Principal Component Analysis of Handwritten Digits

Tyler McDonnell
tyler@cs.utexas.edu

The University of Texas at Austin
Department of Computer Science

Abstract

Principal Component Analysis (PCA) is a widely used tool in data analysis and machine learning. It involves a linear transformation of a set of correlated variables to an alternative representation which emphasizes the variance among observations. Practically, it has many applications, including data compression for machine learning and data visualization. Here, I apply Principal Component Analysis to the classic problem of handwritten digit recognition.

1. Introduction

Principal Component Analysis (PCA) is a powerful technique used for data exploration and compression in data analysis and machine learning. Effectively, it reduces the dimensionality of observational data by eliminating redundancy. By reducing the dimensionality of observational data, we can construct more demonstrative visualizations for low dimensional data sets or reduce storage and processing requirements in high dimensional data sets. In some cases, dimensionality reduction may even improve the accuracy of predictive models.

Here, I have two goals. First, I'll develop a simple classification algorithm to recognize handwritten digits in images. Second, I will introduce the high-level steps of PCA and show how it can be usefully applied to this problem to reduce memory and processing overhead. But will it come at the cost of accuracy?

2. Dataset

For this exercise, I used the MNIST database of handwritten images, a widely used dataset for introductory machine learning experimentation. The database is split into a training set and test set of images. The training set consists of 28x28 images of handwritten digits, one per image, and corresponding labels indicating which digit [0-9] is captured in each image. The test set is structured similarly, but only contains 10,000 images. These images have been pre-processed and gray-scaled, so that the digits are fit and centered in the 28x28 image.

In machine learning, a *feature* is a single property of an observation which we use to build predictive models. In this case, the pixels of the image will be the features used to build our predictive model. Intuitively, we will try to uncover

patterns between the intensity of pixels of images and the digit being displayed in the image. Instead of thinking of an image as a 28x28 grid, we can equivalently think of it as a vector, or list, of 784 features, or pixels. From here on out, I will consider our input, or training data, to be an X by N matrix A, where X corresponds to the number of features in an example, initially 784, and N is the number of training examples used from the available set of $60,000$.

3. K-Nearest Neighbors

To reiterate, our first goal is to deploy an algorithm that can recognize handwritten digits in images given a batch of examples. The K-Nearest Neighbors (KNN) algorithm is one such simple classification algorithm. Intuitively, K-Nearest Neighbors takes an observation $\mathbf{x}$ and classifies it according to examples in our training data that are similar or "close" to $\mathbf{x}$ in some sense. Since $\mathbf{x}$ is an input vector of dimensionality X, we can think of it as a point in $\mathbb{R}^X$. Thus, one simple formulation of KNN is to find the K examples from our training set that are closest to $\mathbf{x}$ in $\mathbb{R}^X$ in terms of Euclidean distance, and take the simple majority among the labels of these "neighbors".

The Euclidean distance in $\mathbb{R}^n$ is defined as:

$$d(p,q) = \sum_{i=1}^{n} (q_i - p_i)^2$$

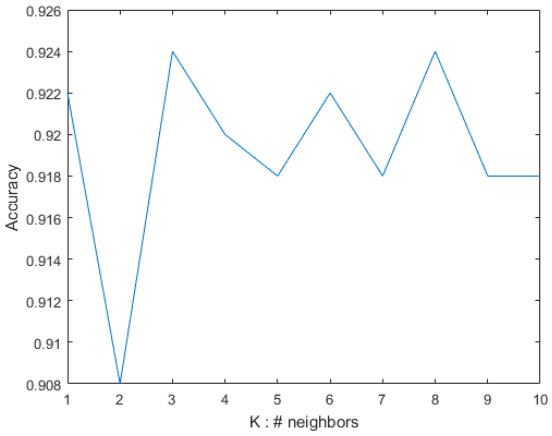

Figure 1: KNN classification accuracy for various values of K.

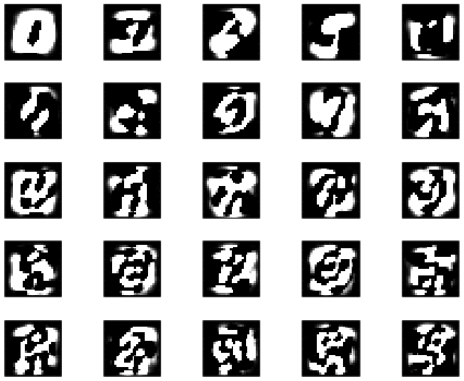

Figure 2: First 25 eigenvectors of A for $N = 60,000$.

Figure 1 shows the accuracy of the KNN algorithm for classifying 500 test cases using 10,000 training examples. As you can see, even this very simplistic model yields over 90% digit recognition accuracy for the K shown!

4. Principal Component Analysis

Now, let's explore how PCA can help us. I won't dig deep into the mathematics of PCA, but I have attached some fantastic resources below. Recall that our goal with PCA is to transform a set of correlated variables into a com-

pressed, dimensionally reduced representation that eliminates redundancy and emphasizes variance. Mathematically, this amounts to computing the covariance matrix, Σ, of our X by N input training data, A, and subsequently finding the eigenvectors of Σ. Note that before computing Σ, you would typically perform some preprocessing on A, such as mean normalization and feature scaling to maximize variance evenly among features. In this exercise, I mean normalize A, but I do not perform feature scaling since each pixel, or feature, may occupy the same range of values. Mean normalization simply means computing a vector containing the mean value of each feature among our training examples, and subtracting that from each example in our training set prior to processing. After computing the eigenvectors, or principal components, of Σ, we sort them in descending order by eigenvalue. I will call this resultant sorted matrix of eigenvectors U. Finally, we select the first T of these eigenvectors as the basis for our new system, which I will call $U_{reduced}$. Figure 2 shows the first 25 eigenvectors of $U_{reduced}$, built with the entire 60,000 image training set.

For clarity, here is the high-level outline of the PCA algorithm, given a matrix of training data A:

1. Mean normalize A.
2. Compute the covariance matrix of the mean normalized input data, $\Sigma = \frac{1}{N-1} * AA^T$.
3. Find the eigenvectors and eigenvalues of Σ.
4. Sort the eigenvectors in descending eigenvalue order and normalize them. Let U be the resultant matrix.
5. Let $U_{reduced}$ be the first T column vectors of U. $U_{reduced}$ is the new basis for our system.

Once we have $U_{reduced}$, we can do some interesting stuff. For example, we can project a new observation, $\mathbf{x}$, into the "eigenspace" defined by $U_{reduced}$, by mean normalizing $\mathbf{x}$ and then computing $U_{reduced}^T\mathbf{x}$. This transforms our original point in $\mathbb{R}^{784}$ into a point, $\mathbf{x}_{reduced}$, in some reduced data space $\mathbb{R}^T$.

I claimed earlier that one of the applications of PCA is lossy compression. Here, the data has clearly been compressed, as $\mathbf{x}$ has changed from a vector of length 784 to a vector of length $T < 784$. But have we lost important data? The first row of Figure 3 shows a particular image that has been projected into the eigenspace defined by $U_{reduced}$ and subsequently reconstructed in the original data space for various values of T. To return to the original data space, we compute $U_{reduced}\mathbf{x}_{reduced}$. We have clearly lost data in the compression, but we even for fairly small values of T, we retain the "meat" of the image.

4.1. Singular Value Decomposition

One widely used implementation of PCA uses the singular value decomposition (SVD) of the covariance matrix rather than eigenvalue decomposition. The steps are identical to the algorithm above, except SVD automatically provides components in descending order and is more "stable," in an esoteric mathematical sense. As a practical comparison, the experimental results that follow are identical for both SVD and eigenvalue decomposition.

Figure 3: Comparison of reconstructed images for PCA and PCA_t for $T = 2, 20, 100, 700$, respectively.

4.2. Dimensionality of Data vs. Number of Examples

Observe that the PCA algorithm above first computes an X by X covariance matrix. In many cases, the dimensionality of the data may be extremely large, and computing this matrix may be computationally intractable. Rather than finding the eigenvectors of this X by X system, we can instead compute the eigenvectors of the N by N system:

$$A^T A\mathbf{v} = \mu\mathbf{v}$$

Multiplying both sides by A, we see that if $\mathbf{v}$ is an eigenvector of A, then $A\mathbf{v}$ is an eigenvector of Σ.

$$AA^T A\mathbf{v} = \mu A\mathbf{v}$$

This algebraic trick presents an alternative PCA formulation for high dimensional observational data, in which we can trade off between accuracy and computational complexity by tuning the amount of training data we wish to use. Specifically, we modify step 2 of the above algorithm to compute the "reduced" covariance matrix, $\Sigma_{reduced} = \frac{1}{N-1} * A^T A$. Then, following step 3, we compute Av, where v are the eigenvectors of $\Sigma_{reduced}$. Figure 3 compares the reconstructions produced by standard PCA and this variant, which I'll coin PCA_t, for training-dominated PCA.

5. Experiments

For experimentation, I tested the classification accuracy of the PCA algorithm using 5,000 observations from the MNIST test set. I first projected the training data and these observations into the eigenspace defined by $U_{reduced}$. I then used Euclidean Distance KNN with $K = 8$ to classify the test observations in the reduced eigenspace. Figure shows the results of these experiments compared to the baseline performance of KNN in the original data space.

Figure 4 shows the performance of traditional PCA for various values of N, with $K = 6$ and $T = 500$. I used PCA instead of PCA_t for this experiment, since the size of $\Sigma_{reduced}$ (and thus T) is constrained by N in PCA, and in my tests, PCA and PCA_t performed the same for identical values of T and N. As we might expect, accuracy increases with the number of training examples used. However, we also quickly reach a point of diminishing returns, where increases in computational requirements may outweigh gains in accuracy. Perhaps surprisingly, KNN performs identically in the

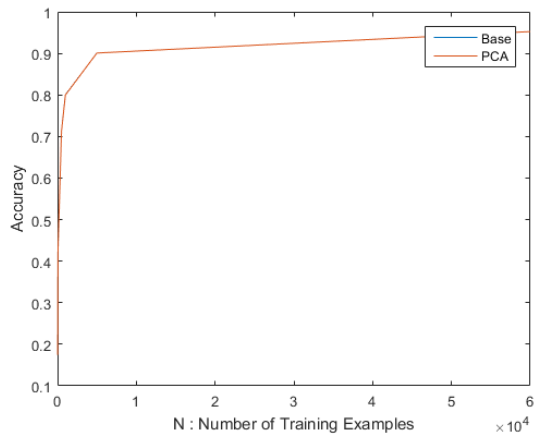

Figure 4: Accuracy for various N.

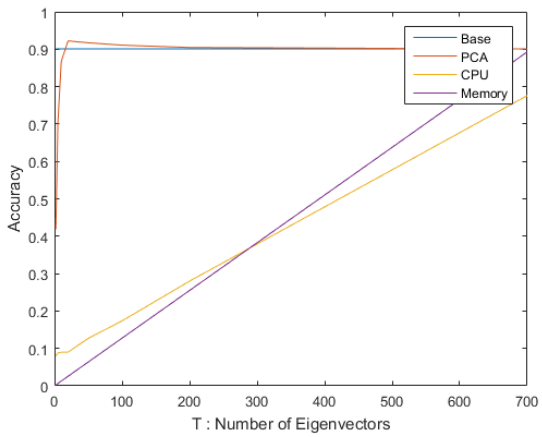

Figure 5: Accuracy for various T.

original data space and the PCA-generated eigenspace for this value of T.

Figure 5 is perhaps the most powerful result in this paper. For this experiment, I held K and N constant, with $K = 8$ and $N = 5000$, and tested different values of T. Even for very low values of T, PCA quickly begins to perform very well. In fact, the optimal performance is reached when the dimensionality is reduced all the way from $\mathbb{R}^{784}$ to $\mathbb{R}^{20}$! Also depicted on this graph are two lines representing the memory usage and CPU time of PCA with respect to the original data space. Remarkably, for $T = 20$, PCA reduces the memory overhead by 98% and CPU time by 90%, while providing better accuracy than KNN in the full data space.

5.1. Bonus: More Fun With Eigenspaces

PCA opens up many new possibilities in terms of potential classification methods. Here, I introduce another approach which I coined "Eigenfun." Rather than construct an eigenspace using training examples from all classifications, I used standard PCA, constructed a separate eigenspace for each classification (i.e., [0-9]), and projected all training examples of each classification into that classification's eigenspace. To classify a new example, I projected it into each eigenspace, performed KNN with $K = 8$, and classified it according to the minimum mean Euclidean Distance among the nearest neighbors in each eigenspace.

Figure 6 compares the accuracy of Eigenfun to PCA and the baseline for $N = 60,000$. Similar to standard PCA, Eigenfun performs well even at low dimensionalities, but unlike PCA, accuracy in this case monotonically increases with T and eventually outperforms both PCA and the baseline at higher dimensionalities.

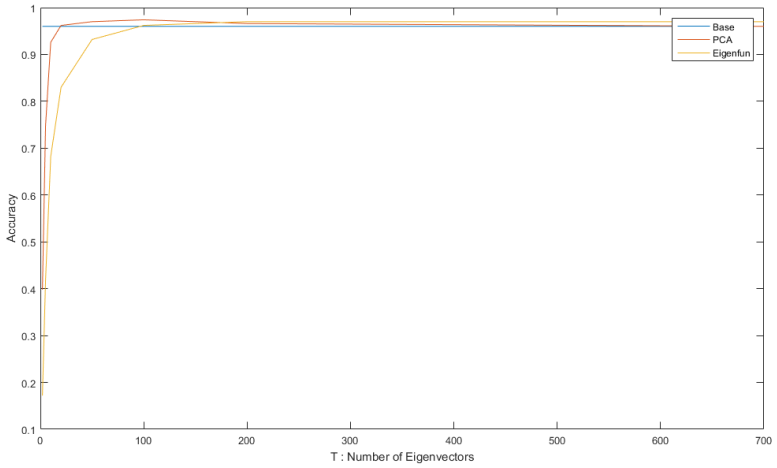

Figure 6: Comparison of base, PCA, and Eigenfun for $N = 60,000$.

6. Conclusion

Principal Component Analysis (PCA) is a powerful machine learning tool that can be used to visualize and compress high dimensional data. It accomplishes this by eliminating redundancy and emphasizing variance, thus allowing us to build more efficient predictive models given large amounts of observational data. Here, I applied PCA to the problem of handwritten digit recognition and showed how it can be used to vastly reduce the memory overhead and CPU time for classification using a simple, but remarkably capable classification algorithm. More importantly, this case study highlights a high degree of redundancy in high dimensional data that is typical of many applications in machine learning. PCA provides a simple, powerful, and flexible framework for data exploration in these cases.

PCA Resources

1. Jon Shlens. A Tutorial on Principal Component Analysis

2. Andrew Ng. Principal Component Analysis, Coursera Lecture.