Report No. UIUCDCS-R-2006-2716 **UILU-ENG-2006-1748**

Learning with Tensor Representation

by

Deng Cai, Xiaofei He, and Jiawei Han

April 2006

Learning with Tensor Representation[*]

Deng Cai[†] Xiaofei He[‡] Jiawei Han[†]

[†] Department of Computer Science, University of Illinois at Urbana-Champaign

[‡] Yahoo! Research Labs

Abstract

Most of the existing learning algorithms take vectors as their input data. A function is then learned in such a vector space for classification, clustering, or dimensionality reduction. However, in some situations, there is reason to consider data as tensors. For example, an image can be considered as a second order tensor and a video can be considered as a third order tensor. In this paper, we propose two novel algorithms called **Support Tensor Machines** (STM) and **Tensor Least Square** (TLS). These two algorithms operate in the tesnor space. Specifically, we represent data as the second order tensors (or, matrices) in $\mathcal{R}^{n_1} \otimes \mathcal{R}^{n_2}$, where $\mathcal{R}^{n_1}$ and $\mathcal{R}^{n_2}$ are two vector spaces. STM aims at finding a maximum margin classifier in the tensor space, while TLS aims at finding a minimum residual sum-of-squares classifier. With tensor representation, the number of parameters estimated by STM (TLS) can be greatly reduced. Therefore, our algorithms are especially suitable for small sample cases. We compare our proposed algorithms with SVM and the ordinary Least Square method on six databases. Experimental results show the effectiveness of our algorithms.

1 Introduction

The problem of data representation has been at the core of machine learning. Most of the traditional learning algorithms are based on the *Vector Space Model*. That is, the data are represented as vectors $\mathbf{x} \in \mathbb{R}^n$. The learning algorithms aim at finding a linear (or nonlinear) function $f(\mathbf{x}) = \mathbf{w}^T\mathbf{x}$ according to some pre-defined criteria, where $\mathbf{w} = (w_1, \cdots, w_n)^T$ are the parameters to estimate. However, in some situations, there might be reason to consider data as tensors. For example, an image is essentially a second order tensor, or matrix. It is reasonable to consider that pixels close to each other are correlated to some extent. Similarly, a video is essentially a third order tensor with the third dimension being time. Also, two consecutive frames in a video are probably correlated.

[*] The work was supported in part by the U.S. National Science Foundation NSF IIS-03-08215/IIS-05-13678. Any opinions, findings, and conclusions or recommendations expressed in this paper are those of the authors and do not necessarily reflect the views of the funding agencies.

In supervised learning settings with many input features, overfitting is usually a potential problem unless there is ample training data. For example, it is well known that for unregularized discriminative models fit via training-error minimization, sample complexity (i.e., the number of training examples needed to learn "well") grows linearly with the Vapnik-Chernovenkis (VC) dimension. Further, the VC dimension for most models grows about linearly in the number of parameters (Vapnik, 1982), which typically grows at least linearly in the number of input features. All these reasons lead us to consider new representations and corresponding learning algorithms with less number of parameters.

In this paper, we propose two novel supervised learning algorithms for data classification, which are called **Support Tensor Machines** (STM) and **Tensor Least Square** (TLS). Different from most of previous classification algorithms which take vectors in $\mathbb{R}^n$ as inputs, STM and TLS take the second order tensors in $\mathcal{R}^{n_1} \otimes \mathcal{R}^{n_2}$ as inputs, where $n_1 \times n_2 \approx n$. For example, a vector $\mathbf{x} \in \mathbb{R}^n$ can be transformed by some means to a second order tensor $\mathbf{X} \in \mathcal{R}^{n_1} \otimes \mathcal{R}^{n_2}$. A linear classifier in $\mathbb{R}^n$ can be represented as $\mathbf{w}^T\mathbf{x} + b$ in which there are $n + 1$ ($\approx n_1 \times n_2 + 1$) parameters ($b, w_i, i = 1, \cdots, n$). Similarly, a linear classifier in the tensor space $\mathcal{R}^{n_1} \otimes \mathcal{R}^{n_2}$ can be represented as $\mathbf{u}^T\mathbf{X}\mathbf{v} + b$ where $\mathbf{u} \in \mathcal{R}^{n_1}$ and $\mathbf{v} \in \mathcal{R}^{n_2}$. Thus, there are only $n_1 + n_2 + 1$ parameters. This property makes STM and TLS especially suitable for small sample cases.

Recently there have been a lot of interests in tensor based approaches to data analysis in high dimensional spaces. Vasilescu and Terzopoulos (2003) have proposed a novel face representation algroithm called Tensorface. Tensorface represents the set of face images by a higher-order tensor and extends Singular Value Decomposition (SVD) to higher-order tensor data. Some other researchers have also shown how to extend Principal Component Analysis, Linear Discriminant Analysis, Locality Preserving Projection, and Non-negative Matrix Factorization to higher order tensor data (Cai et al., 2005; He et al., 2005; Shashua & Hazan, 2005; Ye et al., 2004). Most of previous tensor based learning algorithms are focused on dimensionality reduction. In this paper, we extend SVM and Least Square based ideas to tensor data for classification.

The rest of this paper is organized as follows: Section 2 gives some descriptions on tensor space model for data representation. The Tensor Least Square (TLS) and Support Tensor Machines (STM) approaches for classification in tensor space are described in Section 3. The experimental results on six datasets from UCI Repository are presented in Section 4. In Section 5, we provide a general algorithm for learning functions in tensor space. Finally, we provide some concluding remarks and suggestions for future work in Section 6.

2 Tensor based Data Representation

Traditionally, a data sample is represented by a vector in high dimensional space. Some learning algorithms are then applied in such a vector space for classification. In this section, we introduce

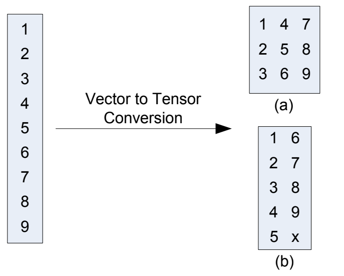

Figure 1: Vector to tensor conversion. 1~9 denote the positions in the vector and tensor formats. (a) and (b) are two possible tensors. The 'x' in tensor (b) is a padding constant.

a new data representation model: Tensor Space Model (TSM)[1].

In Tensor Space Model, a data sample is represented as a tensor. Each element in the tensor corresponds to a feature. For a data sample $\mathbf{x} \in \mathbb{R}^n$, we can convert it to the second order tensor (or matrix) $X \in \mathbb{R}^{n_1 \times n_2}$, where $n_1 \times n_2 \approx n$. Figure 1 shows an example of converting a vector to a tensor. There are two issues about converting a vector to a tensor.

The first one is how to choose the size of the tensor, *i.e.*, how to select n_1 and n_2. In figure 1, we present two possible tensors for a 9-dimensional vector. Suppose $n_1 \geq n_2$, in order to have at least n entries in the tensor while minimizing the size of the tensor, we have $(n_1 - 1) \times n_2 < n \leq n_1 \times n_2$. With such a requirement, there are still many choices of n_1 and n_2, especially when n is large. Generally all these (n_1, n_2) combinations can be used. However, it is worth noticing that the number of parameters of a linear function in the tensor space is $n_1 + n_2$. Therefore, one may try to minimize $n_1 + n_2$. In other words, n_1 and n_2 should be as close as possible.

The second issue is how to sort the features in the tensor. In vector space model, we implicitly assume that the features are independent. A linear function in the vector space can be written as $g(\mathbf{x}) = \mathbf{w}^T\mathbf{x} + b$. Clearly, the change of the order of the features has no impact on the function learning. In tensor space model, a linear function can be written as $f(X) = \mathbf{u}^T X \mathbf{v} + b$. Thus, the independency assumption of the features no longer holds for the learning algorithms in the tensor space model. Different feature sorting will lead to different learning result in the tensor space model.

A possible approach for sorting the features is using the $\mathbf{w}$ learned by a vector classifier. Each w_i in $\mathbf{w}$, where $i = 1, \cdots, n$, corresponds to a feature. Thus, the problem of sorting features in the tensor can be converted to fill these n elements w_i into the $n_1 \times n_2$ tensor. We divide w_i into three groups: positive, negative and zero. The corresponding features in the same group tend to be correlated. Suppose each group has l_k elements, where $k = 1, 2, 3$. For each group, we sort l_k elements by their absolute value and get $(|w_1^k| \geq \cdots \geq |w_{l_k}^k|)$. We take the first n_1 elements to form the first column of the tensor and the next n_1 elements to form the second column of the tensor

[1]Note that, in this paper our primary interest is focused on the second order tensors. However, the TSM presented here and the algorithms presented in the next section can also be applied to higher order tensors.

and so on. For each group, we will fill $\lfloor l_k/n_1 \rfloor$ columns. The remainders in each group will be put together to fill the remaining entries in the tensor.

In this paper, we take $n_1 \approx n_2 \approx \sqrt{n}$ which we called square tensors. The features are filled into the tensor entries by using the way we described above. The better ways of converting a data vector to a data tensor with theoretical guarantee will be left for our future work.

3 Classification with Tensor Representation

Given a set of training samples $\{\mathbf{X}_i, y_i\}, i = 1, \cdots, m$, where $\mathbf{X}_i$ is the data point in order-2 tensor space, $\mathbf{X}_i \in \mathcal{R}^{n_1} \otimes \mathcal{R}^{n_2}$ and $y_i \in \{-1, 1\}$ is the label associated with $\mathbf{X}_i$.

Let $(\mathbf{u}_1, \cdots, \mathbf{u}_{n_1})$ be a set of orthonormal basis functions of $\mathbb{R}^{n_1}$. Let $(\mathbf{v}_1, \cdots, \mathbf{v}_{n_2})$ be a set of orthonormal basis functions of $\mathbb{R}^{n_2}$. Thus, $\mathbf{X}$ can be uniquely written as:

$$\mathbf{X} = \sum_{ij} (\mathbf{u}_i^T \mathbf{X} \mathbf{v}_j) \mathbf{u}_i \mathbf{v}_j^T$$

A linear classifier in the tensor space can be written as $f(\mathbf{X}) = \mathbf{u}^T \mathbf{X} \mathbf{v} + b$, where $\mathbf{u} \in \mathbb{R}^{n_1}$ and $\mathbf{v} \in \mathbb{R}^{n_2}$ are two vectors. The problem of linear classification in tensor space model is to find the $\mathbf{u}$ and $\mathbf{v}$ based on a specific objective function.

In the following two subsections, we introduce two novel classifiers in tensor space model based on different objective functions, which are Tensor Least Square classifier and Support Tensor Machines. One is the tensor generalization of least square classifier and the other is the tensor generalization of support vector machines.

3.1 Tensor Least Square Analysis

The least square classifier might be one of the most well known classifiers (Duda et al., 2000; Hastie et al., 2001). In this subsection, we extend the least square idea to tensor space model and develop a Tensor Least Square classifier (TLS).

The objective function in tensor least square analysis is as follows[2]:

$$\min_{\mathbf{u},\mathbf{v}} \sum_{i=1}^{m} \left(\mathbf{u}^T X_i \mathbf{v} - y_i \right)^2 \tag{1}$$

[2]Note that, we need to add a constant column (or row) to each data tensor to fit the intercept.

By simple algebra, we see that:

$$\begin{aligned}
& \sum_{i=1}^{m} \left(\mathbf{u}^T X_i \mathbf{v} - y_i\right)^2 \\
= & \sum_{i=1}^{m} \left(\mathbf{u}^T X_i \mathbf{v} \mathbf{v}^T X_i^T \mathbf{u} - 2 y_i \mathbf{u}^T X_i \mathbf{v} + y_i^2\right) \\
= & \mathbf{u}^T \left(\sum_{i=1}^{m} X_i \mathbf{v} \mathbf{v}^T X_i^T\right) \mathbf{u} - \mathbf{u}^T \left(2 \sum_{i=1}^{m} y_i X_i\right) \mathbf{v} \\
& + \sum_{i=1}^{m} y_i^2 \qquad (2)
\end{aligned}$$

Similarly, we also have:

$$\begin{aligned}
& \sum_{i=1}^{m} \left(\mathbf{u}^T X_i \mathbf{v} - y_i\right)^2 \\
= & \sum_{i=1}^{m} \left(\mathbf{v}^T X_i^T \mathbf{u} \mathbf{u}^T X_i \mathbf{v} - 2 y_i \mathbf{v}^T X_i^T \mathbf{u} + y_i^2\right) \\
= & \mathbf{v}^T \left(\sum_{i=1}^{m} X_i^T \mathbf{u} \mathbf{u}^T X_i\right) \mathbf{v} - \mathbf{v}^T \left(2 \sum_{i=1}^{m} y_i X_i^T\right) \mathbf{u} \\
& + \sum_{i=1}^{m} y_i^2 \qquad (3)
\end{aligned}$$

Requiring the derivative of Eqn (2) with respect to $\mathbf{u}$ to be zero, we get:

$$\left(\sum_i X_i \mathbf{v} \mathbf{v}^T X_i^T\right) \mathbf{u} = \left(2 \sum_i y_i X_i\right) \mathbf{v}$$

Thus, we get:

$$\mathbf{u} = \left(\sum_i X_i \mathbf{v} \mathbf{v}^T X_i^T\right)^{-1} \left(2 \sum_i y_i X_i\right) \mathbf{v} \qquad (4)$$

Similarly, we can get:

$$\mathbf{v} = \left(\sum_i X_i^T \mathbf{u} \mathbf{u}^T X_i\right)^{-1} \left(2 \sum_i y_i X_i^T\right) \mathbf{u} \qquad (5)$$

Notice that $\mathbf{u}$ and $\mathbf{v}$ are dependent on each other, and can not be solved independently. We can use an iterative approach to compute the optimal $\mathbf{u}$ and $\mathbf{v}$, *i.e.*, we first fix $\mathbf{u}$, and compute $\mathbf{v}$ by Equation (5); Then we fix $\mathbf{v}$, and compute $\mathbf{u}$ by Equation (4). The convergence proof of such an iterative approach is given in Section 3.3.

3.2 Support Tensor Machines

Support Vector Machines are a family of pattern classification algorithms developed by Vapnik (1995) and collaborators. SVM training algorithms are based on the idea of *structural risk minimization* rather than *empirical risk minimization*, and give rise to new ways of training polynomial, neural network, and radial basis function (RBF) classifiers. SVM has proven to be effective for many classification tasks (Joachims, 1998; Ronfard et al., 2002).

Specifically, SVM try to find a decision surface that maximizes the margin between the data points in a training set. The objective function of linear SVM can be stated as:

$$\begin{aligned} \min_{\mathbf{w},b,\xi} \quad & \frac{1}{2}\mathbf{w}^T\mathbf{w} + C\sum_{i=1}^{m}\xi_i \\ \text{subject to} \quad & y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1 - \xi_i, \\ & \xi_i \geq 0, \quad i = 1, \cdots, m. \end{aligned} \tag{6}$$

Our Support Tensor Machines (STM) is fundamentally based on the same idea. As we discussed before, A linear classifier in the tensor space can be naturally represented as follows:

$$f(\mathbf{X}) = \mathbf{u}^T\mathbf{X}\mathbf{v} + b, \quad \mathbf{u} \in \mathbb{R}^{n_1}, \mathbf{v} \in \mathbb{R}^{n_2} \tag{7}$$

Equation (7) can be rewritten through matrix inner product as follows:

$$f(\mathbf{X}) = <\mathbf{X}, \mathbf{u}\mathbf{v}^T> + b, \quad \mathbf{u} \in \mathbb{R}^{n_1}, \mathbf{v} \in \mathbb{R}^{n_2} \tag{8}$$

Thus, the large margin optimization problem in the tensor space is reduced to the following:

$$\begin{aligned} \min_{\mathbf{u},\mathbf{v},b,\xi} \quad & \frac{1}{2}\|\mathbf{u}\mathbf{v}^T\|^2 + C\sum_{i=1}^{m}\xi_i \\ \text{subject to} \quad & y_i(\mathbf{u}^T\mathbf{X}_i\mathbf{v} + b) \geq 1 - \xi_i, \\ & \xi_i \geq 0, \quad i = 1, \cdots, m. \end{aligned} \tag{9}$$

We will now switch to a Lagrangian formulation of the problem. We introduce positive Lagrange multipliers $\alpha_i, \mu_i, i = 1, \cdots, m$, one for each of the inequality constraints (9). This gives Lagrangian:

$$\begin{aligned} L_P \;=\; & \frac{1}{2}\|\mathbf{u}\mathbf{v}^T\|^2 + C\sum_i \xi_i - \sum_i \alpha_i y_i \left(\mathbf{u}^T\mathbf{X}_i\mathbf{v} + b\right) \\ & + \sum_i \alpha_i - \sum_i \alpha_i\xi_i - \sum_i \mu_i\xi_i \end{aligned}$$

Note that

$$\begin{aligned} \frac{1}{2}\|\mathbf{u}\mathbf{v}^T\|^2 &= \frac{1}{2}trace\left(\mathbf{u}\mathbf{v}^T\mathbf{v}\mathbf{u}^T\right) \\ &= \frac{1}{2}\left(\mathbf{v}^T\mathbf{v}\right) trace\left(\mathbf{u}\mathbf{u}^T\right) \\ &= \frac{1}{2}\left(\mathbf{v}^T\mathbf{v}\right)\left(\mathbf{u}^T\mathbf{u}\right) \end{aligned}$$

Thus, we have:

$$\begin{aligned} L_P &= \frac{1}{2}\left(\mathbf{v}^T\mathbf{v}\right)\left(\mathbf{u}^T\mathbf{u}\right) + C\sum_i \xi_i \\ &\quad -\sum_i \alpha_i y_i \left(\mathbf{u}^T\mathbf{X}_i\mathbf{v} + b\right) \\ &\quad +\sum_i \alpha_i - \sum_i \alpha_i\xi_i - \sum_i \mu_i\xi_i \end{aligned}$$

Requiring that the gradient of L_P with respect to $\mathbf{u}$, $\mathbf{v}$, b and ξ_i vanish give the conditions:

$$\mathbf{u} = \frac{\sum_i \alpha_i y_i \mathbf{X}_i \mathbf{v}}{\mathbf{v}^T\mathbf{v}} \tag{10}$$

$$\mathbf{v} = \frac{\sum_i \alpha_i y_i \mathbf{u}^T \mathbf{X}_i}{\mathbf{u}^T\mathbf{u}} \tag{11}$$

$$\sum_i \alpha_i y_i = 0 \tag{12}$$

$$C - \alpha_i - \mu_i = 0, \quad i = 1, \cdots, m \tag{13}$$

From Equations (10) and (11), we see that $\mathbf{u}$ and $\mathbf{v}$ are dependent on each other, and can not be solved independently. In the following, we describe a simple yet effective computational method to solve this optimization problem.

We first fix $\mathbf{u}$. Let $\beta_1 = \|\mathbf{u}\|^2$ and $\mathbf{x}_i = \mathbf{X}_i^T\mathbf{u}$. Thus, the optimization problem (9) can be rewritten as follows:

$$\begin{aligned} \min_{\mathbf{v},b,\xi} \quad & \frac{1}{2}\beta_1\|\mathbf{v}\|^2 + C\sum_{i=1}^m \xi_i \\ \text{subject to} \quad & y_i(\mathbf{v}^T\mathbf{x}_i + b) \geq 1 - \xi_i, \\ & \xi_i \geq 0, \quad i = 1, \cdots, m. \end{aligned} \tag{14}$$

It is clear that the new optimization problem (14) is identical to the standard SVM optimization problem. Thus, we can use the same computational methods of SVM to solve (14), such as (Graf et al., 2004; Platt, 1998; Platt, 1999).

Once $\mathbf{v}$ is obtained, let $\beta_2 = \|\mathbf{v}\|^2$ and $\tilde{\mathbf{x}}_i = \mathbf{X}\mathbf{v}$. Thus, $\mathbf{u}$ can be obtained by solving the following optimization problem:

$$\begin{aligned} \min_{\mathbf{u},b,\xi} \quad & \frac{1}{2}\beta_2\|\mathbf{u}\|^2 + C\sum_{i=1}^m \xi_i \\ \text{subject to} \quad & y_i(\mathbf{u}^T\tilde{\mathbf{x}}_i + b) \geq 1 - \xi_i, \\ & \xi_i \geq 0, \quad i = 1, \cdots, m. \end{aligned} \tag{15}$$

Again, we can use the standard SVM computational methods to solve this optimization problem. Thus, $\mathbf{v}$ and $\mathbf{u}$ can be obtained by iteratively solving the optimization problems (14) and (15). In our experiments, $\mathbf{u}$ is initially set to the vector of all ones.

3.3 Convergence Proof

In this section, we provide a convergence proof of the iterative computational method in STM and TLS algorithms. We have the following theorem:

Theorem 1 *The iterative procedure to solve the optimization problems (14) and (15) will monotonically decreases the objective function value in (9), and hence the STM algorithm converges.*

Proof Define:

$$f(\mathbf{u}, \mathbf{v}) = \frac{1}{2}\|\mathbf{u}\mathbf{v}^T\|^2 + C\sum_{i=1}^{m} \xi_i$$

Let $\mathbf{u}_0$ be the initial value. Fixing $\mathbf{u}_0$, we get $\mathbf{v}_0$ by solving the optimization problem (14). Likewise, fixing $\mathbf{v}_0$, we get $\mathbf{u}_1$ by solving the optimization problem (15).

Notice that the optimization problem of SVM is convex, so the solution of SVM is globally optimum (Burges, 1998; Fletcher, 1987). Specifically, the solutions of equations (14) and (15) are globally optimum. Thus, we have:

$$f(\mathbf{u}_0, \mathbf{v}_0) \geq f(\mathbf{u}_1, \mathbf{v}_0)$$

Finally, we get:

$$f(\mathbf{u}_0, \mathbf{v}_0) \geq f(\mathbf{u}_1, \mathbf{v}_0) \geq f(\mathbf{u}_1, \mathbf{v}_1) \geq f(\mathbf{u}_2, \mathbf{v}_1) \geq \cdots$$

Since f is bounded from below by 0, it converges. ∎

Similarly, for Tensor Least Square algorithm, since the solution for least square is globally optimum, iterative procedure (4) and (5) will monotonically decreases the objective function value in (1), and hence the TLS algorithm converges.

3.4 From Matrix to High Order Tensor

The TLS and STM algorithms described above take order-2 tensors, *i.e.*, matrices, as input data. However, these algorithms can also be extended to high order tensors. In this section, we briefly describe the extension of these algorithms to high order (> 2) tensors. we take STM as an example.

Let (T_i, y_i), $i = 1, \cdots, m$ denote the training samples, where $T_i \in \mathcal{R}^{n_1} \otimes \cdots \otimes \mathcal{R}^{n_k}$. The decision function of STM is:

$$f(T) = T(\mathbf{a}^1, \mathbf{a}^2, \cdots, \mathbf{a}^k) + b$$

$$\mathbf{a}^1 \in \mathbb{R}^{n_1}, \mathbf{a}^2 \in \mathbb{R}^{n_2}, \cdots, \mathbf{a}^k \in \mathbb{R}^{n_k}$$

where

$$T(\mathbf{a}^1, \mathbf{a}^2, \cdots, \mathbf{a}^k) = \sum_{\substack{1 \leq i_1 \leq n_1 \\ \vdots \\ 1 \leq i_k \leq n_k}} T_{i_1, \cdots, i_k} a^1_{i_1} \times \cdots \times a^k_{i_k}$$

As before, $\mathbf{a}^1, \cdots, \mathbf{a}^k$ can also be computed iteratively.

We first introduce the l-mode product of a tensor T and a vector $\mathbf{a}$, which we denote as $T \times_l \mathbf{a}$. The result of l-mode product of a tensor $T \in \mathcal{R}^{n_1} \otimes \cdots \otimes \mathcal{R}^{n_k}$ and a vector $\mathbf{a} \in \mathbb{R}^{n_l}, 1 \leq l \leq k$ will be a new tensor $B \in \mathcal{R}^{n_1} \otimes \cdots \otimes \mathcal{R}^{n_{l-1}} \otimes \mathcal{R}^{n_{l+1}} \otimes \cdots \mathcal{R}^{n_k}$, where

$$B_{i_1,\cdots,i_{l-1},i_{l+1},\cdots,i_k} = \sum_{i_l=1}^{n_l} T_{i_1,\cdots,i_{l-1},i_l,i_{l+1},\cdots,i_k} \cdot a_{i_l}$$

Thus, the decision function in higher order tensor space can also be written as:

$$f(T) = T \times_1 \mathbf{a}^1 \times_2 \mathbf{a}^2 \cdots \times_k \mathbf{a}^k + b$$

The optimization problem of STM in high order tensors is:

$$\begin{aligned} \min_{\mathbf{a}^1,\cdots,\mathbf{a}^k,b,\xi} \quad & \frac{1}{2}\|\mathbf{a}^1 \otimes \cdots \otimes \mathbf{a}^k\|^2 + C\sum_{i=1}^{m} \xi_i \\ \text{subject to} \quad & y_i(T_i(\mathbf{a}^1, \mathbf{a}^2, \cdots, \mathbf{a}^k) + b) \geq 1 - \xi_i, \\ & \xi_i \geq 0, \quad i = 1, \cdots, m. \end{aligned} \tag{16}$$

Here $\|\mathbf{a}^1 \otimes \cdots \otimes \mathbf{a}^k\|$ denotes the tensor norm of $\mathbf{a}^1 \otimes \cdots \otimes \mathbf{a}^k$ (Lee, 2002).

First, to compute $\mathbf{a}^1$, we fix $\mathbf{a}^2, \cdots, \mathbf{a}^k$. Let $\beta_2 = \|\mathbf{a}^2\|^2, \cdots, \beta_k = \|\mathbf{a}^k\|^2$. We then define $\mathbf{t}_i = T_i \times_2 \mathbf{a}^2 \cdots \times_k \mathbf{a}^k$. Thus, the optimization problem (16) can be reduced as follows:

$$\begin{aligned} \min_{\mathbf{a}^1,b,\xi} \quad & \frac{1}{2}\beta_2 \cdots \beta_k \|\mathbf{a}^1\|^2 + C\sum_{i=1}^{m} \xi_i \\ \text{subject to} \quad & y_i({\mathbf{a}^1}^T \mathbf{t}_i + b) \geq 1 - \xi_i, \\ & \xi_i \geq 0, \quad i = 1, \cdots, m. \end{aligned} \tag{17}$$

Again, we can use the standard SVM computational methods to solve this optimization problem. Once $\mathbf{a}^1$ is computed, we can fix $\mathbf{a}^1, \mathbf{a}^3, \cdots, \mathbf{a}^k$ to compute $\mathbf{a}^2$. So on, all the $\mathbf{a}^i$ can be computed in such iterative manner.

4 Experiments

To evaluate the performance of tensor based classifiers, we performed experiments on six datasets from UCI Repository and report results for all of them. These six datasets include BREAST-CANCER, DIABETES, IONOSPHERE, MUSHROOMS, SONAR, as well as the ADULT data in a representation produced by Platt (1999). All of these datasets are binary categories and the features were scaled to $[-1, 1]$. The preprocessed datasets can also be downloaded from LIBSVM dataset webpage[3].

[3] `http://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/`

Table 1: Classification accuracies on various data sets. n is the number of features and m is the total number of examples

Data set	n	m	Train	TLS	LS	TLS vs LS	STM	SVM	STM vs SVM
Adult	123	32561	0.1%	63.6	60.7	$\gg$	76.2	76.5	$\sim$
Breast-cancer	10	683	1%	87.6	85.5	$\gg$	94.3	92.7	$\gg$
Diabetes	8	768	1%	60.3	59.0	$\sim$	66.2	66.5	$\sim$
Ionosphere	34	351	5%	75.4	72.4	$\gg$	77.4	76.4	$>$
Mushrooms	112	8124	0.1%	72.2	72.4	$\sim$	85.7	83.1	$\gg$
Sonar	60	208	5%	60.7	60.3	$\sim$	65.1	62.9	$\gg$

"$\gg$" or "$\ll$" means P-value ≤ 0.01 in the paired T-test on the 50 random splits
"$>$" or "$<$" means $0.01 <$ P-value ≤ 0.05
"$\sim$" means P-value > 0.05

All the datasets were randomly split into training and testing sets. Notice that the training sets are pretty small since we are particularly interested in the performance on the small training size cases. We averaged the results over 50 random splits and reported the average performance.

4.1 Experimental Results on TLS and LS

We used the *regress* function in statistics toolbox of Matlab 7.04 to solve the least square problems in both Tensor Least Square classifier (TLS) and Least Square classifier (LS).

Table 1 summarized the results. Besides the average accuracy, a *paired T-test* on the 50 random splits is also reported, which indicates whether the difference between two systems is significant. We can see that in 3 out of 6 datasets, TLS is better than LS. In the remaining 3 datasets, both algorithms are comparable.

To get a more detailed picture of the performance of TLS with respect to the training set size. We tested TLS and LS over various training sizes (from 1% to 10%) on BREAST-CANCER dataset. The results are shown in Figure 2. As can be seen, when the training set is small (1%, 2% and 3%), TLS outperforms LS. As the number of training samples increases, TLS tends to get same results with LS.

4.2 Experimental Results on STM and SVM

In this experiment, we compared the classification performance of Support Tensor Machines (STM) and traditional Support Vector Machines (SVM). We used the LIBSVM system (Chang & Lin, 2001) and tested it with the linear model.

For both STM and SVM, there is a parameter C need to be set. We use cross-validation on the

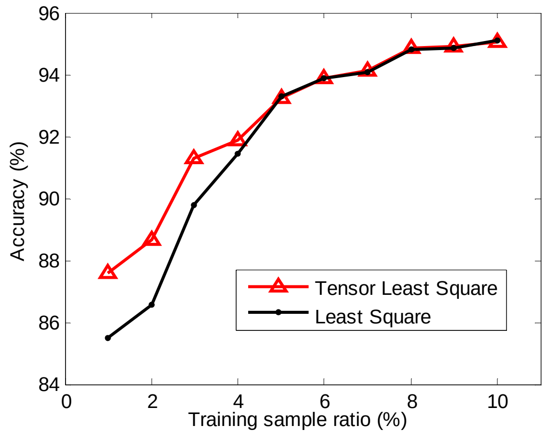

Figure 2: Classification accuracy with respect to the training sample size of TLS and LS on BREAST-CANCER dataset. As can be seen, when the training set is small (1%, 2% and 3%), TLS outperforms LS. As the number of training samples increases, TLS tends to get same results with LS.

training set to find the best parameter C.

Table 1 summarized the results. We can see that in 4 out of 6 datasets, STM is better than SVM. In the remaining 2 datasets, both algorithms are comparable. We also tested STM and SVM over various training size (from 1% to 10%) on BREAST-CANCER dataset. The results are shown in Figure 3. As can be seen, when the training set is small (1%, 2% and 3%), STM outperforms SVM. As the number of training samples increases, STM tends to get same results with SVM.

In all the cases, large margin classifiers (STM and SVM) are better than least square classifiers (TLS and LS).

Both of these two experiments suggest that classifiers based on tensor representation are particularly suitable for small sample problems. This might be due to the fact that the number of parameters need to be estimated in tensor classifiers is $n_1 + n_2$ which can be much smaller than $n_1 \times n_2$ in vector classifiers.

5 The General Algorithm for Learning Functions in Tensor Space

In the above sections, we have developed an iterative algorithm for solving the optimization problems of STM and TLS. It can be noticed that these two algorithms share the similar iterative procedure. Actually, such iterative algorithm can be generalize to solve a broad family of tensor version of traditional vector based learning algorithms.

Suppose the linear classifier in vector space is $g(\mathbf{x}) = \mathbf{w}^T\mathbf{x} + b$ and its objective function is

$$\min V(\mathbf{w}, \mathbf{x}_i, y_i, i = 1, \cdots, m) \tag{18}$$

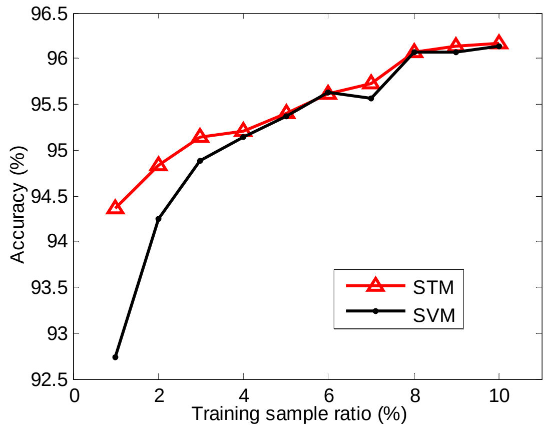

Figure 3: Classification accuracy with respect to the training sample size of STM and SVM on BREAST-CANCER dataset. As can be seen, when the training set is small (1%, 2% and 3%), STM outperforms SVM. As the number of training samples increases, STM tends to get same results with SVM.

The corresponding linear classifier in tensor space is $f(\mathbf{X}) = \mathbf{u}^T\mathbf{X}\mathbf{v} + b$ with objective function

$$\min T(\mathbf{u}, \mathbf{v}, \mathbf{X}_i, y_i, i = 1, \cdots, m) \tag{19}$$

The algorithmic procedure for solving the optimization problem (19) is formally stated below:

1. **Initialization:** Let $\mathbf{u} = (1, \cdots, 1)^T$.

2. **Computing v:** Let $\mathbf{x}_i = \mathbf{X}_i^T\mathbf{u}$. The tensor classifier can be rewritten as

$$f(\mathbf{X}) = \mathbf{u}^T\mathbf{X}\mathbf{v} + b = \mathbf{x}^T\mathbf{v} + b = g'(\mathbf{x})$$

   Thus, the optimization problem of (19) can be rewritten as the optimization problem in vector space:

$$\min V'(\mathbf{v}, \mathbf{x}_i, y_i, i = 1, \cdots, m) \tag{20}$$

   Note: Any computational method for solving (18) can also be used here.

3. **Computing u:** Once $\mathbf{v}$ is obtained, let $\tilde{\mathbf{x}}_i = X_i\mathbf{v}$. Similarly, The tensor classifier can be rewritten as

$$f(\mathbf{X}) = \mathbf{u}^T\mathbf{X}\mathbf{v} + b = \mathbf{u}^T\tilde{\mathbf{x}} + b = g''(\tilde{\mathbf{x}})$$

   Thus, $\mathbf{u}$ can be computed by solving the same vector space optimization problem:

$$\min V''(\mathbf{u}, \tilde{\mathbf{x}}_i, y_i, i = 1, \cdots, m) \tag{21}$$

   Note: As above, Any computational method for solving (18) can also be used to solve (21).

4. **Iteratively computing u and v:** By step 2 and 3, we can iteratively compute $\mathbf{u}$ and $\mathbf{v}$ until they tend to converge.

   Note: As long as the solution of optimization problem (18) is globally optimum, the iterative procedure described here will converge. The convergence proof will be similar to the proof we given in Section 3.3.

6 Conclusions

In this paper we have introduced a tensor framework for data representation and classification. In particular, we have proposed two new classification algorithms called **Support Tensor Machines** (STM) and **Tensor Least Square** (TLS) for learning a linear classifier in tensor space. Our experimental results on 6 databases from UCI Repository demonstrate that tensor based classifiers are especially suitable for small sample cases. This is due to the fact that the number of parameters estimated by a tensor classifier is much less than that estimated by a traditional vector classifier.

There are several interesting problems that we are going to explore in the future work:

1. In this paper, we empirically construct the tensor. The better ways of converting a data vector to a data tensor with theoretical guarantee need to be studied.

2. Both STM and TLS are linear methods. Thus, they fail to discover the nonlinear structure of the data space. It remains unclear how to generalized our algorithms to nonlinear case. A possible way of nonlinear generalization is to use kernel techniques.

3. In this paper, we use an iterative computational method for solving the optimization problems of STM and TLS. We expect that there exist more efficient computational methods.

References

Burges, C. J. C. (1998). A tutorial on support vector machines for pattern recognition. *Data Mining and Knowledge Discovery*, *2*, 121–167.

Cai, D., He, X., & Han, J. (2005). *Subspace learning based on tensor analysis* (Technical Report). Computer Science Department, UIUC, UIUCDCS-R-2005-2572.

Chang, C.-C., & Lin, C.-J. (2001). *LIBSVM: a library for support vector machines.* Software available at `http://www.csie.ntu.edu.tw/~cjlin/libsvm`.

Duda, R. O., Hart, P. E., & Stork, D. G. (2000). *Pattern classification.* Hoboken, NJ: Wiley-Interscience. 2nd edition.

Fletcher, R. (1987). *Practical methods of optimization.* John Wiley and Sons. 2nd edition edition.

Graf, H. P., Cosatto, E., Bottou, L., Dourdanovic, I., & Vapnik, V. (2004). Parallel support vector machines: The cascade SVM. *Advances in Neural Information Processing Systems 17*.

Hastie, T., Tibshirani, R., & Friedman, J. H. (2001). *The elements of statistical learning*. Springer-Verlag.

He, X., Cai, D., & Niyogi, P. (2005). Tensor subspace analysis. *Advances in Neural Information Processing Systems 18*.

Joachims, T. (1998). Text categorization with support vector machines: Learning with many relevant features. *ECML'98*.

Lee, J. M. (2002). *Introduction to smooth manifolds*. Springer-Verlag New York.

Platt, J. C. (1998). Using sparseness and analytic QP to speed training of support vector machines. *Advances in Neural Information Processing Systems 11*.

Platt, J. C. (1999). *Fast training of support vector machines using sequential minimal optimization*, 185–208. Cambridge, MA, USA: MIT Press.

Ronfard, R., Schmid, C., & Triggs, B. (2002). Learning to parse pictures of people. *ECCV'02*.

Shashua, A., & Hazan, T. (2005). Non-negative tensor factorization with applications to statistics and computer vision. *Proc. 2005 Int. Conf. Machine Learning (ICML'05)*.

Vapnik, V. N. (1982). *Estimation of dependences based on empirical data*. Springer-Verlag.

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

Vasilescu, M. A. O., & Terzopoulos, D. (2003). Multilinear subspace analysis for image ensembles. *IEEE Conference on Computer Vision and Pattern Recognition*.

Ye, J., Janardan, R., & Li, Q. (2004). Two-dimensional linear discriminant analysis. *Advances in Neural Information Processing Systems 17*.