Learning Predictors from Multidimensional Data with Tensor Factorizations

Soo Min Kwon, Anand D. Sarwate[*]

Abstract

Statistical machine learning algorithms often involve learning a linear relationship between dependent and independent variables. This relationship is modeled as a vector of numerical values, commonly referred to as weights or predictors. These weights allow us to make predictions, and the quality of these weights influence the accuracy of our predictions. However, when the dependent variable inherently possesses a more complex, multidimensional structure, it becomes increasingly difficult to model the relationship simply through a vector. In this paper, we address this issue by investigating machine learning classification algorithms with multidimensional (tensor) structure. By imposing tensor factorizations on the predictors, we can better model the relationship, as the predictors would take the form of the data in question. We empirically show that our approach works more efficiently than the traditional machine learning method when the data possesses both an exact and an approximate tensor structure. Additionally, we show that estimating predictors with these factorizations also allow us to solve for fewer parameters, making computation more feasible for multidimensional data.

1 Introduction

Machine learning classification algorithms are widely used in many applications, examples including fraud and spam detection. The objective of these algorithms is to model a linear relationship between the independent (e.g. card transactions, amount spent) and dependent (e.g. fraud or not fraud) variables. This relationship is generally modeled by finding a hyperplane that best separates the two classes of data, as shown in Figure 1. The hyperplane is constructed of weights and biases, which can simply be interpreted as the slope and intercept, respectively. One can solve or estimate these values by finding the parameters that most accurately describes the observed data points.

[*]The authors are with the Department of Electrical and Computer Engineering, Rutgers, The State University of New Jersey, New Brunswick, NJ 08854, `smk330@scarletmail.rutgers.edu`, `ads221@soe.rutgers.edu`. Code available at `https://github.com/soominkwon/Machine-Learning-with-Tensor-Factorizations`

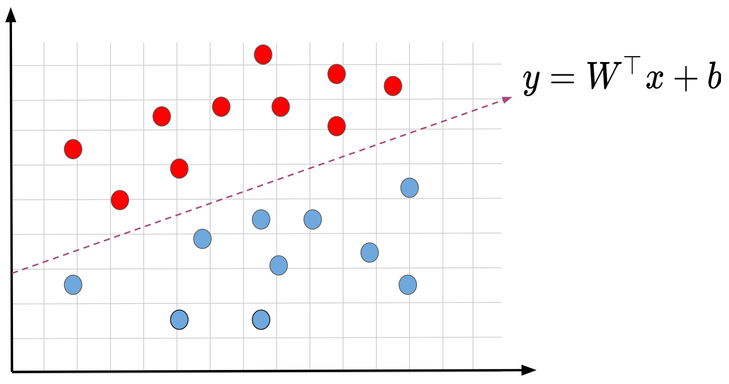

Figure 1: Visualization of learning a line (or hyperplane in higher dimensional space) that best separates two classes of data. Machine learning algorithms estimate these weights, W, and bias, b, through empirical risk minimization.

In machine learning, we solve for these parameters (or predictors) through empirical risk minimization (ERM). The ERM framework tries to estimate the parameters that minimizes the "risk" or error of a loss function between the true and computed predictors given data. The minimization of this loss function measures the "closeness" of the predictors, where a smaller objective function value would account for a more accurate model. There are many different machine learning classification algorithms, and each algorithm has a different loss function. However, since these loss functions try to model a *linear* relationship, this implicitly requires our data to be vectorized. If our data samples were to be multidimensional, vectorization can make estimation of accurate predictors much more challenging. For example, consider a different scenario in which one would like to make movie recommendations for a user given the number of movies watched in a certain genre. This data can easily be stored in the form of a matrix, where the rows represent each user and the columns represent the movie genre. However, what happens when a user's movie preferences change over time? As shown in Figure 2, we can capture this third variable (and many others) by modelling the observations in the form of a tensor, as it matches the structure of the data. Clearly, the structure of this tensor is significant for accurate data analysis. If the orderings of the movies watched were swapped for two given users, incorrect recommendations could be made. Vectorizing this data does not account for these types of structures, making inference much more challenging.

There are many modern applications that involve analyzing data with intrinsically many more dimensions, including medical imaging [1, 2], image processing [3, 4], and seismic data analysis [5]. In most of these settings, the objective is similar to that of the traditional machine learning goal: to formulate a problem of prediction to establish an association between the tensor covariates (independent variable) and outcomes (dependent variable).

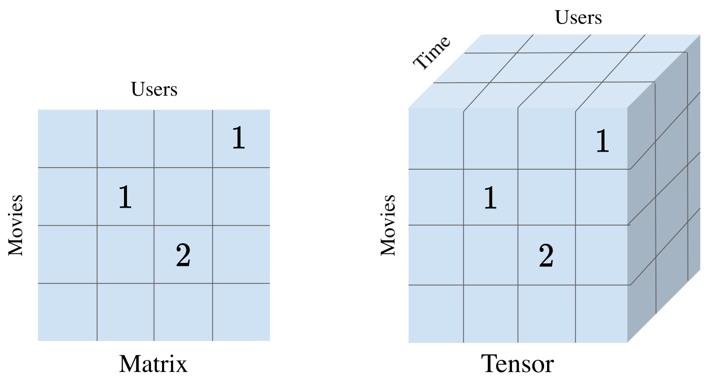

Figure 2: Example of modeling observations: left – matrix, right – tensor

However, as previously mentioned, most machine learning frameworks are formulated for vector spaces, making statistical inference challenging for tensor data. In addition, in most of these domains, the tensor data also exhibits high dimensions. For example, in medicine, tensor data samples may be of dimensions $128 \times 128 \times 128$ or greater. Naively turning this array into a vector for traditional machine learning would result in solving for $128^3 = 2,097,152$ coefficients. In this scenario, vectorization not only destroys the structure of the data, but also makes computation infeasible.

Related Works. Recently, work on tensor-based machine learning approaches uses tensor factorizations to reduce the number of coefficients to be estimated [6, 7, 8]. Specifically, tensor decompositions are imposed on the coefficients as a scheme of feature selection or dimensionality reduction. Integrating such decomposition structures solves for low rank approximations of the true predictor, rather than the vector counterpart. Zhou et al. [8] proposes a tensor regression model with additional independent variables for predicting continuous values given fMRI data. For parameter estimation, they propose a maximum likelihood (ML) approach using a block relaxation algorithm, which we adopt to formulate tensor classification models. Tan et al. [7] proposes a logistic tensor regression model with a ℓ_1 norm regularization to induce sparsity. We observe that this technique efficiently exploits structure, which motivates us to generalize and formulate more classification problems with different regularization (e.g. ℓ_2 norm), and on different datasets.

Our Contribution. In this paper, we investigate the performance of machine learning classifiers with a CANDECOMP/PARAFAC (CP) decomposition structure on the coefficients/predictors. We have seen in previous literature that these methods work efficiently for solving linear regression and logistic regression coefficients [7, 8]. We solve classification problems, namely Support Vector Machines and Logistic Regression on both synthetic and real data. The rest of this paper is organized as follows. We first discuss some tensor algebra and notation that will be used throughout this paper. Then, we propose the objective functions as well as a short analysis of the CP structured machine learning problems. We motivate and show results of our approach by fixing and solving for the true predictor. We compare the results from the tensor structured algorithm as opposed to the unstructured vector algorithm. Our contributions can be summarized as follows:

- We perform experiments to show that our structured method works more efficiently than the traditional method when the true predictor exhibits both an approximate and exact low rank structure.
- We show that our structured approach solves for fewer coefficients more efficiently than the traditional approach with a dimensionality reduction step (e.g. Principal Component Analysis).
- We develop algorithms to solve machine learning problems with decomposition of n-dimensional tensors with an alternating minimization scheme.

2 Preliminaries

We dedicate this section to discuss some of the concepts used throughout this paper. Due to the theoretical nature of this work, the technical description may require some mathematical maturity. The reader interested in the empirical findings can skip to Section 4.

For a complete introduction to tensors, see the comprehensive survey of Kolda and Bader [9] and Rabanser et al. [10]. Tensors are simply defined as multidimensional arrays, and these two terms will be used interchangeably. We will denote vectors with lower case letters (x), matrices with capital letters (X), and tensors as bold capital letters ($\mathbf{X}$).

2.1 Tensor Reorderings

Let $\mathbf{X}$ be a third-order tensor of dimensions $\mathbf{X} \in \mathbb{R}^{3\times3\times2}$ with the two frontal slices defined by $X_1, X_2 \in \mathbb{R}^{3\times3}$:

$$X_1 = \begin{bmatrix} 2 & 8 & 14 \\ 4 & 10 & 16 \\ 6 & 12 & 18 \end{bmatrix}, \quad X_2 = \begin{bmatrix} 1 & 7 & 13 \\ 3 & 9 & 15 \\ 5 & 11 & 17 \end{bmatrix}.$$

Vectorization. We can create a vector from any matrix or tensor by stacking the row or column elements into a row or column vector, respectively. For example, vectorizing the tensor $\mathbf{X}$ by its columns would yield the following column vector:

$$\text{vec}(\mathbf{X}) = \begin{bmatrix} 2 \\ 4 \\ \vdots \\ 15 \\ 17 \end{bmatrix},$$

where we stack the columns from the first frontal slice, X_1 and the second frontal slice, X_2. The dimensions of the resulting vector would be $x \in \mathbb{R}^{18}$.

Matricization. The n-mode matricization (or unfolding) of a tensor $\mathbf{Y} \in \mathbb{R}^{a_1 \times a_2 \times \ldots \times a_N}$ is denoted as $Y_{(n)}$, where $Y_{(n)}$ has the columns of the n-mode fibers. Consider the same tensor $\mathbf{X}$ from the previous example. Then the three n-mode matricizations are the following:

$$X_{(1)} = \begin{bmatrix} 2 & 8 & 14 & 1 & 7 & 13 \\ 4 & 10 & 16 & 3 & 9 & 15 \\ 6 & 12 & 18 & 5 & 11 & 17 \end{bmatrix},$$

$$X_{(2)} = \begin{bmatrix} 2 & 4 & 6 & 1 & 3 & 5 \\ 8 & 10 & 12 & 7 & 9 & 11 \\ 14 & 16 & 18 & 13 & 15 & 17 \end{bmatrix},$$

$$X_{(3)} = \begin{bmatrix} 2 & 4 & 6 & 8 & 10 & 12 & 14 & 16 & 18 \\ 1 & 3 & 5 & 7 & 9 & 11 & 13 & 15 & 17 \end{bmatrix}.$$

One can think of matricization as a generalization of vectorization but to matrices. Since our example $\mathbf{X}$ is a third-order tensor, we have three matrices from matricization, one for each mode.

2.2 Vector & Matrix Products

Outer Product. Let a and b be two vectors of dimensions $a \in \mathbb{R}^n$ and $b \in \mathbb{R}^n$,

$$a = \begin{bmatrix} a_1 & a_2 & \ldots & a_n \end{bmatrix}, \; b = \begin{bmatrix} b_1 & b_2 & \ldots & b_n \end{bmatrix}.$$

The outer product of a and b, denoted as $a \circ b$, is a matrix of dimensions $(a \circ b) \in \mathbb{R}^{n \times n}$,

$$a \circ b = ab^\top = \begin{bmatrix} a_1 b_1 & \ldots & a_1 b_n \\ \vdots & \ddots & \vdots \\ a_n b_1 & \ldots & a_n b_n \end{bmatrix}.$$

Note that this outer product is not only limited to vectors, and can be generalized to matrices and tensors as well.

Kronecker Product. Let A and B be two matrices of dimensions $A \in \mathbb{R}^{m \times n}$ and $B \in \mathbb{R}^{j \times k}$,

$$A = \begin{bmatrix} a_{11} & \dots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{m1} & \dots & a_{mn} \end{bmatrix}, \quad B = \begin{bmatrix} b_{11} & \dots & b_{1k} \\ \vdots & \ddots & \vdots \\ b_{j1} & \dots & b_{jk} \end{bmatrix}.$$

The Kronecker product of A and B, denoted as $A \otimes B$, is a matrix of dimensions $(A \otimes B) \in \mathbb{R}^{mj \times nk}$,

$$A \otimes B = \begin{bmatrix} a_{11}B & \dots & a_{1n}B \\ \vdots & \ddots & \vdots \\ a_{m1}B & \dots & a_{mn}B \end{bmatrix}.$$

In essence, the Kronecker product is computed by multiplying every element in the first matrix, A, by the entire second matrix, B.
Khatri–Rao Product. The Khatri–Rao product is the columnwise Kronecker product. Consider two (different) matrices $A \in \mathbb{R}^{m \times n}$ and $B \in \mathbb{R}^{p \times n}$. The Khatri–Rao product of A and B, denoted as $A \odot B$, is a matrix of dimensions $(A \odot B) \in \mathbb{R}^{mp \times n}$,

$$A \odot B = \begin{bmatrix} a_1 \otimes b_1 & a_2 \otimes b_2 & \dots & a_n \otimes b_n \end{bmatrix}.$$

Here, we are taking the Kronecker product between every column vector from A and B. Note that if A and B itself were column vectors, i.e. $n = 1$, then the Khatri–Rao product is equivalent to the Kronecker product, $A \odot B = A \otimes B$.

2.3 Tensor Decomposition

Tensor decompositions are generalizations of matrix factorizations to multidimensional arrays [13]. We introduce one tensor factorization scheme that is important in understanding the setting of our algorithm. In the matricized form, we show that this factorization has useful properties to be solved with an alternating minimization scheme.

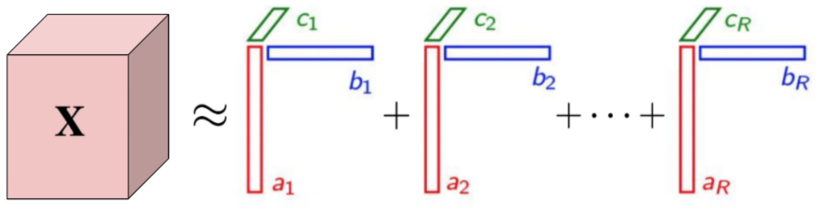

Figure 3: Graphical representation of the CANDECOMP/PARAFAC decomposition – low rank approximation of a third–order tensor

CANDECOMP/PARAFAC (CP) Decomposition. The objective of the CP decomposition is to express a tensor as the sum of component rank–one tensors, i.e. vectors, as depicted in Figure 3. For example, consider a third-order tensor $\mathbf{X} \in \mathbb{R}^{D_1 \times D_2 \times D_3}$. We can approximate this tensor as the following

$$\mathbf{X} \approx \textstyle\sum_{r=1}^{R} a_r \circ b_r \circ c_r,$$

where "$\circ$" denotes the outer product, R represents the rank (positive integer), and $a_r \in \mathbb{R}^{D_1}$, $b_r \in \mathbb{R}^{D_2}$, and $c_r \in \mathbb{R}^{D_3}$ for $r = 1, \dots, R$. We can formalize this decomposition as the following optimization problem:

$$\underset{\hat{\mathbf{X}}}{\text{minimize}} \quad ||\mathbf{X} - \hat{\mathbf{X}}||, \quad \text{subject to} \quad \hat{\mathbf{X}} = \sum_{r=1}^{R} a_r \circ b_r \circ c_r, \tag{1}$$

where $\hat{\mathbf{X}}$ would represent a low rank approximation of $\mathbf{X}$.

The factor matrices or CP factors are matrices with the rank–one tensors as entries. From the previous three-dimensional case, $A \in \mathbb{R}^{D_1 \times R}$ would be an estimated CP factor with entries

$$A = \begin{bmatrix} a_1 & a_2 & \dots & a_R \end{bmatrix}.$$

With these definitions and the products defined previously, we can formulate some useful properties for the third–order case:

$$\begin{aligned} \hat{X}_{(1)} &= (C \odot B)A^\top, \\ \hat{X}_{(2)} &= (C \odot A)B^\top, \\ \hat{X}_{(3)} &= (B \odot A)C^\top. \end{aligned} \tag{2}$$

These relationships can easily be generalized to n-mode tensors, but for the purposes of this paper, n=3 will suffice. We will show how we can use these equations for our alternating minimization algorithm in the following sections. There also are other useful tensor factorizations, such as the Tucker decomposition, which is explained in detail in the survey paper [9].

2.4 Machine Learning Optimization Problems

Many machine learning algorithms can be framed as empirical risk minimization (ERM) problems. The empirical risk is defined in terms of a risk, or loss function $\ell(\cdot)$. For linear classifiers, the loss of a linear predictor w on the data sample (x_i, y_i) can be written as $\ell(w^\top x_i, y_i)$ and the average empirical risk as $\frac{1}{n}\sum_{i=1}^{n} \ell(w^\top x_i, y_i)$. We discuss these loss functions for some common classifiers and how we can use them to solve tensor structured ERM problems.

Support Vector Machines. Consider a dataset with n samples, i.e. $\{(x_i, y_i)\}_{i=1}^{n}$, where $y_i \in \{-1, 1\}$. Support Vector Machine (SVM) or maximum margin linear classifier is a binary classifier that finds a hyperplane to best separate the data, while making minimal margin violations [11]. SVM uses a loss function called the *hinge loss* function, defined by

$$\ell(w^\top x_i, y_i) = \max[0, 1 - y_i(w^\top x_i)],$$

where w is the coefficients of the separating hyperplane. With a penalty (or regularizer), we can mathematically formulate SVM as the following ERM problem:

$$\underset{w}{\text{minimize}} \quad \frac{1}{n}\sum_{i=1}^{n} \max[0, 1 - y_i(w^\top x_i)] + \frac{\lambda}{2}||w||^2. \tag{3}$$

The regularization term, λ, is used to penalize the features, and hence weights, that do not necessarily contribute to the prediction outcome. Here, we are considering the ℓ_2 penalty, but there are other regularizers such as the ℓ_1 penalty. We use these regularization terms in our loss function to estimate a more accurate model.

Logistic Regression. Similarly, consider a dataset with n samples, i.e. $\{(x_i, y_i)\}_{i=1}^n$, where $y_i \in \{-1, 1\}$. The objective of Logistic Regression (LOGIT) is the same as SVM, with a different loss function called the *logistic loss* function, defined by

$$\ell(w^\top x_i, y_i) = \frac{1}{1 + \exp\left(-y_i(w^\top x_i)\right)}.$$

The logistic loss function takes the form of the sigmoid function. With a regularization term, we can define Logistic Regression as the following ERM problem:

$$\underset{w}{\text{minimize}} \quad \frac{1}{n}\sum_{i=1}^{n} \log(1 + \exp\left(-y_i(w^\top x_i)\right)) + \lambda||w||^2. \tag{4}$$

We only introduce the objective function of these two classifiers, as we will construct the CP structured algorithm with these functions in the following section. Note that we do not include the bias term in our hyperplane equation, as it can be modeled in $w^\top$ as a column vector.

3 Problem Formulation

In this section, we propose our tensor-based classifiers in the form of an ERM framework. In general, we structure our linear predictors ($w^\top$) to admit a CP decomposition, in which we can reconstruct to make classifications. We also discuss the metrics that we will be investigating to evaluate the performance of our models.

3.1 CANDECOMP/PARAFAC Structured Classifiers

Support Vector Machines. Consider a dataset $\{(\mathbf{X}_i, y_i)\}_{i=1}^n$, where $\mathbf{X}_i \in \mathbb{R}^{D_1 \times \cdots \times D_N}$ denotes a tensor data sample with $y_i \in \{-1, 1\}$. By imposing the constraints from (1) onto the predictors of (3), we can formulate the following optimization problem:

$$\underset{W_1,\ldots,W_N}{\text{minimize}} \quad \frac{1}{n}\sum_{i=1}^{n} \max[0, 1 - y_i(\langle \sum_{r=1}^{R} W_1^{(r)} \circ W_2^{(r)} \circ \ldots \circ W_N^{(r)}, \mathbf{X}_i\rangle)]. \tag{5}$$

The traditional ERM problem for SVM in (3) solves for one vector predictor of dimensions $w \in \mathbb{R}^{D_1 \ldots D_N}$. The problem in (5), which we call "CP-SVM", solves for N matrix-valued predictors of dimensions $W_i \in \mathbb{R}^{D_i \times R}$, for $i = 1, \ldots, N$. As a concrete example, let each tensor sample be dimensions $\mathbf{X}_i \in \mathbb{R}^{5\times 5\times 5}$ and $R = 3$. The traditional problem would solve for $5\times5\times5 = 125$ coefficients, whereas the structured problem would solve for $3\times(5\times3) = 45$ coefficients. As the dimensions increase, the structured problem substantially reduces the number of parameters/coefficients to be estimated.

Logistic Regression. Similarly, consider a dataset $\{(\mathbf{X}_i, y_i)\}_{i=1}^n$, where $\mathbf{X}_i \in \mathbb{R}^{D_1 \times \ldots \times D_N}$ denotes a tensor data sample with $y_i \in \{-1, 1\}$. By imposing the constraints from (1) onto the predictors of (4), we can solve the following ERM problem:

$$\underset{W_1,\ldots,W_N}{\text{minimize}} \quad \frac{1}{n}\sum_{i=1}^{n} \log(1 + \exp(-y_i(\langle \sum_{r=1}^{R} W_1^{(r)} \circ W_2^{(r)} \circ \ldots \circ W_N^{(r)}, \mathbf{X}_i\rangle))) \tag{6}$$

This new framework, which we call "'CP-LOGIT', solves for fewer parameters, similar to CP-SVM.

In practice, we solve for the weights using numerical optimization methods such as gradient descent. However, solving for the weights in this new CP-structured paradigm is a non-trivial task. In order to solve for the coefficients in (5) and (6), we adopt an alternating minimization algorithm similar to the block relaxation algorithm proposed in Zhou et al. [8]. At each iteration, we update block W_i, while keeping the rest of the blocks fixed. To see this, when updating $W_i \in \mathbb{R}^{D_i \times R}$, we can rewrite the inner product in (5) and (6) with the properties mentioned in (2):

Algorithm 1 CP Alternating Minimization (Zhou et al. [8])

```
Require: Dataset $\{(\mathbf{X}_i, y_i)\}_{i=1}^n$ with $\mathbf{X} \in \mathbb{R}^{D_1 \times \ldots \times D_N}$, $y_i \in \{-1, 1\}$ and $R$
1: Initialize: $A_i \in \mathbb{R}^{D_i \times R}$ for $i = 1, \ldots, N$
2: repeat
3: for $i = 1, \ldots, N$ do
4:     $A_i^{(t+1)} = \underset{A_i}{\text{argmin}} \quad \ell(\mathbf{X}, y, A_1^{(t+1)}, \ldots, A_{i-1}^{(t+1)}, A_i, A_{i+1}^{(t)}, \ldots, A_N^{(t)}) + \lambda||A_i||^2$
5: end for
6: until $\ell(\theta^{(t+1)}) - \ell(\theta^{(t)}) < \epsilon$
```

$$\langle \sum_{r=1}^{R} W_1^{(r)} \circ \ldots \circ W_N^{(r)}, \mathbf{X}_i \rangle = \langle W_i, \mathbf{X}_{(i)}(W_D \odot \ldots \odot W_{i+1} \odot W_{i-1} \odot \ldots \odot W_1) \rangle.$$

This alternating minimization algorithm is summarized in Algorithm 1, in which $\ell(\cdot)$ represents the ERM problem to be minimized, θ represents a collection of all the parameters, and λ is the regularization parameter. The parameter λ was tuned by hand, but can also be determined through cross validation. To understand the CP structured algorithm, consider the loss function in (5) with $N = 3$. When updating W_2, we rewrite the inner product $\langle \sum_{r=1}^{R} W_1^{(r)} \circ W_2^{(r)} \circ W_3^{(r)}, \mathbf{X}_i \rangle$ as $\langle W_2, \mathbf{X}_{(2)}(W_3 \odot W_1) \rangle$. Note that this equation follows from the property of tensor algebra as shown in (2). We perform this algorithm for all the factor matrices until the stopping criteria is met.

The alternating minimization algorithm is useful for several reasons. First, in practice, this algorithm almost always converges to at least a local minimum [12, 13, 8]. To find the best solution, the algorithm can be ran several times with different initial factor matrices. Second, the low rank optimization problem over the factor matrices is non-convex [14]. Thus, this problem becomes difficult to solve using common unconstrained solvers, such as gradient descent. In literature, there are two ways to handle the non-convexity of this optimization problem. One way is to relax the rank constraint by adding a convex regularization term that induces low rank (e.g. trace norm, nuclear norm) [15, 16]. The other solution is to employ this alternating minimization algorithm, as the optimization over one matrix, while holding the others fixed is convex. We chose to explore this procedure following Zhou et al. [8], as the algorithm is straightforward to implement using statistical software such as MATLAB or Python.

3.2 Performance Metrics

We evaluate the performance of our models using several measures with different sample sizes. The following four metrics help determine the measure of "closeness" between the true and estimated predictors.

1. The **Mean Squared Error (MSE)** for n data samples and true predictor W is computed as

$$\text{MSE} = \frac{1}{n} ||W - \hat{W}||^2 \tag{7}$$

   where $\hat{W}$ is the estimated predictor from solving the ERM problem.

2. The **cosine distance** (or similarity) [17] for true predictor W is computed as

$$\cos(\theta) = \frac{\langle W \cdot \hat{W} \rangle}{||W|| \cdot ||\hat{W}||} \tag{8}$$

   where $\hat{W}$ is the reconstructed predictor from solving the ERM problem. Mathematically, the cosine similarity measures the cosine of the angle between two vectors projected in a n-dimensional space. As the angle, θ, between the two vectors become smaller, the cosine similarity will approach a value of 1. As the angles become farther apart (perpendicular), the cosine similarity will approach a value of 0.

3. The **reconstruction error** for true predictor W and estimated tensor predictor $\hat{W}$ is defined as

$$\text{Reconstruction Error} = \frac{||W - \hat{W}||_\text{F}}{||W||_\text{F}}, \tag{9}$$

where $|| \cdot ||_\text{F}$ denotes the Frobenius norm, a matrix generalization of the ℓ_2 norm .

4. The **classification accuracy** for n test samples is simply defined as the following:

$$\text{Accuracy} = \frac{\text{\# of correct predictions}}{\text{total \# of predictions made } (n)}. \tag{10}$$

After solving for $\hat{W}$, we make predictions on test data and compare $\hat{y}_i$ to the true y_i. Before comparing the labels, we use the sign function to quantize our values to $\hat{y}_i \in \{-1, 1\}$.

4 Experiments

We used two types of data for our experiments: synthetic data and the Modified National Institute of Standards and Technology (MNIST) database [18]. The MNIST database is a benchmark dataset used widely in machine learning that consists of $60,000$ samples of handwritten digits from 0 to 9. The objective of both experiments is to compare the performance between the CP-structured algorithms and the traditional algorithms, which were implemented using software packages TensorLy [19] and SciPy [20]. For all experiments, we use a Python environment on a Macbook Pro with 2.2 GHz Intel Core i7 and 16 GB RAM.

4.1 Synthetic Data

For synthetic data, we generated univariate y_i responses with different sample sizes according to the following model:

$$y_i = \langle X_i, W \rangle + \epsilon_i, \tag{11}$$

where X_i is drawn independently and identically distributed (iid) from $\mathcal{N}(0, 1)$, ϵ is a noise term drawn iid from $\mathcal{N}(0, 1)$, and W is the fixed predictor as shown in Figure 4. The objective was to observe if our models defined in (5) and (6) can identify the true signal W given (X_i, y_i).

Performance Comparison. To measure the "closeness" and classification accuracy between the true model and the predicted model, we use performance metrics defined in (7),

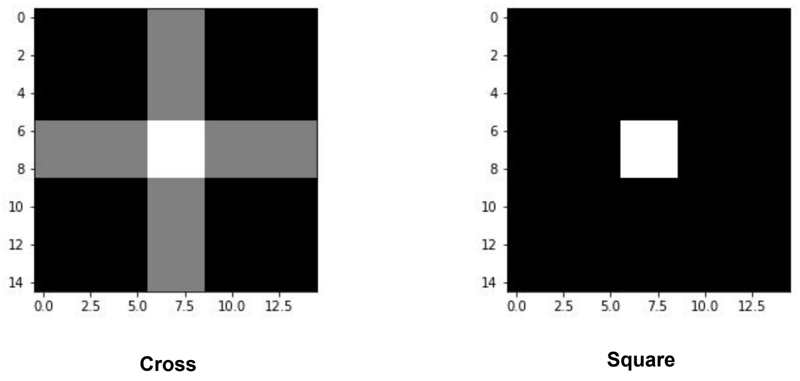

Figure 4: Two 15×15 images used as true predictors W to generate synthetic data

(8), (9), and (10). We compute these metrics at different sample sizes and show that as the number of samples increases, the performance of the traditional vector approach converges to the performance of the CP structured model. These results are displayed in Figures 5 and 6. In Figure 5, we can visually see that the predictors from our method solves for the true predictors more accurately. For example, in the case of $n = 500$ from row 1, the "cross" figure is more accurately portrayed using the CP method (right) than the traditional method (middle). This would allow us to make more accurate predictions, as the estimated weights more closely follow the true weights. In Figure 6, we can see that the MSE for both algorithms is relatively the same throughout *all* sample sizes. For the cosine distance, we can see that the CP structured algorithm approaches a value of 1 very quickly, which indicates that there is a strong similarity between the estimated and the true coefficients. The reconstruction error and classification accuracy both generally have gaps in the figures, but lessen as the sample sizes increase. We can conclude that these results depend on the sample size, as more samples can decrease the number of hyperplanes that separates the data, predicting coefficients closer to the true model. Based on the trends of the graphs in Figure 6, we also hypothesize that if the variance of the noise (ϵ) distribution was higher, the CP structured algorithms would also perform better than the traditional method.

Results with PCA. The CP structured algorithm significantly reduces the number of predictors to be estimated. To solve for less coefficients using the traditional method, we can perform Principal Component Analysis (PCA) on the dataset before using the algorithm. We use PCA on X with an energy capture of 95%, which reduces the number of coefficients from 225 to 189. However, even with this minimal reduction, we can see in Figure 6 that

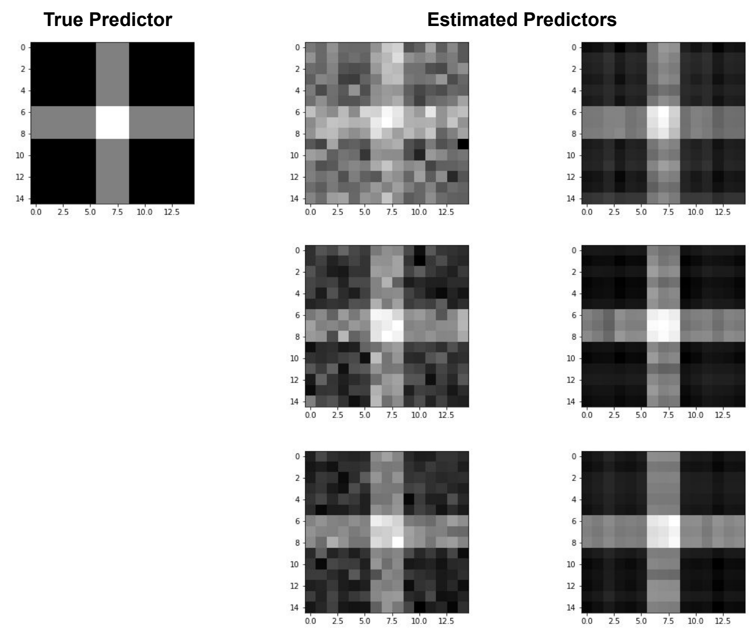

Figure 5: Reconstructed predictors from both algorithms: left – true predictor, middle – reconstructed predictor from traditional method with increasing sample sizes ($n = 500, 1000, 1500$), right – reconstructed predictor from CP-structured Logistic Regression with increasing sample sizes ($n = 500, 1000, 1500$)

there is a notable decrease in performance throughout most metrics. The MSE seems unaffected, but the other three metrics start to see a gap between the traditional method with no PCA and the CP structured algorithm. A possible explanation for this phenomenon is that PCA does not capture tensor data efficiently in lower dimensional space. If we were to decrease the energy capture, the gap in performance would grow larger even for a bigger sample size. We predict that as the dimensions of the data increases, PCA would not be an efficient feature learning method for parameter reduction, favoring the CP structured methods.

4.2 MNIST Dataset

The objective of the MNIST dataset experiment was to observe which algorithm would be more efficient to use when the true predictor exhibited an "approximate" low rank structure. In the previous experiment, the two images used as the true predictor had an exact low rank structure, as it could easily be computed through an outer product of two matrices. Similar to the synthetic data setup, we generated univariate y_i responses with sample size

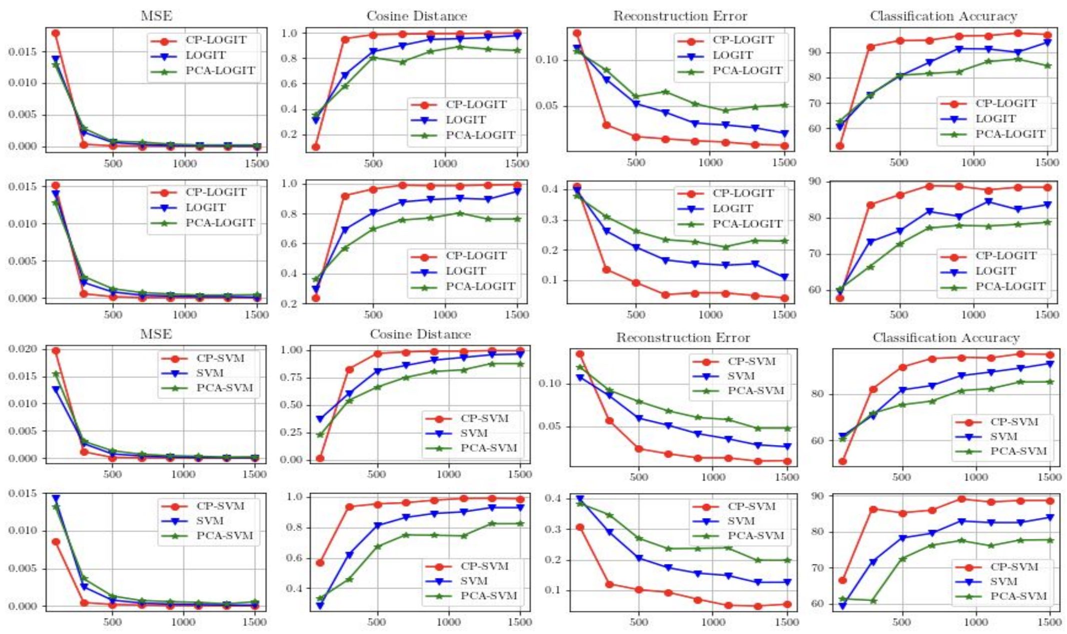

Figure 6: Variation of performance (y-axis) with different sample sizes (x-axis) for SVM and LOGIT. Columns 1-4 represent plots for the MSE, Cosine Distance, Reconstruction Error, and Classification Accuracy, respectively. Row 1, 2: Performance metrics for LOGIT with predictors as cross and square, respectively. Row 3, 4: Performance metrics for SVM with predictors as cross and square, respectively. Predictors of cross and square is as shown in Figure 4

$n = 750$ with the model defined in (11). However, for the true predictor, W, we chose a “1” from the MNIST dataset, as it exhibits “approximate” low rank structure. We compared the CP-structued algorithms to the traditional algorithms using different rank values. These results are shown in Table 1.

Performance Comparison. We use the same performance metrics defined for the previous experiment and display the results in Table 1. From this table, we can conclude that both CP structured algorithms gave favorable results when the CP rank was 2. This shows that we can approximate a ”1” from the MNIST dataset with matrices of rank 2. However in all cases from rank 1 to 3, the structured algorithms gave more favorable results. This proves to show that if the true predictor exhibits an approximate low rank structure, it may be beneficial to use the structured algorithms for classification.

Method	MSE	Cos Distance	Reconstruction Error
SVM	0.00128	0.51832	0.00053
CP-SVM (R=1)	0.00088	0.66727	0.00044
CP-SVM (R=2)	**0.00026**	**0.90259**	**0.00024**
CP-SVM (R=3)	0.00039	0.85099	0.00029
LOGIT	0.00120	0.54759	0.00051
CP-LOGIT (R=1)	0.00089	0.66483	0.00044
CP-LOGIT (R=2)	**0.00028**	**0.89590**	**0.00024**
CP-LOGIT (R=3)	0.00033	0.87807	0.00026

Table 1: Performance metrics between the traditional and structured algorithms for the MNIST dataset experiment. The bolded values represent the "best" performance throughout each method, where R represents the rank of the CP structured algorithm for each experiment.

5 Conclusion

In this paper, we explored tensor-based classification models using a tensor decomposition. We proposed two algorithms that imposed a CANDECOMP/PARAFAC factorization structure on the predictors of traditional classification algorithms, namely Support Vector Machines and Logistic Regression. Imposing these structures on traditional algorithms allowed us to exploit the structure of the data, while solving for fewer parameters. We showed with different performance metrics that our proposed method increased accuracy and overall solved a more accurate estimation of the weights. The experiments showed that the CP algorithm performed best when the true predictor had either an approximate or an exact low rank structure. We also showed that solving for fewer parameters using PCA compromised the performance of the traditional method. We predict that PCA would not generalize well to data with multidimensional structure, favoring the CP structured algorithms. However, we believe that it would be interesting if one could show if and when PCA could be better than using CP structure. This could possibly be a case when the data in question is known to be linear, as PCA is a linear feature learning method. An example could be using structured data for prediction where it is known a priori that the features have a linear relationship. However, due to time constraints, we were not able to explore this possibility in detail. We also think it would be interesting to test these algorithms on more datasets. In addition, we believe an exciting direction for future research could be to exploit tensor decompositions on other applications such as deep learning. However, it is not clear how one could approach this problem, as deep learning algorithms have non-convex loss functions. We leave this up to the audience to explore for future exploration.

References

[1] R. Luis-García, C.-F. Westin, and C. Alberola-López, "Gaussian mixtures on tensor fields for segmentation: Applications to medical imaging," *Computerized medical imaging and graphics : the official journal of the Computerized Medical Imaging Society*, vol. 35, pp. 16–30, 10 2010.

[2] L. O'Donnell and C.-F. Westin, "An introduction to diffusion tensor image analysis," *Neurosurgery clinics of North America*, vol. 22, pp. 185–96, viii, 04 2011.

[3] W. Guo, I. Kotsia, and I. Patras, "Tensor learning for regression," *IEEE Transactions on Image Processing*, vol. 21, pp. 816 – 827, 03 2012.

[4] C. Jia, Y. Kong, Z. Ding, and Y. Fu, "Latent tensor transfer learning for rgb-d action recognition," in *MM '14*, 2014.

[5] N. Kreimer and M. Sacchi, "Tensor completion via nuclear norm minimization for 5d seismic data reconstruction," vol. 78, 09 2012, pp. 1–5.

[6] X. Li, H. Zhou, and L. Li, "Tucker tensor regression and neuroimaging analysis," 2013.

[7] X. Tan, Y. Zhang, S. Tang, J. Shao, F. Wu, and Y. Zhuang, "Logistic tensor regression for classification," 10 2012, pp. 573–581.

[8] H. Zhou, L. Li, and H. Zhu, "Tensor regression with applications in neuroimaging data analysis," *Journal of the American Statistical Association*, vol. 108, no. 502, p. 540–552, Jun 2013. [Online]. Available: http://dx.doi.org/10.1080/01621459.2013.776499

[9] T. Kolda and B. Bader, "Tensor decompositions and applications," *SIAM Review*, vol. 51, pp. 455–500, 08 2009.

[10] S. Rabanser, O. Shchur, and S. Günnemann, "Introduction to tensor decompositions and their applications in machine learning," 11 2017.

[11] G. James, D. Witten, T. Hastie, and R. Tibshirani, *An Introduction to Statistical Learning: With Applications in R.* Springer Publishing Company, Incorporated, 2014.

[12] J. Bezdek and R. Hathaway, "Convergence of alternating optimization," *Neural, Parallel & Scientific Computations*, vol. 11, pp. 351–368, 12 2003.

[13] S. Wright, "Coordinate descent algorithms," *Mathematical Programming*, vol. 151, 02 2015.

[14] E. Candès and B. Recht, "Exact matrix completion via convex optimization," *Communications of the ACM*, vol. 9, pp. 717–772, 11 2008.

[15] R. Tomioka and T. Suzuki, “Convex tensor decomposition via structured schatten norm regularization,” *Advances in Neural Information Processing Systems*, 03 2013.

[16] K. Wimalawarne, R. Tomioka, and M. Sugiyama, “Theoretical and experimental analyses of tensor-based regression and classification,” *Neural computation*, vol. 28, 09 2015.

[17] H. Nguyen and L. Bai, “Cosine similarity metric learning for face verification,” *ACCV*, vol. 6493, pp. 709–720, 11 2010.

[18] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, “Gradient-based learning applied to document recognition,” *Proceedings of the IEEE*, vol. 86, pp. 2278 – 2324, 12 1998.

[19] J. Kossaifi, Y. Panagakis, A. Anandkumar, and M. Pantic, “Tensorly: Tensor learning in python,” *Journal of Machine Learning Research*, vol. 20, pp. 1–, 02 2019.

[20] P. Virtanen, R. Gommers, T. E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, J. Bright, S. J. van der Walt, M. Brett, J. Wilson, K. Jarrod Millman, N. Mayorov, A. R. J. Nelson, E. Jones, R. Kern, E. Larson, C. Carey, İ. Polat, Y. Feng, E. W. Moore, J. Vand erPlas, D. Laxalde, J. Perktold, R. Cimrman, I. Henriksen, E. A. Quintero, C. R. Harris, A. M. Archibald, A. H. Ribeiro, F. Pedregosa, P. van Mulbregt, and Contributors, “SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python,” *Nature Methods*, vol. 17, pp. 261–272, 2020.

[21] Y. Li, H. Zhu, D. Shen, W. Lin, J. Gilmore, and J. Ibrahim, “Multiscale adaptive regression models for neuroimaging data,” *Journal of the Royal Statistical Society. Series B, Statistical methodology*, vol. 73, pp. 559–578, 09 2011.