Prototype Support Vector Machines: Supervised Classification in Complex Datasets

April Tuesday Shen and Andrea Pohoreckyj Danyluk

Williams College

Abstract. Classifier learning generally requires 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 difficulty, especially in real-world data sets where the underlying model is both unknown and potentially complex, we 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. We present experimental results demonstrating the efficacy of PSVMs in both noiseless and noisy datasets.

Keywords: Ensemble methods, Classification, Support Vector Machines

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 arbitrarily 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. This often requires testing multiple models to find one that works well [1]. 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 this paper, we introduce the ensemble of prototype support vector machines (PSVMs)[1] 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 [3], which trains a separate linear separator specific to each instance in the training set. The PSVM algorithm trains an initial

[1] Cheng et al. [2] refer to their *profile* support vector machines as PSVMs. In this paper, PSVM should be taken to unambiguously refer to our *prototype* support vector machines.

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 rather than to specific exemplars. We present empirical evidence that PSVMs are capable of high classification accuracy in a variety of noiseless and noisy datasets with different class distributions.

The remainder of this paper is organized as follows. In Section 2 we motivate the problem and introduce related work. In Section 3 we describe exemplar SVMs in more detail than we have thus far. In Section 4 we describe our PSVM algorithm. Section 5 details experiments comparing PSVMs with other classifier learning algorithms on a selection of datasets. Finally, we summarize our conclusions and suggest future work.

2 Motivation and Related Work

In supervised machine learning, we can think of a learned 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 associated models are highly accurate fits to certain types of class distributions. For datasets that are linearly separable, linear SVMs [4] are a good choice. Such datasets include, for example, a number of text classification problems [5]. For more complicated class distributions, such as ones where multiple noncontiguous regions are mapped to the same class, C4.5 [6], a common decision tree learning algorithm, is a better choice.

However, regardless of how aptly a given model captures a *particular* class distribution, choosing an inappropriate model can still result in poor classification performance. Of course, every learning algorithm has some inductive bias that limits the set of possible models it explores. It is unreasonable to expect an algorithm to be 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 the data distribution looks 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.

Model selection is even more difficult in datasets with complex class distributions – for instance, ones that are highly nonlinear or contain many small disjuncts – since standard algorithms and models may not be sufficient in these situations. There are many ways to attack the problem of complex class distributions directly. One approach is to reduce or reformulate the feature space, since class boundaries may only seem complex when data is viewed in a particular space. Substantial research has been done in both feature selection (see [7] for feature selection in supervised learning and [8] for feature selection in unsupervised learning) and feature extraction (e.g., principle component analysis, multidimensional scaling, constructive induction [9], and more recently, manifold learning [10]). Unfortunately, all of these approaches still make strong assumptions about the fundamental underlying classifier models.

Other research has been concerned with the class distributions themselves [11]. Some of this work is focused on specific types of difficult distributions, such as highly unbalanced class distributions [12] or distributions that include many small disjuncts [13]. While extremely valuable, this focused work does not tackle the wider array of possible class distributions that present challenges to different models and classification algorithms.

Another approach to handling complex class distributions is to learn classifiers from different subsets of training examples. Such ensemble methods include, for example, AdaBoost [14], which builds a series of models that increasingly focus on examples in the difficult-to-capture regions of the feature space.

Of particular relevance to us are approaches that attempt to approximate complex decision surfaces with an ensemble of hyperplanes. These include exemplar SVMs [3], which we discuss in more detail in Section 3, and localized and profile SVMs [2].

Localized and profile SVMs combine the benefits of SVMs with instance-based methods in order to learn local models of the example space. Localized SVMs train a new SVM model for each test instance using that instance's nearest neighbors from the training set. As expected, this is very slow at test time. Profile SVMs also defer SVM learning to test time, but take advantage of the fact that multiple nearby test examples may require only a single SVM in that region. Profile SVMs use a variation of k-means to cluster training examples based on their relationship to the test examples, before learning a local SVM for each cluster. While profile SVMs are demonstrably more efficient than localized SVMs, they still defer training to test time, which may be unreasonable for many applications. Our approach differs from the localized SVM framework in that it is able to approximate a range of complex decision surfaces with ensembles of linear SVMs without the reliance on test examples of transductive or quasi-transductive [2] approaches.

3 Exemplar Support Vector Machines

In this section we discuss exemplar SVMs (ESVMs) in greater detail, as they are a foundation on which our algorithm is based. Exemplar SVMs were developed for object recognition [3]. 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. 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 tasks provide a good example of the types of complex data distributions we seek to handle. Consider, for instance, four images containing front and side views of bicycles and motorcycles, respectively. The two containing side views are conceivably closer in pixel-value feature spaces than are the front and side views of a motorcycle (or bicycle), yet it is the objects that we wish to identify – i.e., motorcycle or bicycle – not the orientation.

The ESVM framework is well-motivated for object recognition and other domains with complex class distributions for a number of reasons. Learning a number of separate classifiers permits each classifier to focus on a difference feature subset. This can be useful since different regions of the example space may be well-characterized by different combinations of features.

An approach based on ensembles and exemplars also 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.

The ESVM algorithm is an appealing starting point for our work as it leverages the power of both SVMs and ensembles to accommodate complexity. At the same time, the fact that it learns a separate SVM for each training instance makes it unnecessarily time- and space-intensive for many datasets. Our approach begins with ESVMs but then learns a set of SVMs that are tuned to general prototypes, rather than specific exemplars.

4 The Prototype SVM Algorithm

In this section we describe how we 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. 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 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, we do not want the single positive instance to be overwhelmed by negative instances. Furthermore, 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 chooses a sample of the potential negatives that is linearly separable from the exemplar.

Algorithm 1 TRAIN(T): Train an ensemble of PSVMs.

```
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 $V$.
$P \leftarrow [[d_i]$ for $d_i \in D]$ # List of positive sets, each initially just the exemplar.
$N \leftarrow [$CHOOSENEGATIVES$(D, d_i)$ 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 $E_j$.
    $a_j \leftarrow$ TEST$(V, E_j)$ # Accuracy of $E_j$ on $V$.
    $P, N \leftarrow$ SHIFT$(D, E_j, P, N)$
return ensemble $E_j$ with highest accuracy on $V$
```

To accommodate these issues, we choose negatives in the manner described in Algorithm 2. We first find 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, k, of them. The number can be user-selected, although empirically about seven negatives seems to work reasonably well. The k negatives chosen are those 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.

4.2 Shifting

After training the initial ensemble of ESVMs, we shift the boundaries according to Algorithm 3. (Note that the SVMs in Malisiewicz et al.'s original ESVM approach are shifted and generalized as well, though by a different process and not for the purpose of creating prototypes.) Shifting in our PSVM algorithm 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.

Algorithm 2 CHOOSENEGATIVES(D, d_i): Initialize set of negatives for a given data instance.

```
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 $D_i$
$\boldsymbol{n} \leftarrow (x - d_i)/||x - d_i||$ # Normal vector to the hyperplane passing through $d_i$.
for $d_j \in D_i$ do
    if $\boldsymbol{n} \cdot (d_j - d_i) > 0$ then
        Add $d_j$ to $N_i$
        if $|N_i| = k$ then
            return $N_i$
return $N_i$
```

Algorithm 3 SHIFT(D, E, P, N): Adjust and drop models from the ensemble.

```
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
            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 $P, N$
```

Generalization is at the core of what turns exemplar SVMs into prototype SVMs. The algorithm generalizes by adding new instances to the positive sets of models. For a given instance, we define a candidate model to be 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 new 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. 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 randomness and hence robustness to the algorithm.

The final step in the shifting algorithm is to remove models that do not classify any instances positively. Depending, in part, on the choice of regularization parameter, some SVMs may default to classifying all examples as negative. These are not useful for the overall ensemble and are therefore removed. Fortunately, the use of an ensemble provides redundancy; since the algorithm begins with 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 also provides some robustness to noise, as noisy exemplars may be more difficult to separate from their closest negatives.

We perform the shifting procedure 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. The ensemble with the highest classification accuracy on the validation set is retained. Empirically, we find that accuracy generally stabilizes fairly quickly – between 10 and 20 iterations at most.

4.3 Prediction

For validation after each shift and at test time, the ensemble of PSVMs predicts class labels for novel instances as follows. 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). We also generate the probability that the instance should be assigned the model's positive class, as in [15], and sum these probability values rather than the raw votes. The predicted class label assigned by the entire ensemble is simply the class with the maximum sum of weighted votes.

5 Experiments and Results

In this section we discuss our experiments. Overall, our PSVM algorithm performs better than the other algorithms we tested on datasets with the most complicated class distributions, and its performance is not significantly worse than the other algorithms when applied to simpler data distributions. We also demonstrate that our PSVM algorithm degrades gracefully in the presence of noise.

5.1 Experimental Setup

We tested our PSVM algorithm against C4.5 [6], AdaBoost [14] with C4.5 as the base classifier, linear SVMs [4], AdaBoost with linear SVMs as the base classifier, SVMs with a quadratic kernel, and multilayer perceptrons trained with backpropagation [16]. We chose these for their generally good performance and their varied strengths. For each algorithm, we used the implementations provided by Weka [17]. In particular, we used Weka's wrapper of LIBSVM [15] for the SVM experiments, as we also used LIBSVM for our PSVM algorithm.

We used Weka's default parameters for our experiments, except we reduced the training time of multilayer perceptrons from 500 to 200 iterations for reasons of time. Weka's default number of iterations for AdaBoost is quite low, namely 10, but we note that AdaBoosted linear SVMs finished in fewer than 10 iterations in the synthetic datasets and glass, and performance on the other datasets with as many as 100 iterations was statistically identical to performance with 10 iterations.

For PSVMs we 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).

There are three general categories of datasets we used in our experiments. These are listed in Table 1. First are synthetic datasets (see Figure 1), specifically designed to have unusual class distributions to provide a proof of concept of the power of PSVMs. The spirals dataset is a standard benchmark for neural networks originally from CMU [18]; we generated the other two. 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 [19]. The last dataset is a natural language classification task from the Semantic Evaluation (SemEval) workshop [20]. 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.

Table 1: The datasets used in the experiments. The top three are synthetic, with the spirals dataset taken from CMU's Neural Networks Benchmarks [18]. The next four are benchmark datasets from UCI's Machine Learning Repository [19]. The Twitter dataset is from SemEval [20].

	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)
UCI	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)

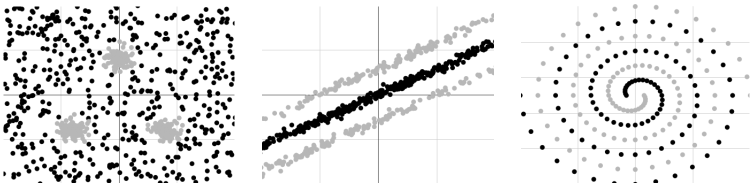

Fig. 1: Synthetic two-dimensional datasets: isolated, striated, and spirals.

The Twitter dataset contained only raw tweets and sentiment labels, and hence we preprocessed and featurized that dataset. Much research has gone into good feature representations for natural language texts and tweets in particular (see, for example, [21] or [22]), but as the focus of our work is not sentiment analysis, we used a basic but reasonable set of features for this data, including single words, links, usernames, hashtags, standard emoticons, and words from the MPQA Subjectivity Lexicon [23].

5.2 Results on Datasets With No Noise Added

For each algorithm, we report the results of ten-fold cross validation on each dataset. Table 2 compares our PSVMs with each of the other algorithms. Also see Table 3 for counts of wins, losses, and ties of PSVMs 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

Table 2: Experiment results. Shown are classification accuracy means with one standard deviation. ○ indicates statistically significant improvement of PSVMs over the other algorithm, ● indicates statistically significant degradation based on a corrected paired T-test at the 0.05 level.

Dataset	PSVM	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 3: The total number of wins, losses, and ties for PSVMs over other algorithms, in noiseless and noisy datasets.

	Noiseless	Noisy
Regular	16 - 13 - 19	14 - 7 - 21

most difficult class distributions. Its performance is especially impressive in the 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; we 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 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.

The spirals dataset is especially difficult for all algorithms, though we would expect good results from SVMs with a radial basis kernel. Of the algorithms tested, multilayer perceptrons are the only other algorithm besides PSVMs that perform relatively well on spirals. 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.

5.3 Results on Datasets With Noise Added

In order to test the robustness of PSVMs to noise, we repeated the experiments described above, on the same datasets but with noise injected into the class labels. These results are reported in Table 4.

Table 4: Results for PSVMs in noisy datasets. Shown are classification accuracy means with one standard deviation. ○ indicates statistically significant improvement, ● statistically significant degradation.

Dataset	PSVM	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)

For injecting noise we followed the same methodology as in [24]: we selected 10% of the instances uniformly and without replacement, then changed their class labels to an incorrect one chosen uniformly. Note that we assume the Twitter dataset is already noisy; because we could not quantify the baseline noise level, we 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. PSVMs retain their advantage in the datasets with the trickiest class distributions, and they degrade gracefully on the benchmark datasets as well. 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). This suggests that the algorithm is not retaining extra models to account for noisy instances. In addition, note that the number of models in the PSVM ensemble for every dataset is less than the number in the baseline ESVM ensemble. Because the final ensemble output by our PSVM algorithm is the one from the best iteration so far as determined by a validation set, it is clear that the unshifted ESVMs (i.e., the first iteration in the PSVM learning process) are never found to be best.

6 Summary and Future Work

In this paper we have described the ensemble of prototype support vector machines (PSVMs), an 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. The issues involved in finding good models are exacerbated when distributions are especially complicated.

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

Table 5: Average number of models in the final ensembles of PSVMs for noiseless and noisy datasets, rounded to the nearest whole number. Also listed is the number of models for the initial ESVM ensemble; these numbers are identical for noiseless and noisy datasets.

	ESVM	Noiseless	Noisy
isolated	720	323	122
striated	720	149	170
spirals	175	47	56
iris	135	54	24
glass	193	35	37
vehicle	761	110	113
segment	2079	453	323
twitter	540	42	X

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 [3]. 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.

Our results demonstrate that PSVMs generally have the highest accuracy among the algorithms we tested in the datasets with the more complex distributions, and good performance in standard benchmark datasets. In addition, the results for noisy datasets provide evidence that PSVMs are more robust to noise than other algorithms that seek to maximize flexibility.

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 size of the validation set, the number of shifting iterations, the size of the initial negative sets, and the probability of mining hard negatives. Optimal settings for these may be dataset-specific and affect the quality of the final ensemble. This remains a limitation of our approach.

There are several interesting directions for future work related to the PSVM algorithm. Further experiments on synthetic, benchmark, and real-world datasets would provide additional information on the capabilities of the algorithm. It would also be worthwhile to explore a variety of modifications to the basic algorithm. For example, we might 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 to effectively perform feature selection in tandem with learning the SVM [25]. Finally, it would be interesting to investigate the connection of the PSVM algorithm with similar algorithms that might utilize an explicit clustering step during training. We hypothesize that the shifting process of PSVMs enables the organic discovery of clusters without needing to specify the number of centroids as in *k*-means. It would be worth exploring to what extent this hypothesis is supported empirically and theoretically.

References

1. Chatfield, C.: Model uncertainty, data mining and statistical inference. Journal of the Royal Statistical Society. Series A (Statistics in Society) **158**(3) (1995)
2. Cheng, H., Tan, P.N., Jin, R.: Efficient algorithm for localized support vector machine. IEEE Trans. Knowl. Data Eng. **22**(4) (2010)
3. Malisiewicz, T., Gupta, A., Efros, A.A.: Ensemble of exemplar-SVMs for object detection and beyond. In: Proceedings of the 2011 International Conference on Computer Vision. ICCV '11 (2011)
4. Cortes, C., Vapnik, V.: Support-vector networks. Machine Learning **20**(3) (1995)
5. Joachims, T.: Text categorization with support vector machines: Learning with many relevant features. In: Proceedings of the 10th European Conference on Machine Learning. ECML '98 (1998)
6. Quinlan, J.R.: C4. 5: programs for machine learning. Morgan Kaufmann (1993)
7. Guyon, I., Elisseeff, A.: An introduction to variable and feature selection. The Journal of Machine Learning Research **3** (2003)
8. Ferreira, A., Figueiredo, M.: Unsupervised feature selection for sparse data. In: Proceedings of the 2011 European Symposium on Artificial Neural Networks. ESANN '11 (2011)
9. Callan, J.P., Utgoff, P.E.: A transformational approach to constructive induction. In: Proceedings of the 8th International Workshop on Machine Learning. ML '91 (1991)
10. Xiao, R., Zhao, Q., Zhang, D., Shi, P.: Facial expression recognition on multiple manifolds. Pattern Recognition **44**(1) (2011)
11. Japkowicz, N.: 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 (2001)
12. Kubat, M., Matwin, S.: Addressing the curse of imbalanced training sets: one-sided selection. In: Proceedings of the 1997 International Conference on Machine Learning. ICML '97 (1997)
13. Holte, R., Acker, L., Porter, B.: Concept learning and the problem of small disjuncts. In: Proceedings of the 1989 International Joint Conference on Artificial Intelligence. IJCAI '89 (1989)
14. Schapire, R.E.: A brief introduction to boosting. In: Proceedings of the 1999 International Joint Conference on Artificial Intelligence. IJCAI '99 (1999)
15. Chang, C.C., Lin, C.J.: LIBSVM: A library for support vector machines. ACM Trans. Intell. Syst. Technol. **2**(3) (May 2011)
16. Rumelhart, D.E., Hinton, G.E., Williams, R.J.: Learning internal representations by error propagation. In Rumelhart, D.E., McClelland, J.L., PDP Research Group, C., eds.: Parallel distributed processing: explorations in the microstructure of cognition, vol. 1. MIT Press, Cambridge, MA, USA (1986) 318–362

17. Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., Witten, I.H.: The WEKA data mining software: an update. SIGKDD Explor. Newsl. **11**(1) (November 2009) 10–18
18. White, M., Sejnowski, T., Rosenberg, C., Qian, N., Gorman, R.P., Wieland, A., Deterding, D., Niranjan, M., Robinson, T.: Bench: CMU neural networks benchmark collection. `http://www.cs.cmu.edu/afs/cs/project/ai-repository/ai/areas/neural/bench/cmu/0.html` (1995)
19. Bache, K., Lichman, M.: UCI machine learning repository. `http://archive.ics.uci.edu/ml` (2013)
20. Wilson, T., Kozareva, Z., Nakov, P., Ritter, A., Rosenthal, S., Stoyanov, V.: Semeval-2013 task 2: Sentiment analysis in twitter. `http://www.cs.york.ac.uk/semeval-2013/task2/` (2013)
21. Barbosa, L., Feng, J.: 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 (2010)
22. Davidov, D., Tsur, O., Rappoport, A.: 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 (2010)
23. Wilson, T., Wiebe, J., Hoffmann, P.: 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 (2005) 347–354
24. Dietterich, T.: An experimental comparison of three methods for constructing ensembles of decision trees: Bagging, boosting, and randomization. Machine Learning **40**(2) (2000)
25. Tan, M., Wang, L., Tsang, I.W.: Learning sparse SVM for feature selection on very high dimensional datasets. In: Proceedings of the 2010 International Conference on Machine Learning. ICML '10 (2010)