CS 464

Introduction to Machine Learning

Homework #2

Kerem Ayöz

21501569 Section:2

Question 1

1.1

SVD (Singular Value Decomposition) is a technique used to calculate eigen-vectors and eigen-values of the covariance matrix (x^Tx) of given matrix (x).

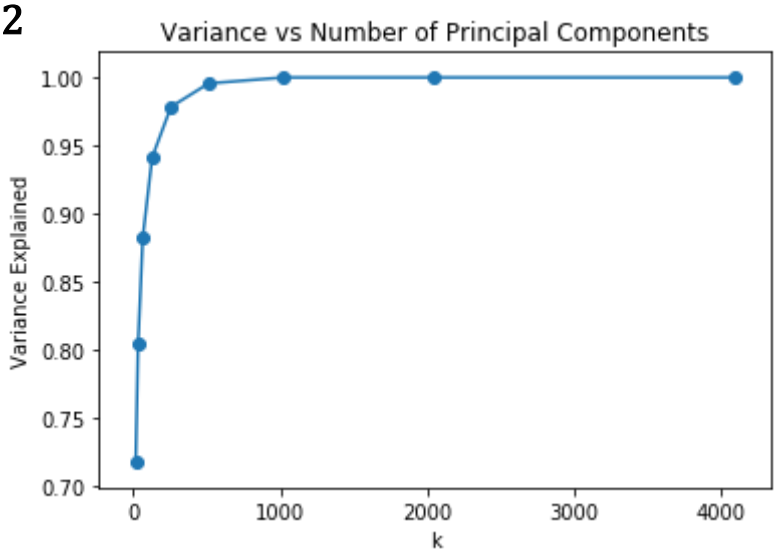
In PCA, we calculate eigen-vectors and eigen values of the covariance matrix (x^Tx) to obtain k-principal components which represents new axes to represent data. However, finding eigen-vectors and eigen-values of covariance matrix may be costly since its dimension is equal to (number of features x number of features) which is huge for image data. To find the eigen-vectors and eigen-values faster, we use SVD on input matrix x.

$$X = U\ S\ V^T$$

$$X^TX = V\ S^T\ U^T\ U\ S\ V^T = V\ S^2\ V^T$$

We can see that S^2 contains eigen-values of covariance matrix in the diagonal and V contains eigen-vectors of covariance matrix in the columns. We obtained these because we diagonalized the covariance matrix by decomposing the input matrix x. This method gives much more efficient running time than first one. Thus, we only need to use input matrix to use SVD to find principal components.

1.2



Values

k = 16 - 0.717453259
k = 32 - 0.803865739
k = 64 - 0.882376722
k = 128 - 0.941317944
k = 256 - 0.978378489
k = 512 - 0.995612619
k = 1024 - 0.999999999
k = 2048 - 0.999999999
k = 4096 - 0.999999999

In this plot, we see that when we use more principal components, we become able to explain more variance in the data. Variance explained ratio shows what percentage of the information we keep after changing transforming the data into another space of axes. When we use a smaller number of principal components, we keep less percentage of the information of data. I obtained this value by summing the eigen values of the selected principal components and dividing them to the total sum of all eigen values. This ratio becomes 1 in theory when we use all principal components.

1.3

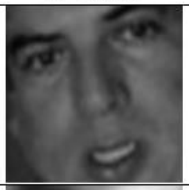
I reconstructed the images by applying inverse transform to the principal components. I used following code to obtain reconstructed images;

```
# Draw reconstructed
samples_idx = [0,1,2,3,7,8]
k_set = [32,128,512]

for k in k_set:
    # Apply PCA
    pca = PCA(k)
    covariance = pca.fit_transform(x)

    # To reconstruct the images, do inverse transform
    proj = pca.inverse_transform(covariance)
    for i in samples_idx:
        new_im = Image.fromarray(np.uint8(np.reshape(proj[i],(64,64))), 'L')
        new_im.save(str(i) + '_' + str(k) + '.png')
```

Then I obtained the following reconstructed images, original images and eigen-faces.

	Aaron Eckhart	Aaron Guiel	Aaron Patterson	Aaron Peirsol	Aaron Pena	Aaron Sorkin
Original						
Eigenfaces						
k=32						
k=128						
k=512						

When we use more eigenfaces, which means principal components, we obtain better reconstructions as we seen from the table above. Since as we deduce from the part 1.2, when we increase number of principal components, we can explain more variance in data which means we lose less information. Thus, we construct images better since we have more information when we use more principal components-eigenfaces.

1.4

As we see from the plot and values in question 1.2, when we use k=128, we can explain the 94.1317944 percent of the variance in data. To obtain 95 percent, we should use k=146 principal components-eigenfaces. This means that we need only 146 transformed features to represent each face. Compression ratio is 146/4096 which is %3.56. This means that if we apply PCA and store only 146 features for each face image, we can reduce the total size of the images to %3.56 of the original size. We keep %95 percent of the information by storing %3.56 percent of information originally needed.

Question 2

2.1

$J(\beta) = || y - X\beta ||^2 = (y - X\beta)^T (y - X\beta) = y^Ty - y^TX\beta - \beta^TX^Ty + \beta^TX^TX\beta$

$\frac{\partial J(\beta)}{\partial \beta} = 0 - y^TX + \beta^T(X^TX) + \beta^T(X^TX) = -2y^TX + 2\beta^TX^TX = 0$

Then $\beta^TX^TX = y^TX \rightarrow (-y^T + \beta^TX^T)X = 0 \rightarrow X^T(-y + X\beta) = 0 \rightarrow X^Ty = X^TX\beta \rightarrow \beta^{RSS} = (X^TX)^{-1}X^Ty$

- x: Training features where each row is a data point and each column represent different feature.
- y: Training labels where each row is label of a data point.
- β: Weights matrix where we have also a bias term. It is calculated from the above formula.

2.2

Since we append a column 1 to the training feature matrix x, the rank of the covariance matrix x^Tx is equal to the number of features of each data point + 1 which makes 9. Thus, rank tells the number of weight terms we will find after calculating the weight matrix, in our case we find 9 weights and one of them is bias term.

```
# Rank of the covariance matrix
rank = np.linalg.matrix_rank(covariance)
print("Rank of covariance matrix is " + str(rank))
```

```
Rank of covariance matrix is 9
```

2.3

After training, I found the weights as following;

β_0 = [35.52032936] (bias)
β_1 = [-0.47656134]
β_2 = [7.00054171]
β_3 = [1.37221717]
β_4 = [-4.24724733]
β_5 = [68.4775968]
β_6 = [286.19995694]
β_7 = [-180.5873212]
β_8 = [24.55935004]

MSE on training data MSE_{train} = 18204.227976890255
MSE on test data MSE_{test} = 40379.06782240535

2.4

Coefficients that I found represents how important each feature to determine the total count of the rental bikes.

Sign of the weight i (β_i) represents the correlation between feature i and label. If it is negative, increasing feature i will decrease the actual label value which means feature i and label are **negatively correlated**. On the other hand, if it is positive, increasing feature i will result in increase in label value. In this case we call feature i and label are **positively correlated**.

Magnitude of the weight i represents the result 1-unit change of feature i in label. For instance, β_8 = 24.55935004 shows that if we increase feature i 1-unit, then the label value will increase 24.55935004. It is also the case for negative weights but this time label decreases. Humidity value is negatively correlated with the total count of rental bikes.

2.5

After training with only humidity feature, I found the weights as following;

β_0 = [348.53430209] (bias)
β_1 = [-277.64064258]

MSE on training data MSE_{train} = 25134.005722252965
MSE on test data MSE_{test} = 48738.79264049626

Humidity value is negatively correlated with the total count of rental bikes.

Question 3

3.1

Optimal learning rate is the value that gives the highest accuracy in training. Optimal learni ng rate is ***0.001*** which gives ***%58.79*** accuracy for training data and gives ***%70.28*** accuracy f or test data. Performance metrics for test data;

Confusion Matrix

	Label = 1	Label = 0
Predicted = 1	1747	986
Predicted = 0	18	627

True Positive: 1747
True Negative: 627
False Positive: 986
False Negative: 18

Performance Metrics

- **Precision:** 0.6392242956458105
- **Recall**: 0.9898016997167138
- **NPV**: 0.9720930232558139
- **FPR**: 0.6112833230006199
- **FDR**: 0.36077570435418954
- **F1**: 0.7847932414406403
- **F2**: 0.9011538854283673

3.2

Training accuracy = % 72.35714285714285
Test accuracy = % 75.39964476021315

Confusion Matrix

	Label = 1	Label = 0
Predicted = 1	1267	333
Predicted = 0	498	1280

True Positive: 1267
True Negative: 1280
False Positive: 333
False Negative: 498

Performance Metrics

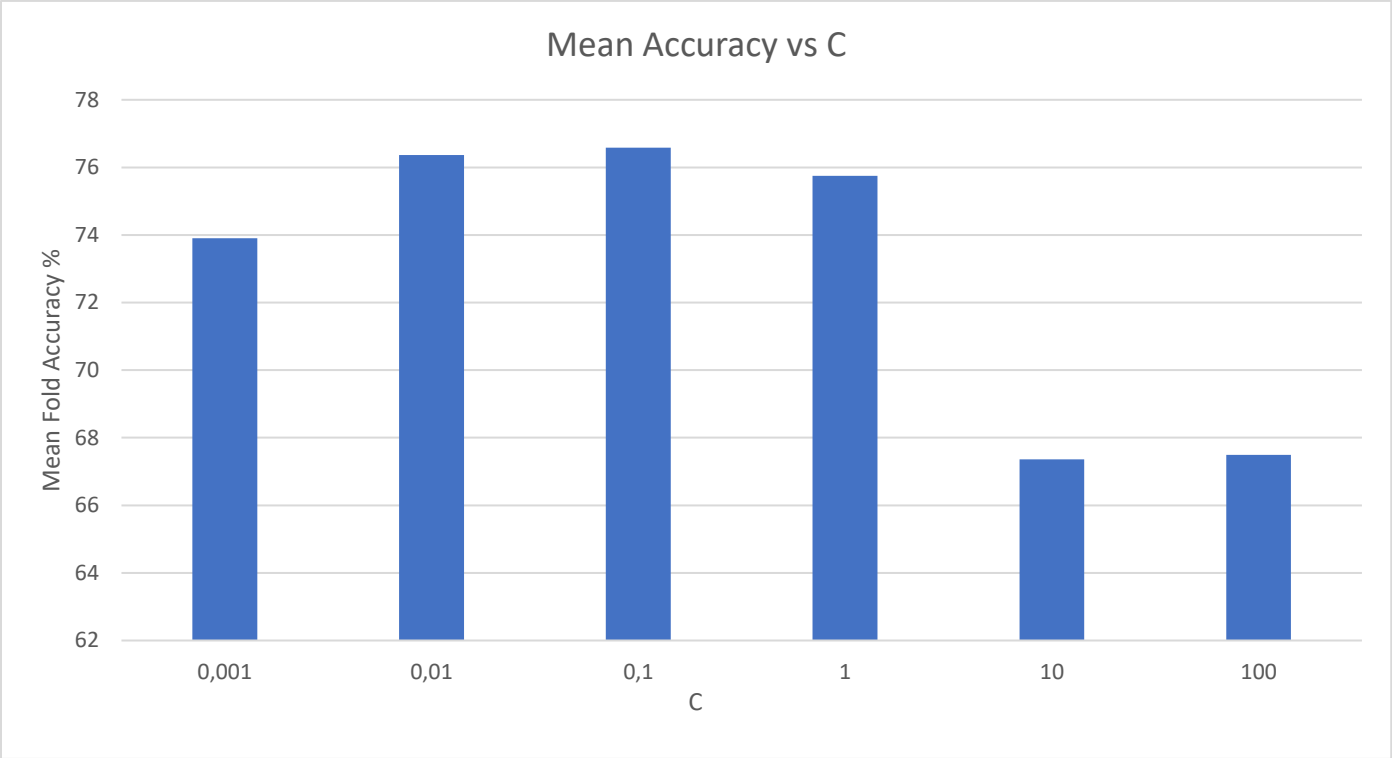
- **Precision**: 0.791875000000
- **Recall**: : 0.71784702549575
- **NPV**: 0.7199100112485939
- **FPR**: 0.20644761314321142
- **FDR**: 0.208125
- **F1**: 1.0490341753343237
- **F2**: 1.0190531177829099

3.3

In case of class imbalance in training data, precision and recall might not be enough to measure performance. If we have few negative or positive samples, precision and recall might be high however performance might be bad.

Question 4

4.1



Optimal C value is the value that gives the highest accuracy in training with 10-fold cross val idation. Optimal C value is ***0.1*** which gives ***%76.67*** mean cross validation accuracy for train ing data and gives ***%69.42*** accuracy for test data. Performance metrics for test data;

Confusion Matrix

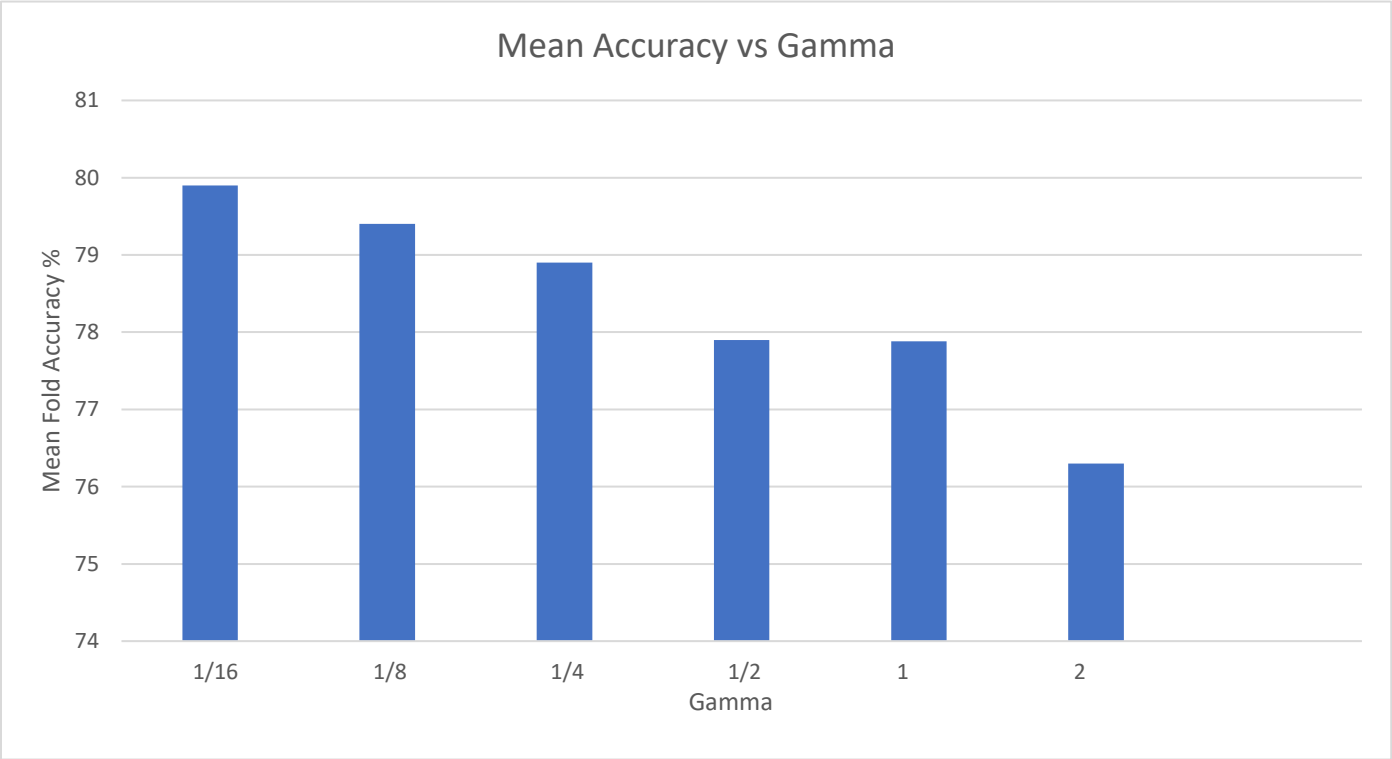
	Label = 1	Label = 0
Predicted = 1	836	104
Predicted = 0	929	1509

True Positive: 836
True Negative: 1509
False Positive: 104
False Negative: 929

Performance Metrics

- **Precision**: 0.8893617021276595
- **Recall**: 0.4736543909348442
- **NPV**: 0.6189499589827727
- **FPR**: 0.06447613143211407
- **FDR**: 0.11063829787234042
- **F1**: 1.3049907578558226
- **F2**: 1.1031250000000001

4.2



Optimal gamma value is value that gives highest accuracy in training with 10-fold cross vali dation. Optimal gamma value is ***1/16*** which gives ***%79.9*** mean cross validation accuracy fo r training data and gives ***%70.54*** accuracy for test data. Performance metrics for test data;

Confusion Matrix

	Label = 1	Label = 0
Predicted = 1	843	73
Predicted = 0	922	1540

True Positive: 843
True Negative: 1540
False Positive: 73
False Negative: 922

Performance Metrics

- **Precision**: 0.9203056768558951
- **Recall**: 0.4776203966005666
- **NPV**: 0.6255077173030057
- **FPR**: 0.04525728456292622
- **FDR**: : 0.07969432314410481
- **F1**: 1.3166728832525176
- **F2**: 1.1064443329989968