CS 464
Introduction to Machine Learning
Homework 2

Mustafa Göktan Güdükbay
21801740, Section 2

1. Principal Component Analysis (PCA) & Digits

I wrote a program in Python to compress digit images using PCA and decompress them back to restore original images. I use the digits image dataset file named digits.csv where the first column in the file represents the digit and the other columns represent the features of the digits. I first make the data mean centered before applying PCA.

1.1. PCA Components and Proportion of Variance Explained (PVE)

I applied PCA to obtain the first 10 principal components. The following output gives the proportion of variance explained (PVE) for each of these 10 components.

Output:

```
Part 1: Proportions of Variance Explained


Component:  1, PVE:  0.10047663329207103
Component:  2, PVE:  0.07544486615471577
Component:  3, PVE:  0.061405161905458035
Component:  4, PVE:  0.054258073927254125
Component:  5, PVE:  0.050312489007778634
Component:  6, PVE:  0.04246363364754702
Component:  7, PVE:  0.03311403768335862
Component:  8, PVE:  0.029502883483561705
Component:  9, PVE:  0.027298577357821815
Component:  10, PVE:  0.022780414032228608
```

Figure 1 shows the images of each of the 10 principal components reshaped to 28×28 matrices.

Discussion:

The first component has the highest PVE value, and as the component number increases, the PVE value decreases. This implies that the top components can explain a more significant portion of the data than the other components. Some components explain the variance better than the other components.

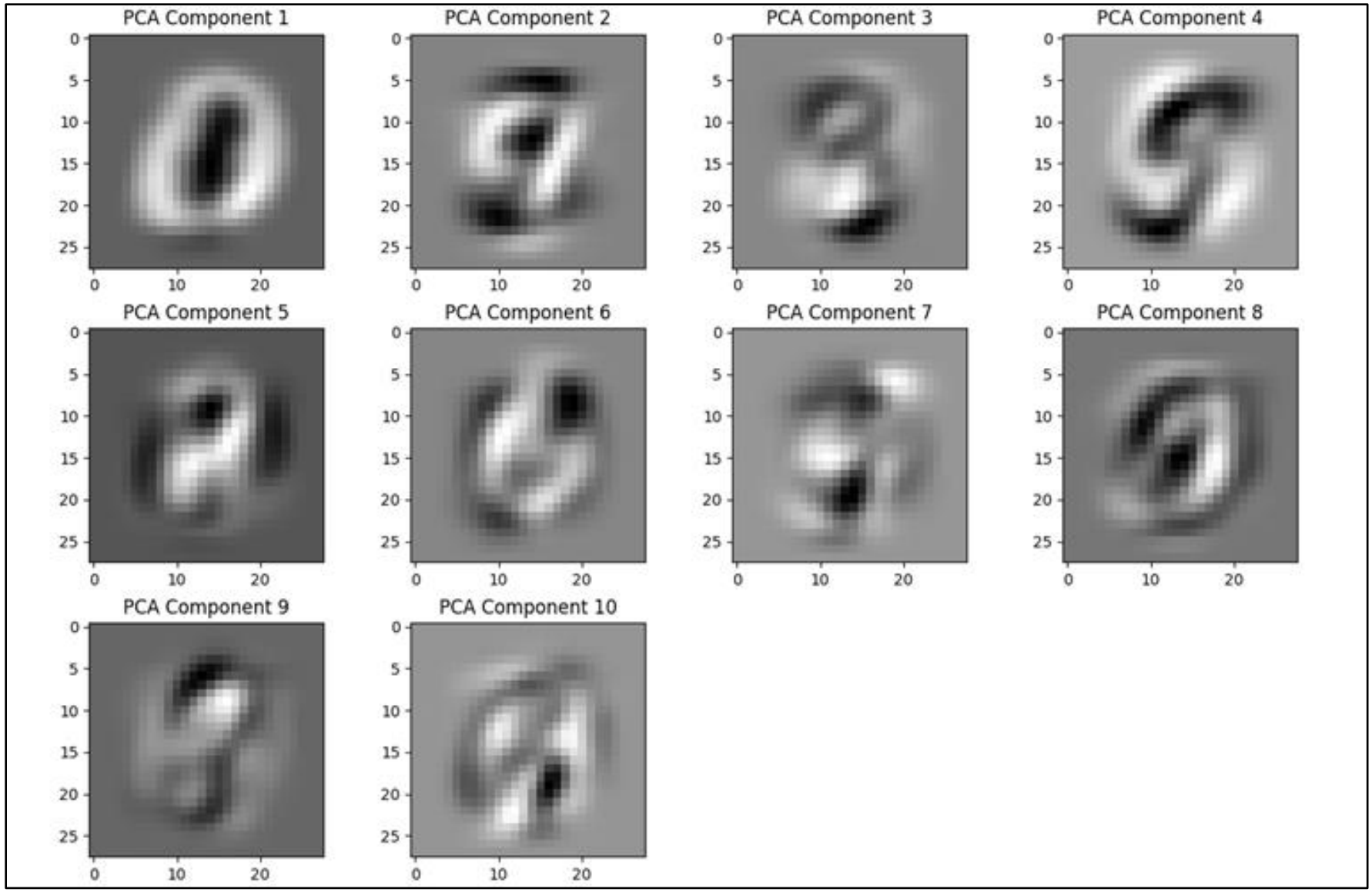

Figure 1 – 10 principal components reshaped to 28×28 matrices.

1.2. First k Principal Components and PVE for k ∈ {8, 16, 32, 64, 128, 256}

I obtain the first k principal components and report the proportion of variance explained (PVE) for k ∈ {8, 16, 32, 64, 128, 256}. The following program output gives these PVE values.

Output:

```
Part 1.2:


k:  8 , Proportion of Variances Explained for Each Component:
[0.10047663 0.07544487 0.06140516 0.05425807 0.05031249 0.04246363
 0.03311404 0.02950288]

Total Proportion of Variances Explained for All the Components:
0.44697777910174497

k:  16 , Proportion of Variances Explained for Each Component:
[0.10047663 0.07544487 0.06140516 0.05425807 0.05031249 0.04246363
 0.03311404 0.02950288 0.02729858 0.02278041 0.02133899 0.02095204
 0.01725898 0.01685758 0.01576077 0.01494562]

Total Proportion of Variances Explained for All the Components:
0.6041707498265494

k:  32 , Proportion of Variances Explained for Each Component:
[0.10047663 0.07544487 0.06140516 0.05425807 0.05031249 0.04246363
 0.03311404 0.02950288 0.02729858 0.02278041 0.02133899 0.02095204
 0.01725898 0.01685758 0.01576077 0.01494562 0.01302506 0.01278467
 0.01180118 0.011483   0.0106641  0.01027178 0.00979715 0.00901867
 0.00888231 0.00826573 0.00799743 0.00765838 0.00717514 0.00687256
 0.00649144 0.00627159]

Total Proportion of Variances Explained for All the Components:
0.7526309375626092

k:  64 , Proportion of Variances Explained for Each Component:
[0.10047663 0.07544487 0.06140516 0.05425807 0.05031249 0.04246363
 0.03311404 0.02950288 0.02729858 0.02278041 0.02133899 0.02095204
 0.01725898 0.01685758 0.01576077 0.01494562 0.01302506 0.01278467
 0.01180118 0.011483   0.0106641  0.01027178 0.00979715 0.00901867
 0.00888231 0.00826573 0.00799743 0.00765838 0.00717514 0.00687256
 0.00649144 0.00627159 0.00590284 0.00581759 0.00544738 0.00541655
 0.00507243 0.00481121 0.00460166 0.00451523 0.00437027 0.00424364
 0.00412964 0.00389631 0.00378241 0.00370461 0.00358284 0.00331726
 0.00326815 0.00311842 0.00304071 0.00297839 0.00288487 0.00278702
 0.00273021 0.00262965 0.00254963 0.00253014 0.00244553 0.00241445
 0.00234363 0.00233339 0.00223071 0.0021553 ]

Total Proportion of Variances Explained for All the Components:
0.8676830035049767
```

k: 128 , Proportion of Variances Explained for Each Component:

```
[0.10047663 0.07544487 0.06140516 0.05425807 0.05031249 0.04246363
 0.03311404 0.02950288 0.02729858 0.02278041 0.02133899 0.02095204
 0.01725898 0.01685758 0.01576077 0.01494562 0.01302506 0.01278467
 0.01180118 0.011483   0.0106641  0.01027178 0.00979715 0.00901867
 0.00888231 0.00826573 0.00799743 0.00765838 0.00717514 0.00687256
 0.00649144 0.00627159 0.00590284 0.00581759 0.00544738 0.00541655
 0.00507243 0.00481121 0.00460166 0.00451523 0.00437027 0.00424364
 0.00412964 0.00389631 0.00378241 0.00370461 0.00358284 0.00331726
 0.00326815 0.00311842 0.00304071 0.00297839 0.00288487 0.00278702
 0.00273021 0.00262965 0.00254963 0.00253014 0.00244553 0.00241445
 0.00234363 0.00233339 0.00223071 0.0021553  0.00208822 0.00202329
 0.00199522 0.00187195 0.00185669 0.00188604 0.00180588 0.00174369
 0.00170453 0.00164066 0.00164326 0.00157377 0.00154559 0.00148286
 0.00143052 0.00139118 0.00134183 0.00137763 0.00131815 0.00131228
 0.00129521 0.00115256 0.00117083 0.00125073 0.00121606 0.00122568
 0.00110545 0.00109889 0.00107662 0.00107223 0.00104666 0.00102063
 0.0010027  0.00099162 0.00096464 0.00094865 0.00092498 0.00093485
 0.00088142 0.00086753 0.00084039 0.0008522  0.00085089 0.00081975
 0.00080513 0.000782   0.00076739 0.00075309 0.00076042 0.00073676
 0.00072691 0.00071049 0.00071482 0.00070077 0.00068777 0.00066883
 0.00066443 0.00065975 0.00065383 0.00064401 0.00063423 0.00062263
 0.0006148  0.00060763]
```

Total Proportion of Variances Explained for All the Components:
0.939243081275845

k: 256 , Proportion of Variances Explained for Each Component:

```
[0.10047663 0.07544487 0.06140516 0.05425807 0.05031249 0.04246363
 0.03311404 0.02950288 0.02729858 0.02278041 0.02133899 0.02095204
 0.01725898 0.01685758 0.01576077 0.01494562 0.01302506 0.01278467
 0.01180118 0.011483   0.0106641  0.01027178 0.00979715 0.00901867
 0.00888231 0.00826573 0.00799743 0.00765838 0.00717514 0.00687256
 0.00649144 0.00627159 0.00590284 0.00581759 0.00544738 0.00541655
 0.00507243 0.00481121 0.00460166 0.00451523 0.00437027 0.00424364
 0.00412964 0.00389631 0.00378241 0.00370461 0.00358284 0.00331726
 0.00326815 0.00311842 0.00304071 0.00297839 0.00288487 0.00278702
 0.00273021 0.00262965 0.00254963 0.00253014 0.00244553 0.00241445
 0.00234363 0.00233339 0.00223071 0.0021553  0.00208822 0.00202329
 0.00199522 0.00187195 0.00185669 0.00188604 0.00180588 0.00174369
 0.00170453 0.00164066 0.00164326 0.00157377 0.00154559 0.00148286
 0.00143052 0.00139118 0.00134183 0.00137763 0.00131815 0.00131228
 0.00129521 0.00115256 0.00117083 0.00125073 0.00121606 0.00122568
 0.00110545 0.00109889 0.00107662 0.00107223 0.00104666 0.00102063
 0.0010027  0.00099162 0.00096464 0.00094865 0.00092498 0.00093485
 0.00088142 0.00086753 0.00084039 0.0008522  0.00085089 0.00081975
 0.00080513 0.000782   0.00076739 0.00075309 0.00076042 0.00073676
 0.00072691 0.00071049 0.00071482 0.00070077 0.00068777 0.00066883
 0.00066443 0.00065975 0.00065383 0.00064401 0.00063423 0.00062263
 0.0006148  0.00060763 0.00059995 0.00059489 0.00058822 0.00058094
 0.00057417 0.00055867 0.00055477 0.00054661 0.0005394  0.00053042
 0.00052553 0.00051832 0.00051119 0.0005035  0.00049851 0.00049047
 0.00048675 0.00047608 0.00047407 0.00046509 0.00046252 0.00045629
 0.00044692 0.00044096 0.00043946 0.00043632 0.00043419 0.00043017
 0.00042573 0.00041641 0.00040729 0.0003912  0.00040138 0.00039974
```

```
0.00038837 0.00034464 0.00038148 0.00037258 0.0003783  0.00035062
0.00036781 0.00036588 0.00035202 0.00035422 0.00035912 0.00036331
0.00035806 0.00033869 0.00034692 0.00033512 0.0003305  0.00032922
0.00032613 0.0003213  0.00032038 0.00031284 0.00031783 0.00031534
0.00031125 0.00030889 0.00030626 0.00030253 0.00029802 0.00029702
0.00029285 0.00028916 0.0002879  0.0002827  0.00028187 0.00027883
0.00027634 0.00027359 0.00027212 0.00026669 0.00026953 0.00026177
0.00026093 0.00025926 0.00025782 0.00025582 0.00020698 0.00020794
0.00025485 0.00025049 0.00025305 0.00025338 0.00024681 0.00020968
0.00021078 0.00021249 0.00024495 0.00024295 0.00023994 0.00023351
0.00023684 0.00023168 0.00021457 0.00024235 0.00022177 0.00021897
0.00021394 0.00023782 0.00022869 0.0002182  0.00022509 0.0002271
0.00022332 0.0002258  0.00020581 0.00020413 0.00020359 0.00015513
0.0001991  0.00015671 0.00015808 0.00015919 0.00019791 0.00019717
0.0001937  0.0001959  0.00016097 0.00019187 0.00016245 0.00016359
0.00019092 0.00018921 0.00018896 0.00016409]

Total Proportion of Variances Explained for All the Components:
0.9804434248981055
```

Figure 2 plots the Proportion of Variances Explained (PVE) for each component for different k values and Figure 3 plots the Total Proportion of Variances Explained (PVE) for different k values.

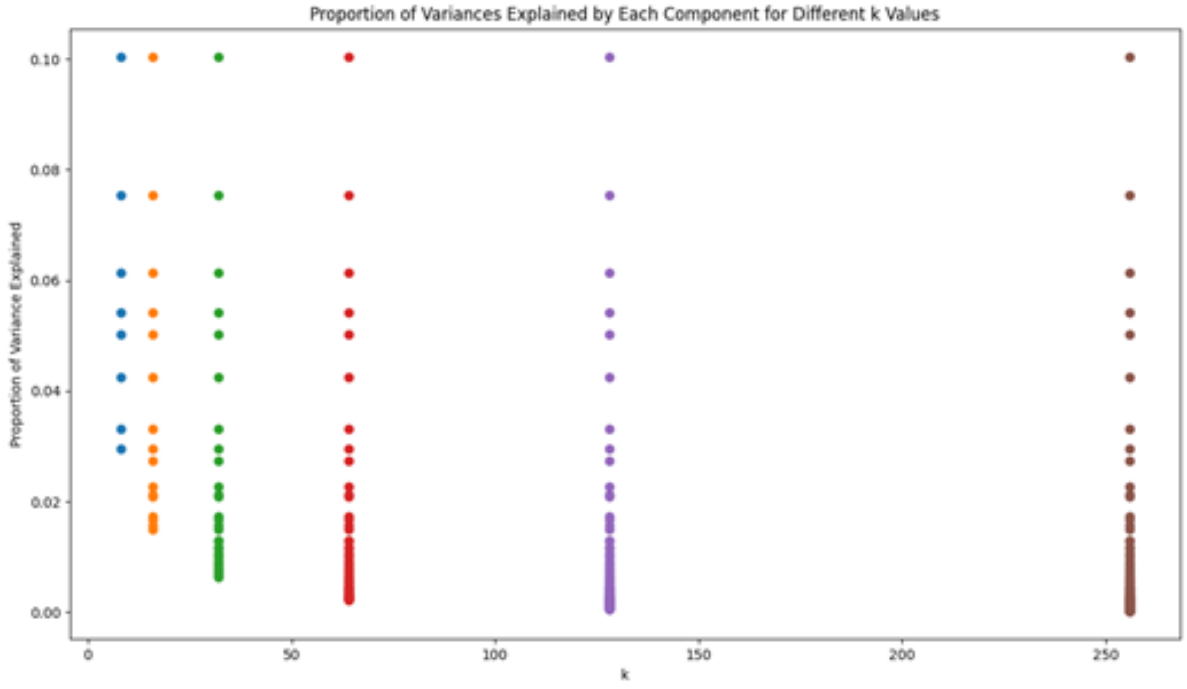

Figure 2 - Proportion of Variances Explained (PVE) for each component for different k values

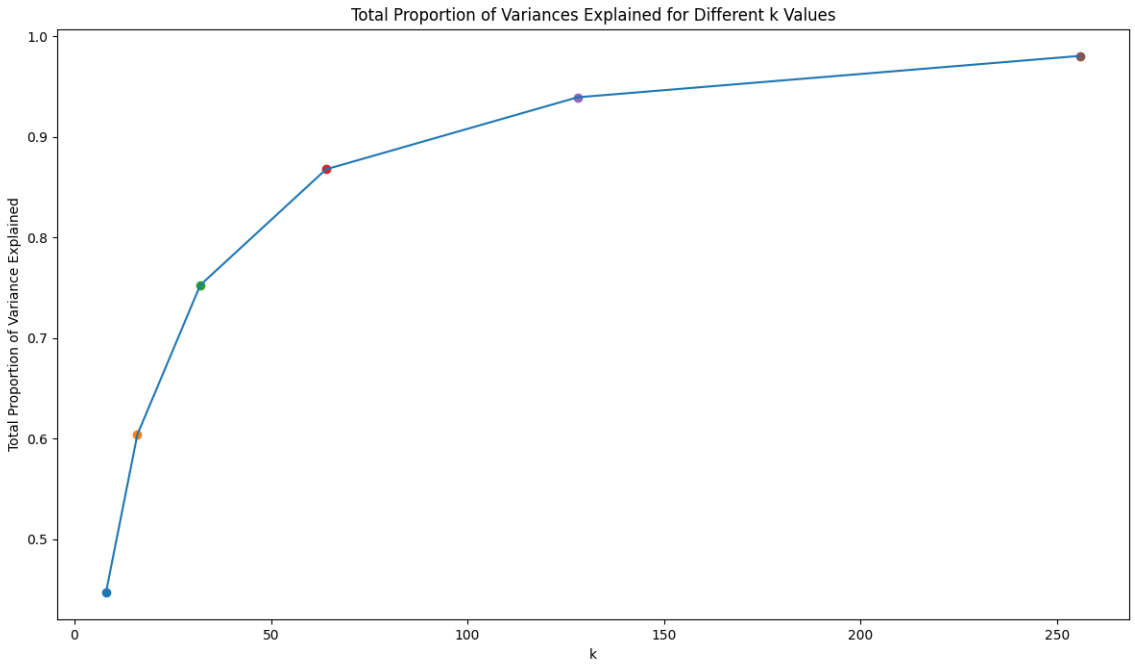

Figure 3 - Total Proportion of Variances Explained (PVE) for different k values

Discussion:

As k increases, the total PVE also increases because there are more principal components included. However, the rate of increase decreases as k increases. After a while, the chart starts to stabilize, which means that including more components would not significantly increase the accuracy.

1.3. Image Reconstruction Using First k Principal Components where k ∈ {1, 3, 5, 10, 50, 100, 200, 300}

We can analyze the images by multiplying the features of the first image with the principal components. We then multiply the obtained results, projections, with the transposed eigenvectors. When we add the mean to the multiplication result and reshape them, we get the images. We use the first k principal components to analyze and reconstruct the first image in the dataset for k ∈ {1, 3, 5, 10, 50, 100, 200, 300}.

Figure 4 shows the original and the reconstructed images. As we can see from the figure, as k increases, the image becomes more similar to the original image. After k = 10, the digit 7 was readable. Although using a higher k value, such as higher than 100, would further increase the accuracy, the improvement will not be very significant as the reconstructed image for k= 100, 200, and 300 are very close to the original image.

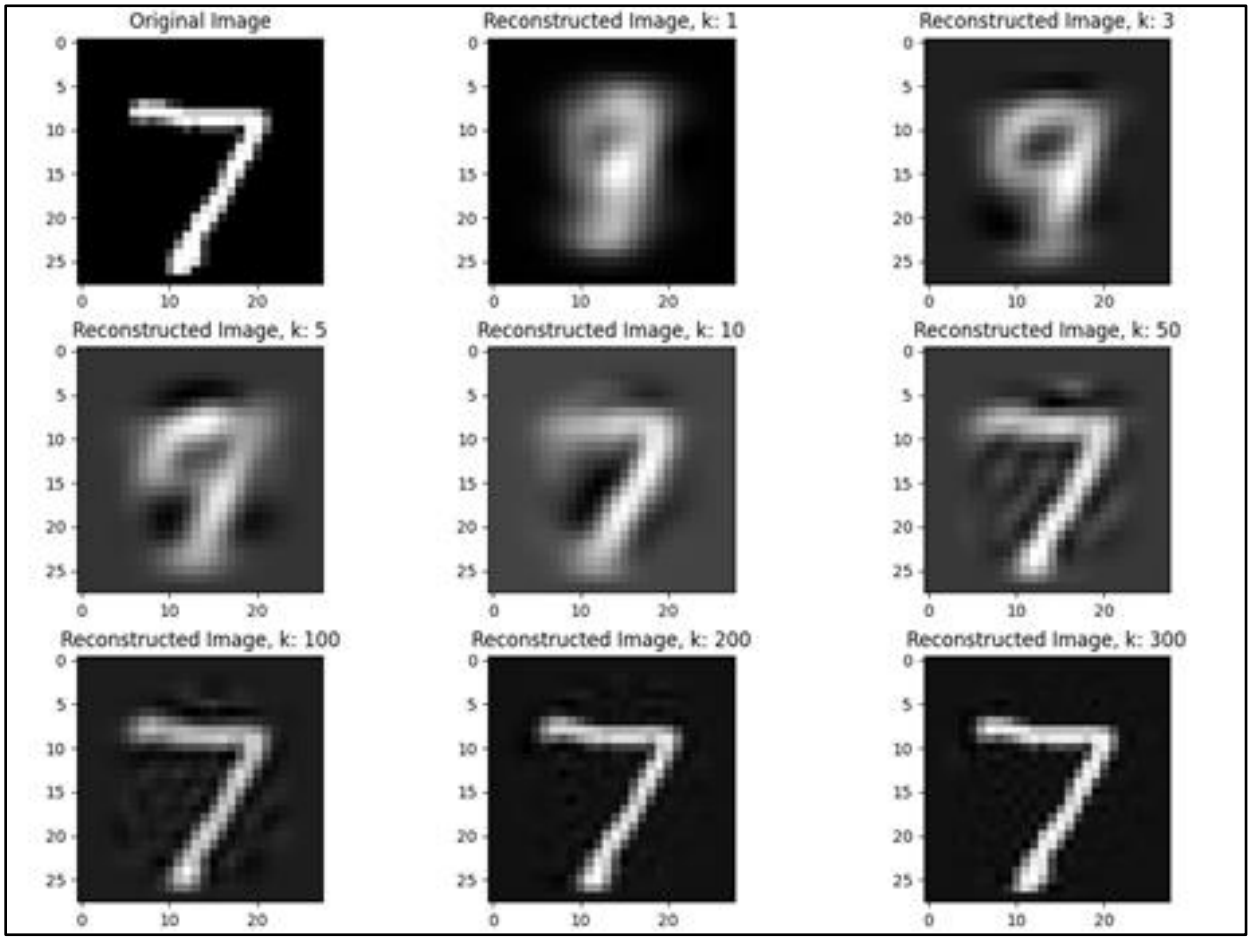

Figure 4- Original and reconstructed images

2. Linear & Polynomial Regression

In this part, we aim to understand which features of the houses are contributing more to the house prices for the dataset that contains information about house prices in King County. I use the files that contains four features of each house and the ground truth that contains the price of each house.

2.1. Derivation of the General Closed Form Solution for Multivariate Regression Model Using Ordinary Least Squares Loss Function

The derivation of the general closed form solution for multivariate regression model using ordinary least squares loss function is given as follows:

$$J_n = \|y - x\beta\|^2$$

$$J_n = (y - x\beta)^T (y - x\beta)$$

$$\frac{dJ_n}{d\beta} = -2x^T y + 2x^T x\beta$$

$$-2x^T y + 2x^T x\beta = 0$$

$$2x^T x\beta = 2x^T y$$

$$x^T x\beta = x^T y$$

$$\beta = (x^T x)^{-1} x^T y$$

The descriptions of the matrices are as follows:

- The β is the weight matrix.
- The x is the feature matrix.
- y is the matrix for ground truths.

2.2. Rank of $x^T x$

We added a column of 1s to the feature vector. The matrix's rank is 5, which tells us that there are five independent rows in the matrix. From this information, we can conclude that the matrix has an inverse, and we can find the weight matrix. It also says that all features are independent.

2.3. Linear Regression Model by Using Only "sqftliving" Feature

Using the formula derived for Question 2.1, I trained a linear regression model by using only "sqftliving" feature provided in the dataset. I use all of the dataset as the training set.

The following output reports the coefficients (weights) of the trained model. It also gives the the calculated Mean Square Error (MSE) value using my model's predictions and ground truth labels.

Output:

```
Part 2.3
Weights:  w0:  -43580.74309447433, w1:  280.62356789744837
MSE:  68351286833.03983
```

Figure 5- Linear regression model for living room size and house price

Figure 5 gives the plot of price vs. “sqftliving” along with my model's predictions on the same plot.

Discussion:

This model successfully estimates the house prices for houses with living room sizes less than 6000 square feet, but after that value, the house prices increased at a higher rate than living room sizes. This model could not successfully estimate house prices for huge living room sizes.

2.4. Polynomial Regression Model by Using Only “sqftliving” Feature

I use polynomial regression to train my model with the only provided feature “sqftliving”. I use the feature x_i and its powers x_i^2. The following output reports the coefficients (weights) of the trained model. It also gives the the calculated Mean Square Error (MSE) value using my model's predictions and ground truth labels. Again, I use all of the dataset as the training set.

Output:

```
Part 2.4

Weights:  w0:  199222.2793054831, w1:  67.99409468579503,
w2:  0.03858126093720162

MSE:  62975083210.965744
```

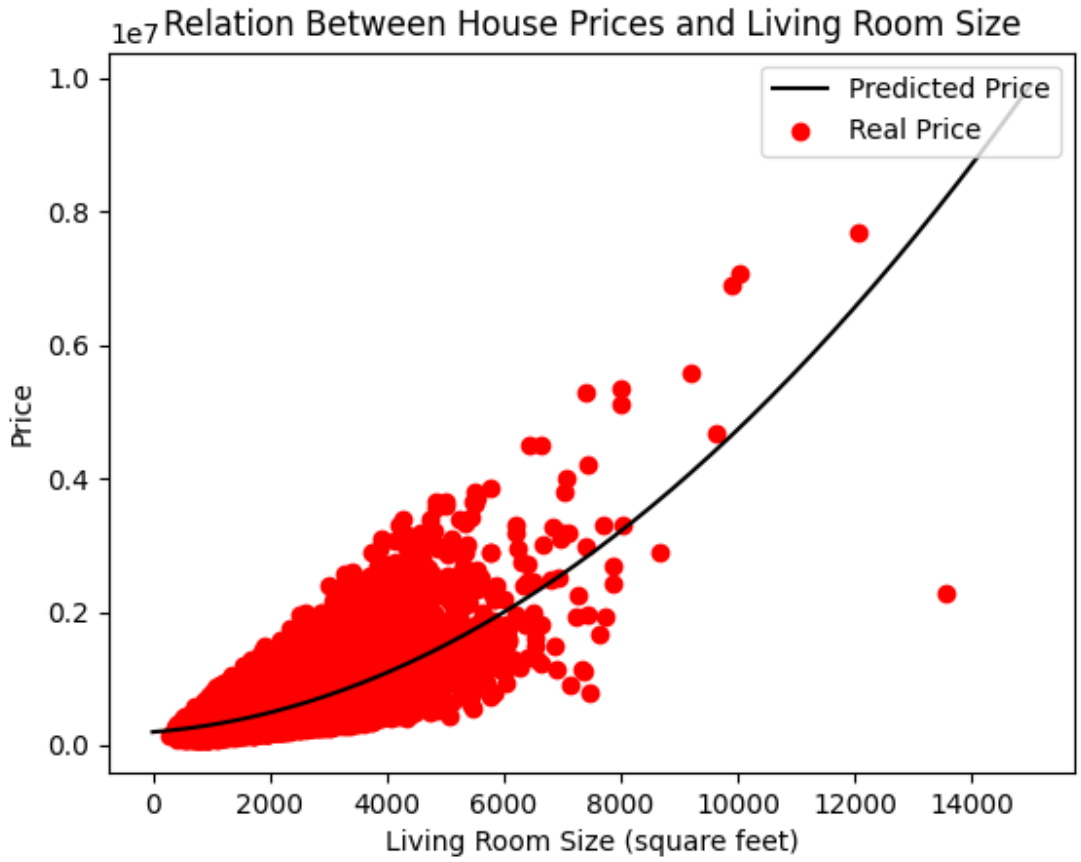

Figure 6 - Polynomial regression model for living room size and house prices

Figure 6 gives the graph of price vs. “sqftliving” along with my model's predictions on the same plot.

Discussion:

The polynomial regression model performs better than the linear regression model for estimating housing prices. The mean squared error (MSE) decreases compared to the linear regression model. This model can predict house prices better than the linear regression model for large living room sizes because as the living room size exceeds 8000 square feet, the house prices increase at a higher rate.

3. Logistic Regression

I wrote a program to detect credit card fraud using logistic regression. I use the dataset provided, a subset of the dataset for credit card fraud detection. The dataset includes twenty-eight features, V1, V2, ... V28, which are the principal components obtained from PCA transformation, and the amount feature that has not been transformed with PCA. The Class column is the response variable, and it takes the value 1 in case of fraud and 0 otherwise. The features and labels for the training and the test datasets are in four different files.

3.1. Full Batch Gradient Ascent Algorithm

I train my logistic regression model by using the full batch gradient ascent algorithm. The amount feature was normalized into the range (0,1) using this formula:

$$x_{normal} = \frac{x - \min(x)}{\max(x) - \min(x)}$$

I run the algorithm was run 1000 times over all of the training data. I tried different learning rates from the given logarithmic scale [10^{-5}, 10^{-4}, 10^{-3}, 10^{-2}, 10^{-1}] and choose the one that works best. The output below reports the accuracy and the confusion matrix using my model on the test set. It also reports the averages of precision, recall, negative predictive value (NPV), false positive rate (FPR), false discovery rate (FDR), F1 and F2 scores.

Output:

```
Full Batch Gradient Ascent, 0.0001 Step Size


Accuracy:  0.9858333333333333
Precision:  0.9764705882352941
Recall:  0.8469387755102041
Negative Predictive Values:  0.9865470852017937
False Positive Rate:  0.0018148820326678765
False Discovery Rate:  0.023529411764705882
F1 Score:  0.907103825136612
F2 Score:  0.870020964360587


                    Actual
Classifier       Fraud   Normal
  Fraud          83      2
  Normal         15      1100
```

Discussion:

The accuracy was high, but it can be misleading because the negative predictive value was too high. When we look at the F1 score and F2 score, we can better understand our model. The F1 and F2 scores can bi considered good because they are around 0.9. So, the model performs good.

3.2. Mini Batch and Stochastic Gradient Ascent Algorithms

I implemented mini-batch gradient ascent algorithm with batch size = 100 and stochastic gradient ascent algorithm to train my logistic regression model. I initialized all weights to random numbers drawn from a Gaussian distribution N(0, 0.01). I used the learning rate chosen in Question 3.1.

Mini Batch Gradient Ascent

The amount feature was normalized into the range (0,1) using this formula:

$$x_{normal} = \frac{x - \min(x)}{\max(x) - \min(x)}$$

I run the algorithm 1000 times over all of the training data. I shuffled the training data at each iteration and used batches of size 100 to update the weights. I used all of the training data at each iteration but with batches of size 100.

Output:

```
Mini Batch Gradient Ascent, 0.0001 Step Size


Accuracy:  0.9858333333333333
Precision:  0.9764705882352941
Recall:  0.8469387755102041
Negative Predictive Values:  0.9865470852017937
False Positive Rate:  0.0018148820326678765
False Discovery Rate:  0.023529411764705882
F1 Score:  0.907103825136612
F2 Score:  0.870020964360587


                    Actual
Classifier       Fraud   Normal
  Fraud          83      2
  Normal         15      1100
```

Discussion:

The accuracy was same with the full batch algorithm. Again, this accuracy can be misleading because the negative predictive value was too high. When we look at the F1 score and F2 score, we can better understand our model. This model had the same F1 and F2 scores with the previous model. However, this model can perform better on a larger test data. The reason for that is there are more updates and randomization of the data at each iteration. When we randomize, we avoid the model from learning the order, and it may cause the weights to converge faster. The random shuffling and more updates can help to avoid local minimas.

Stochastic Gradient Ascent

The amount feature was normalized into the range (0,1) using this formula:

$$x_{normal} = \frac{x - \min(x)}{\max(x) - \min(x)}$$

The algorithm was run 1000 times over the all of the training data. At each iteration, the weights were updated after each sample. So, this algorithm had the highest number of updates compared to the previous algorithms.

Output:

```
Stochastic Gradient Ascent, 0.0001 Step Size


Accuracy:  0.9858333333333333
Precision:  0.9764705882352941
Recall:  0.8469387755102041
Negative Predictive Values:  0.9865470852017937
False Positive Rate:  0.0018148820326678765
False Discovery Rate:  0.023529411764705882
F1 Score:  0.907103825136612
F2 Score:  0.870020964360587
```

	Actual	
Classifier	Fraud	Normal
Fraud	83	2
Normal	15	1100

Discussion:

The accuracy was same with the other two algorithms. Again, this accuracy can be misleading because the negative predictive value was too high. When we look at the F1 score and F2 score, we can better understand our model. The F1 and F2 scores were also same with the previous models.

There are more weight updates in this model, which can help us to avoid local maxima. When we compare all three algorithms for this test data, they have the same confusion matrix. However, stochastic gradient ascent can perform better on larger test data because there are more updates and the data are randomized. More updates can help us to avoid local minima.

3.3. Performance Metrics

When the test data is imbalanced and if a class is more frequent compared to the other class, then the accuracy, precision, and recall can deceive us. If the data is imbalanced, we can get high accuracy, but this does not mean that our model works properly because our model may predict each sample to one class. On the other hand, NPV, FPR, FDR, F1, and F2 rates give us more information than accuracy when the data is imbalanced. The reason behind this is that they also take into account the balance between classes. F1 and F2, for instance, take false positives and false negatives into account. So, even if the test data is imbalanced, they can give a proper interpretation.