EN 685.621 HW2 - Noboru Hayashi

Q1:

```
import numpy as np
import pandas as pd
import math

# minimum
def my_min(df):
    return df.min().to_numpy()

# maximum
def my_max(df):
    return df.max().to_numpy()

# mean
def my_mean(df):
    return df.sum().to_numpy()/len(df)

# trimmed mean with p
def my_trimmed_mean(df, p):
    ret = [0,0,0,0]
    for i in range(len(ret)):
        arr = df[df.columns[i]].to_numpy()
        arr = np.sort(arr)
        trimmed_arr = arr[p-1:-(p-1)]
        ret[i] = sum(trimmed_arr)/len(df)
    return ret

# standard deviation
def my_std(df):
    n = len(df)
    mu = my_mean(df)
    diff = df - mu
    var = (diff**2).sum()/(n-1)
    return np.sqrt(var).to_numpy()

# skewness
def my_skewness(df):
    n = len(df)
    sigma = my_std(df)
    mu = my_mean(df)
```

```
    diff = df - mu
    a = ((diff**3).sum())/n
    b = sigma ** 3
    return (a/b).to_numpy()

# kurtosis
def my_kurtosis(df):
    n = len(df)
    sigma = my_std(df)
    mu = my_mean(df)
    diff = df - mu
    a = ((diff**4).sum())/n
    b = sigma ** 4
    return (a/b).to_numpy()

# test code
if __name__ == '__main__':
    df = pd.read_csv('iris.csv')
    obs = df.iloc[:, :4]

    print('min: ', my_min(obs))
    print('maz: ', my_max(obs))
    print('mean: ', my_mean(obs))
    print('trimmed mean (p=5): ', my_trimmed_mean(obs, 5))
    print('std: ', my_std(obs))
    print('skewness: ', my_skewness(obs))
    print('kurtosis: ', my_kurtosis(obs))
```

Output:

```
% python 1.py
min:  [4.3 2.  1.  0.1]
maz:  [7.9 4.4 6.9 2.5]
mean:  [5.84333333 3.054      3.75866667 1.19866667]
trimmed mean (p=5):  [5.52, 2.8853333333333313, 3.549333333333334, 1.13]
std:  [0.82806613 0.43359431 1.76442042 0.76316074]
skewness:  [ 0.30864073  0.3274013  -0.26899936 -0.10290596]
kurtosis:  [2.39418747 3.19836812 1.58331681 1.64263162]
```

Q2:

a) Python code to read csv, generate additional observations and plot:

```
def read_class(class_name):
    # read csv file
    df = pd.read_csv('iris.csv')
    # read data of a class
    obs = df.where(df.species == class_name).dropna().iloc[:, 0:4]

    # calculate cov, mean, min & max
    cov = obs.cov()
    mean = obs.mean().to_numpy()
    mn = obs.min().to_numpy()
    mx = obs.max().to_numpy()

    return cov, mean, mn, mx


def gen_rnd(row_size=100, col_size=4):
    # generate random numbers with row_size * col_size
    return np.random.rand(row_size, col_size)


def normalize(rnd_mat, cov, mean, mn, mx):
    # multiply random numbers by cov
    mat = np.dot(rnd_mat, cov)

    # set means
    for i in range(len(mat[0])):
        mat[:, i] /= mat[:, i].mean()
        mat[:, i] *= mean[i]

    # min-max normalize
    df = pd.DataFrame(mn+(mat-mean))
    df.columns = ['sepal_length', 'sepal_width', 'petal_length', 'petal_width']

    return df

def plot_eclipse(x, y, x_new, y_new, class_name):

    ax = plt.subplot(111)

    # real data
```

```
    cov = np.cov(x, y)
    lambda_, v = np.linalg.eig(cov)
    lambda_ = np.sqrt(lambda_)
    for j in range(1, 4):
        ell = Ellipse(xy=(np.mean(x), np.mean(y)),
                  width=lambda_[0]*j*2, height=lambda_[1]*j*2,
                  angle= np.rad2deg(np.arctan2(*v[:,0][::-1])) , color='r')
        ell.set_facecolor('none')
        ax.add_artist(ell)
    plt.scatter(x, y, marker='o', c='r')

    # new data
    cov_new = np.cov(x_new, y_new)
    print(cov_new)
    new_lambda_, v = np.linalg.eig(cov_new)
    new_lambda_ = np.sqrt(new_lambda_)
    for j in range(1, 4):
        ell = Ellipse(xy=(np.mean(x_new), np.mean(y_new)),
                  width=new_lambda_[0]*j*2, height=new_lambda_[1]*j*2,
                  angle= np.rad2deg(np.arctan2(*v[:,0][::-1])), color='b')
        ell.set_facecolor('none')
        ax.add_artist(ell)
    plt.scatter(x_new, y_new, marker='^', c='b')
    plt.show()

if __name__ == '__main__':
    df = pd.read_csv('iris.csv')
    classes = ['setosa', 'versicolor', 'virginica']
    for i in range(3):
        obs = df.where(df.species == classes[i]).dropna().iloc[:, 0:4]
        x = obs.sepal_length
        y = obs.petal_width

        cov, mean, mn, mx = read_class(classes[i])
        rnd_mat = gen_rnd()
        new_obs = normalize(rnd_mat, cov, mean, mn, mx)
        new_obs['species'] = classes[i]

        x_new = new_obs.sepal_length
        y_new = new_obs.petal_width

        plot_eclipse(x, y, x_new, y_new, classes[i])
```

b) Assume there are N records, n features in dataset, to generate m additional observations:

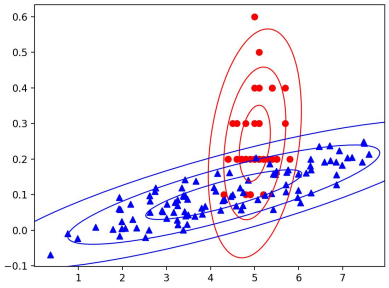
1. Read a csv file: O(N*n)
2. Calculate statistics:
    a. Cov: since cov matrix is calculated by XT*X, with X ~ Nxn matrix, the algorithm operates nxN matrix * Nxn matrix, the running time is O(n*N*n) = O(Nn^2)
    b. Mean, min, max: scan through dataset occurs, O(N) * n = O(Nn)
3. Generate random numbers: O(m) * n = O(mn)
4. Multiply generated number matrix (m x n) with cov matrix (n x n): O(mnn) = O(mn^2)
5. Set mean and normalize of the synthetic data: O(mn)

=> total running time: O(Nn) + O(Nn^2) + O(Nn) + O(mn) + O(mn^2) + O(mn) ~ O(Nn^2) + O(mn^2)

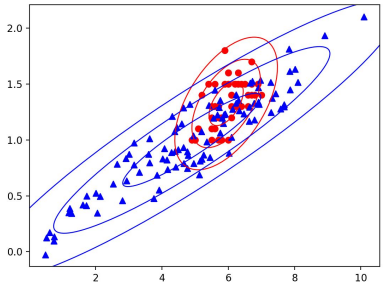
c) The code above will generate 300 additional data

d)

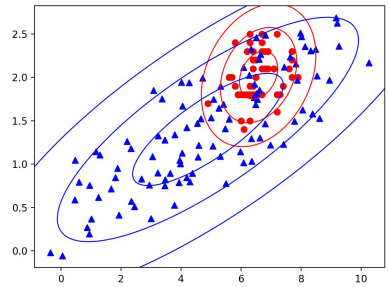
Setosa:



Versicolor:



Virginica:



Q3:

a) Read file:

```
import pandas as pd

# read csv file and take first 100 as samples
df = pd.read_csv('train.csv')
df = df.iloc[:100]
```

b) Reshape:

```
import numpy as np

def read_and_reshape():

    # read csv file and take first 100 as samples
    df = pd.read_csv('train.csv')
    df = df.iloc[:100]

    # extract pixel values
    numbers = df.iloc[:,1:]

    # convert to numpy array and reshape
    # 100 of 28x28 pictures ~ 100x28x28
    mat = numbers.to_numpy().reshape(100, 28, 28)

    return mat
```

c) Assume there are n records with pxp pixel:

1. Reading file contains a list of n image vectors with size p^2: O(n*p^2) = O(np^2)
2. Reshape:
    a. Read elements from a list of image vectors: O(n*p*p) = O(np^2)
    b. Remap to a n x p x p matrix: O(n*p*p) = O(np^2)

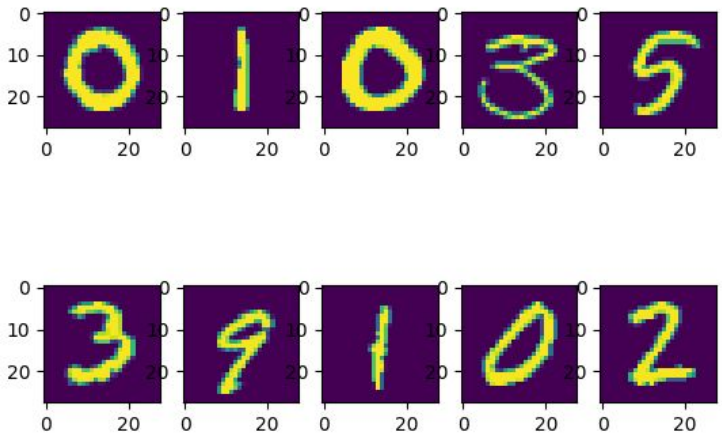
⇒ total running time: O(np^2)

d)

```
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
if __name__ == '__main__':
    mat = read_and_reshape()
    idxs = [1, 2, 4, 7, 8, 9, 11, 12, 17, 22]
    plt.figure()
    for i in range(len(idxs)):
        plt.subplot(2,5,i+1)
        plt.imshow(mat[idxs[i]])

    plt.show()
```

Output:



Q4:

a) Use scipy fftpack package:

```
import pandas as pd
import numpy as np
from scipy.fftpack import dct

if __name__ == '__main__':
    df = pd.read_csv('train.csv')

    # images matrix 42000x784
    images = df.iloc[:, 1:].to_numpy()

    # reshaped dct mat 42000x28x28
    img_dct = dct(images).reshape(42000, 28, 28)
```

b) Access vertical, horizontal, diagonal coeffs:

```
    # extract vertical, horizontal, diagonal
    # 42000x28 for each
    vert = img_dct[:, :, 0]
    hori = img_dct[:, 0, :]
    diag = [img_dct[i].diagonal() for i in range(42000)]
```

c) & d) Use sklearn pca package to perform PCA with 3 top principal components:

```
from sklearn.decomposition import PCA
    # concat vert, hori, diag for PCA
vert_df = pd.DataFrame(vert)
hori_df = pd.DataFrame(hori)
diag_df = pd.DataFrame(diag)

coef_set = pd.concat([vert_df, hori_df, diag_df], axis=1)

    # init PCA with top 3 components
pca = PCA(n_components=3)

    # fit with coef set
pca.fit(coef_set)
```

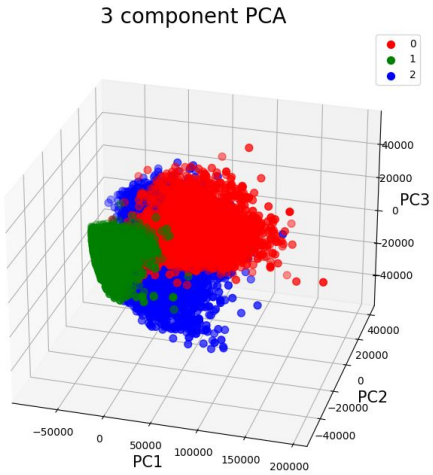
e) Transform DCT transformed data:

```
    # reduce the transformed data
    principalComponents = pca.transform(coef_set)
    principalDf = pd.DataFrame(data = principalComponents
                , columns = ['PC1', 'PC2', 'PC3'])
    finalDf = pd.concat([principalDf, df.label], axis = 1)
```

f) Plot the reduced data with class 0, 1, 2:

```
# Plot the top 2 components on 2d scatter
# since we have 42000 data, just plot datapoint with label 0, 1 & 2
fig = plt.figure(figsize = (8,8))
ax = fig.add_subplot(1,1,1, projection='3d')
ax.set_xlabel('PC1', fontsize = 15)
ax.set_ylabel('PC2', fontsize = 15)
ax.set_zlabel('PC3', fontsize = 15)
ax.set_title('3 component PCA', fontsize = 20)
labels = [0, 1, 2]
colors = ['r', 'g', 'b']
for label, color in zip(labels, colors):
    indicesToKeep = finalDf.label == label
    ax.scatter(finalDf.loc[indicesToKeep, 'PC1']
               , finalDf.loc[indicesToKeep, 'PC2']
               , finalDf.loc[indicesToKeep, 'PC3']
               , color = color
               , s = 50)
ax.legend(labels)
plt.show()
```

Output:



From the visual, it appears that the 3 top principal components have the major variance to separate the 3 classes. For example, data of class 1 having more dense distribution in the 3D scatter, and seems DCT successfully detects the vertical or diagonal line in the original image and maps to reduced domain of the data.