ESE 542 HOMEWORK 6

Anshul Tripathi

Problem 1a.

In this problem, the following code was used:

```
#Load the original data
data = [[0, 1],
        [1, 1],
        [2, 1],
        [2, 3],
        [3, 2],
        [3, 3],
        [4, 5]]
#Scale the data
scaler = StandardScaler()
standardized_data = scaler.fit_transform(data)
print("-------This is the standardized data------")
print(standardized_data)
print(" ")
#Do the PCA
pca = PCA(n_components=2)
projection = pca.fit_transform(standardized_data)
#Output sorted components
print("--------These are the first two principal components by default in
sorted order--------------")
print(pca.components_)
print(" ")
#Output transformed dataset
print("-------New transformed dataset------")
print("Note that the first column here is the new transformed dataset using the
 first principal component")
print(projection)
print(" ")
#Manually verify transformed dataset
print("--------Manually doing the transformation-------------")
print(np.matmul(standardized_data, pca.components_))
print(" ")
```

Listing 1. Problem 1a Code

The outputs of this program are shown on the next page. The outputs contain the answers to the question.

```
-------This is the standardized data------
[[-1.720618   -0.92827912]
 [-0.91766294 -0.92827912]
 [-0.11470787 -0.92827912]
 [-0.11470787  0.51571062]
 [ 0.6882472  -0.20628425]
 [ 0.6882472   0.51571062]
 [ 1.49120227  1.95970037]]

--------These are the first two principal components by default in sorted order
    --------------
[[ 0.70710678  0.70710678]
 [ 0.70710678 -0.70710678]]

-------New transformed dataset------
Note that the first column here is the new transformed dataset using the first
    principal component

[[-1.87305312 -0.5602682 ]
 [-1.30527815  0.00750678]
 [-0.73750317  0.57528175]
 [ 0.28355177 -0.44577319]
 [ 0.34079927  0.63252925]
 [ 0.85132674  0.12200178]
 [ 2.44015666 -0.33127818]]

--------Manually doing the transformation-------------
[[-1.87305312 -0.5602682 ]
 [-1.30527815  0.00750678]
 [-0.73750317  0.57528175]
 [ 0.28355177 -0.44577319]
 [ 0.34079927  0.63252925]
 [ 0.85132674  0.12200178]
 [ 2.44015666 -0.33127818]]
```

Listing 2. Problem 1a Output

As can be seen in this output, the stardized data, the two derived principal components in sorted order and the new transformed dataset are all available above.

Problem 1b.

In this problem, the following code was used:

```
    #Load the original data
    data = [[0, 1],
            [1, 1],
            [2, 1],
            [2, 3],
            [3, 2],
            [3, 3],
            [4, 5]]
    #Do the PCA
    pca = PCA(n_components=2)
    projection = pca.fit_transform(data)
    #Output sorted components
    print("--------These are the first two principal components by default in
    sorted order--------------")
    print(pca.components_)
    print(" ")
    #Output transformed dataset
    print("-------New transformed dataset------")
    print("Note that the first column here is the new transformed dataset using the
     first principal component")
    print("Note here that fit_transform centers the data")
    print(projection)
    print(" ")
    #Manually verify transformed dataset
    print("--------Manually doing the transformation-------------")
    print(np.matmul(data, pca.components_))
    print(" ")
    print("Therefore, PCA is not scacle invariant")
```

Listing 3. Problem 1b Code

The outputs of this program are shown on the next page. The outputs contain the answers to the question. Note that the first columns of the transformed datasets use only the first principal components.

```
--------These are the first two principal components by default in sorted order
    --------------
[[ 0.65908697  0.75206673]
 [ 0.75206673 -0.65908697]]

-------New transformed dataset------
Note that the first column here is the new transformed dataset using the first
    principal component
Note here that fit_transform centers the data
[[-2.37927216 -0.76417402]
 [-1.72018519 -0.01210729]
 [-1.06109821  0.73995944]
 [ 0.44303524 -0.57821451]
 [ 0.35005549  0.83293919]
 [ 1.10212221  0.17385221]
 [ 3.26534264 -0.39225501]]

--------Manually doing the transformation-------------
[[ 0.75206673 -0.65908697]
 [ 1.4111537   0.09297975]
 [ 2.07024068  0.84504648]
 [ 3.57437413 -0.47312747]
 [ 3.48139438  0.93802623]
 [ 4.2334611   0.27893925]
 [ 6.39668153 -0.28716797]]

Therefore, PCA is not scacle invariant
```

Listing 4. Problem 1b Output

As can be seen in this output, the two derived principal components in sorted order and the new transformed dataset (with and without centering) are all available above.Note that the first columns of the transformed datasets use only the first principal components. Hence, note also that the PCA is not scale invariant.

Problem 2a.

In this problem, the following code was used:

```
#Load the dataset
poly_data = pd.read_csv("poly_data.csv",
                        header = None,
                        delim_whitespace=True)
poly_data
X = poly_data.iloc[:, 0].values
Y = poly_data.iloc[:, 1].values
limit = 41
errors = []
for d in range(1, limit):
  #Create polynomial feeeatures
  poly = PolynomialFeatures(degree = d)
  X_poly = poly.fit_transform(X.reshape(-1, 1))
  #Create Regression Model
  model = LinearRegression()
  #Define cross validation method
  cv = KFold(n_splits=10,
             random_state=1,
             shuffle=True)
  #Perform k-fold CV
  scores = cross_val_score(model,
                           X_poly, Y,
                           scoring='neg_mean_squared_error',
                           cv=cv, n_jobs = -1) * -1 #Multiply by -1 to make it
positive
  #Append mean of scores
  errors.append(np.mean(scores))

plt.title("Polynomial Degree Vs Mean Squared Error")
plt.xlabel("Polynomial Degree")
plt.ylabel("Mean Squared Error")
best_degree = errors.index(min(errors)) + 1
plt.plot([i for i in range(1, limit)], errors, color = "blue")
plt.savefig("2a")
plt.show()
print("The best fitting polynomial model is of degree", best_degree)
```

Listing 5. Problem 2a Code

The outputs of this program are shown on the next page. The outputs contain the answers to the question.

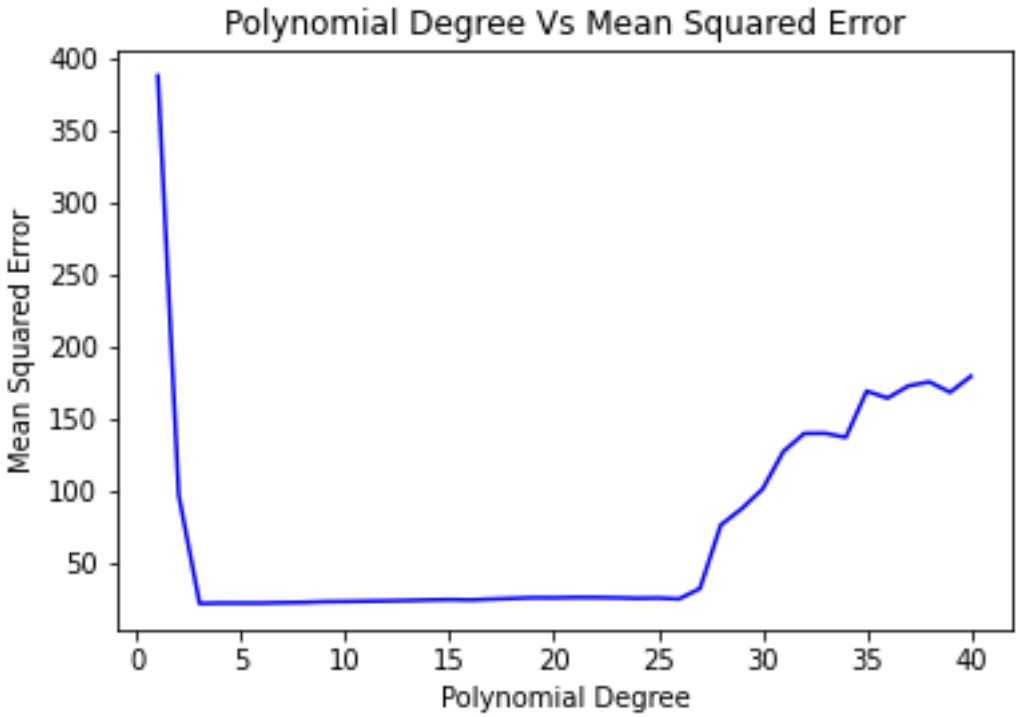

Figure 1. Polynomial Degree Vs Mean Squared Error

```
The best fitting polynomial model is of degree 3
```

Listing 6. Problem 1b Output

As can be seen in this output, all polynomial models between degrees 1 and 40 are tested. During cross validation, for each polynomial degree, we use $k = 10$ folds, evaluate each of their negative mean squared errors on each fold(thereby obtaining 10 such values), multiply them by -1 so that they become positive mean squared errors, and take the mean of these 10 values in order to represent the average mean squared error for that degree polynomial. We then plot this average mean squared error against the polynomial degree, as can be seen in Figure 1. As can be seen in the output as well, the degree 3 polynomial seemed to fit the data the best.

Problem 2b.

In this problem, the following code was used.

```
    poly = PolynomialFeatures(degree = best_degree)
    X_poly = poly.fit_transform(X.reshape(-1, 1))
    model = LinearRegression().fit(X_poly, Y)
    plt.title("Polynomial Regression Degree 3")
    plt.xlabel("Feature")
    plt.ylabel("Label")
    plt.scatter(X, Y, color = 'blue', label = "Real Label")
    plt.scatter(X, model.predict(X_poly), color = 'red', label = 'Regression
    Prediction')
    plt.legend()
    plt.savefig("2b")
    plt.show()
    print("The coefficients are: ", model.coef_)
```

Listing 7. Problem 2b Code

The outputs of this program were as follows:

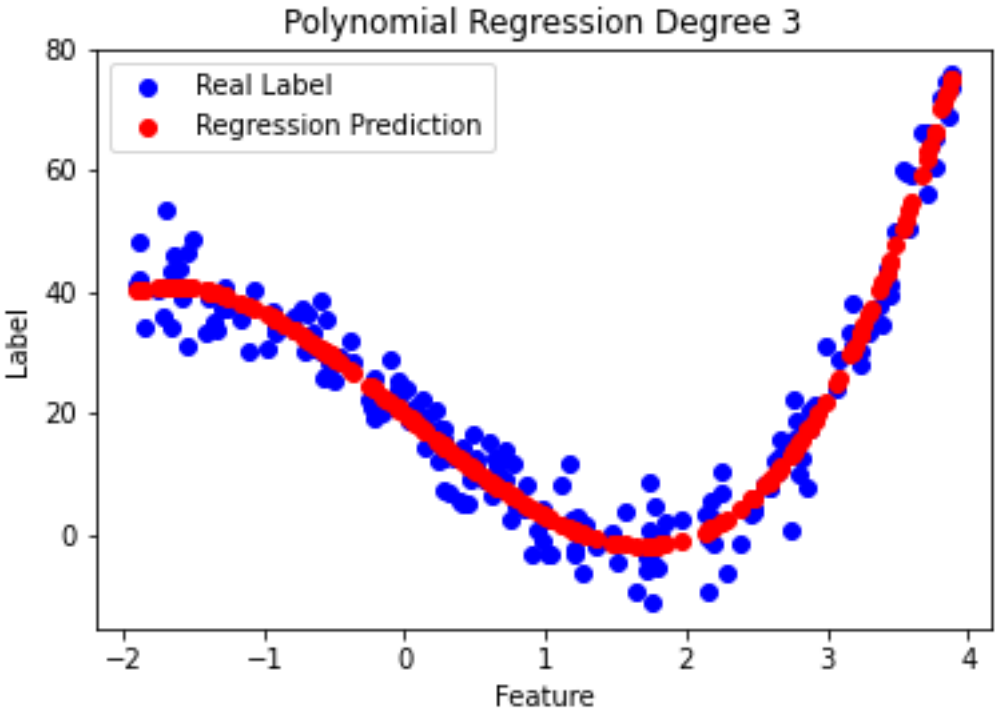

Figure 2. Polynomial Regression Degree 3 Vs Real Labels

```
The coefficients are:  [  0.         -19.03450815  -0.10790684   2.24897906]
```

Listing 8. Problem 2b Output

The coefficients for the 3rd degree polynomial (which was the best fit) as well as the scatter plot of the real labels vs predicted labels is available in the program output above.

Problem 3a. We know the following:

$$\begin{aligned}
\mathbb{P}(Y=+1) = \mathbb{P}(Y=-1) &= \frac{1}{2} \\
\mathbb{P}(X=x|Y=+1) &= \frac{1}{\sqrt{2\pi}} e^{\frac{-(x-5)^2}{2}} \\
\mathbb{P}(X=x|Y=-1) &= \frac{1}{\sqrt{2\pi}} e^{\frac{-(x+5)^2}{2}}
\end{aligned} \tag{1}$$

Therefore, the conditional probabilities can be rewritten as follows:

$$\begin{aligned}
\mathbb{P}(X=x|Y=+1) &= \frac{1}{\sqrt{2\pi}} e^{\frac{-(x-5\cdot(1))^2}{2}} \\
\mathbb{P}(X=x|Y=-1) &= \frac{1}{\sqrt{2\pi}} e^{\frac{-(x-5\cdot(-1))^2}{2}}
\end{aligned}$$

Therefore, we can write a generalized version of the conditional probability as follows:

$$\mathbb{P}(X=x|Y=y) = \frac{1}{\sqrt{2\pi}} e^{\frac{-(x-5y)^2}{2}}$$

Now, we know from elementary probability theory that the joint distribution can be rewritten:

$$\begin{aligned}
\mathbb{P}(X=x, Y=y) &= \mathbb{P}(X=x|Y=y) \cdot \mathbb{P}(Y=y) \\
&= \frac{1}{\sqrt{2\pi}} e^{\frac{-(x-5y)^2}{2}} \cdot \frac{1}{2} \\
&= \frac{1}{2\sqrt{2\pi}} e^{\frac{-(x-5y)^2}{2}}
\end{aligned}$$

Problem 3b.

The following two equations were plotted:

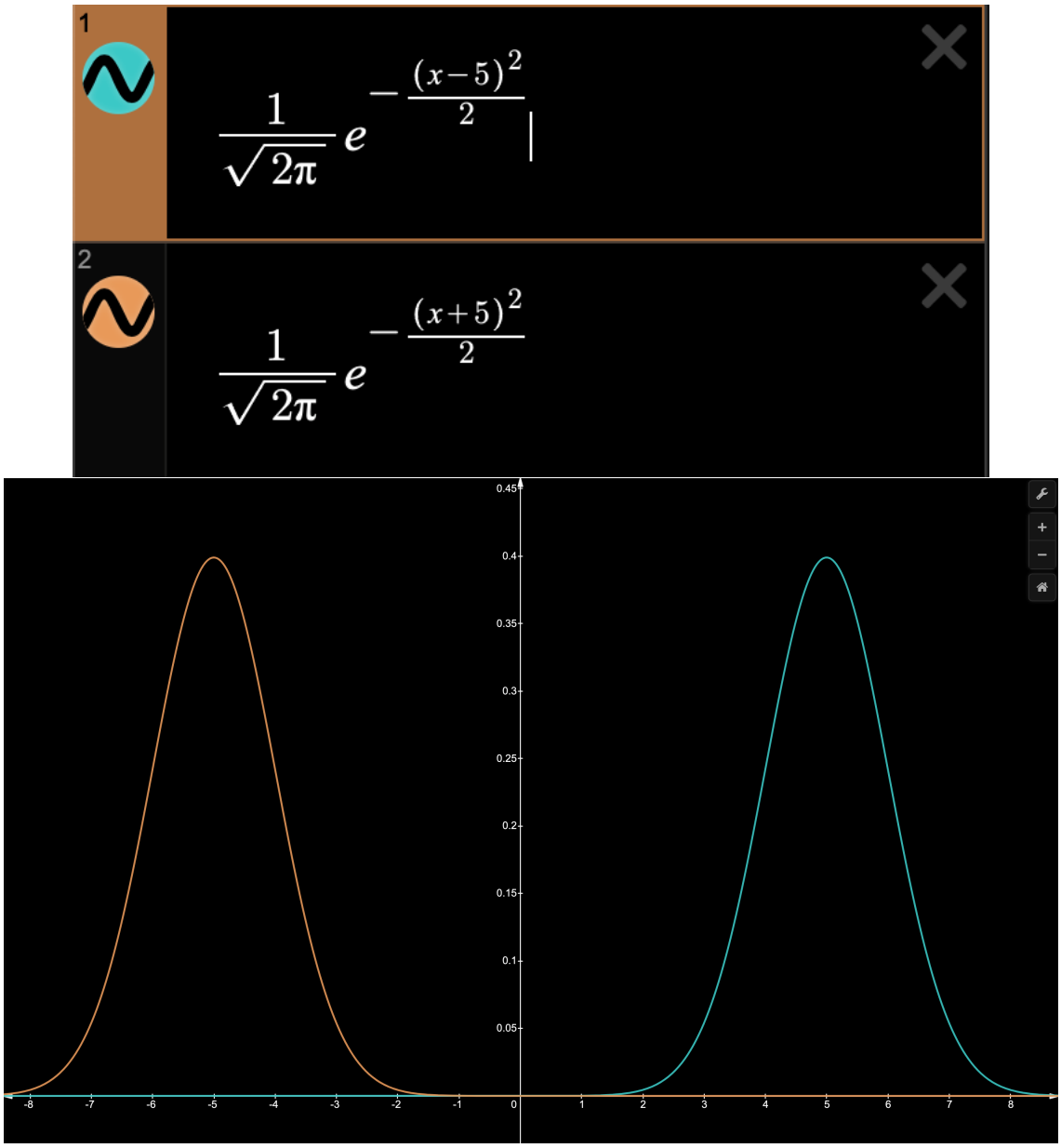

Figure 3. Equations and Plots

As may be seen here, theplots are symmetric around the origin, and have peaks at $x = +5$ and $x = -5$ for equations 1 and 2 respectively.

Problem 3c.

We may write the Bayes' optimal classifier as follows:

$$h^*(x) = \underset{y\in\{-1,+1\}}{\operatorname{argmin}} \mathbb{P}(Y = y|X = x)$$

By Bayes' Rule, we get

$$= \underset{y\in\{-1,+1\}}{\operatorname{argmin}} \frac{\mathbb{P}(X = x|Y = y) \cdot \mathbb{P}(Y = y)}{\mathbb{P}(X = x)}$$

Since the denominator doesn't depend on y

$$= \underset{y\in\{-1,+1\}}{\operatorname{argmin}} \mathbb{P}(X = x|Y = y) \cdot \mathbb{P}(Y = y)$$

Since $P(Y = y) = \frac{1}{2}$ for all values of y

$$= \underset{y\in\{-1,+1\}}{\operatorname{argmin}} \mathbb{P}(X = x|Y = y) \cdot \frac{1}{2}$$

Since the constant $\frac{1}{2}$ has no dependence on y

$$= \underset{y\in\{-1,+1\}}{\operatorname{argmin}} \mathbb{P}(X = x|Y = y)$$

$$= \underset{y\in\{-1,+1\}}{\operatorname{argmin}} \frac{1}{\sqrt{2\pi}} e^{\frac{-(x-5y)^2}{2}} = \begin{cases} -1 & \text{if } \frac{1}{\sqrt{2\pi}} e^{\frac{-(x+5)^2}{2}} > \frac{1}{\sqrt{2\pi}} e^{\frac{-(x-5)^2}{2}} \\ +1 & \text{if } \frac{1}{\sqrt{2\pi}} e^{\frac{-(x+5)^2}{2}} \leq \frac{1}{\sqrt{2\pi}} e^{\frac{-(x-5)^2}{2}} \end{cases}$$

Note here that ties are being broken arbitrarily. Therefore, for the classifier, we have

$$h^*(x) = \begin{cases} -1 & \text{if } \frac{1}{\sqrt{2\pi}} e^{\frac{-(x+5)^2}{2}} > \frac{1}{\sqrt{2\pi}} e^{\frac{-(x-5)^2}{2}} \\ +1 & \text{if } \frac{1}{\sqrt{2\pi}} e^{\frac{-(x+5)^2}{2}} \leq \frac{1}{\sqrt{2\pi}} e^{\frac{-(x-5)^2}{2}} \end{cases}$$

Following the hint, we must convert these conditions into threshold values for x. While we may do this algebraically, we can also refer to the plot in Problem 3c for an easier workaround. It is evident from the equations that have been plotted that $\mathbb{P}(X = x|Y = -1) > \mathbb{P}(X = x|Y = +1)$ when $x < 0$, and conversely, $\mathbb{P}(X = x|Y = -1) \leq \mathbb{P}(X = x|Y = +1)$ when $x \geq 0$.

Therefore, we may finally rewrite the classifier as follows:

$$h^*(x) = \begin{cases} -1 & \text{if } x < 0 \\ +1 & \text{if } x \geq 0 \end{cases}$$

Problem 3d.

We can obtain the classification error rate of the Bayes' Optimal classifer as follows:

$$\mathbb{P}_{(x,y)\sim P}(h^*(x) \neq y) = \mathbb{E}_{(x,y)\sim P}[\mathbb{1}(h^*(x) \neq y)]$$

Give our classifier, there are two possible error cases. First error case is when $x \geq 0$ and the real label is -1. Our classifier would classify this as $+1$. Conversely, the second error case is when $x < 0$ and the real label is $+1$. Our classifier would classify this as -1. Therefore, we can write the above expression as follows:

$$\begin{aligned}
\mathbb{P}_{(x,y)\sim P}(h^*(x) \neq y) &= \mathbb{E}_{(x,y)\sim P}[\mathbb{1}(h^*(x) \neq y)] \\
&= \mathbb{P}(x \geq 0, y = -1) + \mathbb{P}(x < 0, y = +1) \\
&\text{Via conditional probability, we have} \\
&= \mathbb{P}(y = -1) \cdot \mathbb{P}(x \geq 0 | y = -1) + \mathbb{P}(y = +1) \cdot \mathbb{P}(x < 0 | y = -1) \\
&\text{For the conditionals we have } \sigma = 1, \mu_{-1} = -5, \mu_{+1} = +5 \\
&\text{Thus, } P(x \geq 0 | y = -1) = P(x < 0 | y = +1) = 1 - \Phi(5) \\
&\text{Also recal that } P(Y = y) = \frac{1}{2} \forall y \in \{-1, +1\}. \text{ So we have,} \\
&= \frac{1}{2} \cdot (1 - \Phi(5)) + \frac{1}{2} \cdot (1 - \Phi(5)) \\
&= 1 - \Phi(5)
\end{aligned}$$

Problem 3e.

Recall that

$$\begin{aligned}
\mathbb{P}(Y=+1|X=x) &= \frac{\mathbb{P}(X=x|Y=+1)\cdot\mathbb{P}(Y=+1)}{\mathbb{P}(X=x)} \\
&= \frac{\mathbb{P}(X=x|Y=+1)\cdot\mathbb{P}(Y=+1)}{\mathbb{P}(X=x|Y=+1)\cdot\mathbb{P}(Y=+1)+\mathbb{P}(X=x|Y=-1)\cdot\mathbb{P}(Y=-1)} \\
&= \frac{\frac{1}{\sqrt{2\pi}}e^{\frac{-(x-5)^2}{2}}\cdot\frac{1}{2}}{\frac{1}{\sqrt{2\pi}}e^{\frac{-(x-5)^2}{2}}\cdot\frac{1}{2}+\frac{1}{\sqrt{2\pi}}e^{\frac{-(x+5)^2}{2}}\cdot\frac{1}{2}} \\
&= \frac{\frac{1}{2\sqrt{2\pi}}\left(e^{\frac{-(x-5)^2}{2}}\right)}{\frac{1}{2\sqrt{2\pi}}\left(e^{\frac{-(x-5)^2}{2}}+e^{\frac{-(x+5)^2}{2}}\right)} \\
&= \frac{e^{\frac{-(x-5)^2}{2}}}{e^{\frac{-(x-5)^2}{2}}+e^{\frac{-(x+5)^2}{2}}}
\end{aligned}$$

Dividing numerator and denominator by $e^{\frac{-(x-5)^2}{2}}$

$$\begin{aligned}
&= \frac{\frac{e^{\frac{-(x-5)^2}{2}}}{e^{\frac{-(x-5)^2}{2}}}}{\frac{e^{\frac{-(x-5)^2}{2}}}{e^{\frac{-(x-5)^2}{2}}}+\frac{e^{\frac{-(x+5)^2}{2}}}{e^{\frac{-(x-5)^2}{2}}}} \\
&= \frac{1}{1+e^{\frac{(x-5)^2-(x+5)^2}{2}}} \\
&= \frac{1}{1+e^{\frac{x^2-10x+25-x^2-10x-25}{2}}} \\
&= \frac{1}{1+e^{\frac{-20x}{2}}} \\
&= \frac{1}{1+e^{-10x}}
\end{aligned}$$

Recall that the logistic regression data distribution is of the form

$$\mathbb{P}(Y=+1|X=x) = \frac{1}{1+e^{-\beta_0-\beta_1 x}}$$

Therefore, we see that:

$$-\beta_0-\beta_1 x = -10x$$

This equality is therefore upheld when:

$$(\beta_0,\beta_1) = (0,10)$$

Problem 4. In this problem, I am tasked with showing that k-means is suboptimal. Recall that the k-means algorithm works as follows:

(1) Pick a random set of points as cluster centers
(2) Assign each point to its closest cluster
(3) Re-calculate the cluster center as the mean of all points assigned to that cluster. If convergence is obtained and the none of the cluster centers change from their previous value, finish the algorithm and return the current set of clusters.

Following, the second hint, let us begin with the following four points on the 2-dimensional plane with 2 clusters:

- Point A: $(-\sqrt{t}, 1)$
- Point B: $(\sqrt{t}, 1)$
- Point C: $(\sqrt{t}, -1)$
- Point D: $(-\sqrt{t}, -1)$

As may be noted, these four points trace out - in clockwise direction starting at the top left - a rectangle of breadth $2\sqrt{t}$ and height 2, centered at the origin $(0, 0)$. It is vacuously noted that points along the height are strictly closer to each other than points along the breadth for any $t > 1$. This may be proved by observing that the distance between points A and D is the same as the distance between points B and D, which is 2 units, where as the distance between points A and B is the same as the distance between points C and D, which is $2\sqrt{t}$ units, which is stricly $> 2 \forall t > 1$.Therefore, since most similar points should be grouped together, it is evident that the optimal groups would be as follows:

- For the first cluster,
  - Optimal centroid is the midpoint of points in G_1, ie $(\sqrt{t}, 0)$
  - Set of points in this cluster is $G_1 = \{(\sqrt{t}, 1), (\sqrt{t}, -1)\}$
  - Contribution to the objective is the sum of squared distances of each point in G_1 from the current center of $G_1 = 1^2 + 1^2 = 2$
- For the second cluster,
  - Optimal centroid is the midpoint of points in G_2, ie $(-\sqrt{t}, 0)$
  - Set of points in this cluster is $G_2 = \{(-\sqrt{t}, 1), (-\sqrt{t}, -1)\}$
  - Contribution to the objective is the sum of squared distances of each point in G_2 from the current center of $G_1 = 1^2 + 1^2 = 2$

Thus, the optimal objective evaluates to $2 + 2 = 4 = OPT$. However, consider the case where by some stroke of bad luck, the initial random instantiations of the cluster center are $(0, 1)$ and $(0, -1)$. In this case,

- For the first cluster,
  - Set of points in this cluster is $G_1 = \{(-\sqrt{t}, 1), (\sqrt{t}, 1)\}$
  - Contribution to the objective is the sum of squared distances of each point in G_1 from the current center of $G_1 = (\sqrt{t})^2 + (\sqrt{t})^2 = 2t$
- For the second cluster,
  - Set of points in this cluster is $G_2 = \{(-\sqrt{t}, -1), (\sqrt{t}, -1)\}$
  - Contribution to the objective is the sum of squared distances of each point in G_2 from the current center of $G_2 == (\sqrt{t})^2 + (\sqrt{t})^2 = 2t$

Thus, the objective evaluates to $2t + 2t = 4t = t \cdot OPT$. This matches the result indicated by Hint 1. Now, we must generalize to $p \geq 2$ dimensions, n data points, and k clusters as follows; consider the space of all possible points in $\mathbb{R}^p$ of the form:

- $((4n-1)\sqrt{t}, 1, 0, \cdots, 0)$
- $((4n-1)\sqrt{t}, -1, 0, \cdots, 0)$
- $((4n+1)\sqrt{t}, 1, 0, \cdots, 0)$
- $((4n+1)\sqrt{t}, -1, 0, \cdots, 0)$

Here, $t > 1, n \in \mathbb{Z}, 0 \leq n \leq N$. As can be inferred from the 2-dimensional example, the optimal set of centers lie on the x-axis of the form:
$\{((4n-1)\sqrt{t}, 0), ((4n+1)\sqrt{t}, 0)\} \forall n \in \mathbb{Z}, 0 \leq n \leq N$.

In this case, there are $k = 2N$ clusters and $4N$ points. For each cluster, the optimal contribution to the objective is 2. Therefore, $OPT = 2 \cdot 2N = 4N$. However, consider the bad center initialization of the form: $\{(4n\sqrt{t}, 1), (4n\sqrt{t}, -1)\} \forall n \in \mathbb{Z}, 0 \leq n \leq N$. In this case, each cluster contributes $2t$ to the objective, and therefore the total objective evaluates to $2t \cdot 2N = 4tN = t \cdot 4N = t \cdot OPT$.

In case there are $4N + 1, 4N + 2$ or $4N + 3$ data points, we may simply choose to add those offset number of $1, 2$ or 3 data points at some extremely distant locations from any of the otherwise $4N$ neatly arranged points, along with $1, 2$ or 3 other clusters at those distant locations. Thus, this completes the generalization for all values of k, n, p, ie for all quantities of clusters, data points, or dimensions.

Problem 5.

In this problem, the following code was used:

```
from sklearn.datasets import fetch_lfw_people
faces = fetch_lfw_people(min_faces_per_person=60)
_, height, width = faces.images.shape
X = faces.data
n_features = X.shape[1]
pca = PCA(n_components = 150,
          svd_solver="randomized").fit(X)
def custom_plot(images, r = 0, c = 0, prefix = "", question = ""):
    plt.figure(figsize=(3 * c, 3 * r))
    for i in range(r * c):
      plt.subplot(r, c, i + 1)
      plt.title(prefix + " " + str(i+1))
      plt.imshow(images[i].reshape((height, width)), cmap = plt.cm.gray)
      plt.xticks(())
      plt.yticks(())
    plt.tight_layout()
    plt.savefig(question)
    plt.show()
eigenfaces = pca.components_
custom_plot(eigenfaces[:25], r = 5, c = 5, prefix = "Eigenfaces", question = "5a")

custom_plot(X[:10], r = 1, c = 10, prefix = "Face", question = "5b1")
reconstructed = pca.inverse_transform(pca.transform(X[:10]))
custom_plot(reconstructed, r = 1, c = 10, prefix = "Reconstructed", question = "5b2")
```

Listing 9. Problem 5 Code

The 25 eigenfaces as well as 10 original and reconstructed faces are shown on the next page in Figure 3.

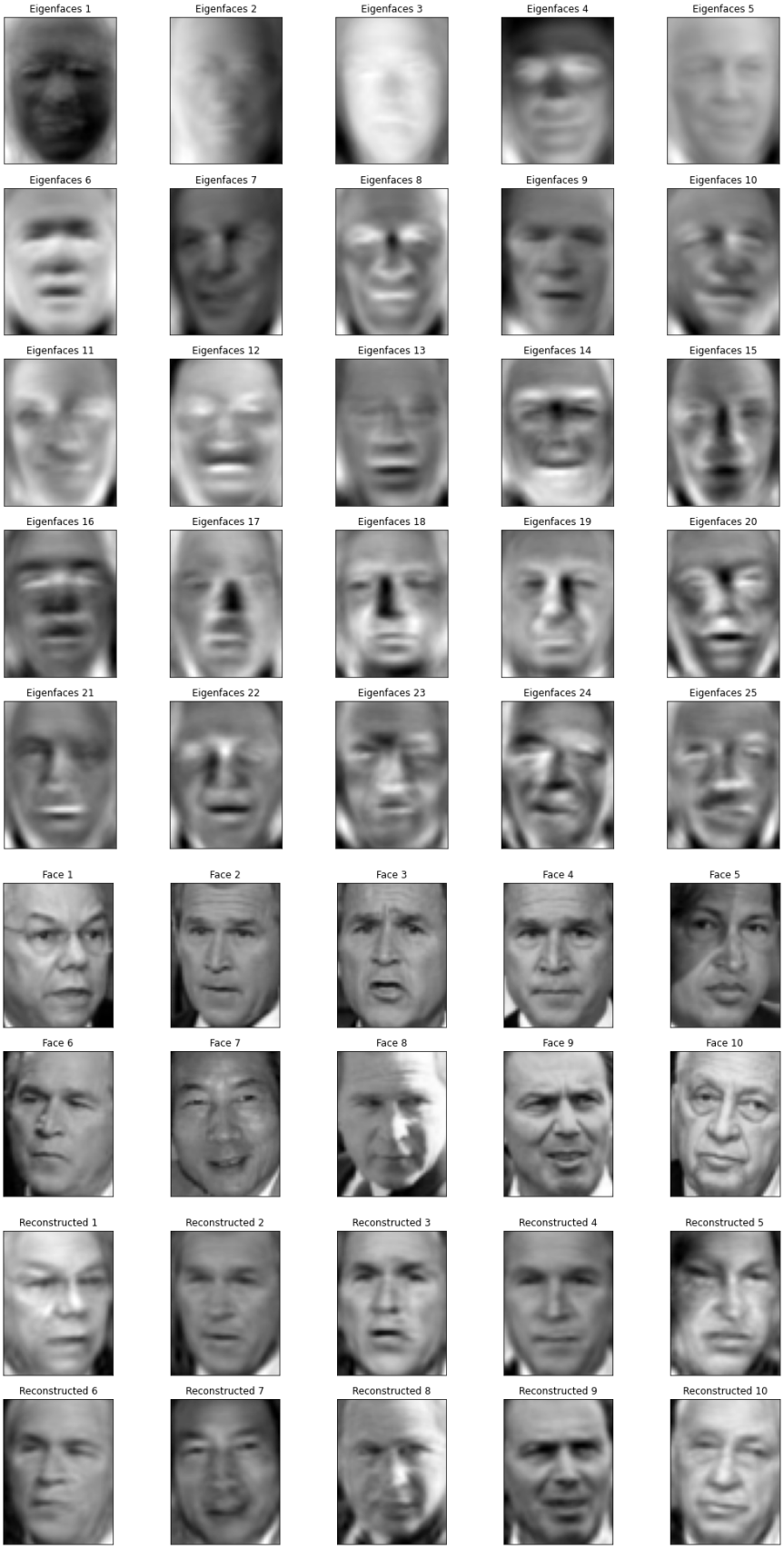

Figure 4. 25 Eigenfaces and 10 Original Vs Reconstructed Faces