ALİ BURAK ERDOĞAN - 21301492

CS464 HW2

Q1)

a)

Our dataset **X** has a shape **m x n,** that is m samples and n features.

Before starting whole PCA derivation process we should make our features have **zero mean value.** That is, we need to generate a *mean vector with n elements* consisting of average values of each feature and then subtract that row-vector from each row of our dataset.

$$X_{m\times n} = X_{m\times n} - (mean(X))_{1\times n}$$

Then, we should compute **covariance matrix** Σ of X using formula below:

$$\Sigma = \frac{1}{m} \sum_{i=1}^{n} ((x^{(i)}) \cdot (x^{(i)})^{T})$$

Then we should extract eigenvectors of this covariance matrix. Each eigenvector has n elements, so our eigenvectors' shape is **n x 1**. And we have exactly n eigenvectors as well.

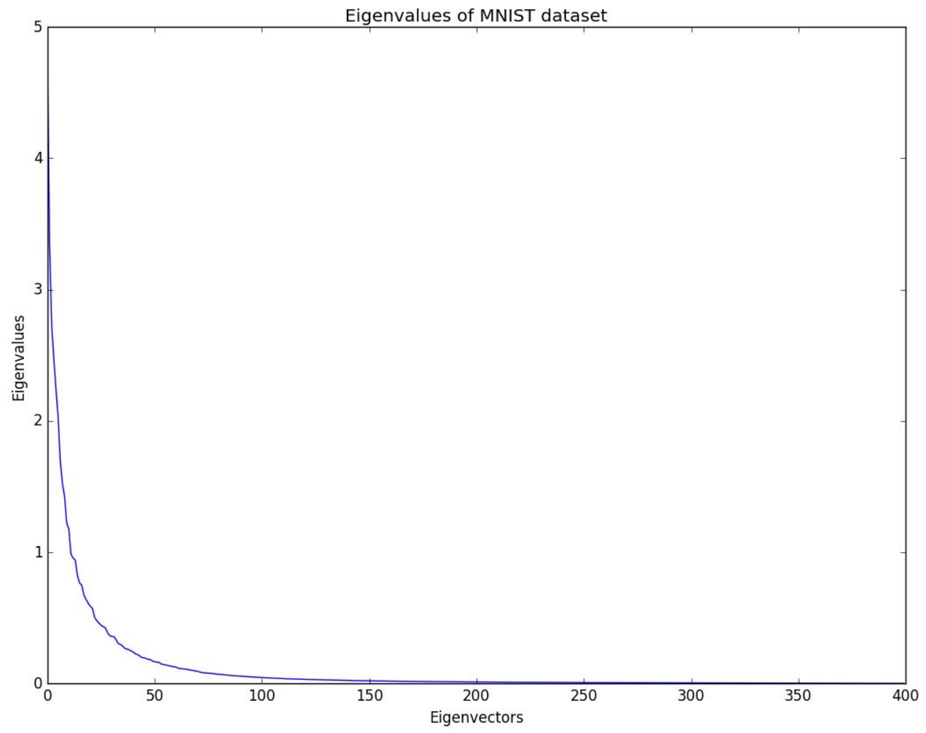
$$U_{n\times n} = eig(\Sigma)$$

We should pick **k eigenvectors** with the highest eigenvalues. This will result in **k principal components.** Then, for projecting into new orthogonal space, multiply those k eigenvectors with each sample. That will give a new sample with reduced number of **k features.**

$$Z_{k\times 1}^{(i)} = U_{k\times n} \times X_{n\times 1}^{(i)}$$

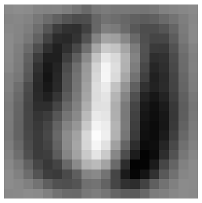
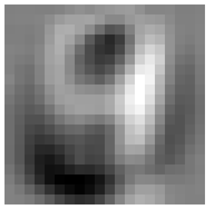
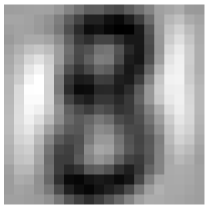
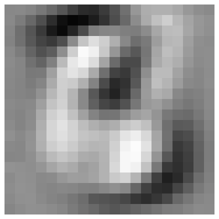
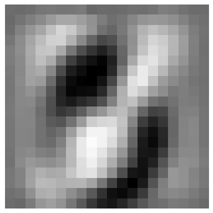
The reason for picking eigenvectors with largest eigenvalues as principal components is high eigenvalue means high variance of a feature. A feature with high variance is very discriminative and gives less reconstruction errors. Therefore, we neglect eigenvectors with low eigenvalues since they are unsuccessful to discriminate and represent our real data, they don't contribute to reconstructing too much.

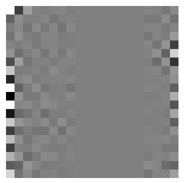
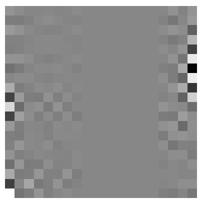
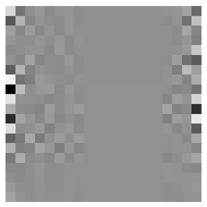
b)



We see from the plot that eigenvectors coming after 20th one has no noteworthy eigenvalues. Choosing the number of principal components “**k”** is actually done by keeping k smallest meanwhile retaining at least %99 of variance .

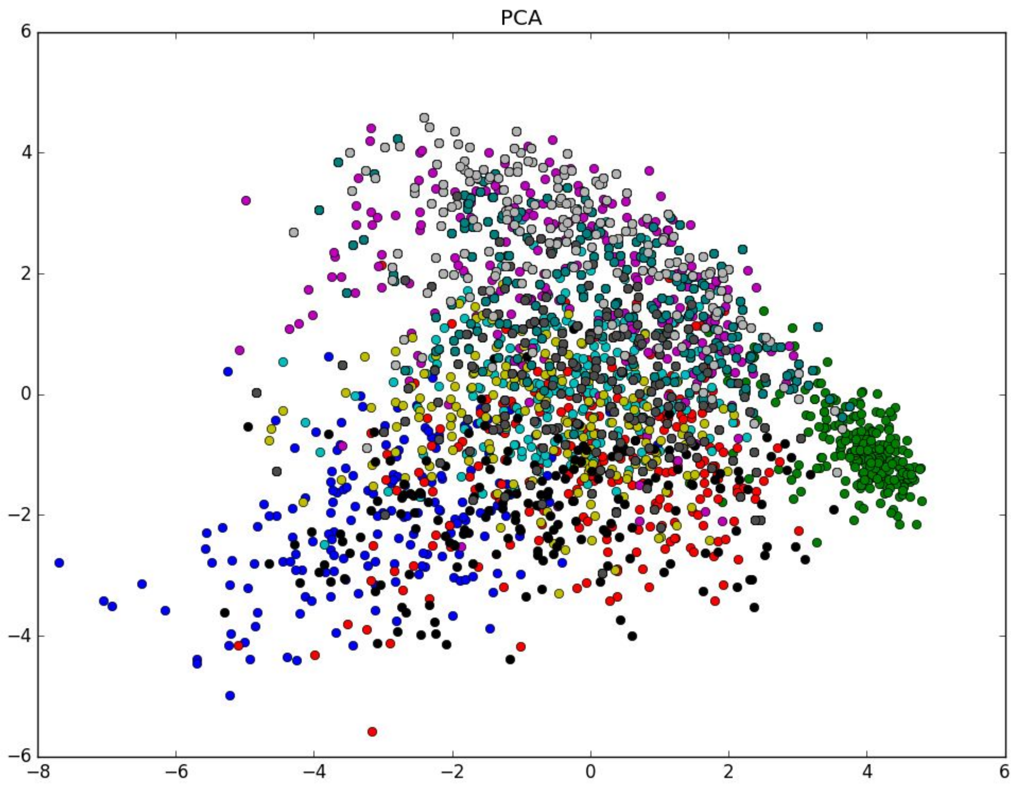
c)

 1st eigenvector	 2nd eigenvector	 3rd eigenvector	 4th eigenvector	 5th eigenvector

Bottom 1 eigvector	 Bottom 2 eigvector	Bottom 3 eigvector	 Bottom 4 eigvector	 Bottom 5 eigvector

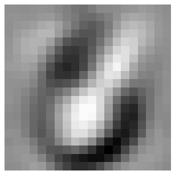
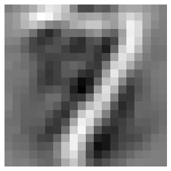
As I explained earlier why we choose eigenvectors with highest eigenvalues, we see the same situation here in these images. Most discriminative and contributive ones are the ones with highest eigenvalues since they show a high variance and low reconstruction error.

d)



This is my scatter plot having 10 different colors for 10 different digit classes.

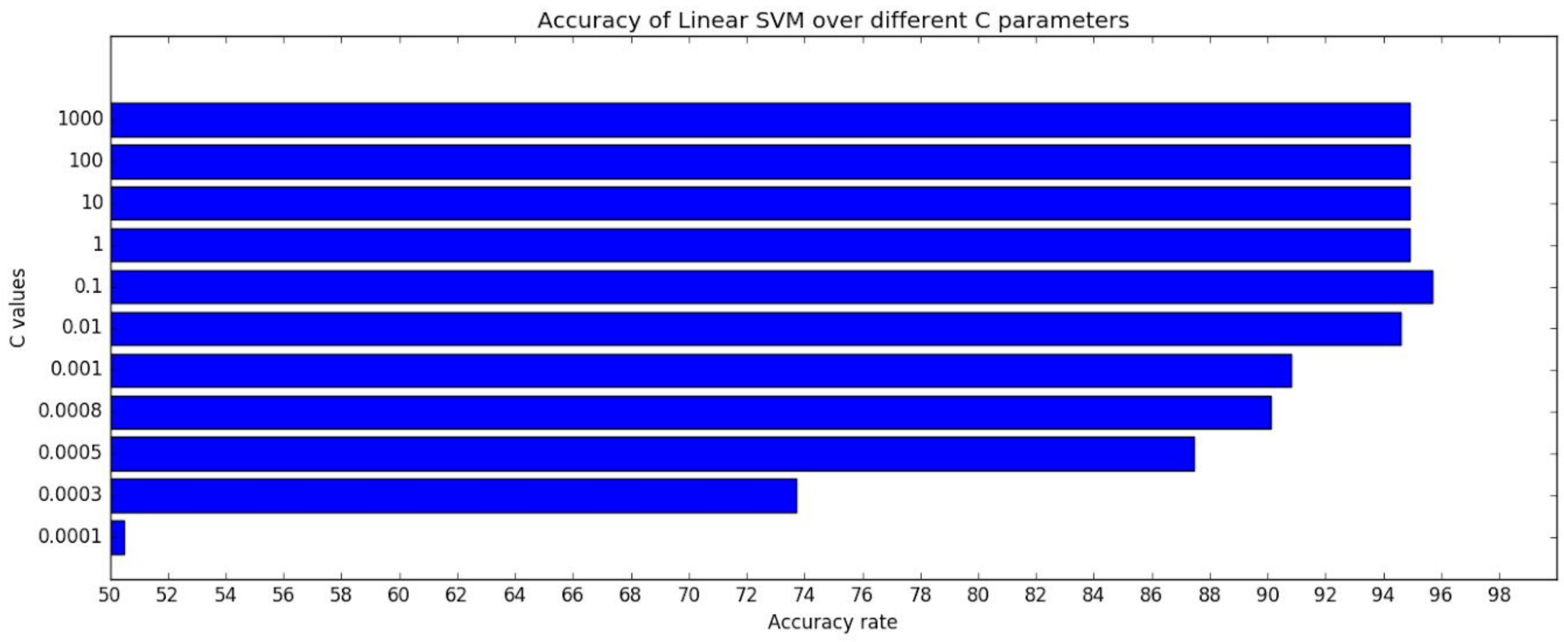
e)

 5 Principal Components	 50 Principal Components	400 Principal Components (full)	Original Image

As we saw in the plot of part B, PCs around 50 is probably enough to retrieve a recognizable reconstruction image. And we also see that using 400 PCs is actually equivalent to original image because we don't perform any feature compression in reality. And using 5 PCs give a very bad result since it's not enough to characterize the samples with that few features.

Q2)

a)



```
/home/burak/.virtualenvs/venv_ml/bin/python3.5
/home/burak/Documents/Courses-2016f/CS464/HW2/hw2_project/main.py
(Linear) C: 0.000100 == 50.556169
(Linear) C: 0.000300 == 73.728513
(Linear) C: 0.000500 == 87.458594
(Linear) C: 0.000800 == 90.123230
(Linear) C: 0.001000 == 90.830709
(Linear) C: 0.010000 == 94.673796
(Linear) C: 0.100000 == 95.718998
(Linear) C: 1.000000 == 94.539525
(Linear) C: 10.000000 == 94.539525
(Linear) C: 100.000000 == 94.539525
(Linear) C: 1000.000000 == 94.539525
Process finished with exit code 0
```

If regularization parameter C is 10^{-4} , it’s the same as a random classifier. As for the values greater than that, we see an increasing performance, but after a little decrease. I think the reason is too small C values result in underfitting (high bias) and C values greater than 0.1 cause very little overfitting.
I chose C = 0.01 for the best fitting parameter for testing step.

Accuracy measurements on test data with C=0.01:

```
/home/burak/.virtualenvs/venv_ml/bin/python3.5
/home/burak/Documents/Courses-2016f/CS464/HW2/hw2_project/main.py
Accuracy: 0.969696969697
F1 score: 0.969690783511
Recall: 0.969696969697
Precision: 0.96988265003

 clasification report:
             precision    recall  f1-score   support

          0       0.96      0.98      0.97       100
          1       0.98      0.96      0.97        98

avg / total       0.97      0.97      0.97       198


 confussion matrix:
 [[98  2]
 [ 4 94]]

Process finished with exit code 0
```

I think that Linear SVM is reasonable when the dataset exhibits a linearly separable distribution. And these high scores I got demonstrates that our dataset is linearly separable.

b)

```
/home/burak/.virtualenvs/venv_ml/bin/python3.5
/home/burak/Documents/Courses-2016f/CS464/HW2/hw2_project/main.py
(Poly) C: 0.000100, degree: 2  == 50.556169
(Poly) C: 0.000100, degree: 3  == 50.556169
(Poly) C: 0.000100, degree: 4  == 50.556169
(Poly) C: 0.000100, degree: 5  == 50.556169
(Poly) C: 0.000100, degree: 6  == 50.556169

(Poly) C: 0.001000, degree: 2  == 50.556169
(Poly) C: 0.001000, degree: 3  == 50.556169
(Poly) C: 0.001000, degree: 4  == 50.556169
(Poly) C: 0.001000, degree: 5  == 50.556169
(Poly) C: 0.001000, degree: 6  == 50.556169

(Poly) C: 0.010000, degree: 2  == 50.556169
(Poly) C: 0.010000, degree: 3  == 50.556169
(Poly) C: 0.010000, degree: 4  == 50.556169
(Poly) C: 0.010000, degree: 5  == 56.448425
(Poly) C: 0.010000, degree: 6  == 67.934405

(Poly) C: 0.100000, degree: 2  == 90.427896
(Poly) C: 0.100000, degree: 3  == 92.383007
(Poly) C: 0.100000, degree: 4  == 93.124565
(Poly) C: 0.100000, degree: 5  == 93.764910
(Poly) C: 0.100000, degree: 6  == 94.304039

(Poly) C: 1.000000, degree: 2  == 95.684919
(Poly) C: 1.000000, degree: 3  == 96.224049
(Poly) C: 1.000000, degree: 4  == 96.898472
(Poly) C: 1.000000, degree: 5  == 97.605952
(Poly) C: 1.000000, degree: 6  == 97.707166

(Poly) C: 10.000000, degree: 2  == 96.696043
(Poly) C: 10.000000, degree: 3  == 96.763178
(Poly) C: 10.000000, degree: 4  == 97.336387
(Poly) C: 10.000000, degree: 5  == 97.437601
(Poly) C: 10.000000, degree: 6  == 97.336387

(Poly) C: 100.000000, degree: 2  == 97.133958
(Poly) C: 100.000000, degree: 3  == 97.235172
(Poly) C: 100.000000, degree: 4  == 97.168037
(Poly) C: 100.000000, degree: 5  == 97.437601
(Poly) C: 100.000000, degree: 6  == 97.336387
Process finished with exit code 0
```

I observe that the polynomial degree affects the accuracy a lot when C=0.01 or C=0.1. However, using C values greater than those causes the polynomial degree *p* to be ineffective for accuracy. I think that best results are given **by choosing C=100 and p=5.**

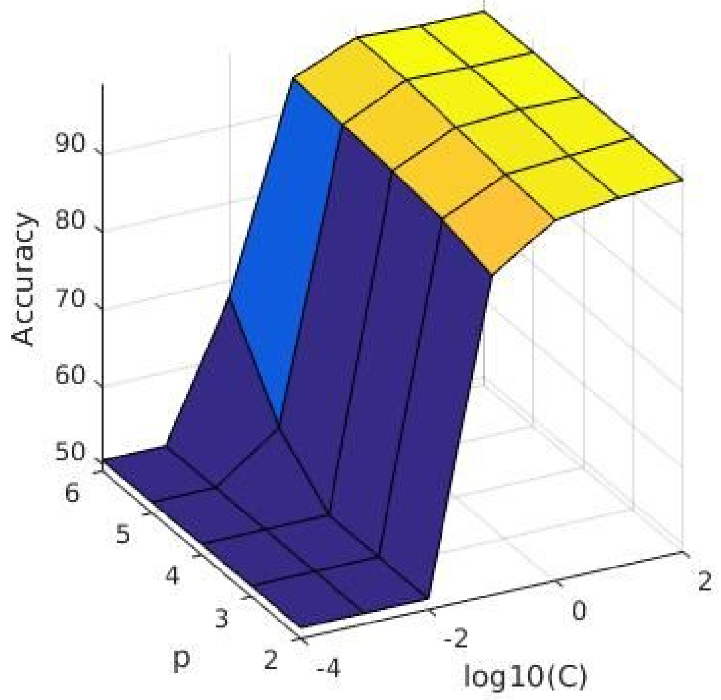
Accuracy measurements on test data with C=100 and p=5:

```
/home/burak/.virtualenvs/venv_ml/bin/python3.5
/home/burak/Documents/Courses-2016f/CS464/HW2/hw2_project/main.py
Accuracy: 0.959595959596
F1 score: 0.959595959596
Recall: 0.959595959596
Precision: 0.959595959596

 clasification report:
             precision    recall  f1-score   support

          0       0.96      0.96      0.96       100
          1       0.96      0.96      0.96        98

avg / total       0.96      0.96      0.96       198


 confussion matrix:
 [[96  4]
 [ 4 94]]

Process finished with exit code 0
```

Using Polynomial kernel SVM is generally better if the dataset exhibits a distribution which cannot be separated with a linear boundary. In this case, I'm not sure whether our dataset is linearly separable or not since the overall scores with polynomial and linear kernels do not differ so much.

Observed differences between linear and polynomial kernel for our dataset:
For C=0.001 linear gave *90.83%* while poly. gave *50% (for different polynomial degrees)*
For C=0.01 linear gave 94.67% while polynomial gave 50-67%
For C=0.1 linear gave 95.71% while polynomial gave 90-94%
For C=1 linear gave 94.53% while polynomial gave 95-97%

Despite accuracy's not changing with different C values, polynomial kernel demonstrated a huge progress in terms of accuracy when I increased C. My assumption is that polynomial kernel was saved from underfitting caused by too small C values.



c)

I had chosen C = 100 and p = 5 in the previous part.

```
935 correct predictions over 1000 samples
Overall accuracy is 93.500000
--------------
Confusion matrix
 [[ 87   0   1   0   0   0   1   0   1   0]
 [   0 118   0   0   1   1   1   0   0   0]
 [   0   0 102   2   2   0   1   4   0   1]
 [   0   0   0  90   0   0   0   0   2   0]
 [   0   0   2   0  77   0   0   0   0   3]
 [   0   1   1   3   1  76   0   0   1   1]
 [   1   1   1   0   0   3  78   0   0   0]
 [   0   2   0   0   1   0   0  96   0   2]
 [   0   0   1   3   2   0   1   1  96   1]
 [   0   2   1   1   7   2   0   1   0 115]]
Accuracy: 0.935
F1 score: 0.935075109785
Recall: 0.935
Precision: 0.936175429676
```

```
 clasification report:
              precision    recall  f1-score   support

          0       0.99      0.97      0.98        90
          1       0.95      0.98      0.96       121
          2       0.94      0.91      0.92       112
          3       0.91      0.98      0.94        92
          4       0.85      0.94      0.89        82
          5       0.93      0.90      0.92        84
          6       0.95      0.93      0.94        84
          7       0.94      0.95      0.95       101
          8       0.96      0.91      0.94       105
          9       0.93      0.89      0.91       129

avg / total       0.94      0.94      0.94      1000


Process finished with exit code 0
```