Permutation Invariant SVMs

Pannagadatta K. Shivaswamy PANNAGA@CS.COLUMBIA.EDU
Department of Computer Science, Columbia University, New York, NY 10027

Tony Jebara JEBARA@CS.COLUMBIA.EDU
Department of Computer Science, Columbia University, New York, NY 10027

Abstract

We extend Support Vector Machines to input spaces that are sets by ensuring that the classifier is invariant to permutations of subelements within each input. Such permutations include reordering of scalars in an input vector, re-orderings of tuples in an input matrix or re-orderings of general objects (in Hilbert spaces) within a set as well. This approach induces permutational invariance in the classifier which can then be directly applied to unusual set-based representations of data. The permutation invariant Support Vector Machine alternates the Hungarian method for maximum weight matching within the maximum margin learning procedure. We effectively estimate and apply permutations to the input data points to maximize classification margin while minimizing data radius. This procedure has a strong theoretical justification via well established error probability bounds. Experiments are shown on character recognition, 3D object recognition and various UCI datasets.

1. Introduction

Support Vector Machines (SVMs) (Vapnik, 1995) are a well established and highly successful tool for classification problems. Given patterns belonging to two classes, SVMs find a generalized large margin separation boundary. Traditional SVMs, however, focus on vectorial input data and are typically applied to datasets in Euclidean spaces, i.e. $\mathbb{R}^n$. However, many interesting applied problems have input spaces that are not simply static vectors in $\mathbb{R}^n$ but have additional structure. Thus, SVMs are often augmented with interesting kernels to handle exotic inputs such as strings (Lodhi et al., 2002), graphs (Mah et al., 2004), and sets (Kondor & Jebara, 2003). These kernel methods make the SVM robust to certain types of nuisance variations the input spaces can undergo and hence help improve accuracy. These nuisance variations include, for instance, strings undergoing insertions or input sets undergoing arbitrary re-orderings of their elements. An alternative approach is to deal with structured input spaces *within* the learning algorithm itself, instead of deferring the responsibility solely to the kernel. For instance, certain conditional independence assumptions on both input and output space can be handled via an extension of the SVM called Maximum Margin Markov Networks (Taskar et al., 2004). Since the learning algorithm is an active element in enforcing the desired invariance or independence assumptions, the resulting algorithm may achieve better accuracy than simply finding a kernel which is constant and invariant to the source of nuisance variations. In this article, we embed an interesting form of invariance, namely permutational invariance, directly into the SVM learning algorithm. For instance, an adversary may have permuted the vectorial inputs in a dataset which prevents us from directly learning an SVM classifier. We thus propose an SVM that jointly uncovers and compensates for these adversarial permutations while learning its decision boundary via the SVM optimization algorithms. Invariance is achieved by optimizing away the source of variation instead of having a kernel that is simply insensitive to it. The motivation is that the robustness or invariance gained by a kernel often comes at the expense of additional loss of information about the input. For instance, an invariant kernel between two sets (Kondor & Jebara, 2003) might consider the spectrum of a matrix of pairwise inner-products between elements in each set. The spectrum is invariant to permutation but it is *also* invariant to rotation. Since rotation is a superset of permutation, such a kernel is overcompensating to achieve invariance.

More specifically, we develop an SVM that can handle datasets where each input can undergo an arbitrary permutation. For instance, if each input to the SVM

Appearing in *Proceedings of the 23rd International Conference on Machine Learning*, Pittsburgh, PA, 2006.

is a vector, the order of the scalar entries in the vector can be shuffled arbitrarily. Alternatively, if each input to the SVM is a matrix, the rows (tuples) of the matrix may be shuffled arbitrarily. The SVM takes a linear combination of the scalars in the input (vector or matrix) to compute a classification, yet, if permutations have been applied to the inputs and are not compensated for, a classical SVM will exhibit poor classification accuracy. While we will focus on matrix permutation, our framework applies more generally to any set of elementary objects (not just a set of scalars or tuples) as long as we can define a kernel between these elementary objects. We will assume that there is no way of knowing a priori the ordering of elements or features in the permutable input set. We may not know which ordering of the features is the correct one - every ordering might be as correct as any other. Therefore, SVM learning algorithms should uncover permutations only to improve classification accuracy and not an alternative criterion.

While permutation may seem like a fairly exotic type of invariance, it is a very useful one in many real datasets. Consider an image based classifier which discriminates between different classes of hand written digits. Each digit can be represented by a set of (x, y) coordinates corresponding to the dark pixels in the image (each pixel is a tuple (x, y)). In this cloud-of-points representation, there is no single correct a priori ordering among the points that compose the digit shape. This representation of a digit is akin to the bag of pixels or bag of tuples representation (Jebara, 2004). One way to represent the point cloud is to have each pixel be a tuple or a row in a matrix and allow the rows of the resultant matrix to be permuted arbitrarily. Clearly, these permutations don't actually change the digit being represented. Similarly, a gray scale image could be represented by a collection of tuples (x, y, I) where x and y denote the location of the pixel and I denotes the intensity of the pixel at (x, y). An SVM needs to compensate for this arbitrary permutation to achieve accurate classification performance. Furthermore, the resulting SVM should be invariant and learn the same classifier despite any such arbitrary permutations.

There have been prior attempts to handle permuted data. For instance, Kirshner et al. (2003) make use of EM algorithm for unsupervised learning and treat each input's permutation as a hidden variable but require exhaustively enumerating all $n!$ permutations. A more efficient approach is to find a single setting of the permutations that maximize likelihood by optimizing a generative unsupervised model (Jebara, 2004). Permutable images and other objects have been handled via a permutational invariant kernel (Kondor & Jebara, 2003) which also remains efficient. Earlier work on removing the invariant aspects of the input spaces has been demonstrated to improve a learning (Simard et al., 1996) yet has not directly handled permutation types of variation. What is lacking in literature, to the best of our knowledge, are large margin discriminative algorithms that factor out nuisance permutations to maximize classification accuracy. In this paper we propose a way to classify data which is permutationally invariant or corrupted by permutation. Our approach is based on well known bounds that relate error probability to the ratio of the radius of the data and the margin of separation. This learning criterion is achieved by interleaving the classical Kuhn-Munkres algorithm (also known as the Hungarian method) into the large margin discriminative SVM learning process. We make use of the Hungarian method to permute the data to achieve large margins and to reduce radius, as the error probability bounds suggest. The proposed method is well suited to bags of objects, data corrupted by permutations, and sparse image datasets where objects undergo 3D translation and rotation.

2. Support Vector Machines

In this paper, we focus on the case where input patterns $\mathbf{x}_i$ are are matrices of size $\mathbb{R}^{m\times d}$ whose m rows are d-dimensional tuples. The rows of each matrix can undergo an arbitrary permutation.

Denote by $\pi : \mathbb{R}^{m\times d} \times \mathbb{R}^{m\times d} \to \mathbb{R}$ the function $\pi(\mathbf{A}, \mathbf{B}) = \sum_{i=1}^{d} \langle \mathbf{A}_i, \mathbf{B}_i \rangle = \sum_{i=1}^{d} \sum_{j=1}^{m} \mathbf{A}_{ij}\mathbf{B}_{ij}$. Here, we have matrices $\mathbf{A}, \mathbf{B} \in \mathbb{R}^{m\times d}$ and the subscript i denotes the i^{th} column of the corresponding matrix while the subscript ij denotes the scalar in the i^{th} row and the j^{th} column of the matrix. When $d = 1$, the function π defines just the dot product.

Assume that we have n observations $(\mathbf{x}_i, y_i)$ drawn i.i.d. (independently and identically distributed) from a distribution over $\mathbb{R}^{m\times d} \times \{\pm1\}$ where $\mathbf{x}_i \in \mathbb{R}^{m\times d}$ is the i^{th} pattern and $y_i \in \{\pm1\}$ is the corresponding label. A linear SVM gives a decision rule $f(\mathbf{x}) = sign(\pi(\mathbf{w}, \mathbf{x}) + b)$ with $\mathbf{w} \in \mathbb{R}^{m\times d}$ and $b \in \mathbb{R}$. Given a matrix $\mathbf{A} \in \mathbb{R}^{m\times d}$, we compute its norm as follows $\|\mathbf{A}\| = \sqrt{\sum_{i=1}^{d} \|\mathbf{A}_i\|^2}$. Here the norm within the square root, $\|.\|$, is just l_2 norm and $\mathbf{A}_i$ denotes the i^{th} column of $\mathbf{A}$. The decision rule parameters $(\mathbf{w}, b)$ are learned by solving the standard SVM equation:

$$\min_{\mathbf{w},b} \frac{1}{2}\|\mathbf{w}\|^2 \quad \text{s.t. } y_i(\pi(\mathbf{w}, \mathbf{x}_i) + b) \geq 1 \;\; \forall 1 \leq i \leq n. \quad (1)$$

The distance of the optimal hyperplane obtained by solving the formulations above to the closest training sample $\mathbf{x}_i$ is called the maximum margin and is denoted by M. Often, it is not possible to strictly separate the two classes in (1). In such cases a slack variable which serves as a penalty is introduced in each of the constraints in (1). Doing so gives the following relaxed version of the SVM (Bennett & Mangasarian, 1993; Cortes & Vapnik, 1995) in which the quantity $C\xi_i$ is the penalty for the patterns that are either within the margin or misclassified:

$$\min_{\mathbf{w},b,\xi} \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{n}\xi_i \tag{2a}$$

$$\text{s.t.} y_i(\pi(\mathbf{w},\mathbf{x}_i)+b) \geq 1-\xi_i, \xi_i \geq 0 \ \ \forall 1 \leq i \leq n. \tag{2b}$$

In addition, the radius R of a dataset is relevant to support vector machines and can be found by considering the smallest hypersphere which encloses all data points. The radius and centroid of the hypersphere are estimated using the following quadratically constrained quadratic program formulation (Weston et al., 2000; Ben-Hur et al., 2001):

$$\min_{\mathbf{c},R,\xi} R^2 + C\sum_{i=1}^{n}\xi_i \tag{3a}$$

$$\text{s. t. } \|\mathbf{c}-\mathbf{x}_i\|^2 \leq R^2+\xi_i, \quad \xi_i \geq 0 \quad \forall 1 \leq i \leq n \tag{3b}$$

where $\mathbf{c} \in \mathbb{R}^{m\times d}$ is the centroid of the hypersphere. The relaxed version (with finite C) makes the hypersphere robust to the outliers in the data.

We state an important result (Vapnik, 1995; Weston et al., 2000) relating the expected probability of error, the maximum margin M and the radius R of the hypersphere enclosing all the training samples in $\mathcal{H}$ space. This theorem is the crux of this paper in that we will use it to motivate a criterion for eliminating nuisance variations (permutations) to minimize the error probability.

Theorem 1 *If images of training data of size n belonging to a hypersphere of size R are separable with margin M, then the expectation of the error probability has the following bound*

$$EP_{err} \leq \frac{1}{n}E\left\{\frac{R^2}{M^2}\right\}, \tag{4}$$

where the expectation is over sets of training data of size n.

Images correspond to the training samples themselves when we consider the input space. If we can make sure that the radius of the hypersphere enclosing the data is small with a large margin of separation, we can expect to have smaller probability of error as suggested by (4).

In this paper, since we are considering permutational invariance, our aim in the following sections will be to reduce the radius R and to increase the margin M by permuting the patterns appropriately.

3. Linear Assignment Problem

At the core of our technique lies the well studied Linear Assignment Problem (LAP) (Papadimitriou & Steiglitz, 1982). For example, consider a situation with N factories and N warehouses, and a matrix of transportation costs from each factory to each warehouse. The LAP problem is to find an assignment of each warehouse to a factory such that the total cost is minimized. Although LAP is traditionally cast as a minimization problem, the maximization problem is equally straightforward. LAP is solvable using the classical Kuhn-Munkres algorithm (also called the Hungarian method) in $O(N^3)$ time. Recent advances (Goldberg & Kennedy, 1995) have made the Kuhn-Munkres algorithm efficient and scalable. Such algorithms produce a permutation or a permutation matrix $\mathbf{A}$ which indicates how the N elements have to be permuted to get the optimal assignment.

Let $\mathbf{P},\mathbf{Q} \in \mathbb{R}^{m\times d}$ and $\mathbf{A} \in \{0,1\}^{m\times m}$ be any permutation matrix such that $\sum_i A_{ij} = 1$ and $\sum_j A_{ij} = 1$. Consider the problem of permuting the $\mathbf{Q}$ matrix to bring it as close as possible to the $\mathbf{P}$ matrix. Formally, we seek a permutation matrix $\mathbf{A}$ which minimizes the quantity $\|\mathbf{P}-\mathbf{AQ}\|^2$. Consider minimizing the following over permutation matrices $\mathbf{A}$:

$$\|\mathbf{P}-\mathbf{AQ}\|^2 = \pi(\mathbf{P}-\mathbf{AQ},\mathbf{P}-\mathbf{AQ}) \tag{5a}$$

$$= \pi(\mathbf{P},\mathbf{P}) + \pi(\mathbf{AQ},\mathbf{AQ}) - 2\pi(\mathbf{P},\mathbf{AQ}) \tag{5b}$$

$$= \pi(\mathbf{P},\mathbf{P}) + \pi(\mathbf{Q},\mathbf{Q}) - 2\pi(\mathbf{P},\mathbf{AQ}). \tag{5c}$$

Above, we used the property that the matrix norm is unchanged under permutation of a matrix's rows.

One consequence of the above (5) is that by maximizing $\pi(\mathbf{P},\mathbf{AQ})$ over $\mathbf{A}$, we find a permutation matrix which reorders the rows of $\mathbf{Q}$ to give the minimum distance matrix from $\mathbf{P}$. The Kuhn-Munkres algorithm effectively finds a matrix $\mathbf{A}$ which is a permutation such that the trace $tr(\mathbf{A}^\top\mathbf{Z})$ is maximized, where $\mathbf{Z}$ is the cost matrix given to the algorithm. The ij^{th} entry of $\mathbf{Z}$ consists of the dot product of the i^{th} row of $\mathbf{P}$ and the j^{th} row of $\mathbf{Q}$. For a given $\mathbf{Z}$ matrix, the solution permutation tells us how to align the rows of the matrices $\mathbf{P}$ and $\mathbf{Q}$ to get the maximum matching, which from (5) gives minimum distance between $\mathbf{P}$ and $\mathbf{AQ}$. In other words, by solving a linear assignment problem, we are computing the closest permutation of a matrix $\mathbf{Q}$ to bring it to $\mathbf{P}$.

Input:	Training dataset - $(\mathbf{x}_i, y_i)_{i=1}^n$, Maximum Iterations - max, Parameter - λ
Output:	Hyperplane - $(\mathbf{w}, b)$ and Centroid - $\mathbf{c}$
Step 0:	Set $j \leftarrow 1$
Step 1	Solve (3) from $(\mathbf{x}_i, y_i)_{i=1}^n$ to find centroid $\mathbf{c}^j$ and the radius R
Step 2	Solve (2) on from $(\mathbf{x}_i, y_i)_{i=1}^n$ to find $(\mathbf{w}^j, b^j)$ and margin M
Step 3	Solve Kuhn-Munkres Algorithm with cost matrix $\lambda y_i \mathbf{w}^j \mathbf{x}_i^\top + \mathbf{c}^j \mathbf{x}_i^\top$ for each i, let the permutation matrix obtained be $\mathbf{A}^{ij}$.
Step 4	Permute $\mathbf{x}_i$ with the permutation matrix obtained, that is, $\mathbf{x}_i \leftarrow \mathbf{A}^{ij}\mathbf{x}_i$.
Step 5	If $j = max$ return $(\mathbf{w}^j, b^j, \mathbf{c}^j)$ else $j \leftarrow j + 1$, goto Step 1

Table 1. Algorithmic description of π-SVM

4. Permutation Invariant SVMs

Consider a dataset that has undergone arbitrary permutations of the rows of each of its matrix-shaped input patterns. For instance, each input pattern is a set where no pre-determined ordering on the elements in the set. To use a classical SVM formulation (2) we would need the data to be reordered feature-wise. We present a version of SVM which we call the Permutation Invariant SVM (π-SVM for short) which is invariant to such feature-wise ordering. To do so, we will combine results from Section 2 and the Section 3 to propose an algorithm training a π-SVM. The bound in (4) suggests that for a smaller R and a larger M the expected probability of error is small. We thus suggest an iterative scheme where we both decrease the radius R and increase the margin M iteratively.

We begin with a training data $(\mathbf{x}_i, y_i)_{i=1}^n$ with the features (or rows) in each matrix-shaped $\mathbf{x}_i$ is arbitrarily permuted. We first find the centroid and radius R of the dataset and the largest margin hyperplane and its margin M. Both of these steps can be made robust to outliers via the C parameters). We then find the permutation using the Kuhn-Munkres algorithm for each example pattern that brings it closer to the centroid in the sphere while ensuring that we only increase its margin from the decision boundary. Thus, we move the data to ensure that R can only be decreased and M can only be increased. We then recompute the centroid, the hyperplane and the new R and M on the permute data and repeat. This process is iterated until some criterion is met. Table 1 summarizes the procedure.

Assume we have a current SVM classifier parameter setting $(\mathbf{w}, b)$ by solving (2). If we permute the rows of $\mathbf{x}_i$ so that left hand side of the inequality (2b) is maximized then we will have a better separation between the patterns belonging to the two classes. For instance, applying the maximization version of the Kuhn-Munkres algorithm to the cost matrix $y_i \mathbf{w} \mathbf{x}_i^\top$, we get a permutation matrix which maximizes $y_i \langle \mathbf{w}, \mathbf{A}\mathbf{x}_i \rangle$. In other words, it gives that permutation of $\mathbf{x}_i$ which is the *most separated* pattern among possible row permutations of $\mathbf{x}_i$.

Conversely, assume that $\mathbf{c}$ is the centroid obtained by solving (3) for a dataset. If we solve Kuhn-Munkres algorithm with the cost matrix $\mathbf{x}_i \mathbf{c}^\top$, we get a permutation matrix $\mathbf{A}$ which permutes the rows of $\mathbf{x}_i$ so that the resulting pattern is as close to the centroid $\mathbf{c}$ as possible among all the row permutation matrices of $\mathbf{x}_i$. That is, we *pull* the pattern close to the centroid by permuting the rows of $\mathbf{x}_i$. If we pull every training pattern towards the centroid, the resulting dataset will have a smaller radius R.

In the iterative procedure outlined in Table 1 we combine both criteria - pulling the patterns towards the centroid while increasing the margin with suitable tradeoff. We use a parameter λ to trade off the relative weights of the two goals - permuting to bring a datum closer to the centroid and permuting to increase its margin. We specifically solve the Kuhn-Munkres for a weighted combination of both inputs, i.e. $\lambda y_i \mathbf{w}^j \mathbf{x}_i^\top + \mathbf{c}^j \mathbf{x}_i^\top$. For high values of λ, this is guaranteed to achieve a margin as large or larger as the one found during the previous iteration of the algorithm. Similarly for low values of λ, the radius of the enclosing hypersphere is guaranteed to be equal to or smaller than the radius during the previous iteration. Ideally, one could perform a bisection search over the Lagrange multiplier λ for each input example by solving the Kuhn-Munkres algorithm over many $\lambda y_i \mathbf{w}^j \mathbf{x}_i^\top + \mathbf{c}^j x_i^\top$ to ensure that we never reduce the margin while we reduce the radius as much as possible. A more efficient alternative is to simply select a λ manually and trade off radius for margin in a fixed way (which is what we implemented in practice). This approach of incrementally increasing the margin and decreasing the radius is similar to Bi and Zhang (2005) where margin is incrementally increased over an uncertainty ball. Our framework addresses permutational uncertainty. Furthermore, instead of just increasing margin, we also simultaneously decrease the data radius (suitably traded off). Once we run the training algorithm in Table 1, we have $(\mathbf{w}, b)$ and $\mathbf{c}$. To predict the label of a test datum $\mathbf{x}$, we proceed as follows,

1. Solve Kuhn-Munkres with the cost matrices $\lambda \mathbf{w}\mathbf{x}^\top + \mathbf{c}\mathbf{x}^\top$ and $-\lambda \mathbf{w}\mathbf{x}^\top + \mathbf{c}\mathbf{x}^\top$, let $\mathbf{A}_+$ and $\mathbf{A}_-$ be the permutation matrices obtained respectively.

2. If $|\pi(\mathbf{w}, \mathbf{A}_+\mathbf{x}) + b| \geq |\pi(\mathbf{w}, \mathbf{A}_-\mathbf{x}) + b|$, then output $sign(\pi(\mathbf{w}, \mathbf{A}_+\mathbf{x}) + b)$ else output $sign(\pi(\mathbf{w}, \mathbf{A}_-\mathbf{x}) + b)$.

In other words, we permute $\mathbf{x}$ as a positive exemplar and then as a negative exemplar to see which permutation achieves a larger (absolute) margin. We then output the sign for $\mathbf{x}$ with $(\mathbf{w}, b)$ after applying the larger-margin permutation to it. While other prediction rules are also sensible, this one works well in practice.

With k iterations of the π-SVM, the running time of training is $O(k(n^3 + n^3 + nm^3))$. In practice, k is just a small constant, therefore, for small values of m, the running time of the algorithm is similar to that of the classical SVM.

5. Experiments

In this section we present experimental results. We present results on a small toy dataset as well as real world datasets. In all these experiments the primal SVM formulation (2) was used for π-SVM and the dual formulation (Burges, 1998) was used for experiments with a Gaussian RBF kernel.

5.1. Illustration with a toy dataset

Figure 1 illustrates the π-SVM on a toy dataset. We generated fifty points from a multivariate normal distribution with mean $[0\ 2]'$ as the positive class and fifty points from a multivariate normal distribution with mean $[0\ 6]'$ as the negative class. Both Gaussians used an identity covariance matrix. We wish to compare the hyperplane obtained on the actual dataset with the one obtained by π-SVM after randomly permuting the features.

The first plot shows the original patterns with the corresponding centroid, enclosing hypersphere (which is a circle in this simple two dimensional case) and the SVM hyperplane obtained with a linear kernel. We used a C value of 10 and λ value of 100 in this experiment. The second plot shows the patterns after permuting the coordinates of every pattern with probability half with the corresponding hyperplane and the centroid on the permuted data. Clearly, the classifier makes a few errors at this point. Our aim was to see if we could recover the hyperplane for the original data from this permuted data. We then performed one iteration on all patterns permuting them to pull towards the centroid and to maximize the margin. The third plot shows the patterns after performing Steps 2 and 3 in Table 1 along with the *old* hyperplane and the *old* bounding circle. The fourth plot shows the updated hyperplane and the updated circle. Not only did π-SVM recover a similar hyperplane on the original uncorrupted data, it gave a better margin compared to the actual dataset. With different values for λ we obtain slightly different margins and radius trade offs. Alternatively, we can do bisection search for λ for each example i within Step 3 to ensure that we never reduce margin as the algorithm iterates.

5.2. Experiment with Object Classification

We demonstrate the effectiveness of our method with classification tasks on three pairs of images from the Amsterdam Library of Object Images (Geusebroek et al., 2005) which consists of images of real everyday objects rotated in 3D. The first experiment consisted of images of planes and darts. The second experiment consisted of two almost identical looking jugs and the third experiment discriminated between two dolls.

In each case, the dataset consisted of 24 images with 12 images of each object. The aim of the experiment was to study the performance of $\pi-SVM$ compared to the other methods. One of the things that we wanted to make sure was that the position of the images was not a classification criterion. Images were moved slightly (both horizontally and vertically) by random number of pixels so as to ensure that the position of the images no longer remained a discriminative criterion. Images were down-sampled to 36×48 (and to 72×96 in case of plane dart as the objects became too small). Figure 2 shows the down sampled images, binarized images (by converting pixels with intensity more than 30 to 255 and others to 0) and the bag-of-pixels images for five instances of each object. We randomly selected 120 pixels (with a check to eliminate duplicates if possible) from the white region of the binarized images (where the intensity value was 255) and retained their spatial coordinates. In this representation, each image or point-cloud is stored as a matrix of $m = 120$ rows and $d = 2$ columns where each row represents a tuple or pixels (x, y) spatial coordinates. We then applied the π-SVM on this dataset since there is no natural ordering of the tuples in any given point-cloud. Any permutation of the rows of the pattern matrices rep-

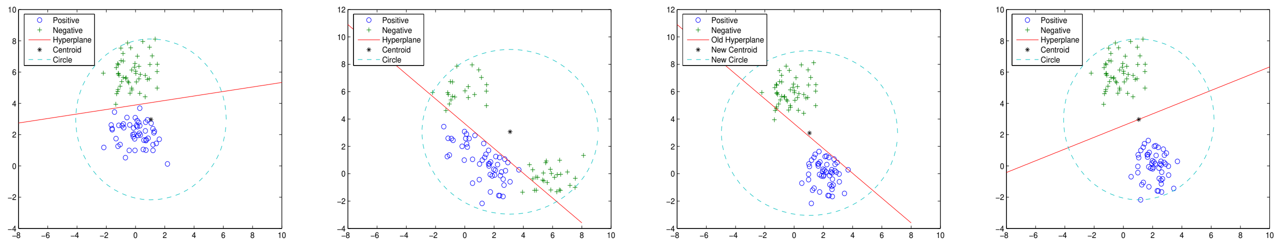

Figure 1. Illustration of π-SVM on a toy dataset.

	Plane/Dart		Jugs		Dolls	
	Acc	Var	Acc	Var	Acc	Var
π-SVM	97.22	0.002	95.83	0.001	98.61	0.006
RBF	66.67	0.014	93.75	0.001	96.52	0.009
Linear	66.67	0.014	91.67	0.001	95.83	0.009
Sorted	95.83	0.003	81.94	0.008	96.52	0.006
Random	88.89	0.004	72.91	0.008	65.28	0.007
Kondor/Jebara	90.97	0.057	71.52	0.003	86.11	0.015

Table 2. **Percentage accuracies with different classifiers on object classification tasks.**

resents the same underlying cloud of pixels.

Comparisons were made by running classical SVMs with Gaussian kernels *and* linear kernels on the binarized images in contrast to the π-SVM on the bag of pixels dataset. In each case, we selected three randomly chosen patterns from each class as training examples and used the remaining 18 patterns as test cases. Classical SVMs were attempted with different values of kernel parameter and C. The π-SVM was run with different values of λ and C. In each case, we noted the mean and the variance of the accuracy over eight randomized runs. Table 2 summarizes results.

It can be noted that π-SVM is the only method which has performed *consistently* well on all all the three datasets. The reason for the performance of π-SVM is the flexibility it has in permuting the tuples. When the objects are nearly identical, it has the ability to match nearby pixels of different images to same or nearby tuples in $\mathbf{w}$ and $\mathbf{c}$, thereby making it much more flexible than traditional classifiers which are rigid and always map the pixels at a certain position always with particular components of $\mathbf{w}$ of the classifier. Our hypothesis is that π-SVM works the best when a linear matching between the images also a matching in a semantically meaningful way. For example, if the images are translated/rotated slightly, doing a maximum matching would still give a matching in a semantically meaningful way in that similar components of the objects get matched with similar components of the other objects. Also, the additional flexibility of permuting the tuples makes the method work better than other methods especially when the number of training samples is low and the training samples contain translated and/or slightly rotated images.

Another quick straw-man approach to handling the bag of pixels or point-cloud representation would be to sort the (x, y) tuples according to their distance from a point (i.e. the origin) and to use the resultant matrix with the classical SVM. We tried this strategy as well as running a classical SVM on randomly sorted features. Both SVMs used a linear kernel. The resulting accuracy was nearly random in case of plane/dart and jugs experiments. Meanwhile, it was 80% on the dolls dataset - still far from the performance of the π-SVM. Clearly, the π-SVM beats such naive sorting strategies as well. Similarly, we compared against an SVM using the kernel on sets (Kondor & Jebara, 2003) which is invariant to ordering. This kernel sometimes works better than naive sorting yet was also outperformed by the π-SVM. Most interestingly, in situations where training data is sparse and not every view of each object is available, the π-SVM performs better because of its ability to match the components in the images.

5.3. Experiments with NIST

We next consider a digit recognition problem. The dataset we used consisted of one hundred patterns of the digits three and nine. Some of the typical dig-

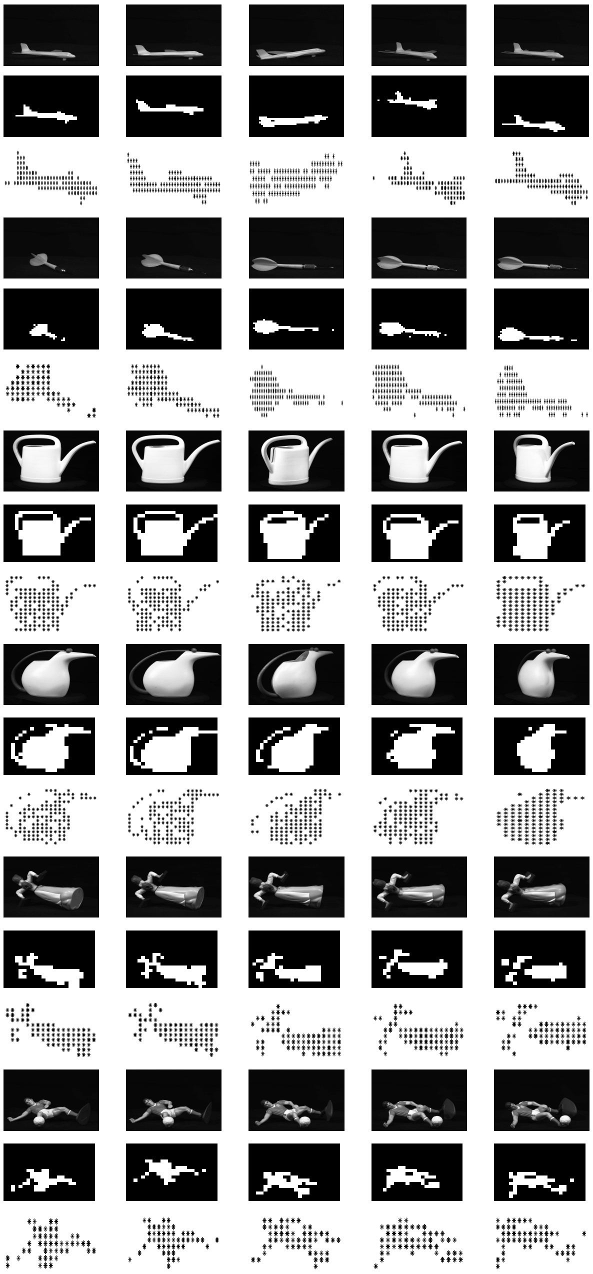

Figure 2. **Plane/Dart,Jugs and Dolls datasets.**

its are shown in the Figure 3. Each digit was represented by the (x, y) coordinates of the dark pixels in the image. This was done by randomly selecting pixel from dark regions in the gray scale image and retaining their (x, y) coordinates (as before). We retained 70 such points for each digit. The point-cloud images are then represented as matrices in $\mathbb{R}^{70\times 2}$. We evaluated the accuracy of the π-SVM over this dataset by doing a ten fold cross validation with different values for λ and C. We compared our method to standard

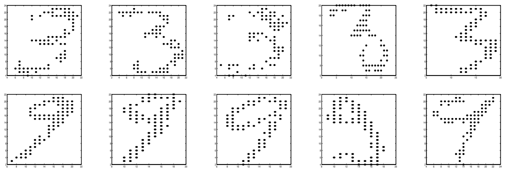

Figure 3. **NIST digits as bags of pixels**

	Linear	Gaussian
π-SVM	97.00	
SVM	91.00	95.00
Sorted	88.00	92.00
Random	69.00	80.00
Kondor,Jebara		94.00

Table 3. Percentage accuracies on the NIST dataset.

SVMs on the vectorized gray-scale digit images using both linear and Gaussian kernels (as is typically done in digit recognition experiments). We also compared our results with classical SVMs with linear and Gaussian kernels on point-cloud data with x, y coordinates under naive or random sorting. The results reported in Table 3 are the best accuracy (averaged over 10 iterations) over runs with different parameters. The π-SVM again performed well.

5.4. Experiments with UCI datasets

We next studied the problem of recovering from random permutations applied to the input vectors in standard UCI datasets. To do so, we randomly permuted the features of heart, ion and Pima datasets from the UCI Machine Learning Repository (Newman et al., 1998) with a *different permutation for each datum.* For these datasets, we have $d = 1$ since each pattern is just a vectors. We ran the π-SVM on arbitrarily permuted features dataset with ten fold cross validation for different values of the parameters λ and C. As before, one quick way of handling such data is to sort the features and to use a classical SVM. We also show the result on the original (uncorrupted) dataset which is an upper limit of how well the π-SVM or any permutation-cleaning algorithm can do. We also show the performance of a classical SVM on the randomly permuted input vectors.

Table 4 summarizes the results. In all the cases the reported results are the best average percentage accuracies over ten randomized runs with different parameters. All methods used a linear kernel. In the heart and ion datasets, our π-SVM was able to almost match the result of SVM on the original uncorrupted

	Ionosphere	Pima	Heart
π-SVM	85.14	69.87	85.18
Actual	85.42	76.75	86.67
Sorted	76.61	67.19	74.89
Random	68.24	65.60	52.36

Table 4. Percentage accuracies on UCI datasets.

datasets. In the case of Pima, the accuracy was slightly lower compared to that on the actual dataset. Nevertheless, the π-SVM performed better compared to the sorted features or using randomized features.

6. Conclusion

We proposed a method for handling the problem of classifying permuted data with SVM. The technique made use of a bound on the expected probability of error. Our proposed π-SVM incrementally searches for and compensates for permutations on each datum by iteratively increasing the margin while keeping tabs on the radius of the enclosing hypersphere. Experiments indicate that π-SVM performed much better than traditional SVMs (with RBF, linear or vector-set kernels) on small datasets for 3D objects classification and digit problems as well as permutationally corrupted UCI datasets. It also outperformed naive ways of handling permuted data such as sorting.

The π-SVM is also kernelizable. In other words, instead of a linear inner product between d-dimensional tuples in each input matrix, we may use a kernel. Thus, π-SVM can deal with sets of permuted Hilbert-space elements. While most attempts to handle uncertainty have focused mainly on margin, integrating our approach with the bounding hypersphere and radius arguments give interesting results and allow us to reliably and incrementally remove sources of nuisance variation such as permutation. We are also considering other SVMs designed to handle different types of variation beyond permutations where the margin and the radius can be incrementally increased and decreased, respectively. We are also exploring ways to solve the problem as one single optimization rather than one for SVM and one for data radius. Other directions include using this framework to linearly transform data for low probability of error. Extending our technique to support vector clustering is also an interesting direction.

7. Acknowledgments

Funded by NSF Grants IIS-0347499 & CCR-0312690.

References

Ben-Hur, A., Horn, D., Siegelmann, H. T., & Vapnik, V. (2001). Support vector clustering. *Journal of Machine Learning Research*.

Bennett, K. P., & Mangasarian, O. L. (1993). Multicategory separation via linear programming. *Optimization Methods and Software*, *3*, 27–39.

Bi, J., & Zhang, T. (2005). Support vector classification with input data uncertainty. *Neural Information Processing Systems 17*.

Burges, C. (1998). A tutorial on support vector machines for pattern recognition. *Knowledge Discovery and Data Mining*.

Cortes, C., & Vapnik, V. (1995). Support-vector networks. *Machine Learning*, *20*, 273–297.

Geusebroek, J. M., Burghouts, G. J., & Smeulders, A. W. M. (2005). The Amsterdam library of object images. *Int. J. Comput. Vision*, *61*, 103–112.

Goldberg, A. V., & Kennedy, R. (1995). An efficient cost scaling algorithm for the assignment problem. *Mathematical Programming*, *71*, 153–178.

Jebara, T. (2004). Kernelizing sorting, permutation and alignment for minimum volume PCA. *COLT*.

Kirshner, S., Parise, S., & Smyth, P. (2003). Unsupervised learning with permuted data. *International Conference on Machine Learning*.

Kondor, R., & Jebara, T. (2003). A kernel between sets of vectors. International Conference on Machine Learning.

Lodhi, H., Saunders, C., Shawe-Taylor, J., Cristianini, N., & Watkins, C. (2002). Text classification using string kernels. *Journal of Machine Learning Research*, *2*, 419–444.

Mah, P., Ueda, N., Akutsu, T., Perret, J., & Vert, J. (2004). Extensions of marginalized graph kernels. *International Conference on Machine Learning*.

Newman, D., Hettich, S., Blake, C., & Merz, C. (1998). UCI repository of machine learning databases.

Papadimitriou, C. H., & Steiglitz, K. (1982). *Combinatorial optimization: Algorithms and complexity*. Prentice-Hall.

Simard, P., LeCun, Y., Denker, J. S., & Victorri, B. (1996). Transformation invariance in pattern recognition-tangent distance and tangent propagation. *Neural Networks: Tricks of the Trade* (pp. 239–27).

Taskar, B., Guestrin, C., & Koller, D. (2004). Max-margin Markov networks. *Neural Information Processing Systems*.

Vapnik, V. (1995). *The nature of statistical learning theory*. Springer-Verlag.

Weston, J., Mukherjee, S., Chapelle, O., Pontil, M., Poggio, T., & Vapnik, V. (2000). Feature selection for SVMs. *Neural Information Processing Systems*.