Machine and Deep Learning
Assignment – 2

1. b) The normal equation,

$$\theta = (x^Tx)^{-1}x^Ty$$

$$\theta \quad = \quad (x^Tx)^{-1}x^Ty \quad = \begin{pmatrix} 1/38 & -5/38 \\ -5/38 & 69/76 \end{pmatrix}\begin{pmatrix} 568 \\ 82 \end{pmatrix} = \begin{pmatrix} 79/19 \\ -11/38 \end{pmatrix} = \begin{pmatrix} 4.15789474 \\ -0.289473684 \end{pmatrix}$$

1. c) Gradient descent is an iterative technique, which means that getting to the global optimum requires numerous iterations. However, it turns out there is a way to solve for the optimal values of the parameter theta in just one step for the particular case of Linear regression, and this algorithm is known as the normal equation. It is only compatible with linear regression and not with any other technique.
   The normal equation is the closed- form solution for the Linear Regression approach, indicating that the best parameters may be derived by simply using a formula that consists of a few matrix multiplications and inversions.

1. d) Implementation of the closed form equation in python

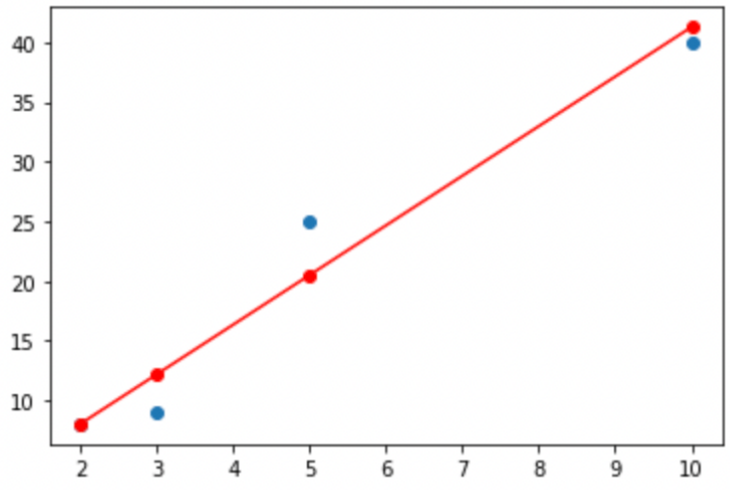
```
#Making imports
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np


#Preprocessing Input Data
x=[2,5,3,10]
y=[8,25,9,40]

#Building the model
mean_x = np.mean(x)
mean_y = np.mean(y)

#total number of values
n = len(x)
```

```
#using the formula to calculate m and c

numer = 0
denom = 0

for i in range(n) :
  numer += (x[i] - mean_x) * (y[i] - mean_y)
  denom += (x[i] - mean_x) ** 2
m = numer/denom
c = mean_y - (m * mean_x)

#Printing the coefficients
print("coefficients:")
print(m,c)

#Making the Predictions
y_pred = m * np.asarray(x) + c
plt.scatter(x,y)  #actual
plt.plot([min(x), max(x)], [min(y_pred), max(y_pred)], color ='red')
plt.scatter(x, y_pred, color = 'red')
plt.show()

#Calculating the root mean squares error

rmse = 0

for i in range(n):
  y_pred = c + m * x[i]
  rmse += (y[i] - y_pred) ** 2
rmse = np.sqrt(rmse/n)
print("Root Mean Squares Error:")
print(rmse)

#Calculating the R2 Score
ss_tot = 0
ss_res = 0
for i in range(n) :
  y_pred = c + m * x[i]
  ss_tot += (y[i] - mean_y) ** 2
  ss_res += (y[i] - y_pred) ** 2
r2 = 1 - (ss_res/ ss_tot)
print("R2 Score:")

print(r2)
```

```
#Closed form Normal Equation

a = np.asarray(x). shape[0]

#appending a column of ones in X to add the bias term.
x = np.append(np.asarray(x).reshape(-1,1),np.ones((a,1)),axis = 1)

x = np.array(x, dtype = 'int16')
y = np.array(y).reshape(-1,1)
theta = np.dot(np.linalg.inv(np.dot(x.T, x)), np.dot(x.T, y))
print("Closed form normal equation solution:", theta)
```

OUTPUT:

```
coefficients:
4.157894736842105 -0.28947368421052744
```



```
Root Mean Squares Error:
2.8307521782623146
R2 Score:
0.9534794897257658
Closed form normal equation solution: [[ 4.15789474]
 [-0.28947368]]
```

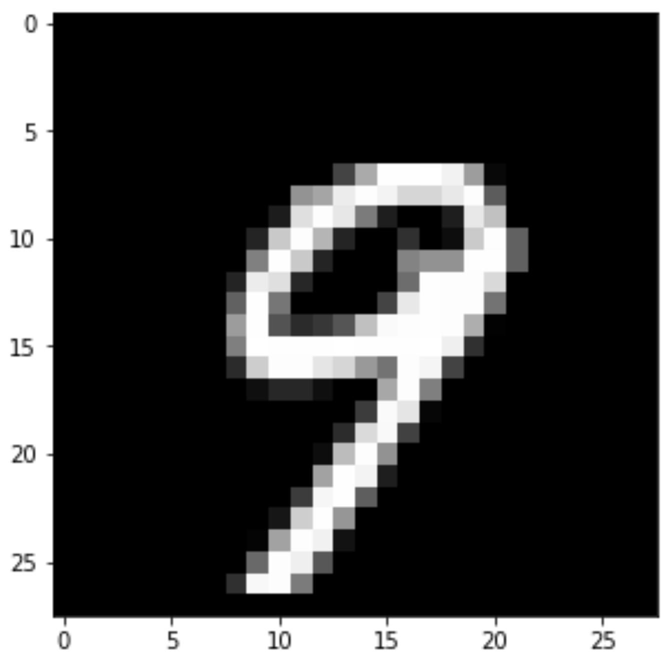
2. PCA Analysis

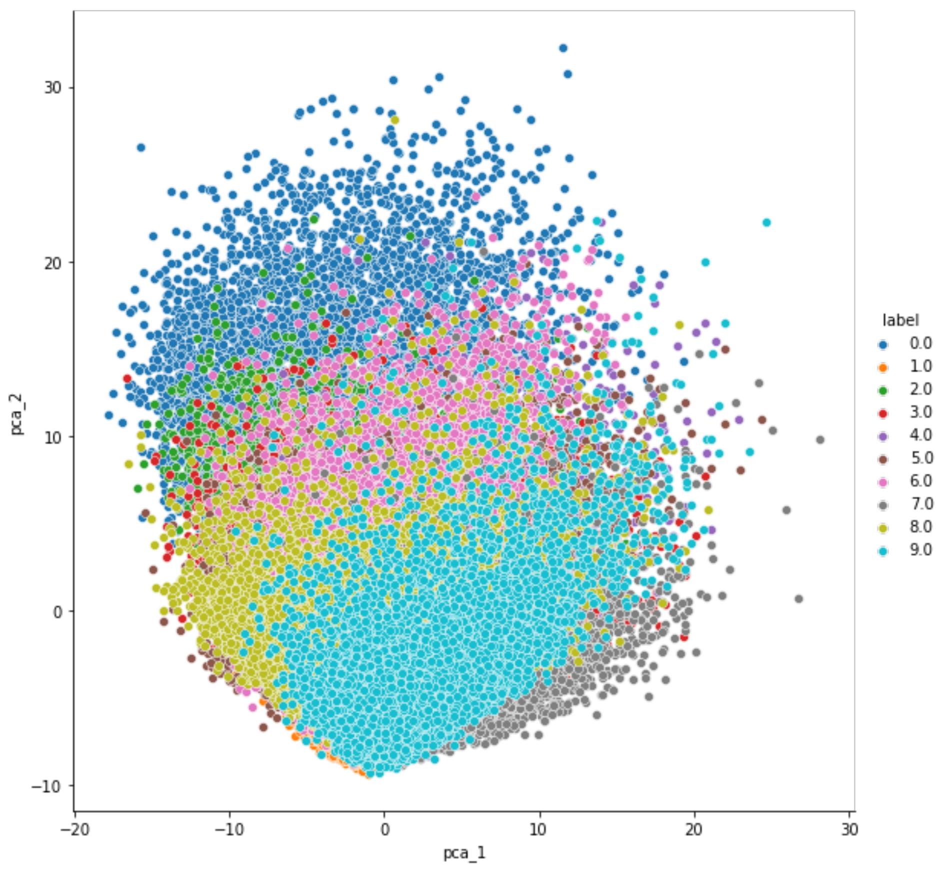
```
#Import Libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import random
from mnist import MNIST
from sklearn.preprocessing import StandardScaler
from scipy.linalg import eigh

The MNIST Dataset is currently being loaded.

mndata = MNIST('C:\\Users\\srija\\OneDrive\\Documents\\mnist')
training_images, training_labels = mndata.load_training()
test_images, test_labels = mndata.load_testing()
print("the shape of the training_images data is : ",
np.array(training_images).shape)
print("the shape of the training_labels data is : ",
np.array(training_labels).shape)
print("the shape of the test_images data is : ",
np.array(test_images).shape)
print("the shape of the test_labels data is : ",
np.array(test_labels).shape)

#From the training dataset, extracting the column label

label = training_labels
ind = np.random.randint(0,20000)
plt.figure(figsize = (20,5))
grid_data =
np.array(pd.DataFrame(training_images).iloc[ind]).reshape(28,28)

#Using the matplotlib imshow() method to plot a random sample data point
from the training dataset.

plt.imshow(grid_data, interpolation = None, cmap = 'gray')
plt.show()

#Column standardization of the training dataset via the
sklearn.preprocessing module's standardScalar class.
#Because after our data has been column standardized, the mean of each
feature is 0 (zero) and the variance is 1.
```

```
#As a result, we begin PCA with the origin point.

scalar = StandardScaler()
std_df = scalar.fit_transform(training_images)
print("Shape of the dataset after the column standardization:",
std_df.shape)

#Using the numpy matmul method, find the co-variance matrix AT * A.

covar_mat = np.matmul(std_df.T, std_df)
print("the dimensions of co-variance matrix after multiplication",
covar_mat.shape)

#The parameter 'eigvals' is defined (low value to high value)
#Finding the top two eigen-values and related eigen vectors for projection
onto a 2D surface
# The eigen values will be returned in ascending order by the eigh
function.

values, vectors = eigh(covar_mat, eigvals = (782,783))
print("Dimensions of eigen vector:", vectors.shape)
vectors= vectors.T
print("Dimensions of eigen vector:", vectors.shape)

#Find two major components by multiplying the two top vectors by the co-
variance matrix. PC1 and PC2 are two different types of computers.

final_df = np.matmul(vectors, stu_df.T)
print("vectors:", vectors.shape, "n", "std_df:", std_df.T.shape,"n",
"final_df:", final_df.shape)

#Stack final df and label vertically, then transpose them to find the
NumPy data table.
#Using PCA, I was able to convert 60000 * 784 data to 60000*4.

final_dfT = np.vstack(final_df, label).T
dataFrame = pd.DataFrame(final_dfT, columns = ['pca_1', 'pca_2', 'label'])
print(dataFrame)

#Now let's use the seaborn Facet Grid technique to visualize the final
data.

sns.FacetGrid(dataFrame, hue = 'label', size = 8)\
.map(sns.scatterplot, 'pca_1','pca_2')\
.add_legend()
```

```
plt.show()
```

OUTPUT:





2. a) Computation of mean and standard deviation

```
xfrom mnist import MNIST
import numpy as np
import matplotlib.pyplot as plt


mnist = MNIST('C:\\Users\\Srija\\OneDrive\\Documents\\mnist')
X_train,y_train = mnist.load_training()

M = np.zeros(28,28,10)
S = np.zeros(28,28,10)

for i in range(9) :
    X_subset = X_train[y_train == i]
    M[:,:,i+1] = np.mean(X_subset, axis = 0)
    S[:,:, i+1] = np.mean(X_subset, axis = 0)
    plt.subplot(2,10,i+1)
    plt.imshow(M[:,:,i+1])
    plt.subplot(2,10,i+11)
    plt.imshow(S[:,:,i+1])

plt.show()
```

OUTPUT:

3. c) **PCA using SVD**

Algorithm:

Step 1 : Define a function, and pass Principal Components (PC), and Mean of Dimension vectors as output arguments. Pass the Input Matrix D as input argument.

Step 2 : Find the Mean between Input Matrix (d) and 2 and assign the value to a variable called mn**.**

Step 3 : Assign the size (D, 2) to m.

Step 4 : Assign the value of D – repmat(mn, 1, m) to D

Step 5 : Assign the value of D'/sqrt(m-1) to y

Step 6 : [u, S, PCno] = svd(Y);

Step 7 : PC = PCno(:, 1: k);

Step 8 : Assign the mean value of mn to dM

Step 9 : End the function

Step 10 : Load MNIST Train Images

Step 11 : Reshape the image Dataset to (28,28,60000)

Step 12 : Apply PCA_SVD

Step 13 : To reduce the train data to 10 dimensions

Step 14 : Reshape principal components to display as image

Step 15 : Display the images doing contrast stretching

3. d)

```
from mnist import MNIST
import numpy as np
import matplotlib.pyplot as plt

def pcs_svd(i_train, i):
    PC = np.dot(i_train, i_train.T)
    reconn = np.dot(PC, i_train)
    normal = np.dot(reconn, PC)
    dif = np.sum(normal=i_train)
    dif = dif/(60000*784)
    diff[i] = dif
    if(name_ == '_main_'):
        i_train =
np.loadtxt('C:\\Users\\srija\\OneDrive\\Documents\\mnist\\t10k-images-
idx3-ubyte')
        diff = np.zeros(784)
    for i in range(784)
    pcs_svd(i_train, i)
    plt.plot(diff)
    plt.show
```

4. d) We can assume that the probability density f(xi) is strictly positive for any observation xi (out of n observations) without much loss of generality (or mass), allowing it to be represented as an exponential.

$$f(x_i) = \exp\left(g(x_i, \theta)\right)$$

If θ=(θj), for a parameter vector.

The type equations are produced by equating the log likelihood function's gradient to zero (which discovers stationary locations of the likelihood, including all inner global maxima if one exists).

$$\sum_i \frac{dg(x_i, \theta)}{d\theta_j} = 0,$$

one for each letter of the alphabet to have a ready solution for any of them, we'd like to be able to separate the xi keywords from the words. The most common approach is to write the equations in the following format:

$$\sum_i \left(\eta_j(\theta)\tau_j(x_i) - \alpha_j(\theta)\right) = \eta_j(\theta)\sum_i \tau_j(x_i) - n\alpha_j(\theta)$$

$$g(x, \theta) = \tau_j(x)\int^{\theta} \eta_j(\theta)d\theta_j - \int^{\theta} \alpha_j(\theta)d\theta_j + B(x, \theta_j')$$