1112 Deep Learning – Homework 1 Due: 10/22, 2023, 11:59pm

111753127 資碩工二 蘇冠華

灰色:程式碼 ；#:註解

1. Please load ‘data.mat’ into your Python code, where you will find $x,y \in R^{1001}$. Now do the following procedures.

1.1. (5%) Plot the data using plot function.

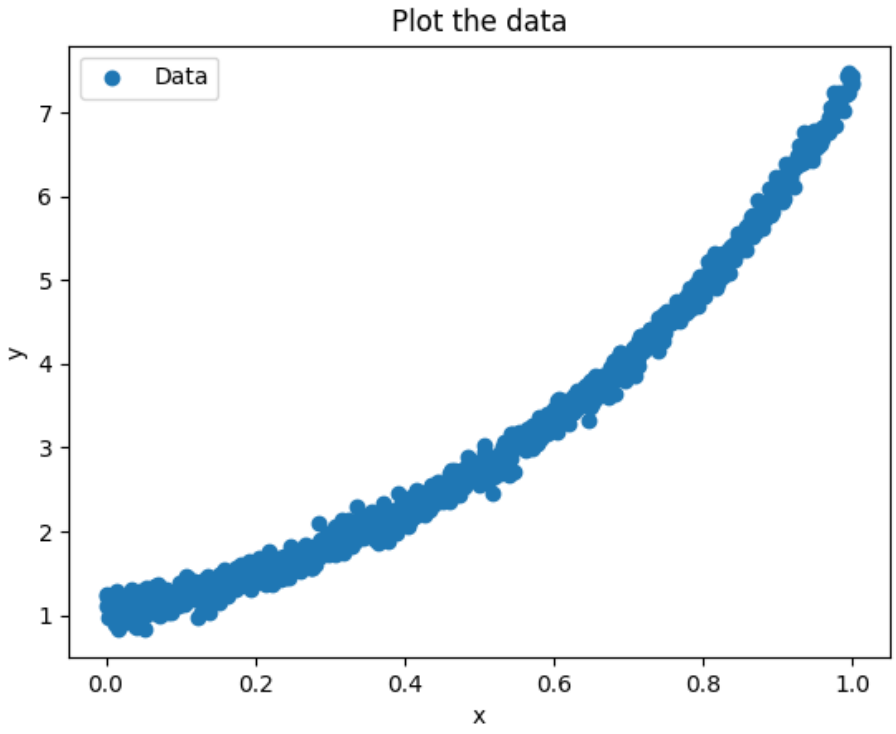
Sol:

Step1. load ‘data.mat’

```
data = loadmat('data.mat')      #先將data.mat上的資料load進程式中
x = data['x'].flatten()         #將多維度資料中的所有元素按照順序排列成一維的序列
y = data['y'].flatten()
```

Step2. Plot the data

```
plt.scatter(x, y, label='Data')  #將step1的x,y資料繪製成分布圖
plt.xlabel('x')、plt.ylabel('y') #設定x、y軸的label名稱
plt.title(' Plot the data ')     #設定圖表title
plt.legend()                     #顯示圖例以方便圖表的閱讀
plt.savefig("Q1_1.png")          #儲存圖片
plt.show()                       #顯示圖表
```



1.2. (**5%**) Compute the least square line $y=\theta \mathbf{0}+x\theta \mathbf{1}$ using the given data and overlay the line over the given data.

Sol:

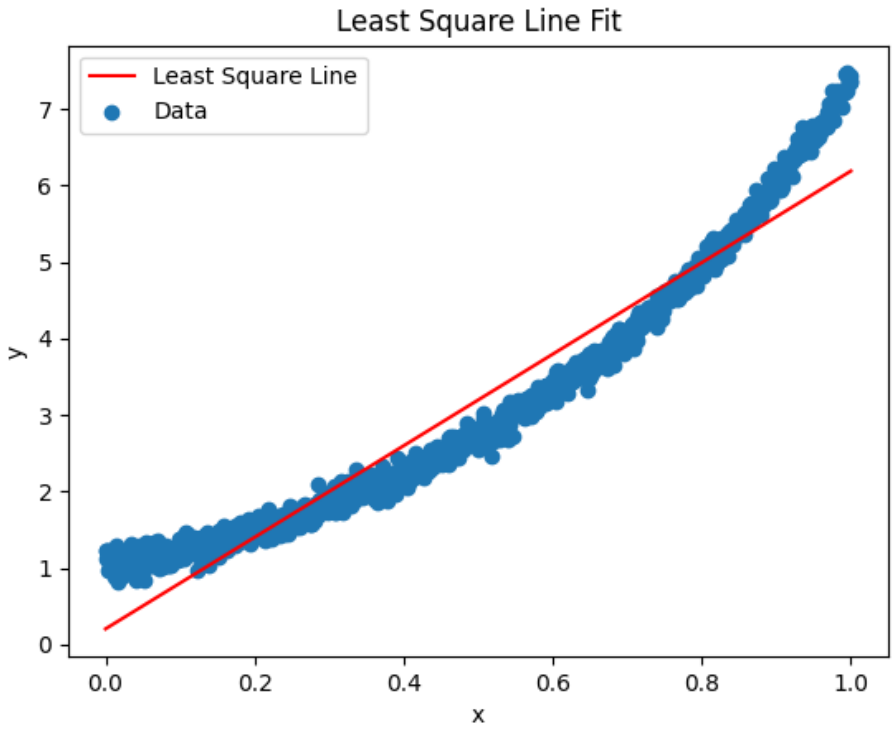
Step1: 計算least square line

```
lr = LinearRegression()          #使用LinearRegression()線性迴歸模型
lr.fit(x.reshape(-1, 1), y)      #用LinearRegression()迴歸模型擬合資料
theta0_line = lr.intercept_      #取得截距
theta1_line = lr.coef_[0]        #取得斜率
```

Step2: Plot the data

```
plt.scatter(x, y, label='Data')  #將step1的x,y資料繪製成分布圖

#將step1算出的截距及斜率繪製成最小平方法的線性迴歸線(並將線設為紅色)
plt.plot(x, theta0_line + x * theta1_line, color='red', label='Least Square Line')
plt.xlabel('x')、plt.ylabel('y') #設定x、y軸的label名稱
plt.title('Least Square Line Fit') #設定圖表title
plt.legend()                     #顯示圖例以方便圖表的閱讀
plt.savefig("Q1_2.png")          #儲存圖片
plt.show()                       #顯示圖表
```



1.3. (**5%**) Compute the least square parabola (i.e. second order polynomial $y=\theta\mathbf{0}+x\theta\mathbf{1}+x\text{^}2\theta\mathbf{2}$) to fit the data.

Sol:

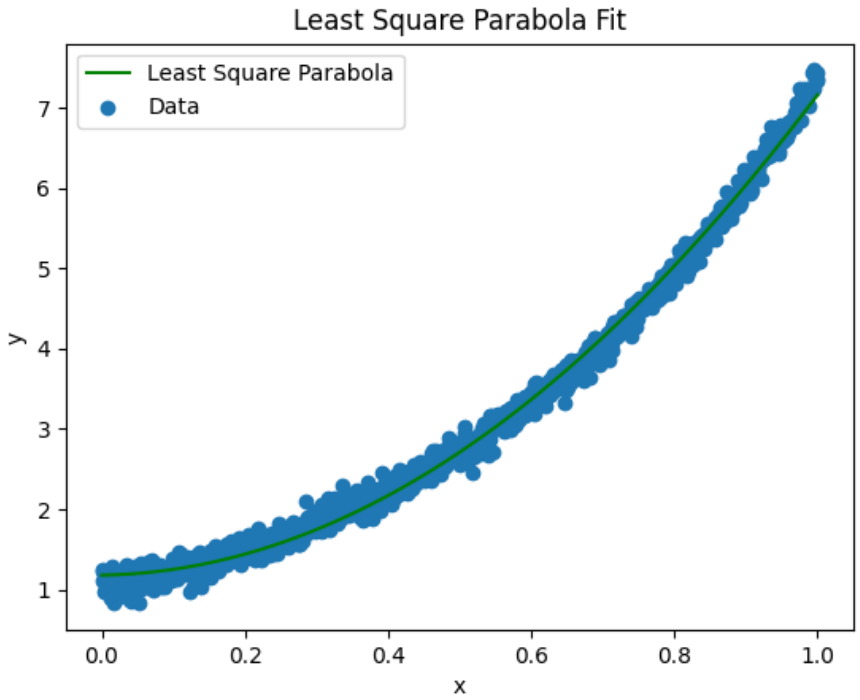
Step1: 計算least square parabola

```
poly = PolynomialFeatures(degree=2) #使用PolynomialFeatures() (degree=2 :二次多項式)
X_poly = poly.fit_transform(x.reshape(-1, 1)) #用PolynomialFeatures()轉換x
lr.fit(X_poly, y)                    #用LinearRegression()迴歸模型擬合轉換後的資料
theta0_parabola = lr.intercept_      #取得截距
theta1_parabola, theta2_parabola = lr.coef_[1:]#取得theta1跟theta2的係數
```

Step2: Plot the data

```
plt.scatter(x, y, label='Data')      #將step1的x,y資料繪製成分布圖

#將step1算出的截距及係數繪製成最小平方法二次多項式迴歸曲線(並將線設為綠色)
plt.plot(x, theta0_parabola + x * theta1_parabola + x**2 * theta2_parabola, color='green',
label='Least Square Parabola')
plt.xlabel('x')、plt.ylabel('y')     #設定x、y軸的label名稱
plt.title('Least Square Parabola Fit')  #設定圖表title
plt.legend()                         #顯示圖例以方便圖表的閱讀
plt.savefig("Q1_3.png")#             #儲存圖片
plt.show()                           #顯示圖表
```



1.4. (**5%**) Compute the least square quartic curve ($y=\theta\mathbf{0}+x\theta\mathbf{1}+x2\theta\mathbf{2}+x3\theta3+x4\theta4$) to fit the data.

Sol:

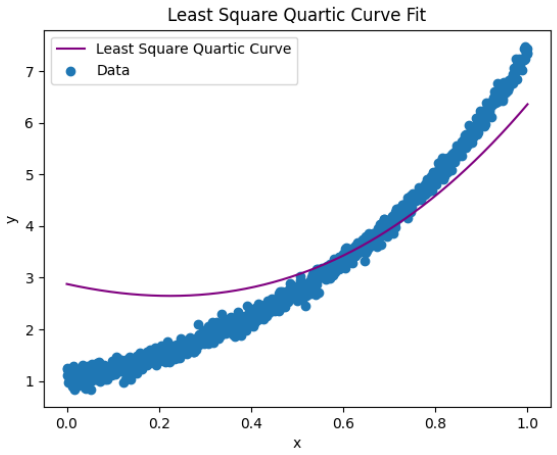
Step1: 計算least least square quartic curve

```
poly = PolynomialFeatures(degree = 4)#使用PolynomialFeatures() (degree=4:四次多項式)
X_poly = poly.fit_transform(x.reshape(-1, 1)) #用PolynomialFeatures()轉換x
lr.fit(X_poly, y)                    #用LinearRegression()迴歸模型擬合轉換後的資料
theta_quartic = lr.coef_             #取得係數
y_quartic = np.polyval(theta_quartic, x)#使用numpy的polyval函數計算四次多項式的值
```

Step2: Plot the data

```
plt.scatter(x, y, label='Data')                  #將step1的x,y資料繪製成分布圖

#將step1算出的係數繪製成最小平方法四次多項式迴歸曲線(並將線設為紫色)
plt.plot(x, y_quartic, color='purple', label='Least Square Quartic Curve')
plt.xlabel('x')、plt.ylabel('y')                 #設定x、y軸的label名稱
plt.title('Least Square Quartic Curve Fit')      #設定圖表title
plt.legend()                                     #顯示圖例以方便圖表的閱讀
plt.savefig("Q1_4.png")                          #儲存圖片
plt.show()                                       #顯示圖表
```



1.5. (**5%**) Explain which formulation (line, parabola, cubic curve) is more suitable for this dataset and why (please calculate the mean square error for these two fitting equations)?

Sol: 原因:根據計算的結果(結果如下)，可見Parabola的MSE最小，所以Parabola(二次多項式)對於此dataset的擬合度最好。

Mean Square Error (Line): 0.20580596682517402

Mean Square Error (Parabola): 0.015744919931207565

Mean Square Error (Quartic Curve): 0.748465669977363

```
Mean Square Error (Line): 0.20580596682517402
Mean Square Error (Parabola): 0.015744919931207565
Mean Square Error (Quartic Curve): 0.748465669977363
```

2. (**25%**) Following the previous two questions, please randomly select 30 data samples for 200 times and plot these 200 lines ($y=\theta\mathbf{0}+x\theta\mathbf{1}$) and quartic curves ($y=\theta\mathbf{0}+x\theta\mathbf{1}+x2\theta\mathbf{2}+x3\theta3+x4\theta4$**)** in two separate figures, one for lines and the other for quartic curves. Explain these visualizations based on the bias and variance.
Sol:
Step1:前置作業:

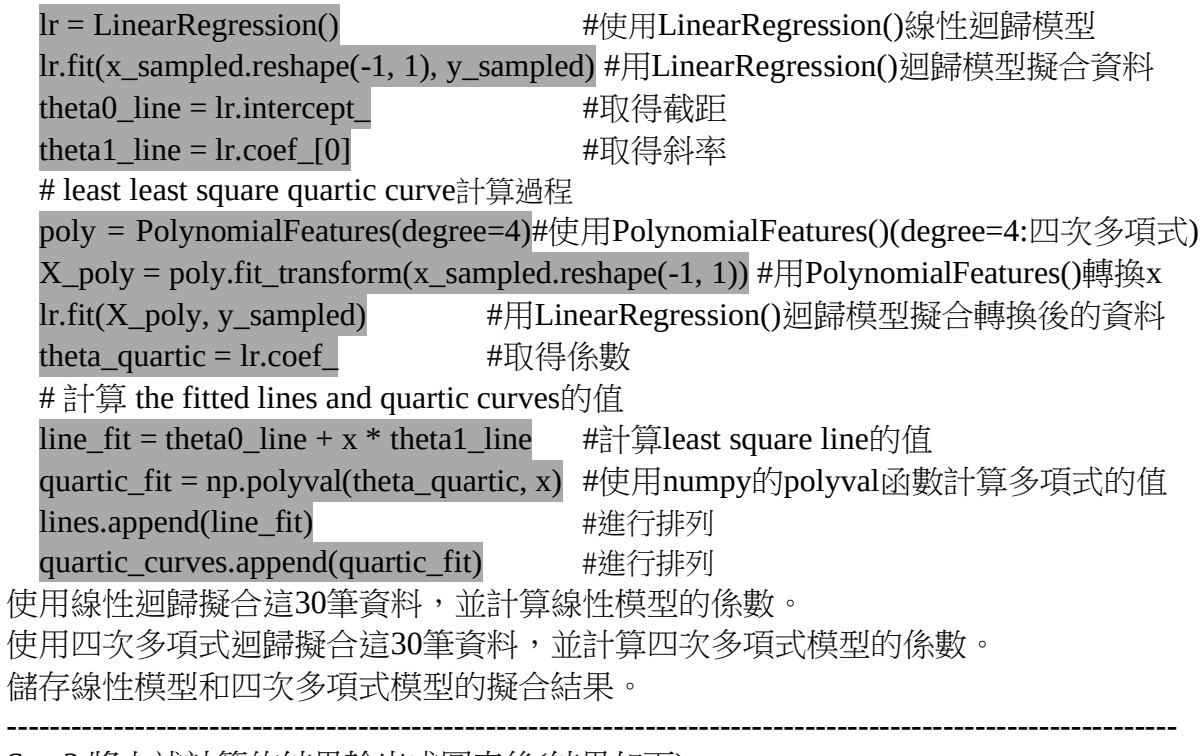
```
data = loadmat('data.mat')              #載入data.mat
x = data['x'].flatten()
y = data['y'].flatten()
lines = []、quartic_curves = []         # 設array來儲存2種結果
num_samples = 30                        # 隨機樣本數 = 30
num_iterations = 200                    # 重複隨機抽樣次數 = 200
```

Step2:

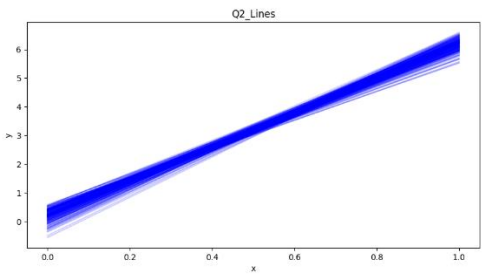
```
for _ in range(num_iterations):     #定義一個for迴圈每次隨機抽樣30筆資料迭代200次。
    random_indices = np.random.choice(len(x), num_samples, replace=False)
    x_sampled = x[random_indices]
    y_sampled = y[random_indices]
    # least square line計算過程
    lr = LinearRegression()                     #使用LinearRegression()線性迴歸模型
    lr.fit(x_sampled.reshape(-1, 1), y_sampled) #用LinearRegression()迴歸模型擬合資料
    theta0_line = lr.intercept_                 #取得截距
    theta1_line = lr.coef_[0]                   #取得斜率
    # least least square quartic curve計算過程
    poly = PolynomialFeatures(degree=4)#使用PolynomialFeatures()(degree=4:四次多項式)
    X_poly = poly.fit_transform(x_sampled.reshape(-1, 1)) #用PolynomialFeatures()轉換x
    lr.fit(X_poly, y_sampled)           #用LinearRegression()迴歸模型擬合轉換後的資料
    theta_quartic = lr.coef_            #取得係數
    # 計算 the fitted lines and quartic curves的值
    line_fit = theta0_line + x * theta1_line    #計算least square line的值
    quartic_fit = np.polyval(theta_quartic, x)  #使用numpy的polyval函數計算多項式的值
    lines.append(line_fit)                      #進行排列
    quartic_curves.append(quartic_fit)          #進行排列
```

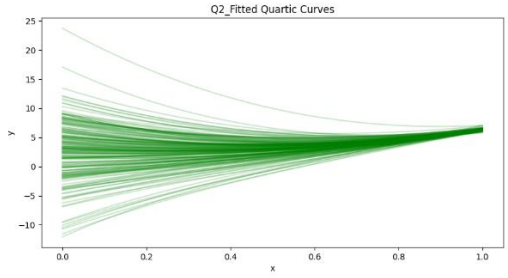
使用線性迴歸擬合這30筆資料，並計算線性模型的係數。
使用四次多項式迴歸擬合這30筆資料，並計算四次多項式模型的係數。
儲存線性模型和四次多項式模型的擬合結果。

Step3:將上述計算的結果輸出成圖表後(結果如下)



解釋: 先解釋bias及variance，
bias:指模型的平均預測值與我嘗試預測的正確值之間的差異。
Variance: 資料分佈值的模型預測的可變性。
lines:由於lines不太靈活，導致模型無法處理較為複雜的關係，所以bias會相對比較高。另外由於這些線彼此非常接近，所以variance會比較低。
quartic curves：由於quartic curves相對於lines更加靈活，可以更好處理及擬合資料點，所以bias相對會比較低。但相對的它可能會造成overfitting，因而導致variance過高。
小結: 論bias:(lines > quartic curves)； 論variance:(lines < quartic curves)





3. (**15%**) In ‘train.mat,’ you can find 2-D points X=[x1, x2] and their corresponding labels Y=y. Please use logistic regression $h(\boldsymbol{\theta})=1/(1+e\text{^}(-\boldsymbol{\theta}\text{^}Tx))$ to find the decision boundary (optimal $\boldsymbol{\theta}\text{^}*$) based on ‘train.mat.” Please report the test error on the test dataset ‘test.mat.’ (percentage of misclassified test samples)
Sol:

Step1:load train.mat 及 test.mat，並將資料格式都處理好

```
# Load training data
train_data = loadmat('train.mat')                                    #load train.mat
X_train = np.column_stack((train_data['x1'], train_data['x2'])) #處理train.mat資料格式
Y_train = train_data['y'].flatten()

# Load test data
test_data = loadmat('test.mat')                                      #load test.mat
X_test = np.column_stack((test_data['x1'], test_data['x2']))  #處理test.mat資料格式
Y_test = test_data['y'].flatten()
```

Step2:訓練模型及預測test data結果

```
logistic_reg = LogisticRegression(solver='liblinear')#創建一個LogisticRegression模型，
                                                     #並使用 'liblinear' solver來訓練模型
logistic_reg.fit(X_train, Y_train)       #使用X_train、Y_train來訓練模型，
                                         #X_train:feature，Y_train:targer
Y_pred = logistic_reg.predict(X_test)#預測模型測試data的結果
```

Step3: 計算 test error

```
test_error = 100 * (1 - accuracy_score(Y_test, Y_pred))
```

test error : 3.33%，結果如下:

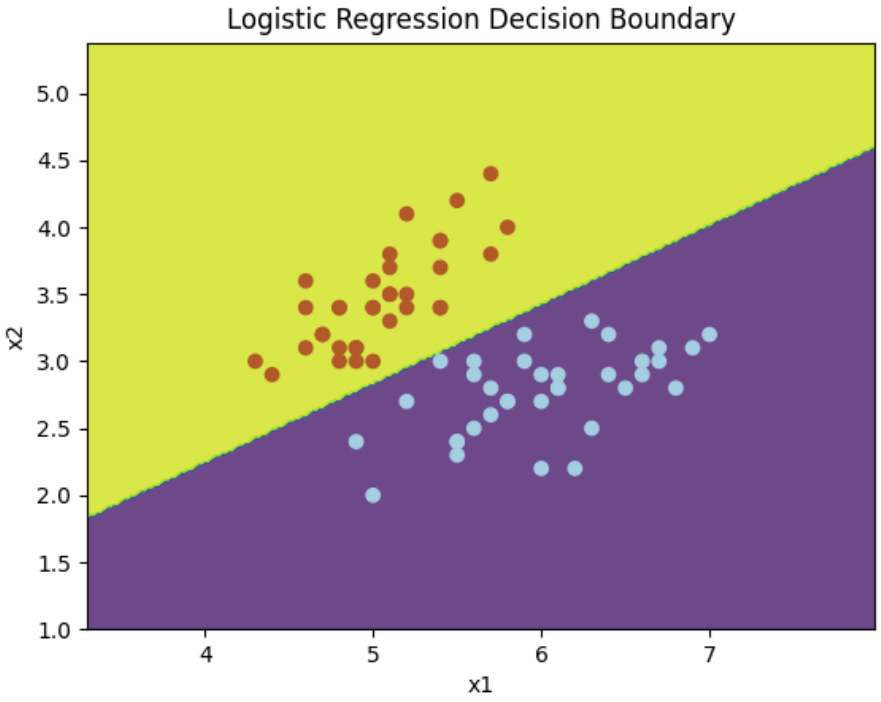
```
--------------------------------
Test Error: 3.33%
--------------------------------
```

Step4:plot the decision boundary

```
#定義x1特徵和x2特徵的最小值及最大值
x_min, x_max = X_train[:, 0].min() - 1, X_train[:, 0].max() + 1
y_min, y_max = X_train[:, 1].min() - 1, X_train[:, 1].max() + 1
#生成網格圖並將上述x1.x2座標丟入
xx, yy = np.meshgrid(np.arange(x_min, x_max, 0.02), np.arange(y_min, y_max, 0.02))
```

#使用 logistic_reg 模型對網格點的坐標進行預測取得每個網格點的類別label
#np.c:將 xx 和 yy 中的坐標值組合成一個特徵矩陣以方便進行預測
```
Z = logistic_reg.predict(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)        #將預測的類別標籤重塑成與 xx 相同形狀的矩陣
```

#繪製decision boundary，如下圖:
```
plt.contourf(xx, yy, Z, alpha=0.8)
plt.scatter(X_train[:, 0], X_train[:, 1], c=Y_train, cmap=plt.cm.Paired)
plt.xlabel('x1')
plt.ylabel('x2')
plt.title('Logistic Regression Decision Boundary')
plt.savefig("Q3_Logistic Regression Decision Boundary.png")
plt.show()
```



4.
4.1. (**5%**) Use the following code to show 50 images in your own dataset.

```
################################################
import numpy as np
import matplotlib.pyplot as plt
amount= 50
lines = 5
columns = 10
number = np.zeros(amount)
for i in range(amount):
number[i] = y_test[i]
# print(number[0])
fig = plt.figure()
for i in range(amount):
ax = fig.add_subplot(lines, columns, 1 + i)
plt.imshow(x_test[i,:,:], cmap='binary')
plt.sca(ax)
ax.set_xticks([], [])
ax.set_yticks([], [])
plt.show()
################################################
```

Sol:

Step1: 前置作業

```
(x_train, y_train), (_, _) = mnist.load_data() #載入 MNIST 數據集
samples_per_digit = 500                        #指定每個數字要選取的樣本數=500
selected_x_train = []                          儲存結果的array
```

Step2: 隨機選取每個數字的樣本

```
for digit in range(10):
    #在每個數字找到訓練集中對應數字的index
    digit_indices = np.where(y_train == digit)[0]
    #使用 numpy.random.choice 函數隨機選擇 samples_per_digit 個不重複的索引。
    selected_indices = np.random.choice(digit_indices, samples_per_digit, replace=False)
    #將選擇的圖片加進selected_x_train陣列中
    selected_x_train.extend(x_train[selected_indices])
```

Step3:將selected_x_train的資料轉換為NumPy陣列格式，並展平為一維陣列

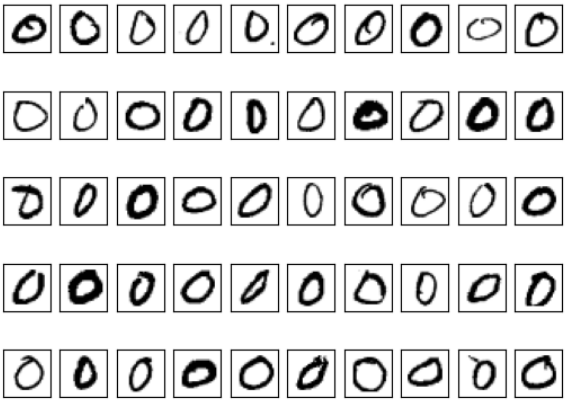
```
selected_x_train = np.array(selected_x_train).reshape(-1, 28 * 28)
```

Step4:將資料整理後輸出成一張圖表，結果如下:

```
#設置要顯示的圖片數量:amount、圖表行數:lines、列數:columns
amount = 50
lines = 5
columns = 10

fig = plt.figure()          #創建大圖表
# 使用迴圈依次將選取的50張圖片添加到圖表中，並設定好每張圖片的行列位置。
for i in range(amount):
    ax = fig.add_subplot(lines, columns, 1 + i)
```

```
    image = selected_x_train[i].astype(np.uint8).reshape(28, 28)#確保資料在正確的範圍內
    plt.imshow(image, cmap='binary') #顯示圖表，cmap='binary' : 設定圖片的顏色映射
    plt.sca(ax)
    ax.set_xticks([])
    ax.set_yticks([])
plt.savefig("Q4_1.png")
plt.show()
```



4.2. (**15%**) Normalize the data (subtracting the mean from it and then dividing it by the standard deviation) and compute the eigenpairs for the covariance of the data (sorted in a descending order based on eigenvalues).

Sol:

Step1: # 對資料做Normalize

```
mean = np.mean(selected_x_train, axis=0)    #計算平均值
std = np.std(selected_x_train, axis=0)          #計算標準差
#進行正規化，後面增加一個極小常數以避免被零除無意義
normalized_x_train = (selected_x_train - mean) / (std + 1e-8)
```

Step2:

```
cov_matrix = np.cov(normalized_x_train.T) #計算covariance matrix
eigenvalues, eigenvectors = np.linalg.eig(cov_matrix) # 計算特徵根及特徵向量
```

Step3:

```
# 將特徵值及特徵根依照大小進行排序
sorted_indices = np.argsort(eigenvalues)[::-1]
eigenvalues = eigenvalues[sorted_indices]
eigenvectors = eigenvectors[:, sorted_indices]
```

Step4:印出前10個特徵根，結果如下:

```
top_eigenvalues = eigenvalues[:10]
print("Top 10 Eigenvalues:")
print(top_eigenvalues)
```

```
Top 10 Eigenvalues:
[40.9078568  30.12824616 26.39500034 20.4164824  17.72514753 15.12294875
 13.99608934 12.25099032 11.5280779   9.90740128]
```

4.3. (**15%**) Please use PCA to reduce the 784 dimensional data to that with 500, 300, 100, and 50 dimensions, and then show 10 decoding results for each digit, respectively. How do you interpret these results?
Sol:
解釋:
PCA 能夠有效地reduce dimensional，它能夠對於資訊的壓縮、視覺化和複雜資料集提供有效的簡化。當減少維度數(從 784 到 500、300、100 、50)時，抓到的資訊量會比較少，因而導致重建品質降低，影像就會失去一些細節導致變得更模糊。
高維度:保留更多的資訊進而產生更清晰的影像，但它們也需要更多的儲存和計算。
低維度:保留的資訊較少，故影像較模糊，但它們的計算效率較快且能夠突顯每個數字的主要特徵點。
我認為降維的選擇應該考慮計算效率和資訊損失之間的平衡點。對於數字辨識等任務，使用較低的維度即可；而對於需要精確影像重建等的應用，才需要運用到較高的維度。
以下為程式碼逐行解釋及成果截圖。

Step1: 定義要測試的reduced dimensions

```
reduced_dimensions = [500, 300, 100, 50]
```

建立一個for迴圈以接續後續步驟
Step2: 使用PCA進行降維，並重建數據:

```
for k in reduced_dimensions:
    pca = PCA(n_components=k)
    reduced_data = pca.fit_transform(normalized_x_train)   #將正規化後的數據降維
    #用inverse_transform()將降維後的數據 reduced_data 重建回原始維度
    reconstructed_data = pca.inverse_transform(reduced_data)
```

Step3: 顯示 10 個不同數字的解碼結果:

```
    #創建一個 10x10 的圖表，以便顯示每個數字(0~9)的解碼結果
    fig, axes = plt.subplots(10, 10, figsize=(10, 10))
    for digit in range(10):
        #對於每個數字找到對應的數字索引 digit_indices
        digit_indices = np.where(y_train == digit)[0]
        #確保digit_indices大於10個，並顯示每個原始圖像和重建圖像
        if len(digit_indices) >= 10:
            for i in range(10): #重建PCA 降維後再還原回原始維度的結果
                ax = axes[digit, i]
                original_image = selected_x_train[digit_indices[i]].reshape(28, 28)
                reconstructed_image = reconstructed_data[digit_indices[i]].reshape(28, 28)
                ax.imshow(reconstructed_image, cmap='binary')
                ax.set_xticks([])
                ax.set_yticks([])
```

Step4:顯示最終結果如下:

```
plt.suptitle(f"Decoding Results (k={k})", fontsize=16)
plt.savefig(f"Q4_3_(k={k})", fontsize=16)
plt.show()
```