Gün Kaynar
22101351

1- PCA & Eigenfaces

Q1.1

Figure 1: First 100 images from the images.csv file are shown

Pandas and numpy were used to get image pixels into arrays. Then, mean of all pixels was found, after that, the mean was subtracted from each pixel of images. So that, with this I have mean centered images.
The second step was to find the covariance matrix and the first 10 principal components of that matrix. For this I have used first 10 eigenvectors with first 10 eigenvalues. I have calculated the Proportion of variance explained by each of the first 10 components, and their total PVE.

Table 1: Proportion of variance explained by first 10 components were calculated.

Principal Component 1	0.2833447489537038
Principal Component 2	0.11027901264243307
Principal Component 3	0.09766803183987764
Principal Component 4	0.061015074869575255
Principal Component 5	0.03217828661264696

Principal Component 6	0.028607248398294885
Principal Component 7	0.020955561849916697
Principal Component 8	0.020521356816013837
Principal Component 9	0.018418297879458354
Principal Component 10	0.014091219567233436

The total proportion of the first 10 principal components was calculated as 0.6870788394291539. Each principal component was reshaped to 48x48 and plotted. The first 10 principal components can be seen below.

Figure 2: From left to right, principal component 1 to principal component 10

Q1.2

I have calculated the total PVE for k = 1, 10, 50, 100, 500. We expected to see an increase in PVE because the variance is brought by every principal component, and if we increase the number of principal components the variance will accumulate.

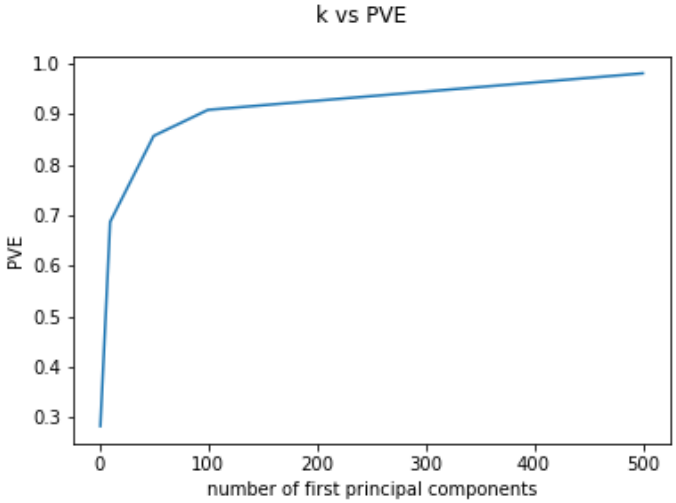

Figure 3: Total PVE increases as number of first principal components increase

Q1.3

I projected the data into a k dimensional subspace of the original feature space. X is 1 by 2304 matrix, and the pixels of the first image. Principal component is k by 2304 matrix. If P is the projected data into k-dimensions then, $P = XPC^T$

Here, we need to multiply projected data with PC. This will give us where data lies approximately. Reconstructed X = P PC. Reconstructed X will have dimensions 1 by 2304.

I have plotted the reconstruction of the first image for k = 1, 10, 50, 100, 500. The images can be seen below.

Figure 4: The original image

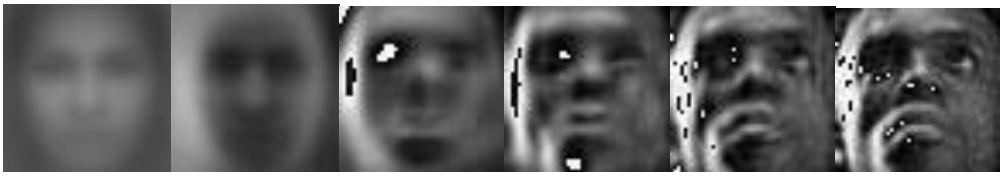

Figure 5: The reconstruction of the first image

The image gets closer to the original image as we increase the number of principal components used in the reconstruction. Because we get nearly all the variance when k = 500. As can be seen from the Figure 3 the PVE is nearly 1 which is exactly 1 in the original image.

2- Linear & Polynomial Regression

Q2.1

The cost function of ordinary least squares for multivariate linear regression was given below.

$J_n = ||y - X\beta||^2 = (y - X\beta)^T (y - X\beta)$

First, we need to show the open form of the equation:

$J_n = y^T y - y^T X\beta - X^T\beta^T y + X^T\beta^T X\beta$

Here, $y^T X\beta$ and $X^T\beta^T y$ are 1x1 matrices. So, the equation derives to:

$J_n = y^T y - 2\beta^T X^T y + X^T\beta^T X\beta$

If we take the derivative for β and set the equation to 0, we can solve for β.

$0 - X^T y + X^T X\beta = 0$

If we take -XT y to the right:

$(X^T X)\beta = X^T y$

Lastly, we solve this equation for β:

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

Matrices X and y are as follows:

$$\boldsymbol{X} = \begin{bmatrix} 1 & x_1^{(1)} & \dots & x_d^{(1)} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x_1^{(i)} & \dots & x_d^{(i)} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x_1^{(n)} & \dots & x_d^{(n)} \end{bmatrix} \quad \boldsymbol{y} = \begin{bmatrix} y^{(1)} \\ y^{(2)} \\ \vdots \\ y^{(n)} \end{bmatrix}$$

Where d is the number of features, n is the number of samples.

This is the vectorized implementation, and it works much better than iterating single sample approach, however X sometimes is not invertible. This can be overcome by simply pseudo-inverting it.

Q2.2

The rank of the matrix (X^TX) is 13. This matrix is invertible since n>d where d is the number of features and n is the number of samples. At this point, we have no clue whether the features are linearly independent or not. So, we are not sure about the invertibility of the matrix (X^TX).

Q2.3

I have trained a model with using only LSTAT feature, the whole dataset was used as training set. I have added bias to the training set so that our model can find W_0. After adding bias, we have a matrix of shape 506 by 2. Our y matrix is of shape 506 by 1.
I have applied the closed form solution from the Question 2.1, and the coefficients were calculated as:
W_0 = 34.55384088
W_1 = -0.95004935
Where W_0 is the weight of the bias and W_1 is the weight of the LSTAT feature.

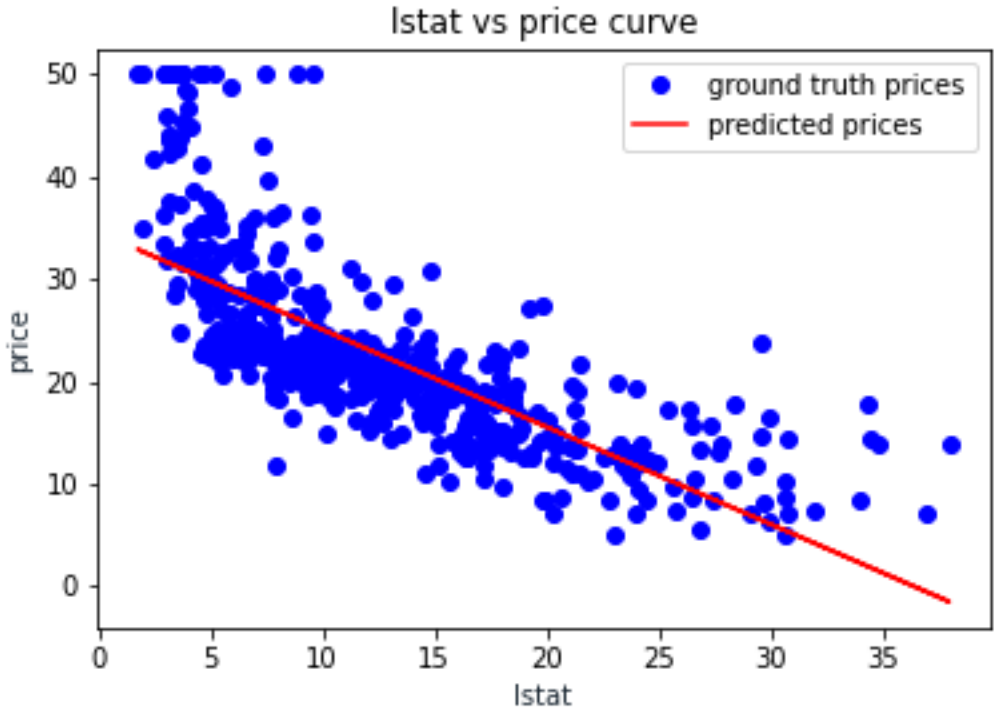

Figure 6: Predicted prices with calculated coefficients and ground truth prices are shown

The MSE was calculated as 38.482967229894165.

By looking to the Figure 6 and the MSE, we can conclude that our model is underfitting to the data. There are features that we need to take into account, and also linear model is not fit to LSTAT feature. We need to increase the complexity of our model by making our $h(x)$ polynomial.

Q2.4

I have trained the model again with only LSTAT feature, but this time after adding bias, I also added $(\text{LSTAT})^2$ feature. The whole dataset was used as training set and the coefficients were calculated as:

$W_0 = 42.86200733$

$W_1 = -2.3328211$

$W_2 = 0.04354689$

Where W_0 is the weight of the bias, W_1 is the weight of the LSTAT feature and W_2 is the weight of the $(\text{LSTAT})^2$ feature.

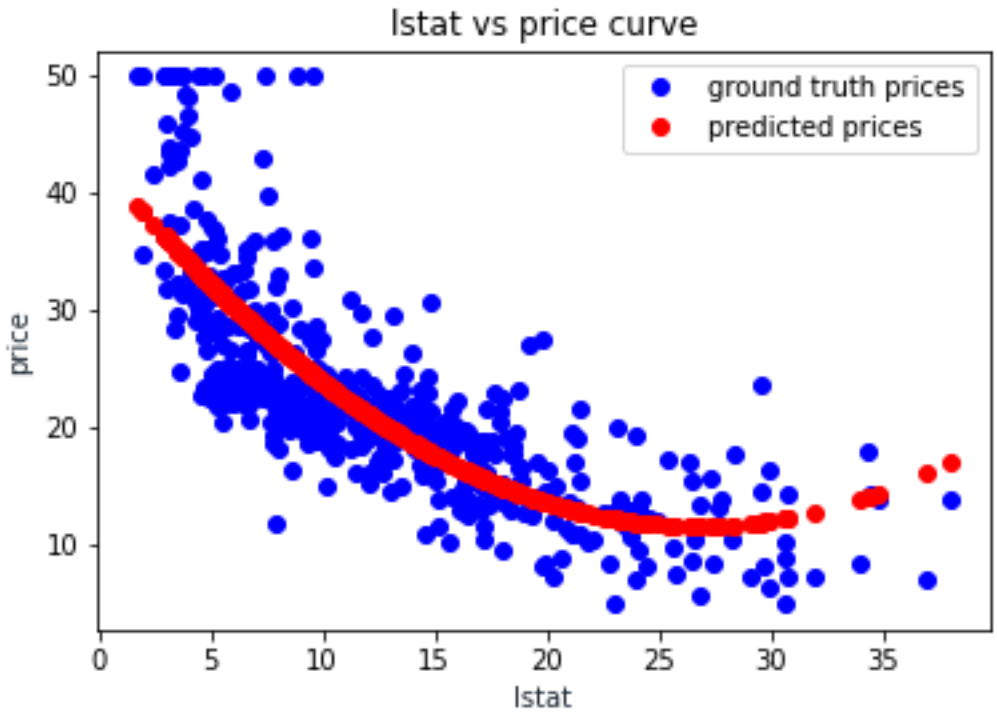

Figure 7: Predicted prices with calculated coefficients and ground truth prices are shown

The MSE was calculated as 30.33052007585372.

We can see that MSE has dropped 21% with the more complex model. Even though in Figure 7, the new model represents better than the first one, we need to increase the number of features used. Because the MSE seems very high.

3- Logistic Regression

Q3.1

I have normalized all features for the gradient ascent with min-max normalization. I have trained a full-batch gradient ascent logistic regression model. In this model, the algorithm tries to increase the log-likelihood. I have tried different learning rates and the log likelihood in 1000 iterations for different learning rates are shown below.

Table 2: The log-likelihoods after 1000 iterations for different learning rates

Learning rate	Log-likelihood after 1000 iterations
0.1	-2.4966976826008159312
0.01	-0.62707493977271158356
0.001	-0.6349472119633693431
0.0001	-0.65772844305975249117
0.00001	-0.66590861175284216003

The best learning rate for my model was 0.01. Then, I have tested my model on test-set.

The accuracy in this model was found 0.676. Other performance metrics can be seen below.
Precision = 0.570
Recall = 0.652
NPV = 0.760
FPR = 0.309
FDR = 0.430
F1 score = 0.608
F2 score = 0.634

The confusion matrix is shown below.
Table 3: Confusion matrix of our model's predictions on test-set

	Predicted Positive	Predicted Negative
Actual Positive	45	24
Actual Negative	34	76

Q3.2

I have trained the same model but this time I have initialized w0, w1, w2 and w3 to random numbers in Gaussian distribution N(0.0,0.01). This time I did not use full-batch gradient ascent but mini-batch gradient ascent and stochastic gradient ascent. My mini-batch size was 100.
I have then tested my model on test-set. The performance metrics of my mini-batch gradient ascent model is as follows:
Accuracy = 0.413
Precision = 0.397
Recall = 1.000
NPV = 1.000
FPR = 0.955
FDR = 0.603
F1 score = 0.568
F2 score = 0.767

The confusion matrix can be seen below.
Table 4: Confusion matrix of our model's predictions on test-set

	Predicted Positive	Predicted Negative
Actual Positive	69	0
Actual Negative	105	5

Then I tested my stochastic batch gradient ascent model and performance metrics are shown below.
Accuracy = 0.676
Precision = 0.570
Recall = 0.652
NPV = 0.760
FPR = 0.309
FDR = 0.430
F1 score = 0.608
F2 score = 0.634

Table 5: Confusion matrix of our model's predictions on test-set

	Predicted Positive	Predicted Negative
Actual Positive	45	24
Actual Negative	34	76

Q3.3

F1 score is more informative when we want balanced precision and recall.
F2 score is more informative when we want balanced precision and recall but recall is more important for us. i.e., if we want our model to find all positive classes but we do not value that much that our predictions on positive classes are wrong.

If positive and negative classes are not equally distributed, to say if there is a class imbalance, then accuracy, precision and recall become uninformative. In this case, if precision for negative class is more important for us then NPV is informative, if precision for both classes is important for us then FDR will become informative.