Improving SVM Accuracy by Training on Auxiliary Data Sources

Pengcheng Wu WUWU@CS.ORST.EDU
Thomas G. Dietterich TGD@CS.ORST.EDU
School of EECS, Oregon State University, Corvallis, OR 97331

Abstract

The standard model of supervised learning assumes that training and test data are drawn from the same underlying distribution. This paper explores an application in which a second, auxiliary, source of data is available drawn from a different distribution. This auxiliary data is more plentiful, but of significantly lower quality, than the training and test data. In the SVM framework, a training example has two roles: (a) as a data point to constrain the learning process and (b) as a candidate support vector that can form part of the definition of the classifier. The paper considers using the auxiliary data in either (or both) of these roles. This auxiliary data framework is applied to a problem of classifying images of leaves of maple and oak trees using a kernel derived from the shapes of the leaves. Experiments show that when the training data set is very small, training with auxiliary data can produce large improvements in accuracy, even when the auxiliary data is significantly different from the training (and test) data. The paper also introduces techniques for adjusting the kernel scores of the auxiliary data points to make them more comparable to the training data points.

1. Introduction

When training data are very scarce, supervised learning is difficult. Recently, many researchers have been exploring other sources of information that might allow successful learning from very small training samples. These efforts include learning by exploiting background knowledge (e.g., Clark & Matwin, 1993) and learning from a mixture of supervised and unsupervised data (e.g., Bennett & Demiriz, 1999; Blum & Mitchell, 1998). In this paper, we investigate another source of additional information: auxiliary supervised data drawn from a distribution different from the target distribution.

Auxiliary data are often available in machine learning application problems. For example, in medical applications, data may have been gathered in different countries or with somewhat different definitions of the class labels. In financial analysis, data may have been gathered in earlier years or with slightly different definitions of the attributes (e.g., the definitions of "productivity" and "consumer price index" change over time). A challenge for machine learning is to find ways of exploiting this data to improve performance on the target classification task.

The utility of auxiliary data can be understood through a bias/variance analysis. Because the real training data is scarce, a learned classifier will have high variance and therefore high error. Incorporating auxiliary data can reduce this variance, but possibly increase the bias, because the auxiliary data is drawn from a different distribution than the real data. This analysis also suggests that as the amount of real training data increases, the utility of auxiliary data should decrease.

This paper was inspired by an application in image classification for botany. Suppose you are hiking in the forest, and you encounter an interesting plant. You wonder what this plant is, so you clip off a leaf, take it home, and scan it using your scanner. Then you go to a web-based classification service, upload the image, and the server classifies the leaf and then provides information about the plant. We would like to provide such a service for a large range of plant species. The research described in this paper is part of this effort.

In this plant image classification task, the primary classification task is to determine the species of an isolated leaf, given an image of that leaf. To obtain

Appearing in *Proceedings of the 21st International Conference on Machine Learning*, Banff, Canada, 2004.

training (and test) data for this task, we collected individual leaves from 4 species of maple trees and 2 species of oak trees and scanned these leaves to obtain high-resolution color images. This is a time-consuming process, and it is expensive to obtain a large number of training examples for each species.

There is an alternative source of training data: plant specimen collections. At many universities, including ours, there is an Herbarium—a collection of dried plant specimens. Each specimen consists of an entire branch of a plant (stems, leaves, flowers, seed pods, and sometimes even roots) along with a label indicating genus, species, date and site of collection, and so forth. These specimens differ in many ways from isolated leaves. First, the specimens are old and dried, so they are discolored. Second, each specimen typically contains several leaves, and these leaves typically overlap and occlude each other. Third, the other plant parts (stems, flowers, seeds) are not useful for the primary isolated-leaf classification task. Nonetheless, the question arises of whether there is some way that we can exploit these plant specimens to help train a classifier for isolated leaves.

This paper explores a general solution to this problem within the framework of support vector machines. We consider two different ways in which auxiliary training data can be incorporated into (a form of) support vector machines, and we experimentally evaluate these methods. The paper begins with a description of the main approach. This is followed by presentation of our particular application problem. Then the experiments and their results are presented. A discussion of the results and conclusions completes the paper.

2. Exploiting auxiliary training data

Suppose we are given N^p training examples $(\mathbf{x}_i^p, y_i^p)$ for $i = 1, \ldots, N^p$ for our primary supervised learning problem, where $\mathbf{x}_i^p$ is a description of the ith training example, and y_i^p is the corresponding class label. The superscript p indicates the "primary" learning task. In addition, suppose we are given N^a auxiliary training examples $(\mathbf{x}_i^a, y_i^a)$ for $i = 1, \ldots, N^a$. We will assume that these training examples are somehow similar to the primary task, but they should be treated as weaker evidence for the design of a classifier.

Most learning algorithms can be viewed as seeking an hypothesis h that minimizes some loss function $L(h(\mathbf{x}), y)$ between the predicted class label $h(\mathbf{x})$ and the observed label y. Often, this can be formulated as finding the hypothesis h that minimizes the objective function

$$J(h) = \sum_i L(h(\mathbf{x}_i), y_i) + \lambda\, D(h),$$

where $D(h)$ is a complexity penalty to prevent overfitting, and λ is an adjustable parameter that controls the tradeoff between fitting the data (by minimizing the loss) and hypothesis complexity.

A natural approach to exploiting auxiliary training data would be to change the objective to have a separate term for fitting the auxiliary data

$$J'(h) = \sum_i^{N^p} L(h(\mathbf{x}_i^p), y_i^p) + \gamma \sum_i^{N^a} L(h(\mathbf{x}_i^a), y_i^a) + \lambda D(y).$$

The parameter γ (presumably less than 1) controls how hard we try to fit the auxiliary data. Cross-validation or hold-out methods could be applied to set γ and λ.

2.1. Auxiliary data with k-nearest neighbors

In many learning algorithms, the training data play two separate roles. Not only do they help define the objective function $J(h)$, but they also help define the hypothesis h. In the k-nearest neighbor algorithm (kNN), for example, $h(\mathbf{x})$ is defined in terms of the k training data points nearest to $\mathbf{x}$. The parameter k is chosen to minimize $J(k)$ where J is the leave-one-out cross-validation estimate of the loss. In this setting, we can now consider two different roles for the auxiliary data. First, when choosing k, we can include the auxiliary data in the objective function as $J'(k)$. Second, we can include the auxiliary data in the set of potential neighbors. In other words, the auxiliary data can be used both to evaluate a candidate classifier using J' and also to define the classifier.

To include the auxiliary data in the set of potential neighbors, we found it best to separately compute the K^p nearest primary neighbors and the K^a nearest auxiliary neighbors, and then take a weighted combination of the votes of these neighbors. Specifically, let $V^p(c)$ and $V^a(c)$ be the number of votes for class c from the primary and auxiliary nearest neighbors. Then the overall vote for class c is defined as

$$V(c) = \theta(V^p(c)/K^p) + (1-\theta)(V^a(c)/K^a).$$

The parameter θ controls the relative importance of the two types of neighbors. If $\theta = 1$, then only the primary nearest neighbors are voting, if $\theta = 0.5$ and $K^p = K^a$, then equal importance is given to primary and auxiliary neighbors, and if $\theta = 1$, then only the auxiliary neighbors determine the classification. The parameters θ, K^p, and K^a must be set by internal cross-validation to optimize the objective function (J or J').

2.2. Auxiliary data with support vector machines

Now let us consider support vector machines and related methods. A support vector classifier has the form

$$y = \begin{cases} 1 & \text{when } \sum_j \alpha_j y_j K(\mathbf{x}_j, \mathbf{x}) + b \geq 0 \\ -1 & \text{otherwise} \end{cases},$$

where the α_j and b are learned parameters and the function $K(\mathbf{x}_j, \mathbf{x})$ is a kernel function that in some sense measures the similarity between the test example $\mathbf{x}$ and the training example $\mathbf{x}_j$. Training examples for which $\alpha_j > 0$ are called *support vectors*.

The α and b values are learned by solving a convex optimization problem. In this paper, we will consider linear programming support vector machines (LP-SVMs) (Mangasarian, 2000) since they encourage sparser solutions than the usual SVM quadratic regularization penalty. This sparseness reduces the number of kernels evaluated at classification time (Graepel et al., 1999):

$$\begin{aligned} & Minimize: \sum_j \alpha_j + C \sum_i \xi_i \\ & s.t.: \quad y_i \left(\sum_j y_j \alpha_j K(\mathbf{x}_j, \mathbf{x}_i) + b \right) + \xi_i \geq 1 \;\; \forall i \\ & \qquad\quad \alpha_j \geq 0 \;\; \forall j. \end{aligned}$$

The objective function includes one term, $\sum_j \alpha_j$, that penalizes the complexity of the classifier and another term, $C \sum_i \xi_i$, that measures how poorly the classifier fits the training data. The slack variables ξ_i will be positive precisely for those training examples that the classifier does not classify correctly with a margin of at least 1.

As with kNN, there are two possible roles for an auxiliary training example. It can be considered as a potential support vector (indexed by j), and it can be included as a constraint to be satisfied in the optimization problem (indexed by i). This results in the following optimization problem:

$$\text{Minimize: } \sum_j^{N^p} \alpha_j^p + \sum_j^{N^a} \alpha_j^a + C^p \sum_i^{N^p} \xi_i^p + C^a \sum_i^{N^a} \xi_i^a$$

subject to:

$$y_i^p \left(\sum_j^{N^p} y_j^p \alpha_j^p K(\mathbf{x}_j^p, \mathbf{x}_i^p) + \sum_j^{N^a} y_j^a \alpha_j^a K(\mathbf{x}_j^a, \mathbf{x}_i^p) + b \right) + \xi_i^p \geq 1 \;\; i = 1, \ldots, N^p$$

$$y_i^a \left(\sum_j^{N^p} y_j^p \alpha_j^p K(\mathbf{x}_j^p, \mathbf{x}_i^a) + \sum_j^{N^a} y_j^a \alpha_j^a K(\mathbf{x}_j^a, \mathbf{x}_i^a) + b \right) + \xi_i^a \geq 1 \;\; i = 1, \ldots, N^a$$

$$\alpha_j^p \geq 0 \;\; j = 1, \ldots, N^p \qquad \alpha_j^a \geq 0 \;\; j = 1, \ldots, N^a$$

This can be simplified in two ways. First, we can remove the auxiliary training examples from the constraints by deleting the second set of constraints (involving ξ^a) and setting $C^a = 0$. This gives an LP-SVM in which the auxiliary examples are only used as support vectors. This increases the expressive power of the classifier, but it is still trained only to classify the primary examples correctly.

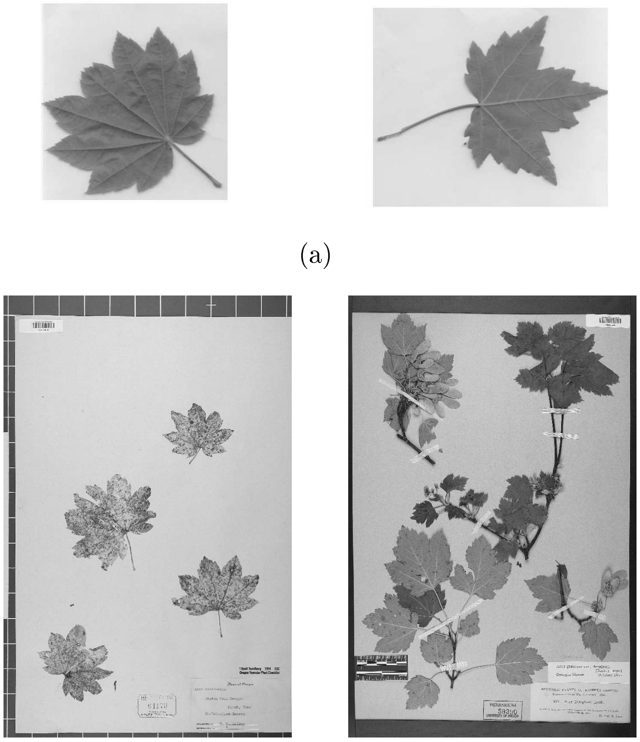
Alternatively, we can keep the constraints but delete the auxiliary examples from the set of candidate support vectors by deleting all terms involving $\sum_j^{N^a}$ in the constraints and the objective function. The resulting SVM will be defined using only primary training examples as support vectors, but it will have been trained to classify both primary and auxiliary examples well.

In the remainder of this paper, we will evaluate experimentally which of these three configurations (both, support-vectors only, and constraints-only) gives the best results on our isolated leaf classification problem.

3. Application: Leaf Classification

Figure 1 shows examples of isolated leaves and Herbarium specimens. Rather than extract feature vectors, we compare leaf shapes to one another directly as follows. First, each image is thresholded to obtain a binary image (1 for plant pixel and 0 otherwise). Then the boundary of each region is traversed, and the shape of the boundary is converted into a sequence of local curvatures. Let (x_j, y_j) be the coordinates of the jth point on the boundary of a region. Define angle θ_j as the angle between the line segments (x_{j-10}, y_{j-10})–(x_j, y_j) and (x_j, y_j)–(x_{j+10}, y_{j+10}). The sequence of angles forms a loop. To compare two leaves, we apply dynamic programming algorithms to align their angle sequences and compute a distance between them. Similar "edit distance" methods have been applied many times in pattern recognition and bioinformatics (Durbin et al., 1998; Milios & Petrakis, 2000; Petrakis et al., 2002).

We employ three different dynamic programming algorithms. The first algorithm is applied to compare two isolated leaves. Let $\{\theta_i : i = 1, \ldots, N\}$ be the angle sequence of the first leaf, and $\{\omega_j : j = 1, \ldots, M\}$ be the angle sequence of the second leaf. We will duplicate the angle sequence of the second leaf so that j



(a)

(b)

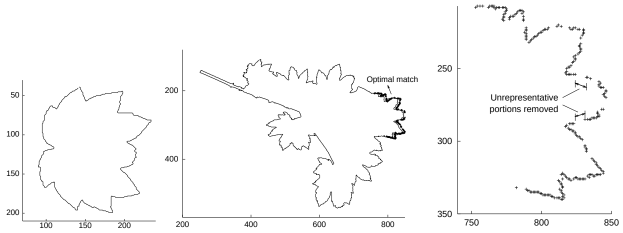
Figure 1. Plant leaf images: (a) Isolated leaves; (b) Herbarium leaves

goes from 1 to $2M$ (and $\omega_j = \omega_{j+M}$ for $j = 1, \ldots, M$).

Let F be an N by $2M$ matrix of costs oriented to lie in the first quadrant. We can visualize an alignment of the two leaves as a path that starts in some location $F_{1,k}$ and matches the first angle θ_1 of the first leaf to angle ω_k of the second leaf. This path then moves upward (increasing i) and to the right (increasing j) until it ends in some position $F_{N,k'}$, where it matches the last angle θ_N of the first leaf to angle $\omega_{k'}$ of the second leaf. Cell $F_{i,j}$ stores the total cost of the minimum-cost path from $F_{1,k}$ for any k to $F_{i,j}$. The F matrix can be filled in by traversing the matrix according to the rule

$$F_{i,j} := \min \begin{cases} F_{i-1,j-1} + d_{ij} \\ F_{i,j-1} + W_1 + d_{ij} \\ F_{i-1,j} + W_2 + d_{ij} \end{cases}$$

where $d_{ij} = (\theta_i - \omega_j)^2$ is the cost of matching the two angles, W_1 is the cost of a horizontal move that skips ω_j and W_2 is the cost of a vertical move that skips θ_i. In our experiments, $W_1 = W_2 = 150$. Note that the match is constrained to match all of the θ angles, but that the match may stop before it has matched all of the ω angles or it may wrap around and match some ω's twice. The final matching score is $(\min_j F_{N,j})/\sqrt{N^2 + (k' - k)^2}$.



(a) (b) (c)

Figure 2. Generating herbarium segments: (a) An isolated example; (b) An herbarium region showing the longest match to (a); (c) Close-up of the longest match.

This first dynamic programming algorithm works well for comparing isolated leaves, but it works very badly for comparing isolated leaves to herbarium samples or herbarium samples to each other. The problem is that a region of an herbarium sample can be very large and contain multiple, overlapping leaves. We decided, therefore, to use our isolated training examples as "templates" to identify parts of the herbarium samples that are most likely to correspond to a single leaf. Specifically, we take each isolated training example and match it to each segment of each herbarium sample of the same species. The purpose of this match is to find the longest contiguous partial match of the isolated leaf against some part of the herbarium sample. This partial match will be called an *herbarium segment*, and it will play the role of the auxiliary training data in our experiments. The process is illustrated in Figure 2.

The dynamic program for extracting herbarium segments works as follows. Let $\{\theta_i : i = 1, \ldots, N\}$ be the sequence of angles extracted from the isolated leaf and $\{\omega_j : j = 1, \ldots, M\}$ be the angle sequence extracted from one connected region of an herbarium sample of the same species. Let S be a $2N \times 2M$ matrix of costs. A match will consist of a path that starts at any arbitrary point (i_s, j_s) in the lower left $N \times M$ matrix and terminates at some arbitrary point (i_e, j_e) above and to the right. S is filled according to the rule

$$S_{i,j} := \max \begin{cases} S_{i-1,j-1} + \gamma - d_{ij} \\ S_{i,j-1} + \gamma - W_1 - d_{ij} \\ S_{i-1,j} + \gamma - W_2 - d_{ij} \\ 0 \end{cases}$$

As before, $d_{ij} = (\theta_i - \omega_j)^2$ is the cost of matching θ_i to ω_j, W_1 is the cost of a horizontal move (skipping

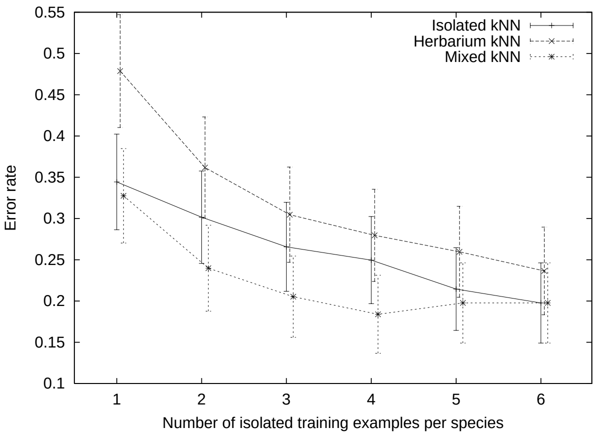


Figure 3. kNN error rates (error bars show 95% confidence interval)

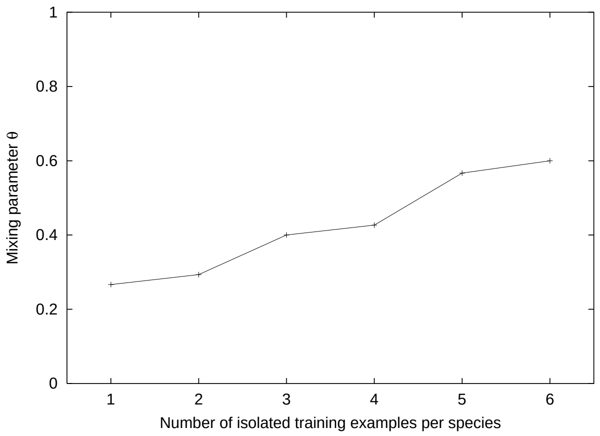


Figure 4. Chosen mixing rates θ for kNN

ω_j) and W_2 is the cost of a vertical move (skipping θ_i). The important thing in this formula is γ, which is the "reward" for extending the match one more angle. The match begins and ends at points where 0 is the largest of the four options in the max. It is easy to keep track of the longest match in the array and to extract the corresponding sequence of angles from the herbarium region, $(\omega_{j_s}, \ldots, \omega_{j_e})$, to form an herbarium segment. Empirically the value of γ is varied within the range 250 ± 64 in each matching process until a good match is found, that is, the ratio of the length of the matched angle sequences is not less than $1/\sqrt{2}$ and not greater than $\sqrt{2}$. Finally, the extracted segment is post-processed to remove angles skipped (by horizontal moves) during the match.

The third dynamic program matches herbarium segments to each other and to isolated training examples. It is identical to the first algorithm, except that we do not permit wrap-around of herbarium segments.

4. Experiments

We collected isolated leaves and photographed herbarium samples for six species—four maples (*Acer Circinatum, Acer Glabrum, Acer Macrophyllum,* and *Acer Negundo*) and two oaks (*Quercus Kelloggii* and *Quercus Garryana*). There are between 30 and 100 primary examples and herbarium specimens for each species.

Because we are interested in cases where primary data is especially scarce, we choose 6 isolated training examples at random from each class and retained the remaining examples as the (isolated) test set.

We generated learning curves by varying the size of the training set from 1 to 6 examples per species. For training sets of size $m \leq 6$, there are $\binom{6}{m}$ possible distinct training sets, so we report the error rate averaged over all of these.

In each run, the auxiliary data is obtained by matching the isolated examples in the training set against all regions of all herbarium samples from the same species. Because the parameter γ is sensitive to tuning and the length ratio constraint is strict, only 1 out of 5 matching processes produces a usable hebarium segment. Thus for each primary training set, we have an auxiliary data set roughly 10 times as large.

4.1. kNN Experiments

Figure 3 shows the learning curves for kNN. In all cases, the values of K^p (the number of primary nearest neighbors), K^a (the number of auxiliary nearest neighbors), and θ (the mixing coefficient) were set to optimize a lexicographical objective function consisting of four quantities. The most important quantity was the leave-one-out number of isolated examples misclassified. Ties were then broken by considering the leave-one-out number of herbarium segments misclassified. Remaining ties were broken to reduce the error margin (number of votes for the winning class $-$ number of votes for the correct class) on the isolated examples and finally to reduce the error margin on the herbarium samples. The figure shows that for small samples, mixing the herbarium examples with the isolated training examples gives better performance, but the differences are not statistically significant. If we classify isolated test examples using only the herbarium segments, the results are significantly worse for small training sets.

Figure 4 shows the values chosen for the mixing pa-

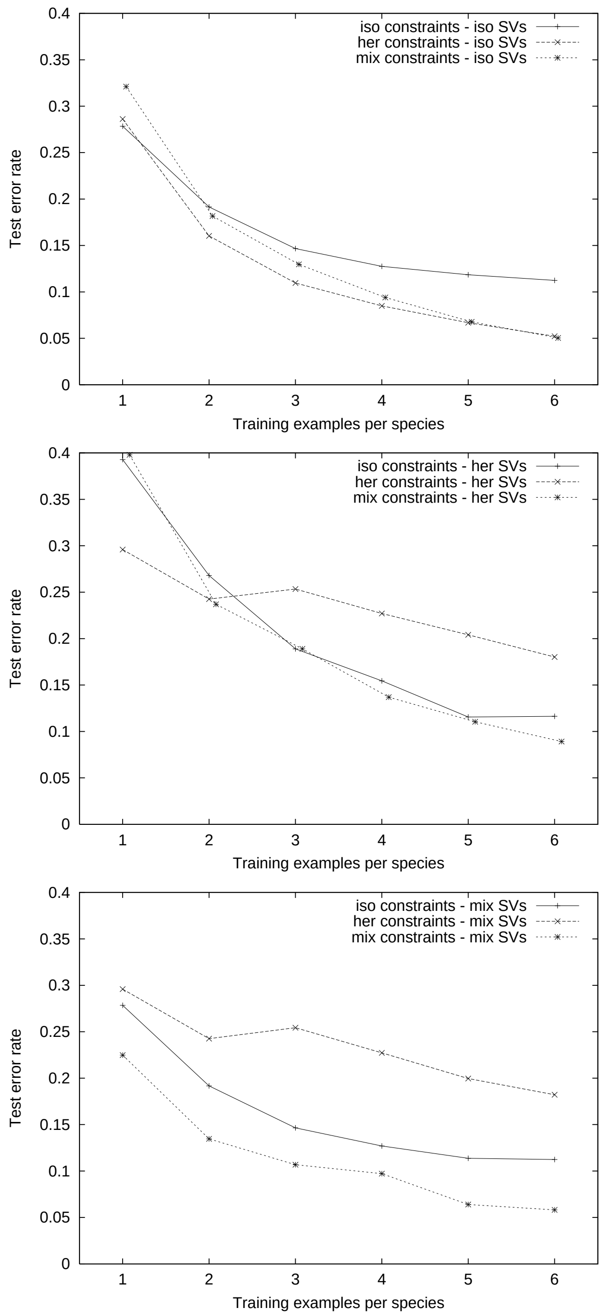


Figure 5. Learning curves for 9 configurations of LP-SVMs. From top to bottom: isolated support vectors, herbarium support vectors, and mixed support vectors.

rameter θ. We can see that for samples of size 1 (per species), approximately 75% of the weight is given to the auxiliary neighbors, whereas for samples of size 6, only 40% of the weight is given to the auxiliary neighbors. This accords with our intuition that as the sample gets larger, the variance (due to the small sample of isolated leaves) decreases and hence, the auxiliary neighbors become less useful.

4.2. LP-SVM Experiments

To apply SVMs, we must first convert the edit distance computed by the dynamic programming algorithms into a kernel similarity function. We employed the simple transformation $K(\mathbf{x}_i, \mathbf{x}_j) = 1/(\text{edit distance})$. However, it should be noted that this kernel is not a Mercer kernel. First of all, it is not symmetric, $K(\mathbf{x}_i, \mathbf{x}_j) \neq K(\mathbf{x}_j, \mathbf{x}_i)$, because the dynamic programming algorithm does not treat the two angle sequences identically (one is required to wrap around exactly, while the other is not). Second, we verified that some of the eigenvalues of the kernel matrix are negative, which would not be true for a Mercer kernel. The practical consequences of this are not clear, and other authors have found that empirical "kernels" of this sort work very well (Bahlmann et al., 2002). However, from a theoretical standpoint, unless a kernel is a Mercer kernel, there is no equivalent higher-dimensional space in which the learned decision boundary is a maximum-margin hyperplane (Cristianini & Shawe-Taylor, 2000).

There are nine possible configurations for our LP-SVMs. The constraints can include only isolated leaves, only herbarium segments, or both. The support vectors can include only isolated leaves, only herbarium segments, or both.

Figure 5 plots learning curves for these nine configurations. We note that, first, the overall best configuration is to combine mixed constraints and mixed support vectors. In short, the auxiliary data are useful both for representing the classifier and for training the classifier. Second, for samples of size 1, it is very important to have both mixed constraints and mixed support vectors. This is exactly what is predicted by a bias/variance analysis. Small samples have high variance, so it is better to mix in the auxiliary data to reduce the variance, even if this introduces some bias. Third, for samples of size 4, 5, and 6, it is very important to have mixed constraints, but it is OK to use just isolated training examples as support vectors. Hence, the auxiliary data is still important. One possible explanation is that 6 examples per species is still not enough data to eliminate the need for auxiliary training data. This is supported by the kNN experiments, where the best θ value was only 0.6 even with 6 examples per class.

To assess the statistical significance of the results, we applied McNemar's test to perform pairwise comparisons of various configurations. These comparisons confirm that the three trends mentioned above are statistically significant.

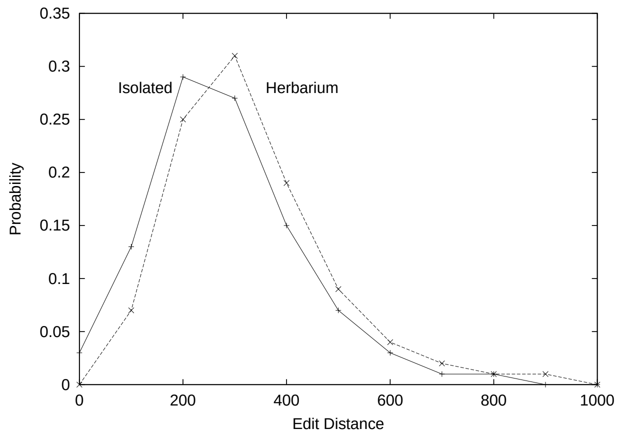


Figure 6. Comparison of edit distance scores for isolated leaves and herbarium segments. The herbarium histogram has been truncated. An additional 1.1% of the herbarium edit distances extend from 1000 to 3200.

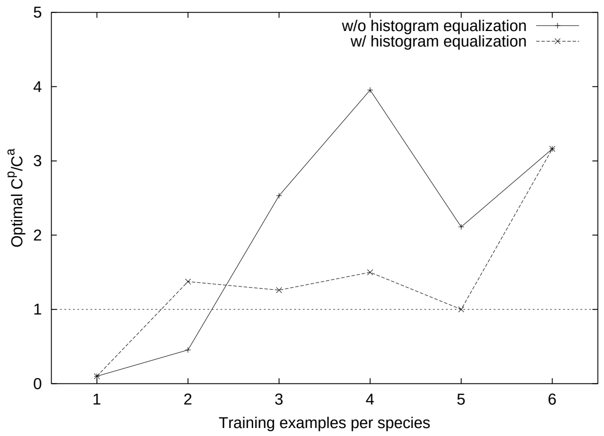


Figure 7. Trained values of C^p/C^a

4.3. Histogram Equalization

Figure 6 compares the distribution of the edit distances computed between all of the examples (isolated and herbarium) and (a) the isolated leaves or (b) the herbarium segments. The herbarium distances are larger and 1,998 segments (1.1%) have edit distances larger than 1000 (beyond the right edge of the figure). We suspected that if we could make these distributions more comparable, performance might improve.

We applied the following histogram equalization technique: Each distance computed with an herbarium segment was transformed by taking the logarithm and then scaling these to have the same range as the isolated edit distances. This eliminates the very large edit distance scores and shifts the distribution lower.

Histogram equalization has no effect on the kNN algorithm, since our kNN algorithm handles the primary and auxiliary data separately. For LP-SVMs, histogram equalization had no statistically significant effect on either the error rates or the relative merits of the 9 different configurations. The best configuration is still the mixed-constraints/mixed-SV configuration. We did find, however, that histogram equalization changed the number of support vectors found by the LP-SVM. At a sample size of 1, histogram equalization cuts the number of herbarium support vectors by more than half and doubles the number of isolated support vectors. At a sample size of 6, the number of isolated support vectors is unchanged, but the number of herbarium support vectors is reduced by roughly an order of magnitude. An explanation for this is that with the "outlier" herbarium segments reduced by histogram equalization, fewer herbarium support vectors were needed to fit them. However, since test set performance is measured strictly on isolated leaves, this reduction in herbarium support vectors has relatively little impact on the error rate.

Another effect of histogram equalization was to change the relative sizes of C^p and C^a, the complexity control parameters of the LP-SVM. Figure 7 plots the ratio C^p/C^a. Without histogram equalization, we can see that C^p was much larger than C^a for sample sizes greater than 2, so much more weight was being placed on fitting the primary training examples than on fitting the auxiliary ones. With histogram equalization, the ratio stays closer to 1, which indicates that roughly equal weight was being placed on primary and auxiliary training examples.

5. Conclusions

This paper has described a methodology for exploiting sources of auxiliary training data within the kNN and LP-SVM learning algorithms. We have shown that auxiliary data, drawn from a different distribution than the primary training and test data, can significantly improve accuracy. For the LP-SVM, Figure 8 shows that when training on only 1 example per class, auxiliary data reduces the error rate from 27.8% to 22.5%, a reduction of nearly 20%. When training on 6 examples per class, the error rate decreases from 11.2% to 5.8%, a reduction of 48%.

This paper has also shown how SVMs can be trained to classify objects based on shape, by using boundary curvature edit distances as a kind of kernel function. By using separate C^p and C^a parameters in the SVM, we can adjust the relative importance of fitting the two data sources. Edit distances have been used with the kNN classifier for many years. Our results suggest that

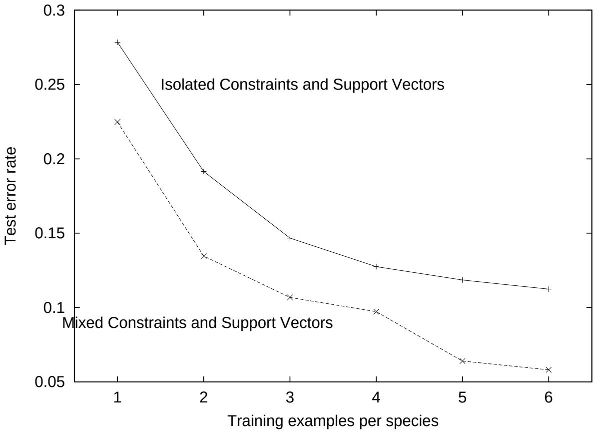


Figure 8. Comparison of training on primary data only (Isolated Constraints and Support Vectors) with training on both primary and auxiliary data (Mixed Constraints and Support Vectors).

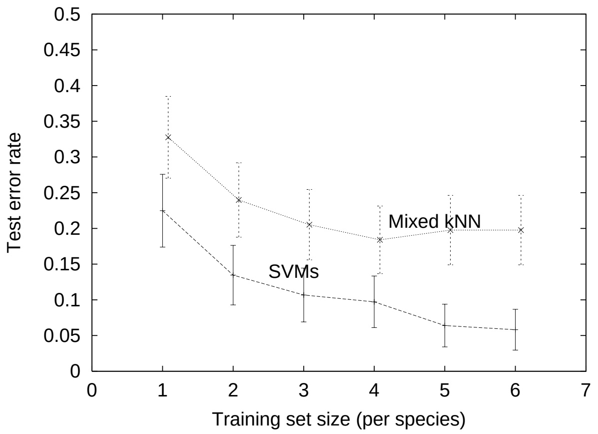


Figure 9. Performance of mixed kNN vs mixed SVMs.

SVMs may be able to give significant improvements in performance over such kNN classifiers. Figure 9 shows that SVMs reduce the error rates by 34.6% (training size 1) and 70.7% (training size 6).

Clearly, the more we can make the auxiliary data resemble the primary data, the more useful it will be. In our application problem, we showed how to apply the primary training examples as templates to extract similar shape segments from the auxiliary data. In addition, we found that equalizing the distance distributions of the two data sources reduced the number of support vectors.

It is easy to imagine ways of extending other learning algorithms to exploit auxiliary data sources. For example, decision tree algorithms could use auxiliary data for attribute selection, split threshold selection, and tree pruning. Neural network algorithms could train on auxiliary data, but with reduced penalties for misclassification. There are several interesting directions to pursue for exploiting auxiliary data in Bayesian network classifiers.

References

Bahlmann, C., Haasdonk, B., & Burkhardt, H. (2002). On-line handwriting recognition with support vector machines—a kernel approach. *8th International Workshop on Frontiers in Handwriting Recognition* (pp. 49–54). Los Alamitos, CA: IEEE.

Bennett, K., & Demiriz, A. (1999). Semi-supervised support vector machines. *Advances in Neural Information Processing Systems 11* (pp. 368–374). MIT Press.

Blum, A., & Mitchell, T. (1998). Combining labeled and unlabeled data with co-training. *Proc. 11th Annu. Conf. on Comput. Learning Theory* (pp. 92–100). ACM Press, New York, NY.

Clark, P., & Matwin, S. (1993). Using qualitative models to guide inductive learning. *Machine Learning: Proceedings of the Tenth International Conference* (pp. 49–56). San Francisco, CA: Morgan Kaufmann.

Cristianini, N., & Shawe-Taylor, J. (2000). *An introduction to support vector machines (and other kernel-based learning methods).* Cambridge University Press.

Durbin, R., Eddy, S., Krogh, A., & Mitchison, G. (1998). *Biological sequence analysis: Probabilistic models of proteins and nucleic acids.* Cambridge: Cambridge University Press.

Graepel, T., Herbrich, R., Scholkopf, B., Smola, A., Bartlett, P., Robert-Muller, K., Obermayer, K., & Williamson, B. (1999). Classification on proximity data with LP–machines. *Proceedings of the Ninth International Conference on Artificial Neural Networks* (pp. 304–309).

Mangasarian, O. (2000). Generalized support vector machines. In A. J. Smola, P. L. Bartlett, B. Schlkopf and D. Schuurmans (Eds.), *Advances in large margin classifiers*, 135–146. Cambridge, MA.: MIT Press.

Milios, E., & Petrakis, E. (2000). Shape retrieval based on dynamic programming. *IEEE Transactions on Image Processing*, *8*, 141–146.

Petrakis, E., Diplaros, A., & Milios, E. (2002). Matching and retrieval of distorted and occluded shapes using dynamic programming. *IEEE PAMI*, *24*, 1501–1516.