PCA

MNIST data given in this PCA exercise consists of 60,000 training images and 10,000 test images, both are constructed by 28x28 centred, size-normalized handwritten digit signals, from which each of rasterized images vector 784x1 can be obtained, both for training and test images. The first task involves visualization of projected data after dimension reduction using PCA to 2 and 3 dimensions respectively. Before getting eigenvectors (new projection axis), a covariance matrix should be obtained for the training image matrix:

$$S = \frac{1}{n}\sum_{i=1}^{n}\left((x_i - \bar{x})(x_i - \bar{x})^T\right) = a_1^T S a_1;\ \ given\ z_1 = a_i^T x$$

x_i represents the i-th training image (rasterized) and $\bar{x}$ represents the average training image that also has been rasterized. Covariance matrix *S* will have a size of 784x784, from which we can find 784 corresponding eigenvalues or eigenvectors. In PCA, we can reduce the data dimension to *n*-dimension by choosing *n* largest eigenvalues, and its corresponding eigenvectors will be the new projection axes (dimension) which we can project the data onto them To maximize variances of projected data *var(z_1),* Lagrangian optimization can be used to derive the value of a_1 (with constraint of basis eigenvectors $a_1^T a_1 = 1$) :

$$L = a_1^T S a_1 - \lambda(a_1^T a_1 - 1)$$

$$\frac{\partial L}{\partial a_1} = S a_1 - \lambda(a_1 - 0) \rightarrow (S - \lambda I)a_1 = 0$$

As recalled earlier, value of a_1 that will maximize variances of projected data is the eigenvector of covariance matrix. Subscript '1' in a_1 indicates the eigenvector that corresponds to 1st largest eigenvalue of covariance matrix. In the code, the covariance matrix is obtained as follows ('im_train_cov'):

```
%---------- Calculating Covariance Matrix on Training Data ---------
% Calculate mean image, its size will be 784 by 1
im_train_mean = mean(im_train,2);
im_train_cov = zeros(size(im_train,1),size(im_train,1));
h = waitbar(0,'Please wait...');
index = 0;
for i=1:im_train_count
    index = index + 1;
    im_train_cov = im_train_cov + (im_train(:,i)-im_train_mean)*(im_train(:,i)-im_train_mean)';
    waitbar(index/(length(im_train)),h,sprintf('Generating cov matrix...%2.1f%%',100*index/(length(im_train))));
end
close(h);
```

Figure 1 Calculating covariance matrix

After getting those covariance matrix, a built-in function 'eig' in MATLAB can be simply called and will return eigenvalues and corresponding eigenvectors, which then can be followed up with sorting the eigenvalues and obtain its *n* largest values. With that, we can find corresponding eigenvectors that we can use for projection.

```
%----------- Calculating eigenvalues and eigenvector matrix U -----------
[train_U,train_eig] = eig(im_train_cov);
train_eig = diag(train_eig);

% sorting eigenvalue and eigenvectors
[train_eig, idx_eig] = sort(train_eig,'descend');
train_dump = train_U(:,idx_eig(1));
for i = 2:length(idx_eig)
    train_dump = [train_dump train_U(:,idx_eig(i))];
end
train_U = train_dump;
clear train_dump;
```

Figure 2 Getting eigenvectors and sorting on descending fashion

For 2D projection, the first two eigenvectors are obtained and project the training images data onto them using simple matrix dot. By plotting the per their classes (0-9), we can get visualization of 2D projection of training images.

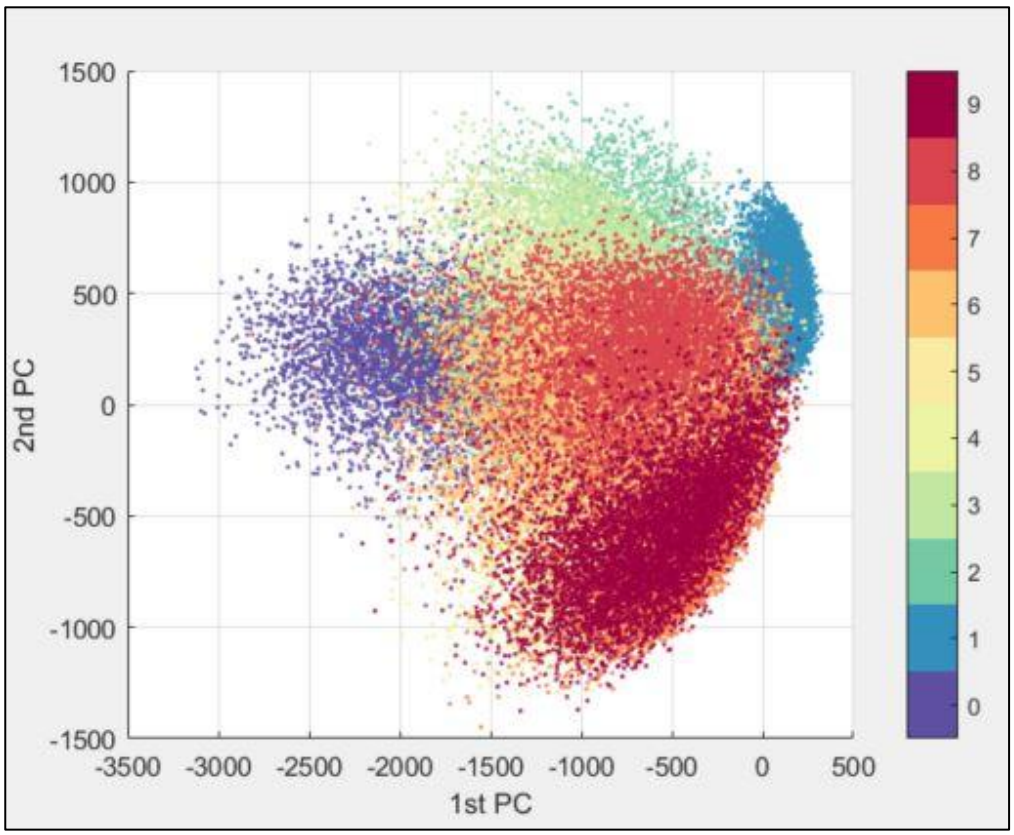

Figure 3 2D representation of training images after performing PCA

The PCA dimensions (1st and 2nd) also can be visually represented by images below, which have been de-rasterized back into 28x28 pixels and its contrast adjusted (by multiplying with scalars) to clearly show the eigenvectors. Note that after being multiplied with scalar, eigenvectors still have the same direction but different length. Multiplication is done such that our visual limitation can see the pixel distribution in eigenvectors.

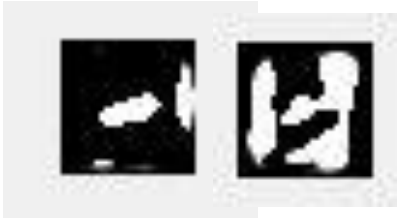

Figure 4 (Left) 1st eigenvector (Right) 2nd eigenvector

Similar steps can be taken for 3D projection, except that first three eigenvectors are to be used to project training data images, which visually illustrated below:

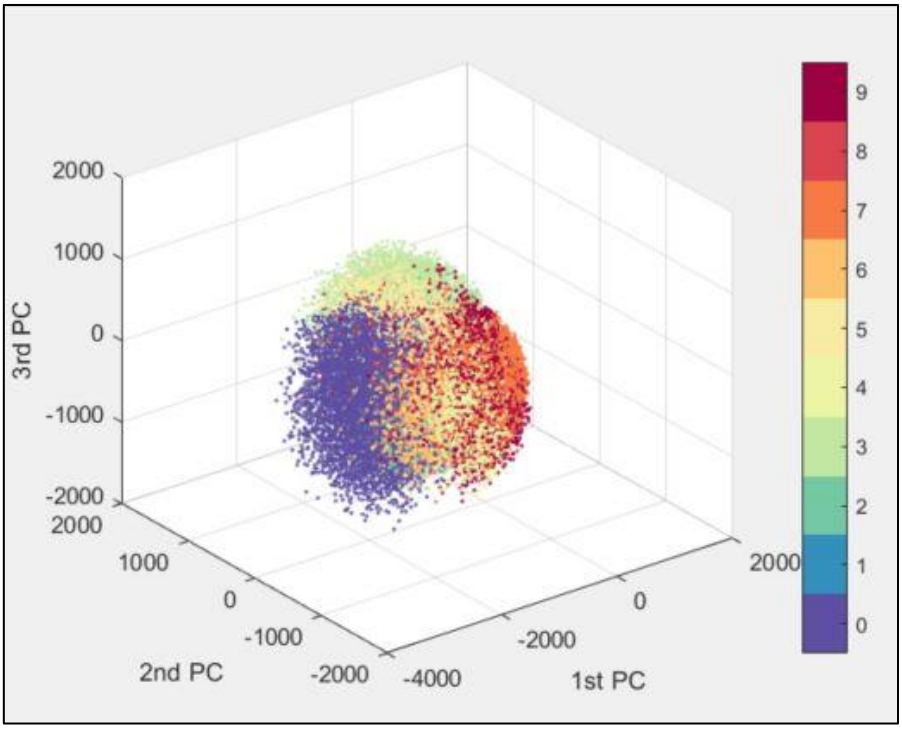

Figure 5 PCA 3D representation of training images

And its eigenvectors are visually represented below:

Figure 6 (Left) 1st eigenvector (Mid) 2nd eigenvector (Right) 3rd eigenvector

Second task in this PCA exercise is to reduce the training images into 40,80,200 dimensions. Those dimensions will be used to project the training images and test images. The label map of training images (which are already known) can be used to determine the new neighbour (test images) using k-NN algorithm below:

1. Calculate distance t_{ij} between data ***i*** in test data and data ***j*** in training data.
2. Repeat step 2 to every single data in test data and create fetch-able matrix from it.
3. Assign **k** as number of neighbours to be taken into k-NN
4. For each data ***m*** in test data.
5. | Sort distance vector $t_{m,:}$
6. | Pick the labels of nearest **k** neighbours based on sorted distance.
7. | Find the majority label ***c*** from that **k** neighbours.
8. | Assign label ***c*** to data ***i*** .
9. End;

For this PCA case, k=28 is empirically the best number of neighbours that has very less error. Other might find other k based on their empirical results.

The results of k-NN algorithm application used on the test images and mapped into label map of training images are tabulated below:

	Test Image Classification Accuracy, k=28
PCA 40 dim	96.88%
PCA 80 dim	95.55%
PCA 200 dim	96.18%

While the number of dimension (reduced) that at least maintain 95% of variances is given below:

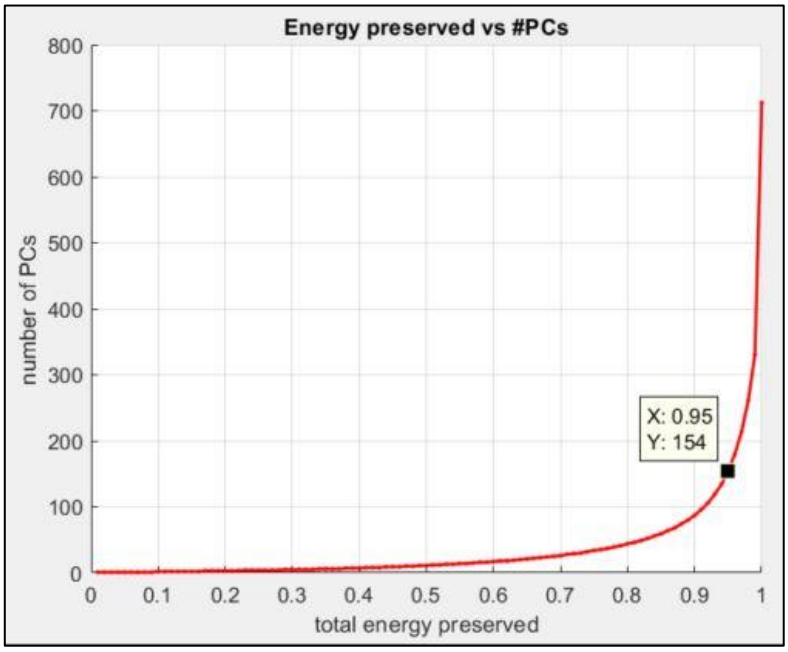

Figure 7 Energy preserved (variance maintained) vs dimension reduction

Found that at least 154 dimensions should be retained, to at least preserve 95% of variances. With 154 dimensions used in the PCA and k-NN algorithm, test image classification had been performed with following error:

	Test Image Classification Accuracy, k=28
PCA 154 dim	96.23%

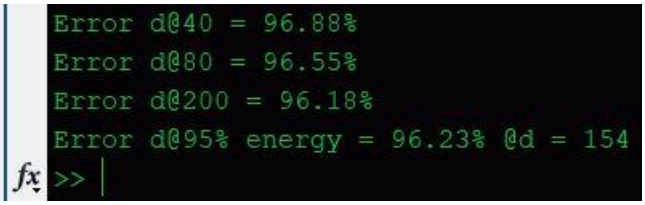

Figure 8 Command window result on the k-NN neighbour

The accuracy of PCA-based k-NN above can never get 100% since PCA by nature omits features that do not characterize the image significantly, but however these non-significant portions are important enough and are the reasons the accuracy can never get 100%. Also, can be seen on the error vs dimension reduction, the accuracy seems better when PCA reduce dimension to lower value (acc@40dim>acc@80dim> acc@154dim> acc@200dim). The possible intuitive reason is that since data are more compact in lower dimension (though features are less), thus applying k-NN classification on a compact neighbour can boost a little bit of classification accuracy, provided that this compact neighbouring still have the necessary features to differentiate. For example, PCA @40 dimension can perform better than PCA@80 dimension since PCA@40 is more compact and it still

likely has the necessary feature for classification. But PCA@5 dimension probably is not better than PCA@40 dimension since these 5 dimensions/projections contain less features (reduced) that possibly omit a certain projection where classes are more distinguishable.

Another criterion that can be used to determine the value of *d* to maintain some data characteristics is: $\frac{\sum_{i=1}^{d}\lambda_i^2}{\sum_{i=1}^{N}\lambda_i^2} > threshold$ .That is also sometimes used in determining the number of hidden neuron in the hidden layer of MLP structure.

LDA

Exercise done in this section utilized LDA method, where the data are again projected onto new axis W such that the distance between classes and at the same time minimize the distance of data of the same class (within-class). In order to perform this optimization, a cost function defined by Fisher is used to achieve optimal distancing between data and between class:

$$J(w) = \left|\frac{\tilde{S}_B}{\tilde{S}_W}\right| = \left|\frac{W^T S_B W}{W^T S_W W}\right|$$

Where S_B is between class variance (scatter) and $S_W$is within class variance (scatter). Thus, the purpose of LDA is to find W such that $J(w)$ maximum, by maximizing the value of $W^T S_B W$ and minimizing the value of $W^T S_W W$. Therefore, W can just simply be obtained through simple derivation with respect to W, and ended up into a form of generalized eigenvalue decomposition problem, that is:

$$\tilde{L}v = \lambda\tilde{B}v \rightarrow (S_B - \lambda_i S_W)W_i^* = 0 \rightarrow (S_W^{-1}S_B - \lambda_i I)W_i^* = 0$$

W_i^* that will maximize $J(w)$ are actually eigenvectors of matrix $S_W^{-1}S_B$.

Since matrix $S_W^{-1}S_B$ likely have more than 10 eigenvectors, we can reduce (pick) two, three, and nine largest eigenvalues and corresponding eigenvectors to reduce the data dimension. Similar to PCA, eigenvectors and eigenvalues of $S_W^{-1}S_B$ can be retrieved using built-in 'eig' in MATLAB, which then we can sort it to find the largest eigenvalues and eigenvectors. For LDA with 2 dimension and 3 dimension, the visual representation of training data are shown below:

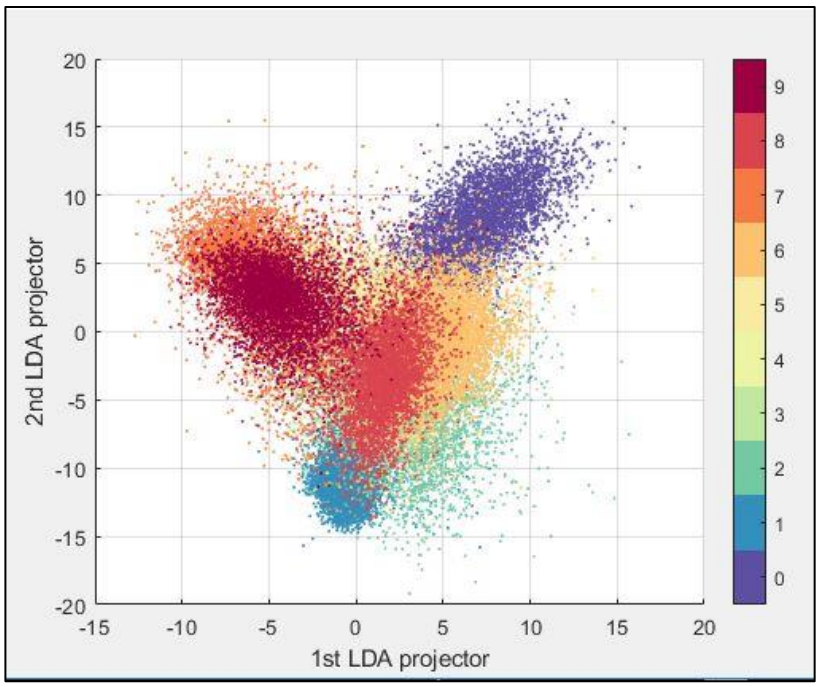

Figure 9 LDA representation with 2 dimensions

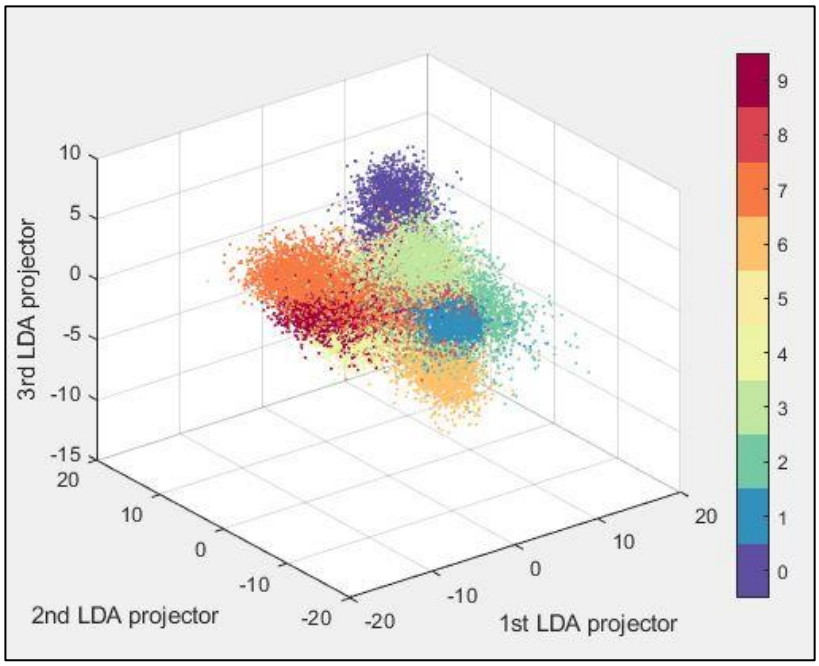

Figure 10 LDA representation with 3 dimensions

Since 9-dimension is way to un-imaginable to be plotted visually, there is no LDA 9-dimension visual representation in this report. However, the results of k-NN test images classification using these dimension-reduced training images are presented below:

*k (number of neighbour used in k-NN) is set at 20 since it is empirically giving quite acceptable result by the time this report is created.

	Test Image Classification Accuracy, k=20
LDA 2 dim	56.63%
LDA 3 dim	74.99%
LDA 9 dim	91.34%

The accuracy of LDA-based k-NN is not as good as PCA-based k-NN, mainly due to the distribution of pixel values in the test images that could be different of that in the training images. And likely within each class itself, the distribution is not unimodal normal, which means that mode in each of classes are not relatively very distinct and its distribution in each dimension (from multidimensional space) doesn't really resemblance gaussian distribution as illustrated below:

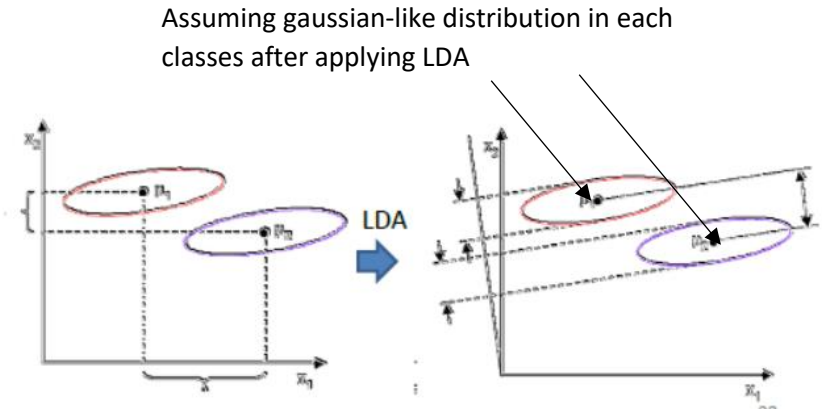

Figure 11 Data distribution resemblances normal-like distribution

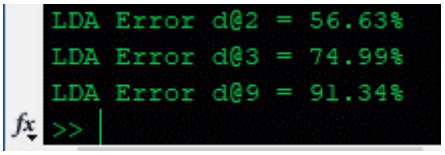

Figure 12LDA-based k-NN Accuracy in command window

The largest dimension that data can be projected onto via LDA is 1 less than the number of classes. Since S_B(between-class scatter) is outer product of 10 mean vector from 10 classes, thus the maximum rank of S_B is 9 (it follows that μ_{10}, let's say, is linear combination of $\mu_{1:9}$ and the training mean μ, thus 9 independent mean vectors as indication that maximum rank of S_B is 9. The result of LDA-based with 9 dimensions is shown above, accuracy@9dim = 91.34%, the highest among the three dimensions examined.

SVM with Linear & Radial Kernels

A new method is again investigated if it can give better classification result on the MNIST handwritten digits. In this exercise, SVM with different kernel function and soft margin are tested. To simplify the direction, PCA pre-processing are performed to reduce the number of training image dimension to: 40,80,200 respectively. And the same tests are to be repeated with different kernel (linear and RBF) and different soft margin, which mathematically are broken down into cases below:

	Soft margin C = 0.01	Soft margin C = 0.1	Soft margin C = 1	Soft margin C = 10
PCA40 linear	Case 1	Case 2	Case 3	Case 4
PCA80 linear	Case 5	Case 6	Case 7	Case 8
PCA200 linear	Case 9	Case 10	Case 11	Case 12
PCA40 RBF	Case 13	Case 14	Case 15	Case 16
PCA80 RBF	Case 17	Case 18	Case 19	Case 20
PCA200 RBF	Case 21	Case 22	Case 23	Case 24

Case 1-12:

Finding α_i such that $J(\alpha) = \sum_{i=1}^{N} \alpha_i - \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N} \alpha_i \alpha_j y_i y_j x_i^T x_j$ with constraints $0 < \alpha_i \leq C$ and $\sum_{i=1}^{N} \alpha_i y_i = 0$. Those α_i can be used to calculate the weights and bias:

Weights : $w_o = \sum_{i=1}^{N} \alpha_{o,i} y_i x_i$ where $\alpha_{o,i}$ are alpha values that satisfy $0 < \alpha_i \leq C$. And x_i are support vectors (x_i whose alpha values fall within constraint)

Bias : $b_o = average\ value\ of\ b_{o,i}$ where $b_{o,i} = \frac{1}{y_i} - w_o^T x_i$

Case 13-24:

Finding α_i such that $J(\alpha) = \sum_{i=1}^{N} \alpha_i - \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N} \alpha_i \alpha_j y_i y_j \psi^T(x_i)\psi(x_j)$ with same constraints :

$0 < \alpha_i \leq C$ and $\sum_{i=1}^{N} \alpha_i y_i = 0$. Most of the time, $\psi^T(x_i)\psi(x_j) = K(x_i, x_j)$ to save computation. And in this RBF case, $K(x_i, x_j) = \exp\left(-\frac{|x_i - x_j|^2}{\sigma}\right)$, σ is set at 1850^2 empirically.

Weights : $w_o = \sum_{i=1}^{N} \alpha_{o,i} y_i \psi(x_i)$ where $\alpha_{o,i}$ are alpha values that satisfy $0 < \alpha_i \leq C$. And x_i are support vectors (x_i whose alpha values fall within constraint)

Bias : $b_o = average\ value\ of\ b_{o,i}$ where $b_{o,i} = \frac{1}{y_i} - w_o^T x_i$

The results of all cases are tabulated below:

Classification Accuracy (%)	Soft margin C = 0.01	Soft margin C = 0.1	Soft margin C = 1	Soft margin C = 10
PCA40 linear	82.7	86.4	87.4	87.2
PCA80 linear	86.1	84.3	86.4	88.4
PCA200 linear	82.7	82.4	82.9	82.5
PCA40 RBF	81.9	90.1	94.4	95.3
PCA80 RBF	82.3	90.4	94.7	95.2
PCA200 RBF	82.2	90	94.1	95.1

As observed in the table above, non-linear kernel seems generally dominate the accuracy compared to linear kernel. And increasing its PCA dimension does not significantly change the accuracy of non-linear kernel SVM as shown in lower half of the table above. On the opposite, increasing PCA dimension in linear SVM seems to reduce the accuracy of classification. The possible reason of that is probably due to the test images becoming more non-linear as the number of dimensions are increased. Thus, hyperplane at 40,80,200 most likely still can't linearly separate the test images based on its classes. The slack penalty C on these linear kernels seems improving the accuracy in some cases (case 2-4, case 6-8), which further indicates that test images are linearly non-separable even at higher dimensions.

```
------Overall performance, row = PCAdim/kernel, col = softmargin------
Error using PCA40 and linear kernel = 82.7         86.4         87.4         87.2%
Error using PCA40 and radial kernel = 81.9         90.1         94.4         95.3%
Error using PCA80 and linear kernel = 86.1         84.3         86.4         88.4%
Error using PCA80 and radial kernel = 82.3         90.4         94.7         95.2%
Error using PCA200 and linear kernel = 82.7         82.4         82.9         82.5%
Error using PCA200 and radial kernel = 82.2           90         94.1         95.1%
>>
```

Figure 13 Results of parameter iteration and different kernels usage