods has gone into the training phase and how to most effectively use a given base learning algorithm to capture different aspects of the dataset. There are several methods for doing this, all of which have different strengths in terms of dealing with noise, detecting outliers, and generally accommodating variability (Kotsiantis, 2011).

One way to ensure diversity in a set of classifiers is to train each classifier on the same training data but using different representation spaces. The random subspace method accomplishes this by using different subspaces of the complete feature space (Ho, 1998). The effect of this is that each classifier can focus on a different subset of the available attributes, which may not all be equally important for every data instance. This also has connections to other methods of feature selection, including subspace clustering as described in Section 2.1.1.

In contrast to the random subspace method, bagging tries to ensure a diverse set of classifiers by representing all data in the same space, but training using different samples of the training data (Breiman, 1996). The samples are drawn uniformly with replacement, and the size of a sample is typically the same as the size of the training set. Bagging is particularly effective when the base classification algorithm is unstable, meaning that small changes in the training set can result in large variation in the resulting model. Bagging is also robust to noise, as the sampling method ensures that noisy instances are unlikely to be over-represented in the training data for any base classifier (Dietterich, 2000).

2.2.2 AdaBoost

Boosting is an ensemble method similar to bagging, except that sampling of the training set is not uniform, but biased to attempt to learn complementary classifiers. Its goal is specifically to "boost" a weak learner [1] into a strong learner with arbitrarily high accuracy. The theory behind this method is based in computational learning theory, particularly the probably approximately correct (PAC) learning model (Valiant, 1984).

Perhaps the best-known boosting algorithm is AdaBoost (Schapire, 1999), which iteratively samples a subset from the training data according to some distribution of weights, trains a classifier using that subset, and then recalculates the weights based on what examples are incorrectly categorized by that iteration's classifier. The resulting complete model then uses a weighted vote of the classifiers in the ensemble to perform the final classifications, where votes are weighted according to the accuracies of the individual classifiers.

AdaBoost empirically tends to perform better than bagging, especially in clean datasets, because it increasingly focuses on training examples that could not be captured by classifiers learned on earlier iterations. Unfortunately, this same aspect also makes AdaBoost susceptible to noise in the training data, which can lead the algorithm astray (Dietterich, 2000). This is because mislabeled instances are very likely to be misclassified and hence receive high priority when the training set is resampled.

Finally, note that AdaBoost, like the other ensemble methods mentioned in this section, requires choosing a base classification algorithm. The parameters and inductive bias of the base algorithm necessarily impact the theoretical and practical properties of the ensemble approach, even though combining multiple classifiers still allows for a more flexible overall classifier.

2.3 Alternate Approaches

There is much other research into accommodating unusual and difficult class distributions. For example, class imbalances arise when one class is overrepresented by the data, which can bias the classifier towards overemphasizing that class (Kubat and Matwin, 1997). Outlier detection focuses on finding instances that differ markedly from the rest of the data; these may need to be filtered out as noise or examined further as important special cases (Hodge and Austin, 2004).

Other researchers have acknowledged the fact that data within a single class may be distributed in different clusters around the feature space (Japkowicz, 2001). One way this issue has been framed, particularly for concept learning, is in terms of small disjuncts (Holte et al., 1989). These represent small groups of data that form a relatively minor portion of a given class, hence causing models to either overfit or misclassify these. Note that we can view AdaBoost as learning classification rules for small disjuncts in its later iterations, which earlier iterations of classifiers are likely to have missed; accurately distinguishing these from noise is a significant problem for many learning algorithms in addition to AdaBoost.

Class imbalances, outlier detection, and small disjuncts all suggest that handling difficult class distributions well is critical to performing classification. However, all the methods in these areas are focused on a specific type of difficult distribution; few if any acknowledge the wider array of possible class distributions that present challenges to different models and classification algorithms.

[1] A weak learner is one that is guaranteed only to perform better than chance.

Chapter 3

SVMs and Ensembles of SVMs

In this chapter, I introduce support vector machines (SVMs), which are classifier models that are both empirically tested and theoretically sound. I also discuss exemplar SVMs, the approach on which I base my algorithm, and profile SVMs, another ensemble-learning algorithm that learns linear approximations of nonlinear models. Both demonstrate how an ensemble of linear SVMs can be utilized to learn multiple high-quality linear classifiers that are tuned to local regions of the feature space, a strategy flexible enough to be effective in a variety of class distributions.

3.1 Support Vector Machines

Support vector machines are classifier models with theoretical foundations in statistical learning theory and well-established algorithms for learning them. They have been empirically shown to perform well in many different types of real-world classification domains, such as the categorization of texts (Joachims, 1998) and of fMRI volumes (Mouro-Miranda et al., 2005).

I begin by introducing the simplest task for SVMs, namely a two-class problem where the classes are linearly separable, in which theory and intuition are most easily developed. Afterwards I move on to the nonseparable and nonlinear cases.

3.1.1 Linear SVMs

We are given a training set containing n examples of the form $(\vec{x}_i, y_i)$ where $\vec{x}_i \in \mathbb{R}^m$ is a feature vector and $y_i \in \{-1, +1\}$ is the class label (assuming two classes). The goal of the SVM learning algorithm is to learn the equation of a hyperplane $\vec{w} \cdot \vec{x} + b$ that separates the two classes, where $\vec{w} \in \mathbb{R}^m$ is a vector of feature weights and $b \in \mathbb{R}$ is a bias term. When the class labels are ± 1, this corresponds to using $g(\vec{x}) = sgn(\vec{w} \cdot \vec{x} + b)$ as the decision function. (Note that this and the derivations below follow (Alpaydin, 2010) closely.)

Since there are an infinite number of separating hyperplanes for any linearly separable dataset (see Figure 3.1), we include the additional requirement that the margin be maximized, where the margin is defined as the distance from the hyperplane to the closest training examples. This is a reasonable objective, because the hyperplane with the largest margin is the most generalizable linear discriminant; in other words, if we assume instances of the same class are likely to lie close to each other, the maximum margin hyperplane will be the one with lowest expected classification error on

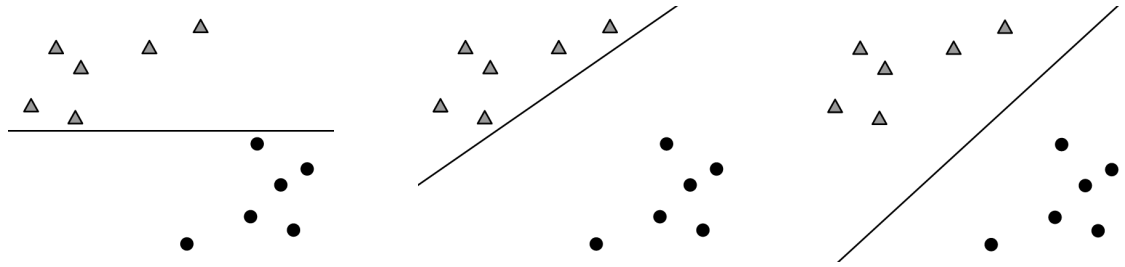


Figure 3.1: Three examples of the infinite number of hyperplanes that perfectly separate the sample data from two classes. The third is the most generalizable and has the largest margin.

unseen data. It is also most robust to noise, which might shift instances by small amounts in any direction.

We can encode these ideas as an optimization problem in the following way. Note that $y_i(\vec{w} \cdot \vec{x}_i + b) > 0$ whenever the hyperplane classifies $\vec{x}_i$ correctly. Furthermore, the signed distance from an instance $\vec{x}_i$ to the plane $\vec{w} \cdot \vec{x} + b$ is $(\vec{w} \cdot \vec{x}_i + b)/||\vec{w}||$ where $||\vec{w}||$ is the norm of $\vec{w}$. Hence maximizing the margin is equivalent to minimizing $||\vec{w}||$. However, note that scaling $\vec{w}$ and b by the same constant does not change the distance. So to determine a unique solution, we solve the following quadratic optimization problem:

$$\begin{aligned} &\min_{\vec{w},b} \ \frac{1}{2}||\vec{w}||^2 \\ &\text{subject to } y_i(\vec{w} \cdot \vec{x}_i + b) \geq 1 \ \forall i. \end{aligned} \tag{3.1}$$

The support vectors are defined to be those $\vec{x}_i$ that satisfy $y_i(\vec{w} \cdot \vec{x}_i + b) = 1$. These are the instances that lie on the margin of the hyperplane and can be considered the most difficult instances to classify.

If the data is not linearly separable, as is often the case, we can introduce slack variables $\xi_i \geq 0$ for each training instance $\vec{x}_i$, representing how far inside the margin (or on the wrong side of the hyperplane altogether) $\vec{x}_i$ is. We then add the sum of these slack variables as a penalty term in the objective function to get the following modified optimization problem:

$$\begin{aligned} &\min_{\vec{w},b} \ \frac{1}{2}||\vec{w}||^2 + C\sum_{i=1}^{n} \xi_i \\ &\text{subject to } y_i(\vec{w} \cdot \vec{x}_i + b) \geq 1 - \xi_i, \ \xi_i \geq 0 \ \forall i. \end{aligned} \tag{3.2}$$

In the nonseparable case, the support vectors also include training examples inside the margin or on the wrong side of the hyperplane, as these are also the difficult cases to classify. The regularization parameter, $C \in \mathbb{R}^+$, determines the relative importance of large margin versus accuracy on the training set: the smaller C is, the more important margin is. Tuning C, and more generally the goal of maximizing the margin, can help make SVMs robust to overfitting, especially in noisy datasets. See Figure 3.2 for a geometric depiction of the various quantities involved.

While the optimization problems in Equations 3.1 and 3.2 can be solved as-is with existing quadratic optimization software, the time complexity of doing so is dependent on the dimension of the feature space, which can be extremely large. Hence we generally solve the optimization problem using Lagrange multipliers and the dual formulation. This also greatly simplifies the process of

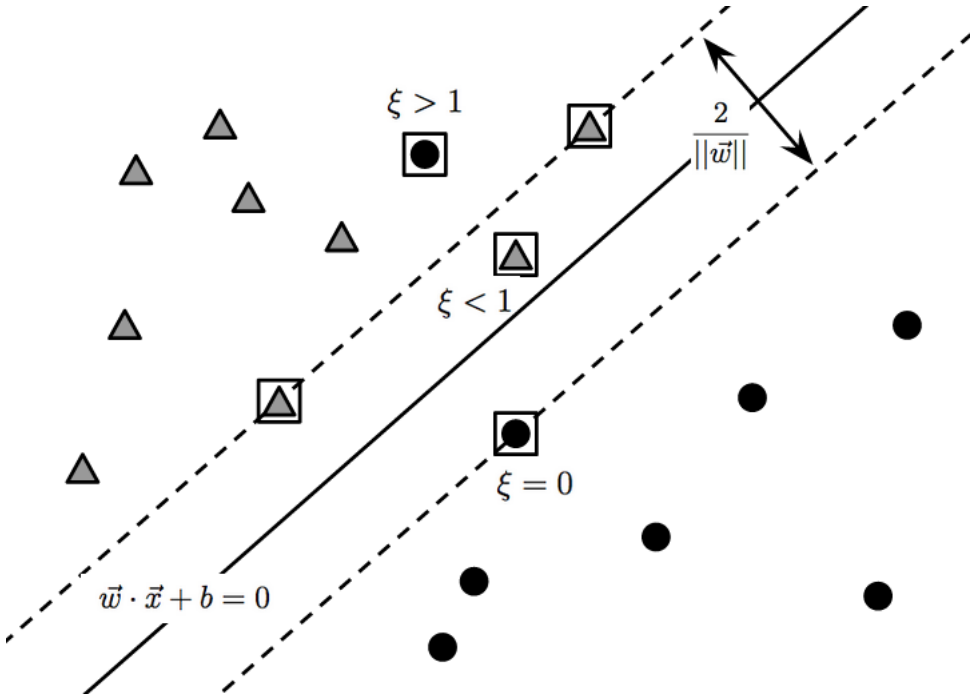


Figure 3.2: A linear SVM in a non-linearly separable dataset with margins indicated with dashed lines and support vectors in boxes. The width of the margin and values of slack variables ξ for certain points are indicated.

learning nonlinear SVMs by enabling the use of kernel functions; see Section 3.1.2 for more details.

The Lagrangian primal corresponding to the SVM optimization problem (for separable problems; adding slack variables is a simple modification of this) is

$$\min_{\vec{w},b} L_P = \frac{1}{2}||\vec{w}||^2 + \sum_i \alpha_i(1 - y_i(\vec{w} \cdot \vec{x}_i + b)) \tag{3.3}$$
$$\text{subject to } \frac{\partial L_P}{\partial \alpha_i} = 0 \ \forall i.$$

The dual formulation is

$$\max_{\alpha_i} \sum_i \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j \vec{x}_i \cdot \vec{x}_j \tag{3.4}$$
$$\text{subject to } \vec{w} = \sum_i \alpha_i y_i \vec{x}_i, \ \sum_i \alpha_i y_i = 0, \ \alpha_i \geq 0 \ \forall i.$$

This reformulation allows us to solve the problem in time depending only on the number of training instances. When Equation 3.4 is solved, most α_i vanish; these correspond to training instances that are far from the margin. We get $\alpha_i > 0$ when $\vec{x}_i$ is on the margin, or in the nonseparable case when $\vec{x}_i$ is inside the margin or misclassified. Hence the weight vector $\vec{w}$ is expressable just in terms of the support vectors.

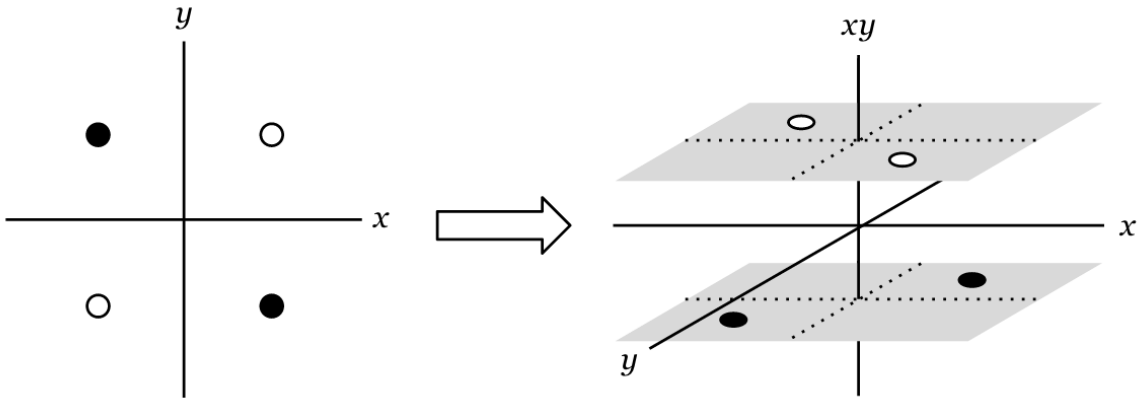


Figure 3.3: Nonlinear mappings to higher-dimensional spaces can make non-linearly separable datasets linearly separable. Here an XOR two-class dataset is mapped from (x, y)-space to (x, y, xy)-space, in which it can be separated by a hyperplane. This corresponds to a nonlinear function in the lower dimensional space.

Multiclass SVMs

While SVMs are fundamentally binary classifiers, they can be extended to handle $k > 2$ classes in a number of different ways. The two most common methods are one-against-all and one-against-one. In one-against-all, we train k SVMs, each with one class as the positive class and all others treated as a single negative class. Since multiple classifiers may classify a given instance positively, the decision function chooses the class for which $\vec{w} \cdot \vec{x} + b$ is maximum. In one-against-one, we train $\binom{k}{2}$ SVMs that separate all pairs of classes. Each classifier then votes for a class, and the class with the most votes is the predicted class.

Note that my PSVM algorithm uses the one-against-one approach as implemented in LIBSVM (Chang and Lin, 2011). Chang and Lin base this choice on the results in (Hsu and Lin, 2002), which show that, especially in the case of linear kernels, one-against-one has significantly better performance than one-against-all or other more complicated methods. For descriptions and analyses of other multiclass SVM approaches, see for instance (Duan and Keerthi, 2005).

3.1.2 Nonlinear SVMs

SVM learning algorithms can be extended to learn nonlinear classifiers through the use of basis functions, which map instances from the input feature space to another space, possibly of much higher dimension. When these mappings are nonlinear, learning a linear discriminant in the new space corresponds to learning a nonlinear discriminant in the original space. See Figure 3.3 for an example of this. Using a basis function $\phi(\vec{x})$, the objective function in Equation 3.4 becomes

$$\max_{\alpha_i} \sum_i \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j \phi(\vec{x}_i) \cdot \phi(\vec{x}_j). \tag{3.5}$$

Explicitly computing these mappings and the dot product in Equation 3.5 can be computationally expensive, especially since the target space might have extremely high dimension. Kernel functions allow us to perform these mappings implicitly, reducing the computational overhead and giving us

Figure 3.4: The idea behind the ESVM algorithm is to learn a separate SVM for each instance in the training set, so that each classifier can be specifically tuned to the visual characteristics of its exemplar without being overly general. For example, motorcycles viewed from the side and from the back visually have nothing in common, but an individual classifier can be trained to recognize a side-view or back-view image of a motorcycle very well, and an ensemble of such classifiers can then recognize motorcycles in general. Figure content from (Malisiewicz et al., 2011).

the power to map into infinite-dimensional spaces. In this case, the objective function becomes

$$\max_{\alpha_i} \sum_i \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j K(\vec{x}_i, \vec{x}_j). \tag{3.6}$$

Different choices of kernel functions result in classifiers of varying flexibility and power. Two popular kernel types are polynomials, or $K(\vec{x}_i, \vec{x}_j) = (\vec{x}_i \cdot \vec{x}_j + 1)^d$ for some degree d; and radial basis functions: $K(\vec{x}_i, \vec{x}_j) = \exp\left[-\frac{||\vec{x}_i - \vec{x}_j||^2}{2s^2}\right]$ for some radius s.

Note that the choice of kernel function is effectively a parameter for the SVM learning algorithm, and a major one at that; choosing a particular kernel restricts the hypothesis space of the algorithm to a certain class of functions, and choosing improperly for a given dataset may result in over- or underfitting.

3.2 Exemplar SVMs

My approach to dealing with difficult class distributions is based on the notion of exemplar SVMs (ESVMs) developed for object recognition (Malisiewicz et al., 2011). In object recognition, the goal is to classify images based on what objects they contain. The ESVM algorithm trains a separate SVM for each exemplar image from the training set, with that exemplar as the sole positive instance and many instances of the other classes as negative instances (see Figure 3.4 for an example). If one of these exemplar SVMs positively classifies a novel instance, then this suggests that the novel instance shares the class label of (i.e., depicts the same object as) that model's exemplar. Thus an ensemble of ESVMs can be used to classify new data instances, either through voting or a more complicated procedure.

Object recognition is a well-studied classification problem that exemplifies the kind of complex class distributions I am seeking to handle in a general way. In particular, the ESVM algorithm is an appealing starting point for this work, as it leverages the power of both SVMs and ensembles to accommodate this complexity.

Figure 3.5: Examples of visual synonymy (left) and polysemy (right). In the former, images are visually similar but present objects belonging to different semantic categories; in the latter, images are of the same object but are visually very different. Image taken from (Malisiewicz and Efros, 2008).

3.2.1 Motivation

The ESVM framework is especially suitable for object recognition for a number of reasons. In this domain, discriminative techniques have seen much success thanks to their ability to effectively handle massive quantities of negative instances by representing them implicitly in the decision boundary rather than storing them explicitly. A powerful discriminative classifier, such as an SVM, permits a greater degree of generalization than nearest-neighbor and other instance-based machine learning algorithms.

However, without the benefit of explicit association provided by nearest-neighbor approaches, these discriminative methods do not directly address the trickiest problems in object recognition, including visual synonymy and polysemy (see Figure 3.5). Visual synonyms are objects that look similar but have different class labels; these are data instances that might lie close together in the feature space but ought to be classified differently. Visual polysemes are objects that look different but have the same class label, corresponding to instances that are distant in feature space but must be classified identically. A classifier that learns a single decision boundary will not necessarily be able to handle these cases correctly.

Explicit association also permits the transfer of meta-data, such as information on image segmentation or object geometry, from the exemplar to a newly classified instance. Thus ESVMs combine the best aspects of discriminative and instance-based algorithms, resulting in an object recognition framework that can go beyond mere image retrieval.

These strengths of ESVMs also make them potentially useful in other domains with complex class distributions. An approach based on ensembles and exemplars provides comprehensive coverage of the feature space, which is critical for classes whose instances could potentially lie in a number of diverse regions of the space. The existence of at least one exemplar in each of these regions can allow the ensemble as a whole to recognize their presence, without the need to create a single overly general classifier to accommodate them.

3.2.2 Algorithm Description

As applied to object recognition, the ESVM algorithm works as follows. Each ESVM is trained after mining a subset of hard negatives from the set of all images that do not contain instances of the exemplar's category. This step can be accomplished via a bootstrapping algorithm, which trains an initial model with an arbitrary negative set and then iteratively retrains models using the

negatives that were incorrectly classified by the previous model (Felzenszwalb et al., 2010). Hard negative mining is used because the potential negatives far outnumber the single positive exemplar of a particular model; hence a subset of the negatives must be chosen to avoid class imbalances, and it is important to choose negatives that are likely to become support vectors.

The outputs of the independently-trained ESVMs must then be tuned or calibrated to make them compatible. This accounts for the fact that each model may report a potentially large number of false positives, since many images may be visually similar without actually containing the same object. The calibration is essentially equivalent to shifting the decision boundary of each SVM based on performance on a validation set. More generalizable models are boosted by having their decision boundaries shifted further from the exemplar, while the decision boundaries of poorly generalizable models are shifted closer to the exemplar.

Exemplar SVMs have achieved notable success in the realm of computer vision, but applications to other domains have been limited. Hajishirzi et al. use ESVMs as a component of an algorithm to learn alignments between sentences and the events they describe (Hajishirzi et al., 2012). A natural language setting such as this is an intriguing choice, as linear SVMs in general are well suited to text classification tasks (Joachims, 1998). However, to the best of my knowledge, there have been no other applications of the ESVM algorithm outside of vision. Hence one of the goals of this thesis has been to start with this algorithm, which handles the difficult class distributions in visual domains very well, remove its vision-specific elements, and adapt it to perform well in any complex dataset.

3.3 Localized and Profile SVMs

Another pair of classification algorithms that use an ensemble of linear SVMs are localized and profile SVMs (Cheng et al., 2010). Here the idea is again to combine SVMs with instance-based methods, specifically the k-nearest neighbor classifier, in order to learn local models of the example space. The localized SVM algorithm trains a new SVM model for each test instance, where the influence of each training instance $\vec{x}_i$ is weighted according to its similarity to the test instance $\overline{x}_j$. This similarity is represented by a function $\sigma(\vec{x}_i, \overline{x}_j)$ taking on real values between 0 and 1.

As expected, the localized SVM algorithm is very slow at test time. Profile SVMs improve this running time by learning a local SVM for each of k clusters of data, so that test instances can be classified by finding the most similar cluster and using that cluster's SVM. If k is much smaller than the number of test instances, then many fewer SVM models must be learned. However, this means that before learning the SVMs, the algorithm must perform a variation of k-means clustering on both training and (unlabeled) test data, called MagKmeans.

There are two main differences between MagKmeans and standard k-means. First, besides grouping together similar data, an additional objective is to balance the class distribution of training data within a single cluster, since the ultimate goal is to learn a local SVM model for each cluster. Second, while k-means clusters the training instances themselves, MagKmeans clusters similarity vectors $\vec{\sigma}_i$ corresponding to training instances $\vec{x}_i$, where each component is equal to $\sigma(\vec{x}_i, \overline{x}_j)$ for a test instance $\overline{x}_j$. Thus the training examples in a single cluster have correlated similarities to every test instance. Note that forming these similarity vectors does not use information about test instances' actual class labels, but it does assume that all test examples are available at once.

While the profile SVM algorithm performs very well, at least in the two-class cases reported in

(Cheng et al., 2010), and also improves in speed over localized SVMs, it still postpones all clustering and training of models until test time, unlike prototype SVMs. It also requires that the number of clusters be chosen ahead of time, which is exactly the kind of assumption we would rather not make; prototype SVMs, as I discuss in Chapter 4, are able to discover clusters of similar data on their own without knowing ahead of time how many there are. Despite these drawbacks and differences, profile SVMs are indicative of the fruitfulness of an approach that learns local linear models for complex class boundaries.

Chapter 4

The Prototype SVM Algorithm

This chapter describes how to train an ensemble of prototype support vector machines (PSVMs). The algorithm begins by training an ensemble of exemplar support vector machines (ESVMs). It then iteratively improves and generalizes the boundaries of the SVMs to achieve the final ensemble of PSVMs. I will also present a variation of this algorithm, in which an AdaBoost-like procedure repeatedly weights and samples the training data to focus on difficult-to-capture regions of the feature space. Such an addition can be particularly helpful in datasets with unusually difficult class distributions.

4.1 Prototype SVMs

There are three major components of the main PSVM algorithm: initialization, shifting of boundaries, and prediction. Algorithm 1 describes the high-level training of PSVMs, showing how these three components are used.

4.1.1 Initialization

The algorithm first trains an ensemble of ESVMs, with one model for each instance in the training set. This requires that the algorithm create a training set of positive and negative examples specific to each ESVM. Initializing positive sets is straightforward, as each set is a single example that serves as the exemplar for the ESVM.

In theory, negative sets could contain every instance of a different class from the exemplar. However, this is problematic for several reasons, as depicted in Figure 4.1. The first is that training any classifier in the presence of class imbalances can cause underrepresented classes to be ignored in favor of high classification accuracy; this problem is exacerbated when one class is represented by a single instance, as the exemplar framework dictates. The second reason is that in spaces with highly variable class distributions, distant regions of the space may have no influence in how local regions should be partitioned, so negatives far from the exemplar might be useless or even detrimental for learning good classifiers. Finally, in general, classes will not be linearly separable, so in order to learn relatively high-quality linear discriminants, the algorithm should choose a sample of the potential negatives that is linearly separable from the exemplar.

TRAIN(T)

Input: set of labeled training data, T
Parameters: number of iterations, s, and fraction of data to hold out for validation, v

Split data into training and validation sets, D and V.
$P \leftarrow [[d_i] \text{ for } d_i \in D]$ *# List of positive sets, each initially just the exemplar.*
$N \leftarrow [\text{CHOOSENEGATIVES}(D, d_i) \text{ for } d_i \in D]$ *# List of negative sets.*
for $j = 0, \ldots, s$ **do**
  $E_j \leftarrow [\,]$ *# The ensemble being trained on this iteration.*
  for $P_i \in P$ and $N_i \in N$ **do**
    Train a linear SVM, using P_i and N_i.
    Add this SVM to E_j.
  $a_j \leftarrow \text{TEST}(V, E_j)$ *# Accuracy of E_j on V.*
  $P, N \leftarrow \text{SHIFT}(D, E_j, P, N)$
return ensemble E_j with highest accuracy on V

Algorithm 1: Train an ensemble of PSVMs.

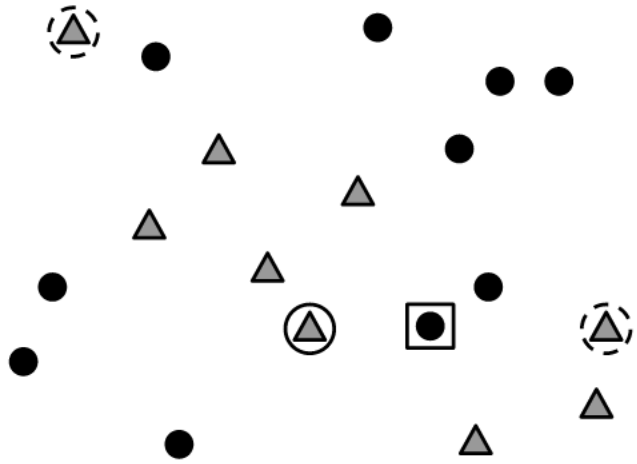


Figure 4.1: Problematic choices for initial negatives. The boxed instance is the exemplar and the instance in a solid circle is the closest negative. The instances in dashed circles are poor choices of negatives, either for being too far away and thus not relevant to local class distribution, or for being in a different direction from the closest negative and thus not linearly separable.

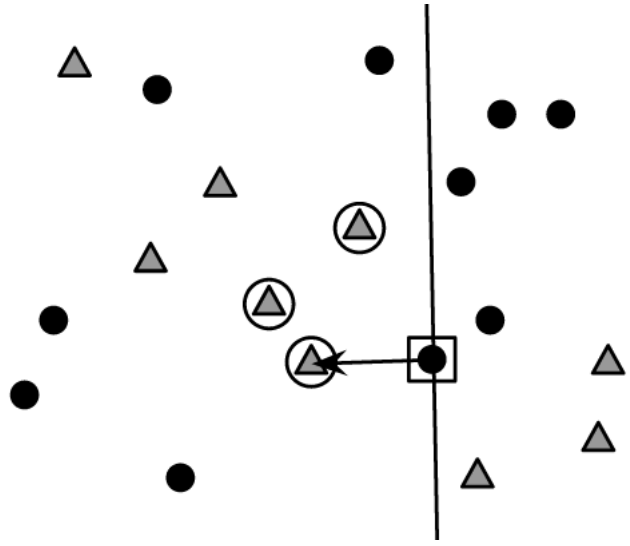


Figure 4.2: How negatives are initialized for the boxed exemplar. The vector to the closest negative as well as the hyperplane normal to it (and containing the exemplar) are shown. Here k is set to three, so the three closest negatives in the correct direction have been selected.

ChooseNegatives(D, d_i)

Input: set of training data, D, and training instance, d_i
Parameters: number of negatives to return, k

$N_i \leftarrow [\ ]$
$D_i \leftarrow$ all instances in D of a different class label from d_i
Compute Euclidean distance from d_i to each element of D_i.
Sort D_i in ascending order by distance.
$x \leftarrow$ closest negative in D_i
$\vec{n} \leftarrow (x - d_i)/||x - d_i||$ *# Normal vector to the hyperplane passing through d_i.*
for $d_j \in D_i$ **do**
  if $\vec{n} \cdot (d_j - d_i) > 0$ **then**
    Add d_j to N_i
    if $|N_i| = k$ **then**
      return N_i
return N_i

Algorithm 2: Initialize set of negatives for a given data instance.

To accommodate these issues, the algorithm chooses negatives in the manner described in Algorithm 2 and depicted in Figure 4.2. The algorithm first finds the negative closest in Euclidean distance to the exemplar. This defines a hyperplane passing through the exemplar and normal to the vector between the exemplar and that negative. The candidate negatives are then those that lie on the positive side of this hyperplane (i.e., the same side of the hyperplane as the closest negative). Note that no margin is enforced, and the hyperplane being considered is only a very rough approximation of the hyperplane that will be learned by the SVM algorithm.

From those candidate negatives, the algorithm chooses only a small number of negatives. This number can be user-selected, although empirically about seven negatives seems to work reasonably well for training the SVMs. This is not surprising; the discriminant that the SVM algorithm learns is expressed in terms of support vectors, which are the examples that lie closest to the margin, so in general omitting examples distant from the hyperplane does not affect which hyperplane is learned.

Finally, this small subset is chosen from the potential negatives closest to the exemplar, so that the training set for each model is kept mostly localized and the training process is not "distracted" by instances in distant regions of the feature space.

Once a negative set for each exemplar has been initialized, the algorithm trains an SVM for each exemplar, as in the ESVM algorithm.

4.1.2 Shifting

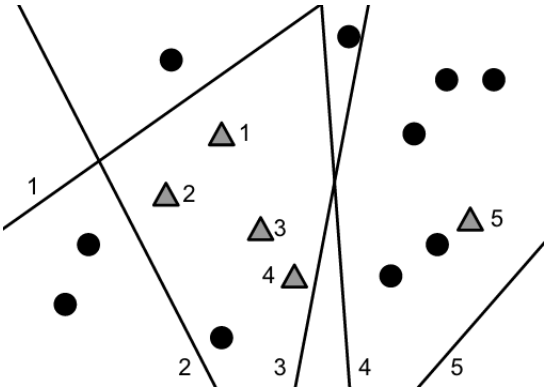
After the algorithm has trained the initial ensemble of ESVMs, the boundaries are shifted according to Algorithm 3. Shifting accomplishes three main goals:

1. It generalizes classifiers from a single exemplar to a cluster of nearby instances.

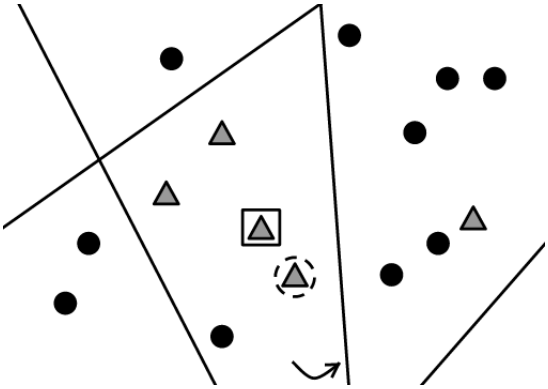
2. It adjusts boundaries that misclassified negative instances.

3. It removes useless classifiers from the ensemble altogether.

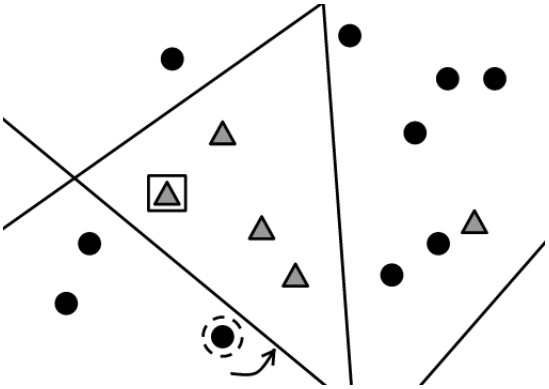
These tasks are depicted in Figure 4.3.



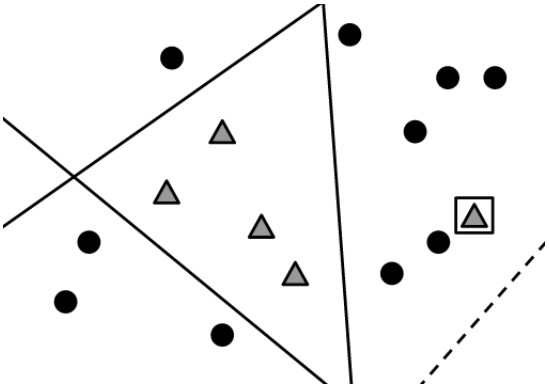
To the left is the initial ensemble of ESVMs. For clarity, only the models whose exemplars are triangles are shown, and the numbers indicate which exemplars correspond to which models.



Here the boxed exemplar's model is generalized to include the circled instance. The new instance becomes a support vector for the new model, which essentially coincides with a pre-existing SVM. We expect this to happen because these two instances are similar.



This image depicts boundary adjustment, where the boxed exemplar's model is shifted to account for the fact that it misclassified the circled instance.



Finally, since the boxed exemplar's model fails to capture any data, it is dropped from the ensemble. This model was poorly placed because the exemplar was so close to its negatives that the SVM algorithm opted to classify the exemplar as a negative (i.e., a noise instance) and optimize margin width instead.

Figure 4.3: How the various pieces of the shifting method take place. Note that the final ensemble already does a better job at capturing the important structure of the class distribution, namely the cluster of triangle instances.

SHIFT(D, E, P, N)

Input: set of training data, D; ensemble of models, E;
positive and negative sets for each model, P and N
Parameters: probability to add to negative set, p

$C \leftarrow$ [[ ] for $d_j \in D$] *# List of candidate models for each d_j.*
for $m_i \in E$ and $d_j \in D$ **do**
  if m_i classifies d_j positively **then**
    if $class(d_j) = class(m_i)$ **then** *# d_j is correctly classified.*
      Compute the distance of d_j to m_i's exemplar.
      Add m_i and its distance to the list of candidates C_j.
    else *# d_j is misclassified, i.e. a hard negative.*
      Add d_j to m_i's negative set N_i with some probability p.
for $d_j \in D$ **do**
  Add d_j to the positive set P_k, where m_k is the closest model in C_j.
for $m_i \in E$ **do**
  if m_i did not classify anything positively **then**
    Remove m_i from the ensemble, by removing P_i from P and N_i from N.
return P, N

Algorithm 3: Generalize, adjust, and drop models from the ensemble.

Generalization is at the core of what turns exemplar SVMs into prototype SVMs. Although training a classifier for each instance is effective for ensuring complete coverage of the feature space, in actuality many instances will co-occur in clusters, and a classifier that acknowledges this will more effectively capture the true class distribution.

The algorithm generalizes by adding new instances to the positive sets of models. For a given instance, a candidate model is one that classifies the instance positively and whose exemplar is of the same class as the new instance. These candidate models are ones that could be improved by adding this instance to their positive sets.

But it could be problematic to add each instance to all of its candidate models. Because linear classifiers simply divide the space into half-spaces, they run a serious risk of overgeneralizing by adding too many positives that may have nothing to do with the original exemplar and its cluster. This is the problem of failing to keep classifiers local that we encountered in Section 4.1.1 as well. To avoid this, if a particular instance is positively classified by multiple models, the algorithm only adds it to the positive set of the model with the closest exemplar. As in the initialization of negatives, this helps keeps each model tuned to a local region rather than attempting to capture wide swaths of the feature space.

The algorithm also improves the models by adding misclassified negative instances to their negative sets. This performs a kind of hard negative mining. If we discover that a model classifies an instance as a positive example when it should be a negative, we need to shift the linear separator to exclude the negative example. To do this, we consider adding the negative instance explicitly to the negative set for that model.

However, again since we are dealing with complicated class distributions and we want to keep models localized, some negatives actually should be classified incorrectly by individual models, and there is no principled way of identifying these in every possible dataset. Hence we only add to the negative set with some probability, as set by the user. This has the additional benefit of adding

TEST(D, E)

Input: set of testing data, D, and ensemble of models, E
1: **for** $m_i \in E$ and $d_j \in D$ **do**
2: **if** m_i classifies d_j positively **then**
3: Record weighted vote for m_i's class.
4: **for** $d_j \in D$ **do**
5: Output class with max votes as prediction for d_j.
6: $a \leftarrow$ classification accuracy over all data in D.
7: **return** a

Algorithm 4: Predict class labels for test data and compute overall accuracy.

randomness and hence robustness to the algorithm. See Figure 4.4 for an illustration of these issues.

The final step in the shifting algorithm is to remove models that do not classify any instances positively. Depending on what regularization parameter is chosen for the SVM learning algorithm, an SVM trained on unbalanced data can simply classify everything into the majority class. Such a classifier does not recognize anything as being in its positive class and thus is not useful for the overall ensemble. Fortunately, the use of an ensemble provides redundancy; since the algorithm learns a classifier for each instance of the training data, some models can be dropped from the ensemble without detriment, unless the class distribution is extremely difficult.

Dropping models is also a key reason why the ensemble of PSVMs is robust to noise. When the exemplar of an ESVM is a noisy instance, the SVM training algorithm has difficulty separating it with a reasonable margin, and thus is often inclined to classify the exemplar negatively and optimize the width of the margin instead. Such a model then classifies nothing as a positive instance and is dropped from the ensemble during shifting. Hence the SVM's proper handling of noise helps the ensemble of PSVMs do the same.

The shifting procedure occurs some number of times as specified by the user. After each shift, the algorithm trains a new ensemble with the new positive and negative sets, then tests this ensemble on a held-out validation set in the manner described in Section 4.1.3. The ensemble with the highest classification accuracy on the validation set is retained. Empirically, I found that a reasonable number of iterations was between 10 and 20, as by this time the accuracy generally has stabilized.

4.1.3 Prediction

Finally, Algorithm 4 describes how the ensemble of PSVMs predicts class labels for novel instances, both for validation after each shift and at test time. For each new instance, if a model classifies it positively, this corresponds to a "vote" for that model's positive class (i.e., the class of its original exemplar). I also generate the probability that the instance should be assigned the model's positive class, as is detailed in (Chang and Lin, 2011), and sum these probability values rather than the raw votes. This allows models to make weighted votes based on their confidence levels. Then the predicted class label assigned by the entire ensemble is simply the class with the maximum sum of these weighted votes.

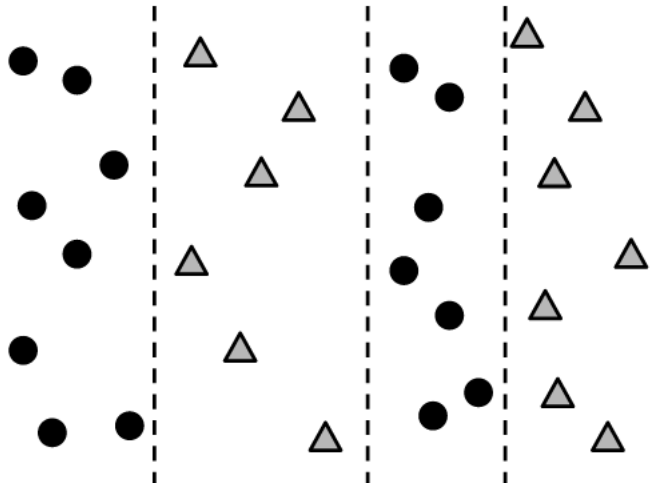
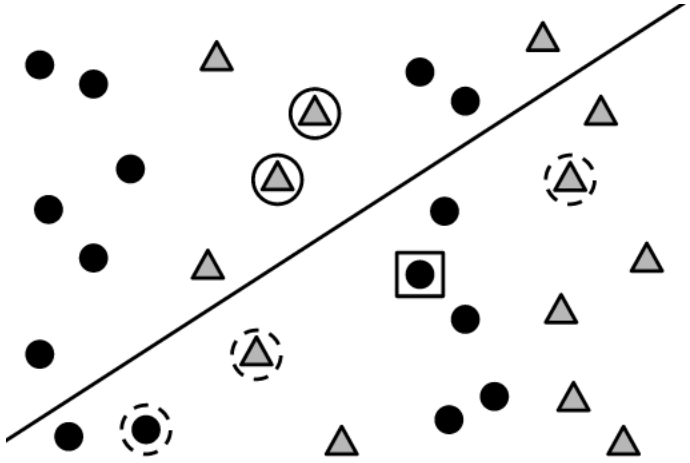


Figure 4.4: Some tricky cases to consider for the shifting algorithm. Note that the data occurs in alternating stripes, as is depicted in the above image, so classifiers that approximate the vertical boundaries can be considered "good models".

Below is the model shown for the boxed exemplar, whose negative set is circled in solid lines. The triangles with dashed circles are negatives that have been misclassified by the model, or what we call hard negatives. One would help lead to a vertical line while the other would be a distraction, but there is no *a priori* way to distinguish between them that could not be foiled by another dataset. Hence the algorithm randomly decides which negatives to include.

The dot with a dashed circle is a positive that could be added to the positive set, but does not provide information about the exemplar's local region and could hinder the process of discovering the vertical line. It is worth noting that in contrast to the negative sets, in the case of positives we do have a principled way of determining relevance for a given model, namely proximity, since the one thing we require from PSVMs is that they capture a local generalization of their initial exemplar.



TrainWithSampling(D)

Input: set of labeled training data, D
Parameters: number of iterations, r; fraction of data to sample initially, f; fraction of data to sample subsequently, f'

```
1: Initialize all weights to 1/n.          # n is the size of D.
2: S ← Sample(D, f)
3: for r iterations do
4:    E ← Train(S)
5:    a ← Test(D, E)                       # Accuracy of the ensemble.
6:    Reweight(D)
7:    S ← S ∪ Sample(D, f')
8: return E
```

Algorithm 5: Train a sequence of ensembles of PSVMs using sampling and reweighting.

4.2 PSVMs with Weighted Sampling

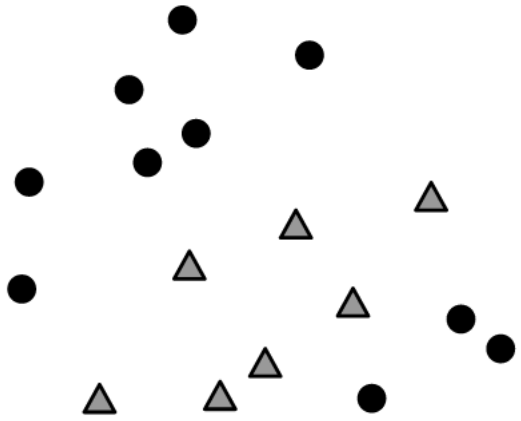
Although the algorithm described in the previous sections can handle a variety of complex data distributions, there are at least two reasons why we might want to modify it. Figure 4.5 shows a simple example in which this is the case. First, there is computational overhead in training a separate SVM for every instance in the training set, especially if the class distribution is not in fact particularly complicated. Hence training an ensemble using just a sample of the training data can provide increased efficiency without necessarily decreasing classification accuracy.

Second, iteratively reweighting and resampling the training set, as AdaBoost does, can help capture difficult regions of the feature space that the PSVM algorithm as presented in Section 4.1 might overlook. The SVM algorithm, particularly with a fairly large regularization parameter, is a powerful discriminative classifier in part due to its robustness to noise. Yet this can backfire in datasets where classes in the feature space truly are fragmented in small groups, which the SVM algorithm might mistake as noise. Effectively, AdaBoost's oversensitivity to noise (Dietterich, 2000) paired with SVM's robustness to noise is capable of modeling truly complex datasets as accurately as possible.

To imitate AdaBoost's ability to focus on hard-to-capture regions of data through reweighting, I implemented a similar sampling and reweighting scheme as a wrapper around the previously described PSVM algorithm; this wrapper method is Algorithm 5. As in AdaBoost, the algorithm repeatedly samples from the training data according to a distribution of weights, trains a classifier (which in this case is an ensemble of classifiers), and then reweights the data based on the performance of that classifier on the training set, where the misclassified instances are more likely to be chosen on the next iteration.

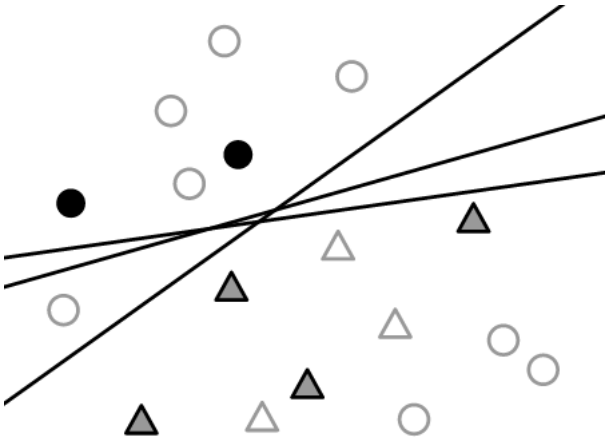
However, there are some notable differences between this method of resampling and AdaBoost. Rather than generate a new classifier using a new sample of data on each iteration and add it to a growing ensemble, the algorithm takes the union of the current data sample and the new sample and retrains the entire ensemble of classifiers from scratch. This is because we already have a principled way of creating the ensemble, using the exemplar approach, and to use boosting to create an ensemble of ensembles would be overkill.

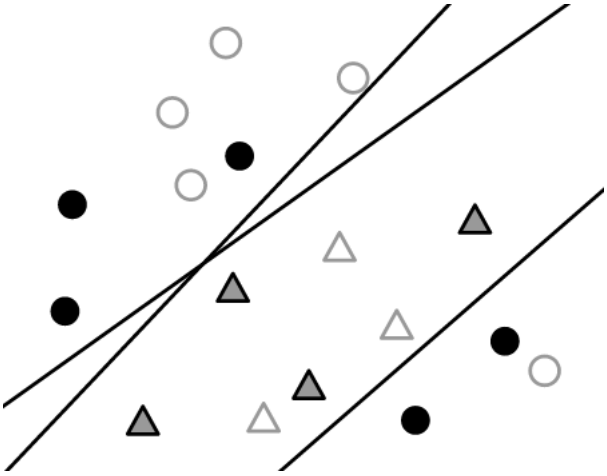
Furthermore, since the algorithm unions the new and old samples on each iteration, there is no need to have positive weight for correctly classified instances, since those are likely to have come



This dataset is mostly linearly separable, so learning a separate ESVM for every instance is overkill, but some care must be taken to include the small cluster in the bottom right.

On the right is the initial sample of data (shown darkened), as well as the final ensemble trained using this sample, where three of the initial ESVMs have been dropped and the others shifted. Note that the small cluster was not represented in the sample and hence has not been captured.





On the left is the next iteration, with the sample shown darkened. This new final ensemble captures the class distribution without the performance overhead of training many unnecessary SVMs.

Figure 4.5: An example of how weighted sampling can help in certain types of datasets.

REWEIGHT(D)

Input: set of training data, D

```
for d_i ∈ D do
    if d_i was correctly classified then
        weight(d_i) = 0
    else
        weight(d_i) = 1/z          # z is the number of misclassified instances.
```

Algorithm 6: Adjust weights of data instances for sampling.

from the old sample and hence will still be included. Thus the weights of these instances are set to zero rather than simply decreased as in AdaBoost. The weights of incorrectly classified instances are then renormalized so that they sum to one. More complicated weighting schemes are certainly possible; for example, the weights could be adjusted in the same way as in AdaBoost (Schapire, 1999). Algorithm 6 details the reweighting procedure.

Adding this level of iteration clearly increases the complexity of the algorithm, both in the sense of intellectual complexity for the programmer and computational complexity. But in practice, I find that more than two or three iterations of sampling are rarely necessary. In fact, as we will see in Chapter 5, AdaBoost rarely improves linear SVMs significantly, possibly due to the fewer degrees of freedom in a linear classifier versus a nonlinear one.

In this version, the number of times that resampling occurs, as well as the percentage of the training set that is sampled or resampled on each iteration, are user-defined parameters for the algorithm. When the entire training set is used and no resampling occurs, this corresponds to the normal version of PSVMs as described in Section 4.1. While ability to sample from different parts of the space can improve the performance of PSVMs, this is not necessarily the case. Also note that in general there is a tradeoff in efficiency between sample size and number of iterations. The resampled version with small sample sizes and a small number of iterations could also be more flexible as well as relatively efficient.

4.3 Analysis

The final goal for this chapter is to determine the asymptotic running time of the PSVM algorithm. I will also describe some practical caveats and factors to consider in addition to the worst-case computational complexity.

The core of the training algorithm (TRAIN, Algorithm 1) is, of course, training one linear SVM for each data instance. If there are n training instances, then training a linear SVM is $O(n^2)$, making the total running time $O(n^3)$. Note that there are algorithms for speeding up or approximating the SVM learning process, such as (Joachims, 2006) and (Franc and Sonnenburg, 2008), that can reduce this running time substantially. Also note that this is a very conservative upper bound in this particular context, as I will discuss later in this section.

Aside from training the SVMs, the running time of training the ensemble of PSVMs depends on the running time of CHOOSENEGATIVES, TEST, and SHIFT. (Note that I assume the iteration constants s in Algorithm 1 and r in Algorithm 5 are small relative to both the amount of data and dimension of the feature space.)

Three main tasks happen for each data instance in ChooseNegatives (Algorithm 2). First Euclidean distances are computed to $O(n)$ negatives; each distance computation is $O(m)$ where m is the dimension of the feature space, so in total this takes $O(nm)$ time. Then the negatives are sorted, which is $O(n \log n)$. Finally the directions of the negatives are checked relative to the closest negative to the exemplar, which is $O(nm)$ due to the dot product. So overall, ChooseNegatives is $O(nm+n \log n))$, which means that choosing negative sets for all of the instances is $O(n^2m+n^2 \log n)$.

For Shift (Algorithm 3), the most time-intensive section is the loop in lines 2–8, during which each training instance is classified by each model in the ensemble, and the distance between instances is computed (in the worst case) every time. For SVMs with a linear kernel, a single classification requires taking a single dot product, so these operations are both $O(m)$. Hence Shift is an $O(n^2m)$ algorithm.

Predicting a class label for a single new instance requires classifying it using each of $O(n)$ linear SVMs, which, as described above, takes $O(m)$ time. So Test (Algorithm 4) is $O(nm)$ for a single novel example.

Finally, note that adding weighted sampling to the PSVM algorithm does not increase asymptotic runtime, as the operations in Algorithm 5 other than Train, like computing the weight distribution and drawing samples, are linear operations. Therefore, for both versions, the overall asymptotic running time to train an ensemble of PSVMs with n training instances and m features is $O(n^3 + n^2m + n^2 \log n)$.

However, note that in practice the running time will virtually always be much lower than this. There are three key assumptions we make for worst-case running time analysis that are unrealistic, though impossible to make theoretical guarantees about without detailed knowledge of precise data distributions.

1. All the training data is used in each of the above methods.

2. In particular, all the training data is used to train each SVM.

3. All the models are retained throughout all the iterations of the algorithm.

Item 1 will certainly never be true in the version of PSVMs with weighted sampling, since a sample of the complete dataset is passed to the Train method. Furthermore, in line 1 of Train the data is split into training and validation sets, and all further methods (besides Test in line 9) are called with just the former. For each of these cases, it is likely that relatively high proportions of the data will be used for training, so assuming a value of n is reasonable for asymptotic runtime.

We assume item 2 when we claim that training a linear SVM takes $O(n^2)$ time, and certainly bizarre datasets are conceivable in which all the data given to Train will be used in the positive and negative sets of all SVMs. However, remember from Section 4.1.1 that initial sets are seeded with a single positive exemplar on the one hand and a small number (e.g. seven) negatives on the other; furthermore, positives are added only with the distance restriction as described in Section 4.1.2, while negatives are only added with some small probability (e.g. 0.5%). Thus the sets of data each model is trained on are generally only small subsets of the full dataset.

Item 3 is also conceivably true in some outlying cases. But in reality, the urge of the SVM algorithm to maintain large margins, and the use of relatively small (initially singleton) positive sets, implies that many SVMs classify nothing positively and are dropped from the ensemble during

the first few iterations. More details on empirically how many models are retained in the final ensemble will be described in Chapter 5. Also note that this behavior collaborates nicely with the growth of positive and negative sets in item 2 to keep running times low: in early iterations, there are many models but small sets, while later on there tend to be fewer models with larger sets.

Chapter 5

Experiments

This chapter describes experimental results comparing the PSVM algorithm against a number of different classification learning algorithms on a variety of datasets. I begin by describing the setup of the experiments and the nature of the datasets and algorithms used. Then I present and discuss my results on both noiseless and noisy data. Overall, the PSVM algorithm performs better than the other algorithms I tested in datasets with the most complicated class distributions, and the performance of the standard PSVM algorithm is not significantly worse than other algorithms when applied to simpler data distributions. PSVMs with weighted sampling also perform very well in complicated datasets but tend to perform worse on simpler datasets as compared to other algorithms.

5.1 Experimental Setup

5.1.1 Methodology

I tested both the basic PSVM algorithm and the version with weighted sampling against the following algorithms.

- C4.5: a decision tree learning algorithm, which recursively splits the data based on the attribute that maximizes information gain with respect to the classes (Quinlan, 1993).
- AdaBoost with C4.5 as the base classifier: see Chapter 2 for a description of AdaBoost.
- SVMs with a linear kernel: see Chapter 3 for a description of SVMs.
- AdaBoost with linear SVMs as the base classifier.
- SVMs with a polynomial (quadratic) kernel.
- Multilayer perceptrons: a neural network classifier, with sigmoid units trained using backpropagation (Rumelhart et al., 1986).

These are all commonly used classifier learning algorithms that perform well on average, but generally have different strengths. C4.5 is a standard algorithm that researchers frequently use for comparisons; it performs particularly well in combination with AdaBoost. Linear SVMs are the state-of-the-art algorithm for learning linear separators. AdaBoosted linear SVMs are a reasonably

	Dataset	Instances	Features (type)	Classes (instances per class)
Synthetic	isolated	800	2 (real)	2 (500 / 300)
	striated	800	2 (real)	2 (400 / 400)
	spirals	194	2 (real)	2 (97 / 97)
Benchmark	iris	150	4 (real)	3 (50 / 50 / 50)
	glass	214	9 (real)	6 (70 / 76 / 17 / 13 / 9 / 29)
	vehicle	846	18 (integer)	4 (212 / 217 / 218 / 199)
	segment	2310	19 (real)	7 (330 each)
Real-world	twitter	600	2715 (integer)	4 (99 / 157 / 30 / 314)

Table 5.1: The datasets used in the experiments. The top three are synthetic, with the first two created by myself and the spirals dataset taken from CMU's Neural Networks Benchmarks (White et al., 1995). The next four are benchmark datasets from UCI's Machine Learning Repository (Bache and Lichman, 2013). The Twitter dataset is from SemEval (Wilson et al., 2013).

close approximation to PSVMs, as both algorithms learn ensembles of linear SVMs that capture different parts of the example space. Polynomial SVMs are common "off the shelf" classifiers, so performance comparable to theirs is indicative of strong performance in a variety of types of data. Finally, multilayer perceptrons are flexible classifiers able to represent arbitrary boolean-valued or continuous functions, given sufficient training data and an appropriate number of hidden nodes. This is a goal close to that of PSVMs, so again comparable performance would suggest PSVMs are achieving that goal.

For each algorithm, I used the implementations provided by Weka (Hall et al., 2009), an open-source library of data mining algorithms. In particular, I used Weka's wrapper of LIBSVM (Chang and Lin, 2011) for the SVM experiments, as I also used LIBSVM for the PSVM algorithm. LIBSVM is a fast and easy-to-use library for support vector classification and regression.

I also used the default parameters provided by Weka for all the algorithms, except I reduced the training time of multilayer perceptrons from 500 to 200 for reasons of time. For PSVMs I used the following default parameters: $v = 25\%$ (percent of data used for validation), $s = 10$ (iterations of shifting), $k = 7$ (number of initial negatives), and $p = 0.5\%$ (hard negative mining probability); see Chapter 4 for details on how these parameters are used. In the sampled version, I started with $f = 50\%$ of the data and resampled $f' = 25\%$, for a total of two samplings. Note that for all the algorithms, these parameters were intentionally chosen to be reasonable but not optimal. Hence these can be considered pessimistic estimates of accuracy, since in practice, parameters would be optimized for the data.

5.1.2 Datasets

There are three general kinds of datasets on which I tested my algorithm; these are listed in Table 5.1. First are synthetic datasets, specifically designed to have unusual class distributions to provide a proof of concept of the power of PSVMs. Figure 5.1 shows what these datasets look like; by being two-dimensional, they also allow easier visualization of both the data itself and the algorithm's behavior. The spirals dataset is a standard benchmark for neural networks originally from CMU

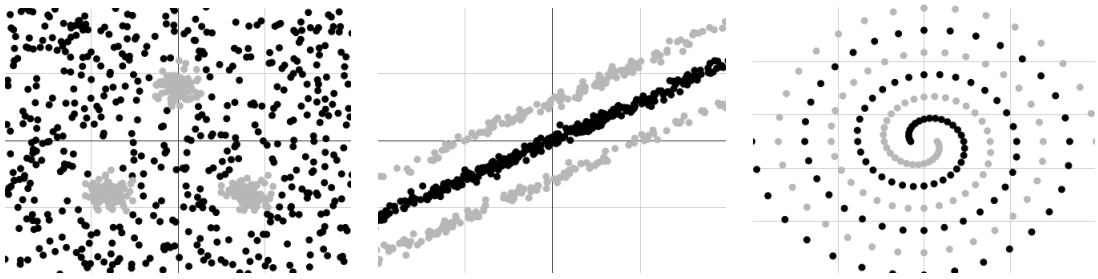


Figure 5.1: Images of the three synthetic two-dimensional datasets on which I ran experiments: isolated, striated, and spirals.

(White et al., 1995); the others I generated myself. In isolated, each cluster is normally distributed and the background data is uniform outside three standard deviations of each cluster's mean. In striated, each stripe is normally distributed with greater standard deviation in one direction.

Next are benchmark datasets from UCI's Machine Learning Repository (Bache and Lichman, 2013). These allow for comparisons with other algorithms in the literature. I selected these datasets without regard to the results of the current study, and they were the only such datasets used.

The last dataset is a natural language classification task from the Semantic Evaluation (SemEval) workshop (Wilson et al., 2013). The data consists of raw Twitter messages, or tweets, and the task is to classify them according to their sentiment as objective (i.e., no sentiment), positive, neutral, or negative. I included this dataset as an example of a challenging real-world domain.

For the benchmark datasets, I used the features provided in those sets. The Twitter dataset contained only raw tweets and sentiment labels, and hence I preprocessed and featurized that dataset. Much research has gone into good feature representations for natural language texts and tweets in particular, but as the focus of this work is not sentiment analysis, I used a basic but reasonable set of features for this data; my feature set included single words, links, usernames, hashtags, standard emoticons, and words from the MPQA Subjectivity Lexicon (Wilson et al., 2005). For more details on sentiment analysis in Twitter, see for instance (Barbosa and Feng, 2010) or (Davidov et al., 2010).

5.2 Results and Discussion

For each algorithm, I report the results of ten-fold cross validation on each dataset. I applied the 10-fold cross-validated t-test to construct a 95% confidence interval for the difference in accuracy rates of the algorithms. Table 5.2 compares standard PSVMs with the other algorithms, and Table 5.3 does the same for PSVMs with weighted sampling. Also see Table 5.4 for counts of wins, losses, and ties over the other algorithms.

Overall, the PSVM algorithm performs about as well as the other algorithms in all datasets, and has significantly higher accuracy in the datasets with the most difficult class distributions. Its performance is especially impressive in the tricky synthetic datasets. This is no surprise; while those datasets were not designed specifically for this algorithm, they were designed to exemplify particularly difficult class distributions. Although PSVMs do not perform as impressively in the other domains, in general they do not perform significantly worse than the other algorithms. The exceptions are glass and segment; I hypothesize that this is because these are the datasets with the largest number of classes, and the extension of SVMs to multiclass domains is not as natural as it

Dataset	PSVM without Sampling	C4.5	Boosted C4.5	Linear SVM	Boosted Linear SVM	Polynomial SVM	Multilayer Perceptron
isolated	90.9 ± 4.07	98.1 ± 1.79 •	98.5 ± 1.46 •	62.5 ± 0.0 ○	62.5 ± 0.0 ○	81.9 ± 4.38 ○	80.6 ± 2.81 ○
striated	97.3 ± 1.66	69.4 ± 13.4 ○	90.1 ± 7.38 ○	48.5 ± 3.94 ○	53.4 ± 16.3 ○	72.8 ± 4.18 ○	75.0 ± 25.0 ○
spirals	21.8 ± 18.9	0.0 ± 0.0 ○	0.0 ± 0.0 ○	0.0 ± 0.0 ○	5.63 ± 8.97 ○	0.0 ± 0.0 ○	20.6 ± 28.9
iris	96.0 ± 4.42	94.0 ± 6.29	94.0 ± 5.54	98.7 ± 2.67	97.3 ± 3.27	96.7 ± 4.47	97.3 ± 4.42
glass	52.8 ± 8.72	69.1 ± 6.40 •	72.9 ± 7.85 •	63.5 ± 8.08 •	63.1 ± 7.73 •	69.7 ± 6.55 •	70.6 ± 8.82 •
vehicle	79.4 ± 4.49	73.8 ± 4.48 ○	75.7 ± 3.56	80.4 ± 4.50	80.4 ± 4.23	80.4 ± 4.53	79.7 ± 4.61
segment	95.2 ± 1.18	97.1 ± 0.93 •	98.1 ± 0.85 •	96.3 ± 0.93 •	96.1 ± 0.89	95.8 ± 1.38	96.2 ± 1.30
twitter	55.8 ± 5.12	54.5 ± 2.89	60.5 ± 7.99	62.2 ± 3.66 •	62.2 ± 4.15 •	52.3 ± 5.54	52.3 ± 5.54

Table 5.2: Results for standard PSVMs. Shown are classification accuracy means with one standard deviation. ○ indicates statistically significant improvement of PSVMs over the other algorithm, • indicates statistically significant degradation.

Dataset	PSVM with Sampling	C4.5	Boosted C4.5	Linear SVM	Boosted Linear SVM	Polynomial SVM	Multilayer Perceptron
isolated	84.0 ± 7.0	98.1 ± 1.79 •	98.5 ± 1.46 •	62.5 ± 0.0 ○	62.5 ± 0.0 ○	81.9 ± 4.38	80.6 ± 2.81
striated	96.9 ± 3.84	69.4 ± 13.4 ○	90.1 ± 7.38 ○	48.5 ± 3.94 ○	53.4 ± 16.3 ○	72.8 ± 4.18 ○	75.0 ± 25.0 ○
spirals	60.7 ± 34.9	0.0 ± 0.0 ○	0.0 ± 0.0 ○	0.0 ± 0.0 ○	5.63 ± 8.97 ○	0.0 ± 0.0 ○	20.6 ± 28.9 ○
iris	86.7 ± 13.0	94.0 ± 6.29	94.0 ± 5.54	98.7 ± 2.67 •	97.3 ± 3.27 •	96.7 ± 4.47 •	97.3 ± 4.42 •
glass	47.6 ± 12.9	69.1 ± 6.40 •	72.9 ± 7.85 •	63.5 ± 8.08 •	63.1 ± 7.73 •	69.7 ± 6.55 •	70.6 ± 8.82 •
vehicle	75.4 ± 3.83	73.8 ± 4.48	75.7 ± 3.56	80.4 ± 4.50 •	80.4 ± 4.23 •	80.4 ± 4.53 •	79.7 ± 4.61 •
segment	94.4 ± 1.86	97.1 ± 0.93 •	98.1 ± 0.85 •	96.3 ± 0.93 •	96.1 ± 0.89 •	95.8 ± 1.38	96.2 ± 1.30 •
twitter	54.0 ± 8.86	54.5 ± 2.89	60.5 ± 7.99	62.2 ± 3.66 •	62.2 ± 4.15 •	52.3 ± 5.54	52.3 ± 5.54

Table 5.3: Results for PSVMs with weighted sampling. Shown are classification accuracy means with one standard deviation. ○ indicates statistically significant improvement, • statistically significant degradation.

is for the other algorithms. The glass domain has an especially small number of instances in certain classes, and since we hold out 25% of the training data for validation after each shift, this may explain PSVM's poor performance in this dataset.

It is worth making a few observations about the performance of the other algorithms on these datasets. Decision trees do quite well in the isolated dataset, which consists of axis-aligned clusters that the C4.5 algorithm is easily able to accommodate. In addition, linear SVMs perform very well on the Twitter dataset; linear SVMs are known to perform well in natural language domains (Joachims, 1998), where feature vectors are typically sparse and high-dimensional. Note how closely these strengths are tied to particular datasets and the particular models these algorithms learn. We see the drawbacks of this specificity clearly in the results; C4.5 falls apart in even a clean dataset with non-axis-aligned clusters, such as striated, and linear SVMs are unable to handle non-linearly separable data.

AdaBoost generally improves the performance of C4.5, as other researchers have noted. However it seems to help linear SVMs very little, hence justifying the use of few iterations in the version of PSVMs with resampling. AdaBoosted linear SVMs finished in fewer than 10 iterations in the

	Noiseless	Noisy
Regular	16 - 13 - 19	14 - 7 - 21
Sampling	14 - 23 - 11	13 - 23 - 6

	Noiseless	Noisy
Regular vs. Sampling	1 - 1 - 6	3 - 1 - 3

Table 5.4: The left table shows the total number of wins, losses, and ties for PSVMs and PSVMs with weighted sampling over all the other algorithms, in noiseless and noisy datasets. The right table shows wins, losses, and ties of regular PSVMs over PSVMs with weighted sampling in noiseless and noisy datasets. Overall, standard PSVMs perform on par with the other algorithms, and perform somewhat above average in the presence of noise. On the other hand, PSVMs with weighted sampling do not perform as well, especially in noisy datasets.

synthetic datasets and glass, and performance on the others with as many as 100 iterations was statistically identical to performance with 10 iterations.

The spirals dataset is especially difficult for all algorithms, though we would expect good results from SVMs with a radial basis kernel. Out of the algorithms I tested, multilayer perceptrons are the only other algorithm besides PSVMs that perform relatively well on spirals. Multilayer perceptrons are capable of representing arbitrary functions, but only given sufficient training time, data, and hidden nodes; their major downsides are needing to define the network structure prior to learning, and their vastly longer training time. The focus of these experiments and this thesis in general is improving classification accuracy, so I did not time the algorithms precisely. However, it is worth noting that multilayer perceptrons took on the order of days to run the full suite of experiments, whereas the other algorithms, including PSVMs, each took on the order of minutes or hours.

PSVMs with weighted sampling tend to perform worse than standard PSVMs on the UCI benchmark and Twitter datasets. It seems likely that unless data distributions are particularly difficult, such as in the spirals dataset, the benefits of using all the available training data outweigh the benefits of sampling and reweighting the data. It is also possible that optimizing the percentage of data sampled and the number of times sampling occurs could improve performance. Though we could determine close-to-optimal parameter settings with grid search, our goal is to devise an algorithm to handle the widest range of complex datasets without the need for significant parameter tuning; in this regard, PSVMs with weighted sampling are not ideal.

Table 5.5 indicates which iteration of shifting is used as the final ensemble (where iteration zero would be the initial ensemble of ESVMs). This shows that the initial ESVM ensemble in fact never has the highest accuracy on the validation set, and so the shifting operation improves the performance of the model and reduces its size. In particular, the reduction in ensemble size implies that the PSVM algorithm will scale better to larger datasets than ESVMs will.

Comparing the sizes of final ensembles is also important in assessing the performance of the two versions of PSVMs. Clearly the sampled version generates smaller ensembles, though typically at the cost of accuracy. Furthermore, it appears that the number of models retained in the final ensemble depends on not only the size of the dataset (which we would expect, given that we learn an initial ESVM for each data instance), but also to some extent the complexity of class boundaries or the degree to which they can be linearly approximated. For instance, if we look at the standard PSVM ensembles in the noiseless datasets, isolated and striated both have 800 instances, but isolated consists of circular clusters while striated consists of linear clusters; and accordingly, the final ensemble in isolated has over twice as many models as the final ensemble in striated. As another

Noiseless datasets	ESVM	PSVM		Sampling	
isolated	720	5	323	6	124
striated	720	8	149	6	88
spirals	175	5	47	4	29
iris	135	7	54	8	13
glass	193	7	35	7	15
vehicle	761	8	110	8	61
segment	2079	8	453	8	172
twitter	540	9	42	7	32

Noisy datasets	PSVM		Sampling	
isolated	6	122	6	70
striated	6	170	6	93
spirals	4	56	6	20
iris	8	24	8	11
glass	8	37	6	20
vehicle	8	113	8	64
segment	8	323	8	164

Table 5.5: Additional statistics for the final ensembles of PSVMs for noiseless and noisy datasets. Also listed is the number of models for the ESVM ensemble, i.e., the number of instances in the training set for each fold; these numbers are identical for noiseless and noisy datasets. In each PSVM column, the left value is average iteration used, the right value is the average number of models in the final ensemble. All averages are rounded to the nearest whole number.

These statistics indicate that the shifting procedure is both improving accuracy and decreasing ensemble size compared to the initial ensemble of ESVMs. In addition, the version with weighted sampling uses a smaller number of models than the standard version, which we would expect because the ensemble is trained using a subset of the data; this is worth considering if ensemble size is a concern.

piece of evidence, note the relatively small number of models in the Twitter dataset, which is best captured by the linear classifiers I tested.

5.3 Noise Experiments

Considering classification in the presence of noise is important when the objective is high performance in real-world datasets. It is especially important given that the goal of the PSVM algorithm, namely to be flexible enough to capture difficult data distributions, is exactly the kind of goal that can result in vulnerability to noisy data. Hence I repeated the experiments described above, on the same datasets but with noise injected into the class labels. These results are reported in Tables 5.6 and 5.7 for regular PSVMs and PSVMs with weighted sampling, respectively.

My method for injecting noise was the same as that of (Dietterich, 2000): I selected 10% of the instances uniformly and without replacement, then changed their class labels to an incorrect one chosen uniformly. Note that I assume the Twitter dataset is already noisy, as it has not been cleaned of noise as the benchmarks have; because I could not quantify the baseline noise level, I did not inject noise into this dataset.

Every algorithm's performance degrades in the presence of noise, as we would expect. (The spirals dataset is an anomaly; the dataset is so small and its class distribution so unusual that adding noise seems to make it easier to partition for most algorithms.) In general, the PSVM ensemble is fairly robust to the presence of noise. It is especially instructive to compare PSVMs with AdaBoost. Recall that later iterations of AdaBoost tend to focus on data that has been misclassified in previous iterations, and hence AdaBoost as a whole can be susceptible to noise. Therefore we would expect the performance of AdaBoost to drop more significantly than that of PSVMs. Furthermore, the

Dataset	PSVM without Sampling	C4.5	Boosted C4.5	Linear SVM	Boosted Linear SVM	Polynomial SVM	Multilayer Perceptron
isolated	71.9 ± 7.71	84.8 ± 4.36 •	83.6 ± 4.65 •	59.3 ± 3.67 ○	57.9 ± 3.71 ○	75.5 ± 4.91	73.9 ± 3.51
striated	76.8 ± 7.42	60.3 ± 7.24 ○	64.3 ± 6.55 ○	52.0 ± 2.86 ○	56.8 ± 7.36 ○	66.8 ± 3.12 ○	67.4 ± 18.5
spirals	29.0 ± 15.4	9.79 ± 6.28 ○	9.79 ± 6.28 ○	11.9 ± 10.2 ○	16.9 ± 17.1	10.3 ± 6.95 ○	17.1 ± 13.4
iris	88.7 ± 5.21	84.7 ± 7.33	83.3 ± 7.45	90.0 ± 6.83	90.0 ± 6.15	90.0 ± 5.37	91.3 ± 6.70
glass	50.0 ± 5.50	62.2 ± 7.46 •	66.3 ± 8.87 •	53.7 ± 8.91	57.5 ± 7.90 •	61.6 ± 8.95 •	61.7 ± 4.14 •
vehicle	68.1 ± 4.43	63.1 ± 1.89 ○	67.6 ± 2.54	71.5 ± 3.57	69.1 ± 4.24	57.4 ± 5.12 ○	71.0 ± 2.61
segment	84.3 ± 2.18	85.5 ± 2.30	85.4 ± 2.23	84.1 ± 1.82	84.0 ± 2.12	67.8 ± 3.95 ○	85.4 ± 2.51

Table 5.6: Results for standard PSVMs in noisy datasets. Shown are classification accuracy means with one standard deviation. ○ indicates statistically significant improvement, • statistically significant degradation.

Dataset	PSVM with Sampling	C4.5	Boosted C4.5	Linear SVM	Boosted Linear SVM	Polynomial SVM	Multilayer Perceptron
isolated	68.8 ± 5.81	84.8 ± 4.36 •	83.6 ± 4.65 •	59.3 ± 3.67 ○	57.9 ± 3.71 ○	75.5 ± 4.91 •	73.9 ± 3.51 •
striated	75.1 ± 10.1	60.3 ± 7.24 ○	64.3 ± 6.55 ○	52.0 ± 2.86 ○	56.8 ± 7.36 ○	66.8 ± 3.12 ○	67.4 ± 18.5
spirals	57.8 ± 28.6	9.79 ± 6.28 ○	9.79 ± 6.28 ○	11.9 ± 10.2 ○	16.9 ± 17.1 ○	10.3 ± 6.95 ○	17.1 ± 13.4
iris	76.7 ± 10.4	84.7 ± 7.33	83.3 ± 7.45	90.0 ± 6.83 •	90.0 ± 6.15 •	90.0 ± 5.37 •	91.3 ± 6.70 •
glass	43.4 ± 9.40	62.2 ± 7.46 •	66.3 ± 8.87 •	53.7 ± 8.91 •	57.5 ± 7.90 •	61.6 ± 8.95 •	61.7 ± 4.14 •
vehicle	59.1 ± 5.69	63.1 ± 1.89	67.6 ± 2.54 •	71.5 ± 3.57 •	69.1 ± 4.24 •	57.4 ± 5.12	71.0 ± 2.61 •
segment	80.6 ± 3.15	85.5 ± 2.30 •	85.4 ± 2.23 •	84.1 ± 1.82 •	84.0 ± 2.12 •	67.8 ± 3.95 ○	85.4 ± 2.51 •

Table 5.7: Results for PSVMs with weighted sampling in noisy datasets. Shown are classification accuracy means with one standard deviation. ○ indicates statistically significant improvement, • statistically significant degradation.

performance of PSVMs with weighted sampling should drop more significantly than that of non-sampled PSVMs, since re-weighting difficult instances is the aspect of AdaBoost that causes poor performance in the presence of noise.

This hypothesis seems to be borne out by the accuracy results. Both PSVM algorithms retain their advantage in the datasets with the trickiest class distributions, and regular PSVMs degrade gracefully on the benchmark datasets as well. PSVMs with sampling perform especially badly in the noisy benchmark data, as expected. More experimentation would be necessary to confirm these results; in particular, running AdaBoost with a larger number of iterations would probably demonstrate its vulnerability to noise more dramatically. It is also worth noting that for the most part, the sizes of the final PSVM ensembles are not affected by the presence of noise (see Table 5.5). This suggests that the algorithm is not retaining extra models to account for noisy instances.

5.4 Summary

In this chapter, I presented experimental results that show the PSVM algorithm performs with high accuracy in a number of datasets with class distributions of varying complexity. This is good evidence that the ensemble of PSVMs is in fact flexible enough to perform well in a variety of datasets without

model selection, with fewer SVMs and higher accuracy than the ensemble of ESVMs. Standard PSVMs are also relatively robust to noise in all datasets, unlike other flexible algorithms that tend to overfit, such as AdaBoost. PSVMs with weighted sampling perform well when class distributions are complicated and generate smaller ensembles, but perform poorly in simpler distributions and are much more sensitive to noise.

Chapter 6

Conclusion and Future Work

6.1 Summary of Contributions

This thesis presents the ensemble of prototype support vector machines (PSVMs), a novel algorithm for performing supervised classification in datasets with complex class distributions. This work is motivated primarily by the problem of model selection. When a data mining practitioner needs to choose a classifier-learning algorithm for a dataset, he most likely has no *a priori* knowledge of the class distributions that the algorithm will need to model. Because different algorithms learn different kinds of models, it is often necessary to try multiple algorithms to determine which performs best on the given data, a process that can be ad-hoc and time-consuming. The issues involved in finding good models are exacerbated when distributions are especially complicated, when many algorithms are liable to either over- or underfit.

In response to these challenges, the PSVM algorithm works by learning an ensemble of linear classifiers tuned to different sets of instances in the training data. Such an ensemble is flexible enough to have high performance in datasets with arbitrarily complex class boundaries with minimal parameter tuning. It accomplishes this by leveraging both the power of SVMs as effective linear classifiers and the power of ensembles to provide flexibility and improve accuracy without the need to specify a particular kernel function. This algorithm is based on the ensemble of exemplar SVMs for object recognition from (Malisiewicz et al., 2011), which learns an SVM for each instance in the training set. The core of the PSVM approach is an initial ensemble of exemplar SVMs, followed by a shifting algorithm that refines linear models, drops unnecessary models, and generalizes models from single exemplars to clusters of similar instances, or prototypes.

I also introduce a variant of the PSVM algorithm that repeatedly learns ensembles of PSVMs using increasingly large samples of the training set, each drawn according to a weight distribution that adapts to the performance of the previous ensembles. This reweighting and resampling process is borrowed from AdaBoost and can help the algorithm focus on the most difficult regions of the feature space, with the costs of not always being able to train on all available data and being potentially more susceptible to noise.

Finally, I report on the results of experiments comparing these two versions of PSVMs against a number of different classification learning algorithms. I ran these algorithms on synthetic, benchmark, and real-world datasets, and in both the absence and presence of noise. The results demon-

strate that PSVMs generally have the highest accuracy among all the algorithms I tested in the datasets with the more complex distributions, and good performance in the more standard datasets. This supports my claim that the PSVM algorithm provides a good balance of flexibility and high performance. In addition, the results for noisy datasets provide evidence that PSVMs are more robust to noise than other algorithms that seek to maximize flexibility. On the other hand, PSVMs with weighted sampling have high accuracy in the classes with the most difficult class distributions but perform poorly in the benchmark sets, and they also degrade more when noise is injected.

The main goal of the PSVM algorithm is to reduce the need to make data-dependent algorithmic decisions before knowing about data distributions. While model selection is no longer necessary with this algorithm, there are still parameters that must be set: the number of shifting iterations, the size of the initial negative sets, and the probability of mining hard negatives, as well as the sampling percentage for the version with weighted sampling. Optimal settings for these may be dataset-specific and affect the quality of the final ensemble. This remains a limitation of this approach.

6.2 Future Work

There are several interesting directions for future work related to the PSVM algorithm. Further experiments on more synthetic, benchmark, and real-world datasets would provide additional information on the capabilities of the algorithm. There is also opportunity for continued empirical study comparing this algorithm against other ensemble-of-SVM algorithms, such as exemplar SVMs (Malisiewicz et al., 2011) or profile SVMs (Cheng et al., 2010). A more thorough study of how the various algorithms degrade in the presence of noise is necessary to ensure the perceived differences are in fact statistically significant. Further experimentation could also confirm some of the conjectures I made in Chapter 5, including how final ensemble sizes vary for datasets with different degree of nonlinearity of decision boundaries.

It would also be interesting to explore a variety of modifications to the basic algorithm. For example, it is possible that combining the final PSVMs in a way besides majority voting could have a significant effect on the algorithm's overall performance. Another modification would be to leverage the ability of ensembles to select feature sets independently for each model. This can be useful since different regions of the example space may be well-characterized by different combinations of features. One elegant way of doing this would be to use 1-norm SVMs, which effectively perform feature selection in tandem with learning the SVM by forcing the weights of certain features in the linear model to be exactly zero; see (Tan et al., 2010) or (Zhu et al., 2004) for more details.

A more complicated change to the algorithm would be to generalize the various Euclidean distance computations to use different distance metrics or other possibly domain-specific measures of similarity. This would affect the initialization of negatives and the generalization of positive sets during shifting; in other words, it would affect what instances are considered part of a model's local region. Since a common aspect of complex datasets is that distance is not a reliable measure of similarity, this could allow the ensemble to more effectively classify this kind of complex data. However, in proposing such additions of domain-specific flexibility, it is important to remember that one of the goals of the algorithm is to avoid the need to make these types of domain-specific choices.

One important thread of research I did not investigate in this thesis is a theoretical proof that an ensemble of linear classifiers trained using the PSVM algorithm can provide accurate linear

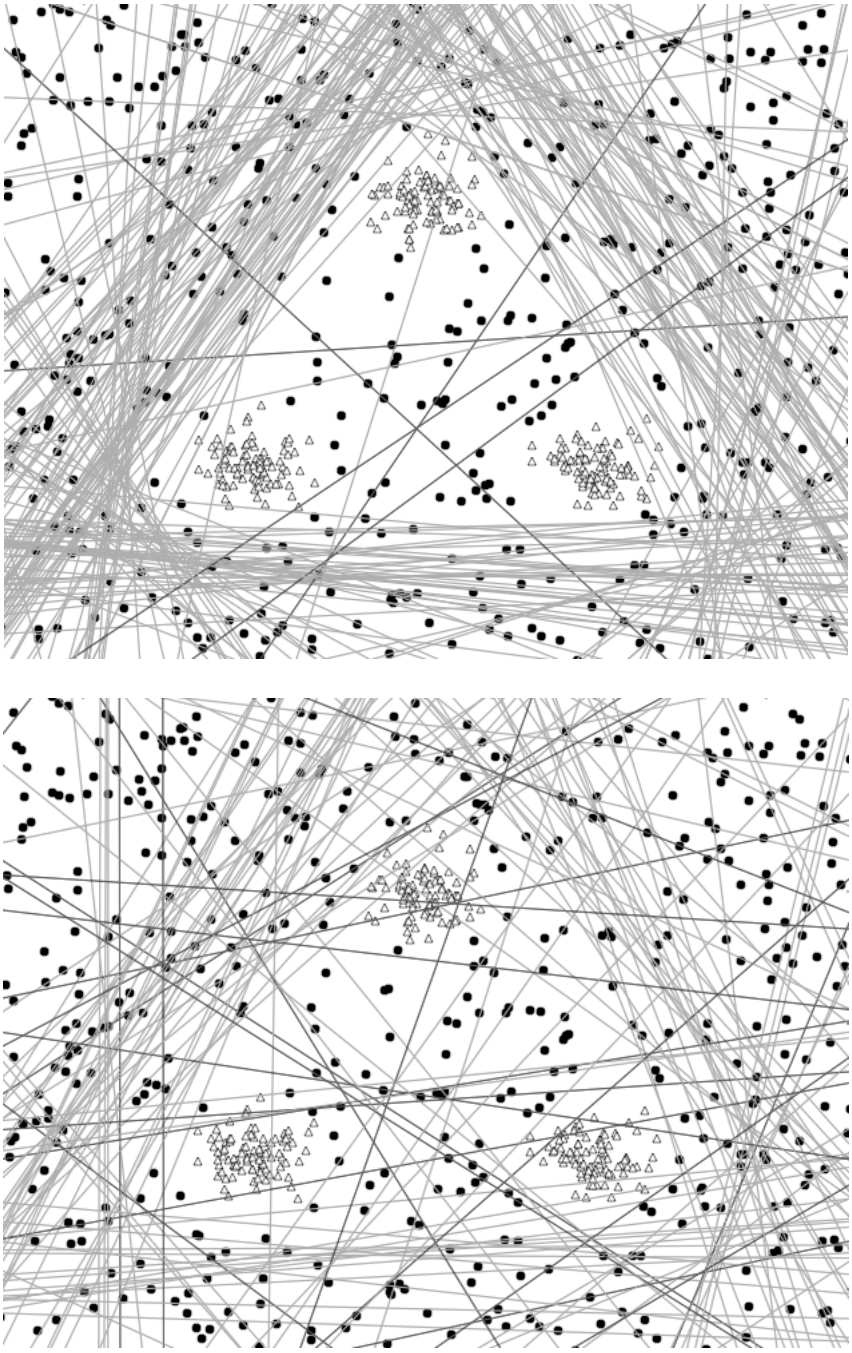


Figure 6.1: Examples of final ensembles in the isolated dataset for standard PSVMs (top) and PSVMs with weighted sampling (bottom). These are certainly not what we would imagine linear approximations of the decision boundaries to look like. However, a coarse approximation, or at least reasonable partitioning of the space, is occurring; for instance, the outline of the triangle enclosing all three clusters is visible, and particularly emphasized by the standard PSVM ensemble, which is less sensitive to structural detail. While the mathematical theory of linear approximation may not strictly apply, there is certainly some interesting reason for why these ensembles of linear classifiers are able to perform so well (with over 95% accuracy on the test set in both cases). Images created using matplotlib (Hunter, 2007).

approximations of certain types of decision surfaces or functions, given some minimum number of training instances and some number of shifting iterations. It is hard to envision what this theory would look like, as so much of the PSVM algorithm depends on the properties of a specific dataset; see Figure 6.1 for images of ensembles suggesting why developing this theory might be difficult. There is certainly empirical evidence that PSVMs can perform with high accuracy in datasets with a variety of different class distributions, and good reasons for this based on our understanding of how SVMs and ESVMs work. Nonetheless, a rigorous argument that shows this must be true or gives bounds on expected performance would be ideal.

Finally, it would be interesting to see how the PSVM algorithm compares with a similar classification algorithm that utilizes a preprocessing clustering step during training. One of the most potentially fruitful ideas this work has explored is the possible connection between supervised learning without explicit model selection and unsupervised learning or clustering. My hypothesis is that the shifting process of PSVMs enables the organic discovery of clusters without needing to specify the number of centroids as in k-means, and it would be worth exploring to what extent this hypothesis is supported empirically and theoretically.

Bibliography

Aggarwal, C., Wolf, J., Yu, P., Procopiuc, C., and Park, J. (1999). Fast algorithms for projected clustering. *ACM SIGMOD Record*, 28(2).

Agrawal, R., Gehrke, J., Gunopulos, D., and Raghavan, P. (1998). Automatic subspace clustering of high dimensional data for data mining applications. *ACM SIGMOD Record*, 27(2).

Alpaydin, E. (2010). *Introduction to Machine Learning (Adaptive Computation and Machine Learning)*. The MIT Press.

Bache, K. and Lichman, M. (2013). UCI machine learning repository. `http://archive.ics.uci.edu/ml`.

Barbosa, L. and Feng, J. (2010). Robust sentiment detection on twitter from biased and noisy data. In *Proceedings of the 23rd International Conference on Computational Linguistics: Posters*, COLING '10, Stroudsburg, PA, USA. Association for Computational Linguistics.

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

Callan, J. P. and Utgoff, P. E. (1991). A transformational approach to constructive induction. In *Proceedings of the 8th International Workshop on Machine Learning*, ML '91.

Chang, C.-C. and Lin, C.-J. (2011). LIBSVM: A library for support vector machines. *ACM Trans. Intell. Syst. Technol.*, 2(3).

Chang, Y., Hu, C., and Turk, M. (2003). Manifold of facial expression. In *Proceedings of the 2003 International Workshop on Analysis and Modeling of Faces and Gestures*, AMFG '03.

Chatfield, C. (1995). Model uncertainty, data mining and statistical inference. *Journal of the Royal Statistical Society. Series A (Statistics in Society)*, 158(3).

Cheng, H., Tan, P.-N., and Jin, R. (2010). Efficient algorithm for localized support vector machine. *IEEE Trans. Knowl. Data Eng.*, 22(4).

Cortes, C. and Vapnik, V. (1995). Support-vector networks. *Machine Learning*, 20(3).

Cui, Y., Fern, X. Z., and Dy, J. G. (2010). Learning multiple nonredundant clusterings. *ACM Trans. Knowl. Discov. Data*, 4(3).

Danyluk, A. P. and Provost, F. J. (1993). Small disjuncts in action: learning to diagnose errors in the local loop of the telephone network. In *Proceedings of the Tenth International Conference on Machine Learning*, pages 81–88.

Davidov, D., Tsur, O., and Rappoport, A. (2010). Enhanced sentiment learning using twitter hashtags and smileys. In *Proceedings of the 23rd International Conference on Computational Linguistics: Posters*, COLING '10, Stroudsburg, PA, USA. Association for Computational Linguistics.

Dietterich, T. (2000). An experimental comparison of three methods for constructing ensembles of decision trees: Bagging, boosting, and randomization. *Machine Learning*, 40(2).

Duan, K.-B. and Keerthi, S. S. (2005). Which is the best multiclass SVM method? an empirical study. In *Multiple Classifier Systems*, pages 278–285. Springer.

Edelman, S. and Shahbazi, R. (2012). Renewing the respect for similarity. *Frontiers in Computational Neuroscience*, 6(45).

Felzenszwalb, P. F., Girshick, R. B., McAllester, D., and Ramanan, D. (2010). Object detection with discriminatively trained part-based models. *IEEE Trans. Pattern Anal. Mach. Intell.*, 32(9).

Ferreira, A. and Figueiredo, M. (2011). Unsupervised feature selection for sparse data. In *Proceedings of the 2011 European Symposium on Artificial Neural Networks*, ESANN '11.

Franc, V. and Sonnenburg, S. (2008). Optimized cutting plane algorithm for support vector machines. In *Proceedings of the 25th International Conference on Machine Learning*, ICML '08, New York, NY, USA.

Guyon, I. and Elisseeff, A. (2003). An introduction to variable and feature selection. *The Journal of Machine Learning Research*, 3.

Hajishirzi, H., Rastegari, M., Farhadi, A., and Hodgins, J. K. (2012). Semantic understanding of professional soccer commentaries. In *Proceedings of the 2012 Conference on Uncertainty in Artificial Intelligence*, UAI '12.

Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., and Witten, I. H. (2009). The WEKA data mining software: an update. *SIGKDD Explor. Newsl.*, 11(1):10–18.

Ho, T. K. (1998). The random subspace method for constructing decision forests. *IEEE Trans. Pattern Anal. Mach. Intell.*, 20(8).

Hodge, V. and Austin, J. (2004). A survey of outlier detection methodologies. *Artificial Intelligence Review*, 22(2).

Holte, R., Acker, L., and Porter, B. (1989). Concept learning and the problem of small disjuncts. In *Proceedings of the 1989 International Joint Conference on Artificial Intelligence*, IJCAI '89.

Hsu, C.-W. and Lin, C.-J. (2002). A comparison of methods for multiclass support vector machines. *Trans. Neur. Netw.*, 13(2).

Hunter, J. D. (2007). Matplotlib: A 2d graphics environment. *Computing In Science & Engineering*, 9(3):90–95.

Japkowicz, N. (2001). Concept-learning in the presence of between-class and within-class imbalances. In *Proceedings of the 14th Biennial Conference of the Canadian Society on Computational Studies of Intelligence: Advances in Artificial Intelligence*, AI '01.

Joachims, T. (1998). Text categorization with support vector machines: Learning with many relevant features. In *Proceedings of the 10th European Conference on Machine Learning*, ECML '98.

Joachims, T. (2006). Training linear SVMs in linear time. In *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '06.

Kotsiantis, S. (2011). Combining bagging, boosting, rotation forest and random subspace methods. *Artif. Intell. Rev.*, 35(3).

Kruschke, J. K. (2006). *Concept Learning and Categorization: Models.* John Wiley & Sons, Ltd.

Kubat, M. and Matwin, S. (1997). Addressing the curse of imbalanced training sets: one-sided selection. In *Proceedings of the 1997 International Conference on Machine Learning*, ICML '97.

Malisiewicz, T. and Efros, A. A. (2008). Recognition by association via learning per-exemplar distances. In *Proceedings of the 2008 Conference on Computer Vision and Pattern Recognition*, CVPR '08.

Malisiewicz, T., Gupta, A., and Efros, A. A. (2011). Ensemble of exemplar-SVMs for object detection and beyond. In *Proceedings of the 2011 International Conference on Computer Vision*, ICCV '11.

Mouro-Miranda, J., Bokde, A. L., Born, C., Hampel, H., and Stetter, M. (2005). Classifying brain states and determining the discriminating activation patterns: Support vector machine on functional MRI data. *NeuroImage*, 28(4).

Parsons, L., Haque, E., and Liu, H. (2004). Subspace clustering for high dimensional data: a review. *SIGKDD Explor. Newsl.*, 6(1).

Quinlan, J. R. (1993). *C4. 5: programs for machine learning*, volume 1. Morgan Kaufmann.

Roweis, S. T. and Saul, L. K. (2000). Nonlinear dimensionality reduction by locally linear embedding. *Science*, 290(5500).

Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986). In Rumelhart, D. E., McClelland, J. L., and Group, P. R., editors, *Parallel distributed processing: explorations in the microstructure of cognition, vol. 1*, chapter Learning internal representations by error propagation, pages 318–362. MIT Press, Cambridge, MA, USA.

Schapire, R. E. (1999). A brief introduction to boosting. In *Proceedings of the 1999 International Joint Conference on Artificial Intelligence*, IJCAI '99.

Tan, M., Wang, L., and Tsang, I. W. (2010). Learning sparse SVM for feature selection on very high dimensional datasets. In *Proceedings of the 2010 International Conference on Machine Learning*, ICML '10.

Tenenbaum, J. B., Silva, V. D., and Langford, J. C. (2000). A global geometric framework for nonlinear dimensionality reduction. *Science*, 290(5500).

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

White, M., Sejnowski, T., Rosenberg, C., Qian, N., Gorman, R. P., Wieland, A., Deterding, D., Niranjan, M., and Robinson, T. (1995). Bench: CMU neural networks benchmark collection. `http://www.cs.cmu.edu/afs/cs/project/ai-repository/ai/areas/neural/bench/cmu/0.html`.

Wilson, T., Kozareva, Z., Nakov, P., Ritter, A., Rosenthal, S., and Stoyanov, V. (2013). Semeval-2013 task 2: Sentiment analysis in twitter. `http://www.cs.york.ac.uk/semeval-2013/task2/`.

Wilson, T., Wiebe, J., and Hoffmann, P. (2005). Recognizing contextual polarity in phrase-level sentiment analysis. In *Proceedings of the conference on Human Language Technology and Empirical Methods in Natural Language Processing*, HLT '05, pages 347–354.

Xiao, R., Zhao, Q., Zhang, D., and Shi, P. (2011). Facial expression recognition on multiple manifolds. *Pattern Recognition*, 44(1).

Zhu, J., Rosset, S., Hastie, T., and Tibshirani, R. (2004). 1-norm support vector machines. In *Advances in Neural Information Processing Systems 16*.