Assignment 1: Eigendigits

Xiyu Wang

xw5638 xiyu_wang@utexas.edu

Feb. 6, 2020

1 Introduction

Image recognition problems can be solved with machine learning technologies including classification, neural networks and so on. The main idea of them are about identifying some specific features the machine learned from the training images, and thus decide what do the test images contain. One of the most import prework of it is to extract the key features and remove redundancy. The ideal result is to know about the weight of different features when doing image recognition, which is exactly what the ML work intend to do. With great prework, machine learning results can be with more accuracy and efficiency. One of the practical prework when doing image recognition is principal component analysis (PCA), which project the data to a lower dimensional subspace with less redundancy.

In this report, the famous handwritten digits image data set MNIST is used to implement a series of image recognition method. First PCA is applied to the image data. The data in eigenspace and the reconstructed images are generated. The following step is to apply k-nearest neighbors classification (KNN) to realize the image recognition and assign a label to each test image. At last, the accuracy of the classification is analyzed and compared with different parameters.

2 Methods

2 (1) PCA

Principal component analysis (PCA) projects the data to a lower dimensional subspace with less redundancy. There are two main steps: finding the eigenspace and reconstructing the data.

2 (1) a Find the eigenspace

The eigenspace is defined mainly by a matrix of eigenvectors, which are orthogonal with each other and the norm of each one is 1. The number of eigenvectors can be adjusted for the need of the accuracy, and the general "importance" of them can be valued according to the eigenvalues. Generally, the eigenvectors can be sorted in the order of eigenvalues, so it's easier to select the most influential ones.

First, calculate the mean m of each feature in training image A and subtract it from A:

Project the square vector A^T*A and find the eigenvectors v of it.

The eigenvectors V of A can be calculated with the trick in slides: V = A*v

Sort the eigenvectors in V according to the eigenvalues in descending order. Select the top T columns as will to remove the redundancy.

Normalize each column of V to make sure norm(V[:, i]) = 1 for all i within the range.

The required function hw1FindEigendigits() return mean vector m and normalized V.

In this step it's easy to plot the eigenvectors in V to observe the shape of the most crucial eigenvectors.

2 (1) b Reconstruction

Reconstruction will provide the eigenspace projection of the image data according to the V in last step, as well as the reconstructed image after removing the unnecessary eigenvectors.

First, reshape the images into x*k shape matrix A with similar requirement of function hw1FindEigendigits() input.

Subtract the mean value m from A. The m is the output of the function.

Project the image in eigenspace with V^T*A.

Project the reconstructed image with $V*V^T*A$, with the shape of x*k.

In this part it's easy to plot the original image and reconstructed image and compare their difference.

2 (2) KNN

K-nearest neighbors classification find the k "closest" images in training images and come out the label according to similar images.

Prepare the training data by project both training images and test images into eigenspace.

Train the KNN model with training images and training labels.

Feed test images into KNN and get the recognition result.

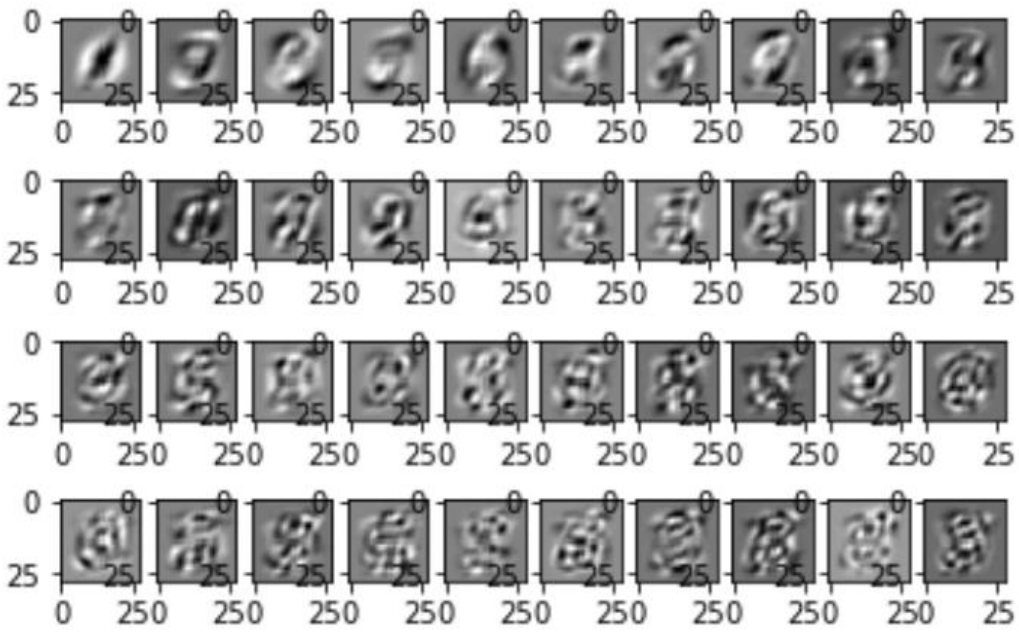
Compare the result and the original labels. Analyze the accuracy.

Adjust some parameters and compare the accuracies with different settings.

3 Results

3 (1) Eigenvectors

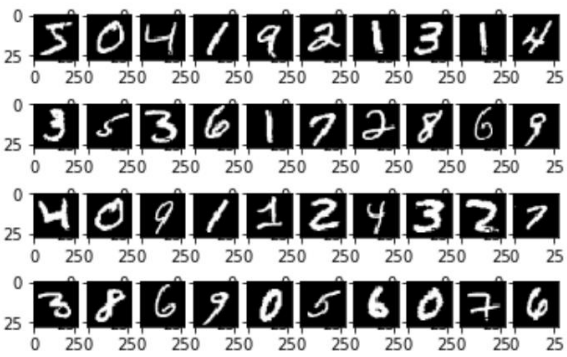
Eigenvectors are sorted by eigenvalues in the matrix V. Since they have the dimension of x = 784 = 28*28, it's easy to plot them as images. The first 40 eigenvectors are as follows:



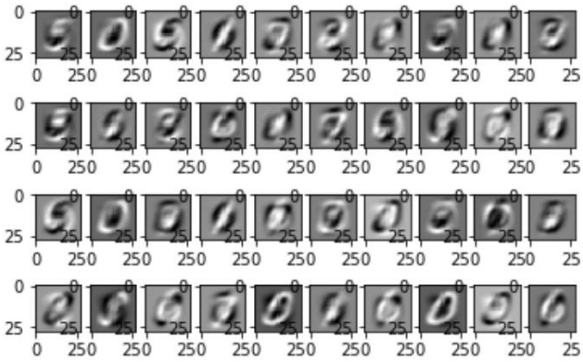
Since the eigenvalues are sorted in descending order, it's easy to notice that the former eigenvectors look more similar with a digit number and are more recognizable. With the order increases, they look more like massive random images. For the simple recognition problem get the first 20 eigenvectors might be enough for a great accuracy.

3 (2) Original image and reconstructed image

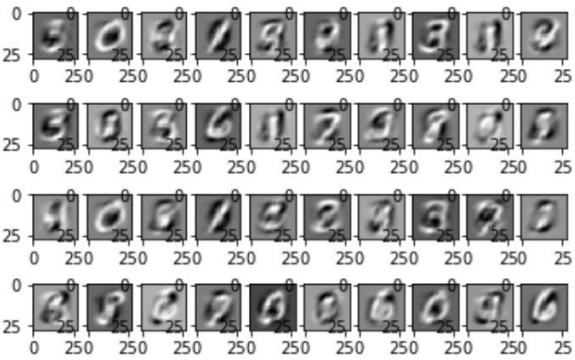
Randomly, the training images are selected to be reshaped as 28*28 and plot for compare. The comparation when selecting top T eigenvectors are as follows when changing T's value:



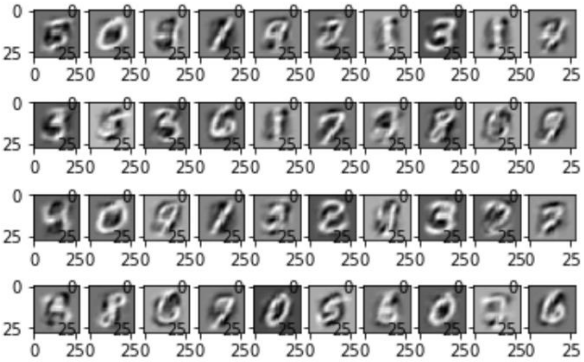
Original Images



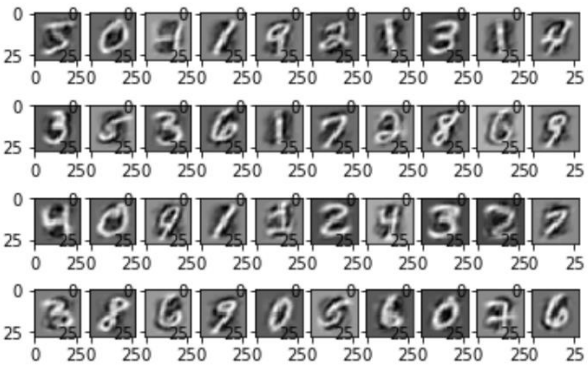
T = 5



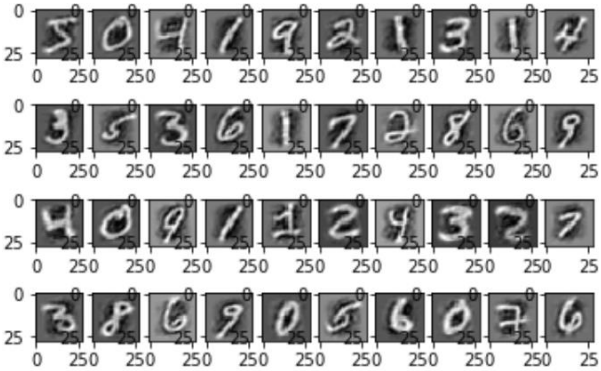
T = 10



T = 20



T = 40



T = 80

During the experiment above, it's easy to find that with T increase the images are closer to the original image and are easier to be recognized. When T = 5 the images look like a combination of the eigenvector images and that do make sense based on the theory. In this case since there only have 10 different digits, it's quite recognizable when T = 10 or T = 15. When T = 40, it becomes very clear and most of the key features are already projected at that time.

3 (3) KNN

Two different variables are included inside of the KNN training and recognition: the size of training image set, and the number T of selected eigenvectors. Separately, two parameters are changed and record the accuracy or the image recognition.

First above all, the accuracy is calculated based on a Boolean: (y_predicted == y_test). That is, set a counter s, and do s++ when there is one pair satisfying (y_predicted == y_test). Go through the label array and the score is the correction rate:

score = s / len(y_test) = number of correct recognition / all test data

which is a number no more than 1.

The test set is fixed as size of 1000.

Consider different T as the legend, training set size as the x-axis and the score as the y-axis. Do experiments and plot it:

The training set size and T value used are:

train_size = [100, 250, 500, 1000, 1500, 2000, 2500, 3000, 3500, 4000, 4500]

T_set = [5, 10, 20, 40, 80]

The scores are shown as follows, with orange colored ones are below 80% and the green colored ones reaches 90%.

T_set\train_size	100	250	500	1000	1500	2000	2500	3000	3500	4000	4500
5	0.486	0.548	0.593	0.627	0.592	0.629	0.622	0.634	0.641	0.645	0.652
10	0.609	0.696	0.741	0.770	0.809	0.820	0.838	0.841	0.843	0.854	0.851
20	0.623	0.728	0.798	0.837	0.857	0.882	0.893	0.896	0.906	0.909	0.909
40	0.637	0.730	0.791	0.840	0.868	0.880	0.890	0.906	0.914	0.917	0.921
80	0.626	0.696	0.778	0.836	0.856	0.877	0.887	0.901	0.900	0.908	0.911

Plot the accuracy scores by different T value:

According to the image, it is obvious that more eigenvectors will lead to higher accuracy, but according to my experiment the calculation difficulty and time spent will increase accordingly. If it's a large scale projection then it might require some special machine (or GPU). Another feature is that when the T is big enough in this data set, the improvement or accuracy will get slower or even stop. For instance, when the T = 80, the accuracy is generally lower than when T = 40. It's possibly because the digit image do not have that many eigenvectors that really matters. The increasement of eigenvectors from 40 to 80 mostly added redundant information, which helps little for more information but makes the KNN more difficult. The drawbacks of it overlapped the benefits. During my experiment when T keep increases above 80, the accuracy would keep decreasing slightly.

From the x-axis direction, generally the accuracy increases with the increase of training size, and the calculation gradually becomes very slow. The accuracy when T = 5 is not so stable, which might be caused by coincidence. The model accuracy is possibly bad when T is small while training data lacks. The test set might happen to fit the simply trained model. If we keep increasing the size of testing set, this unstable part may disappear. For the case T >= 10, the accuracy will increase with the training size, but it will be slower with the increase. For practical use, when the increase of accuracy slowed down, it might not worth it to keep doing it for the high cost of calculation.

4 Summary

PCA can extract eigenvectors and decrease redundancy when selecting the number of eigenvectors.

When selecting the number of eigenvectors, the accuracy will mostly increase when the number increases. There exists a peak and after that the accuracy will not significantly raise. It is because the useful information contained in the eigenvectors are limited and too much variables are difficult to learn from in ML models. It important to analyze a suitable T value according to different problems and data sets.

When selecting the size of training data, basically the accuracy will increase with the size. But the concern is the cost and the requirement. The time cost as well as calculation cost grow very fast

when the training size expends, but the accuracy improvement will get slow when the size is big enough. The idea is to balance the cost and the accuracy, according to the requirement, machine and budget of practical problems. For students or starters, I think set a rate and stop expend the size when the accuracy increase rate is lower than the setting, which would be a good idea.

In this case for MNIST data set, according to my experiment described above, T = 20 to select the top 20 eigenvectors is a great choice and train the KNN with 3000~3500 images, which can get the cognition accuracy of 89.6% to 90.6% with acceptable computation time cost.

Besides, the k value in KNN model is another problem to explore. In my experiment k = 1~3 can work with great accuracy and the increase of k can not significantly increase the score as the former two parameters do. But it's another probability problem and more explore and researches are necessary before making a conclusion.