Homework 1: Eigendigits

Thomas Wei
ThomasW219@gmail.com

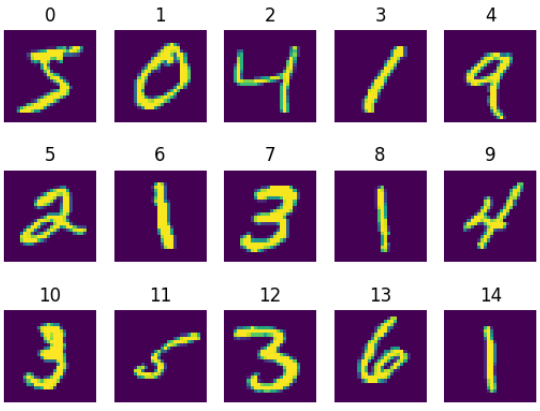
***Abstract*—For many machine learning problems, expression in the standard basis is often not the most natural representation of data. Such representations frequently hide useful correlations in the data and are often higher dimensional than necessary. Principal Component Analysis (PCA) remedies both problems by transforming the data into a new coordinate space made up of the eigenvectors of the sample covariance matrix of the data.**

I. Introduction

For many machine learning problems, expression in the standard basis is often not the most natural representation of data. Such representations frequently hide useful correlations in the data and are often higher dimensional than necessary. Principal Component Analysis (PCA) remedies both issues by transforming the data into a new coordinate space made up of the eigenvectors of the sample covariance matrix of the data. Using this new eigenvector basis allows us to look at the data as a vector of uncorrelated features which are conveniently assigned an importance as well (the magnitude of the corresponding eigenvector). By discarding the coordinates of eigenvectors with eigenvalues close to 0, we can reduce the dimensionality of the problem as well.

In this paper, we perform PCA on the MNIST data set to investigate its usefulness in aiding classification. We will compare the performance of a number of different classifiers on the images in their pixel representation and in their eigenvector representation. In addition, we will also see how discarding the coordinates of eigenvectors with small eigenvalues affects each classifier.

Fig. 1. Images in the MNIST data set



II. Method

A. Principal Component Analysis

We begin the PCA process by first constructing the sample covariance matrix of the data. Assume our data is d dimensional and we have n samples in a $n \times d$ matrix $\mathbf{X}$. The MNIST data set is composed of 28×28 images so we flatten the images into 784-dimensional vectors and use them as rows in the matrix $\mathbf{X}$ Let row i of $\mathbf{X}$ be $\mathbf{x}_i$. Then our sample covariance matrix can be given by:

$$\mathbf{\Sigma} = \frac{1}{n}\sum_{i=1}^{n} \mathbf{x}_i^T \mathbf{x}_i \tag{1}$$

Once we've calculated the sample covariance, we want to find the eigenvectors which will give us the principal directions in which our data is data is distributed in $\mathbb{R}^d$. The eigenvalues will also give the variance of our data along the axis specified by the corresponding eigenvector. The eigenvectors $\{\lambda_i\}$ and eigenvectors $\{\eta_i\}$ can be found by solving the equation:

$$\mathbf{\Sigma}\eta = \lambda\eta \tag{2}$$

Our eigenvectors will be $d \times 1$ column vectors. In practice, we find solutions to (2) using the `numpy.linalg.eig()` function. Since covariance matrices are symmetric and positive definite (except for rare cases) our d eigenvectors will be orthogonal and form a basis for $\mathbb{R}^d$. We will also scale our eigenvectors so they are unit norm.

After attaining the eigenvectors and eigenvalues, we want to transform our data from the coordinate system defined by the standard basis to the one defined by our eigenvectors. In other words, we want to find the coordinates of our data in terms of our eigenvectors.

$$\mathbf{x}_i^T = \eta_1 c_1 + \eta_2 c_2 + \cdots + \eta_d c_d$$

$$\mathbf{x}_i^T = [\eta_1\ \eta_2\ \ldots\ \eta_d] \begin{bmatrix} c_1 \\ c_2 \\ \vdots \\ c_d \end{bmatrix}$$

$$\mathbf{x}_i^T = \mathbf{W}\mathbf{c}$$

We can see from the above derivation that the transformation that maps the space defined by eigenvectors to standard basis is simply the matrix $\mathbf{W}$ who's columns are the eigenvectors we found. Since we found our eigenvectors to be orthogonal, we know that $\mathbf{W}$ is full rank and thus invertible. Furthermore, since $\mathbf{W}$ is an orthogonal matrix, its inverse is simply its transpose. So, to recover the coordinates of our data in

eigenspace from their coordinates in standard space, we simply solve:

$$\mathbf{c} = \mathbf{W}^{-1}\mathbf{x}_i^T = \mathbf{W}^T\mathbf{x}_i^T \tag{3}$$

To perform the same transformation on the entire data matrix X, we can solve:

$$\mathbf{X}' = (\mathbf{W}^T\mathbf{X}^T)^T = \mathbf{X}\mathbf{W} \tag{4}$$

B. Application to classification

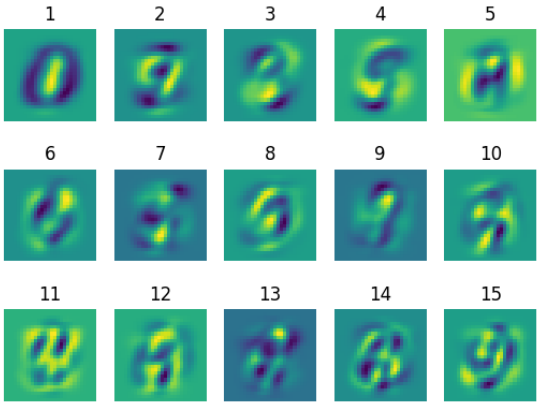
Using PCA, we can create a transformed data matrix and apply the same classification techniques that we would on the original data. In this paper, we test the performance of a k-nearest neighbor classifier on both the transformed data and original. We also do the same for multinomial logistic regression.

III. Results

A. PCA visualization and interpretation

Once we perform PCA on the MNIST and extract the eigenvectors, it is possible to reshape the eigenvectors back into images to get a better idea of how the images are distributed in eigenspace.

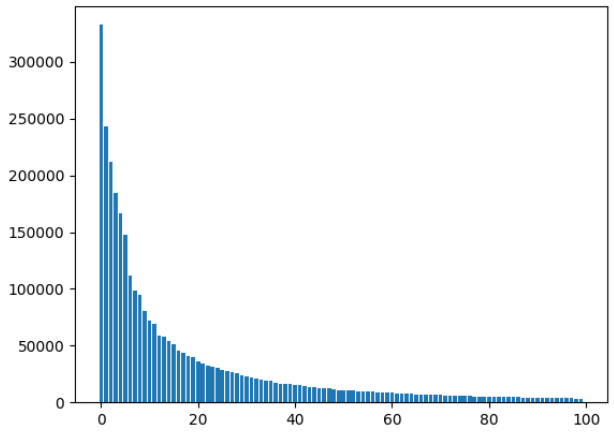
Fig. 2. The 15 eigenvectors with the highest magnitude eigenvalues



In addition, the relative magnitude of the eigenvalues (seen in Fig. 3) indicates that most of the information in the images can be expressed in less than 100 coordinates rather than the 784 coordinates required in standard space.

We can test this by discarding the projections of the images onto eigenvectors with small eigenvalues when reconstructing the images. The reconstructions using the top 10, 50, 100, and 784 eigenvectors can be seen in Fig. 4. Even using only 10 eigenvectors, the reconstructions can still be somewhat reliably identified as the intended digit. At 100 eigenvectors, the reconstructions are nearly completely identical to the original image. When the images are reconstructed using all 784 eigenvectors, unsurprisingly, the original image is perfectly recovered.

Fig. 3. The magnitude of eigenvalues in decreasing order



B. Effects on k-nearest neighbor classification

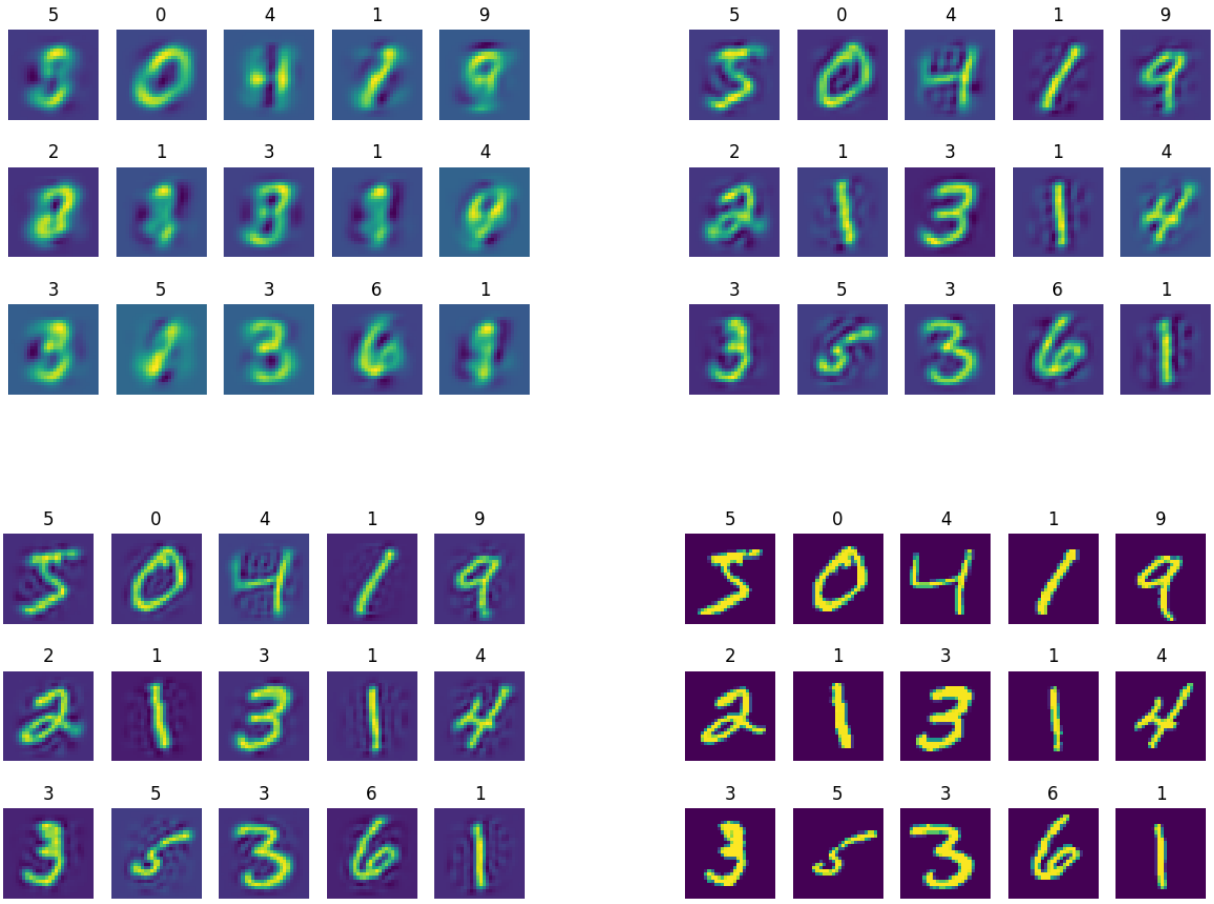
We also compared the accuracy of classifiers trained on the unmodified data set and the transformed data set. For k-nearest neighbors, we used the `sklearn.neighbors.KNeighborsClassifier` implementation and varied the number of neighbors parameter. Besides the number of neighbors, all other parameters were left to their default values. As seen in Fig. 5 the model trained on the top 100 magnitude eigenvectors had a higher test set accuracy than the model trained on raw images at every number of neighbors we tested.

With the number of nearest neighbors fixed at 5, we also tried varying the number of eigenvector coordinates used and the size of the training set. From looking at figure in Fig. 6 we can see that, among the number of dimensions that we tested (10, 20, 50, 100, 200, 500), the maximum test score was when we kept the coordinates of the 50 most significant eigenvectors. At some point between 20 and 100, adding dimensions became detrimental to the test accuracy of the model, it could be that these less significant eigenvectors represent only noise. In Fig. 7, the model that we ran used the top 100 magnitude eigenvectors and predictably its performance increased with the size of the training set. It also outperformed a model trained on the raw images with the same reduced size training set at every size we tested.

C. Effects on multinomial logistic regression

We also tested the effects of transforming the data on multinomial logistic regression. For these experiments, we used the `sklearn.linear_model.LogisticRegression` implementation. All the parameters were left to their defaults except maximum iterations was increased from 100 to 10,000. We varied the number of coordinates of eigenvectors we allowed the model to use and compared its performance to multinomial logistic regression run on the raw images, we can see the results in Fig. 8.

Fig. 4. Upper left: reconstruction of MNIST images using only the top 10 highest magnitude eigenvectors. Upper right: reconstruction using the top 50 highest magnitude eigenvectors. Lower left: reconstruction using the top 100 highest eigenvectors. Lower right: reconstruction using all eigenvectors



Interestingly, the maximum performance for logistic regression was achieved at a different number of dimensions (200) than the maximum for k-nearest neighbor classification (50). The maximum at the top 200 eigenvectors was the only iteration of logistic regression on eigenvector coordinates of images that exceeded the performance of the baseline. Although, the performance of models trained at 200, 500, and 784 dimensions could've been slightly better or worse than the values we ended up with since the model failed to converge within the maximum number of iterations we defined during training in those three cases. Due to computational cost, we only looked at the effect of dimension on logistic regression with default parameters but its conceivable that better results can be obtained with more (and finer grain) hyperparameter search.

IV. Summary

From the experiments done, we can see that PCA has the potential to help make sense of data sets and improve the accuracy of classifiers such as k-nearest neighbors and logistic regression. In addition, PCA can allow us to reduce the dimensionality of data sets without losing important information. This, allows us to save on computational costs while preserving or increasing performance of the machine learning model in use.

Fig. 5. The test set error for a model trained on the raw MNIST images and a model trained on the top 100 eigenvector coordinates for the images

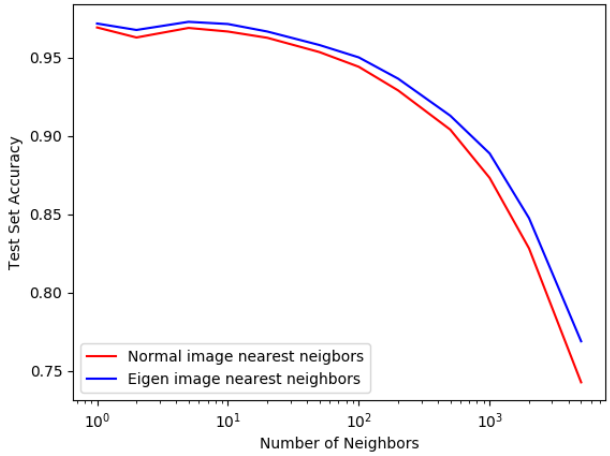


Fig. 6. The test set error for a model trained on a varying number of the top eigenvector coordinates for the images, the baseline trained on raw images is also displayed

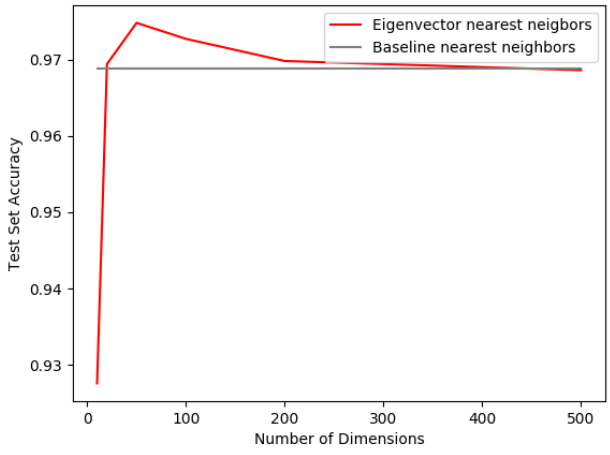


Fig. 7. The test set error for a model where the eigenvectors come from PCA of a varying training set size, a model trained on the same training set but in standard form is also shown

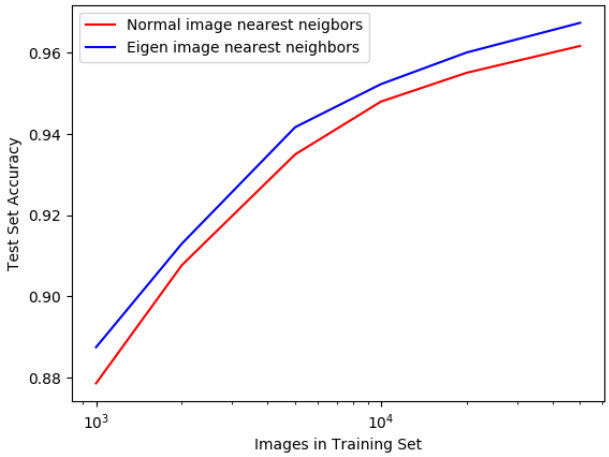


Fig. 8. The test set error for logistic regression trained on a varying number of eigenvector coordinates, the baseline of logistic regression trained on raw images is displayed as well