Homework 6

ECE204 Data Science & Engineering

This notebook uses `X` for multiple problems, and that variable may end up being used for the wrong problem. To start fresh, restart the kernel in the Kernel menu or with the button above.

Import Statements

In [261…

```
import numpy as np
import pandas as pd
from sklearn.preprocessing import StandardScaler
```

Problem 1. On the MNIST dataset, **find the number of principle components that are required to explain Y%** (Y is given to you in Canvas) **of the variance.**

In [262…

```
data = np.load("mnist.npz")
X = data["X"]
X.shape
```

Out[262…

```
(70000, 784)
```

In [263…

```
# On the MNIST dataset, how many principal components are needed to explain 98% of
from sklearn.decomposition import PCA
embedding = PCA()
embedding.fit(X)

embedding = PCA(n_components = 0.98)
X_low = embedding.fit_transform(X)
embedding.n_components_, embedding.explained_variance_ratio_.sum()
```

Out[263…

```
(260, 0.9800213049893107)
```

Problem 2. Consider the MNIST dataset again. The data shape should indicate there are 70,000 samples with 784 features each. Now suppose we want to reduce the number of dimensions in this dataset by an order of magnitude, and find out how the total explained variance changes going from one case to the next.

Specifically, run PCA independently 3 times on this data reducing the data to the dimensions assigned to you in Canvas. **How much variance is explained by this many principal components?**

In [264…
```
data = np.load("mnist.npz")
X = data["X"]
```

In [265…
```
embedding = PCA(n_components = 2)
X_low = embedding.fit_transform(X)
embedding.n_components_, embedding.explained_variance_ratio_.sum()
0.16901560482659653*100
```

Out[265…
```
16.901560482659654
```

Problem 3. Suppose you have a dataset with a large number of dimensions, and want to perform clustering on it in a lower dimensional space.

On the MNIST dataset, **reduce the number of dimensions to 3 (from 784) using PCA, and then run KMeans clustering on this reduced dataset assuming 10 clusters** since we have the prior knowledge that there are 10 categories of digits in MNIST, 0-9.

Which cluster centers are obtained through this process? The cluster centers below are accurate to at least one unit.

`Note:` Set `random_state=42` in both KMeans and PCA. e.g., `PCA(n_components=x, random_state=42)` . This is similar to setting a random seed.

In [31]:
```
data = np.load("mnist.npz")
X = data["X"]
X.shape
```

Out[31]:
```
(70000, 784)
```

In [41]:
```
from sklearn.cluster import KMeans

X_low = PCA(n_components=3, random_state=42).fit_transform(X)
kmeans = KMeans (n_clusters = 10, random_state=42 )
labels_hat = kmeans.fit_predict(X_low)

cluster_centers = kmeans.cluster_centers_
cluster_centers
```

```
C:\ProgramData\Anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:870: FutureWar
ning: The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the va
lue of `n_init` explicitly to suppress the warning
  warnings.warn(
Out[41]: array([[ -94.64831658,  652.73685857,  338.39275519],
               [ 762.61535829, -113.46982348,  702.57124568],
               [ 206.00746714, -387.80816029, -737.58537175],
               [-810.2162059 , -414.68248194,  116.03452355],
               [ 393.67202451,   58.99656311,  -15.68372627],
               [1229.00397561, -271.28009592, -161.92366748],
               [  67.5929418 ,  746.87060358, -382.35558561],
               [-457.58588716,  283.38330233, -196.77507067],
               [   8.11524579, -593.47851781, -125.86574999],
               [-148.40464129, -143.64691583,  489.28665701]])
```

Problem 4.

1. Perform PCA on the data (`X`) in `synthetic1.csv` with `n_components=1` .
2. Now, get the inverse transform on the transformed data to reconstruct your input to it's original dimensions. Let's call this reconstructed data `X_hat` .

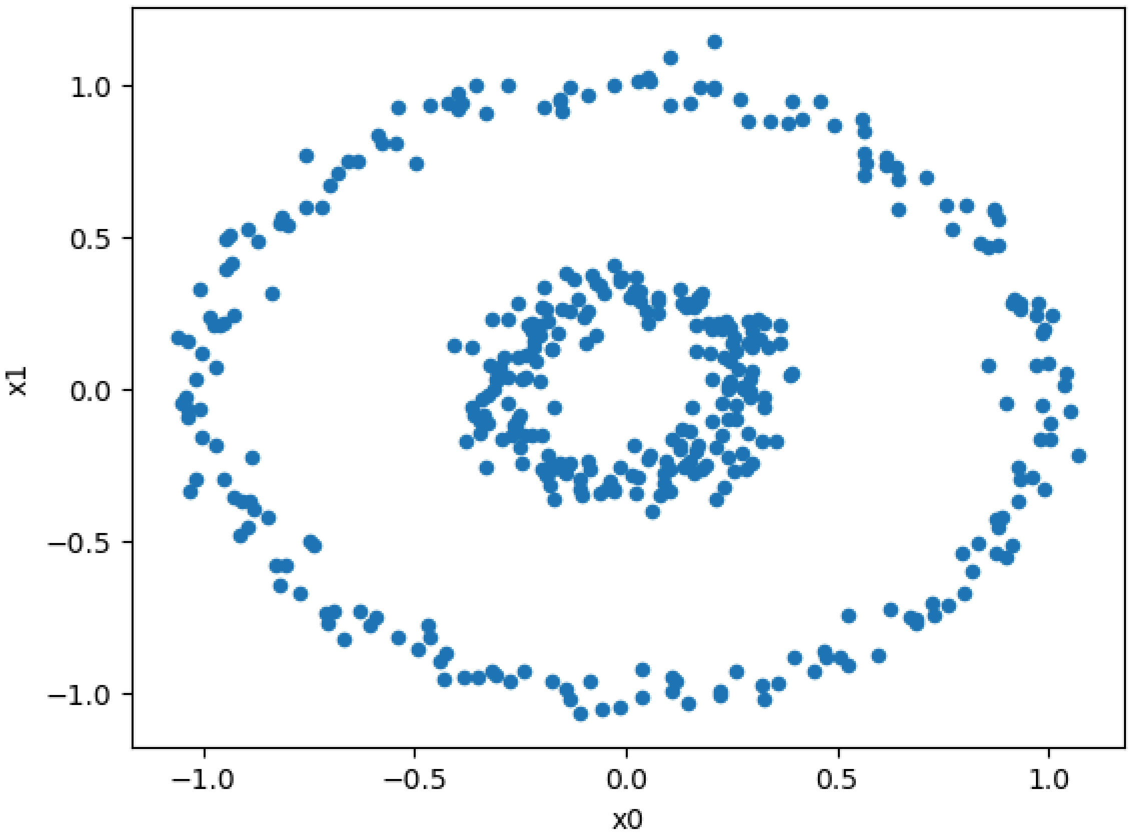
For sanity check: What shape is the data in `X` , and what is the shape of `X_hat` = inverse_transform(fit_transform(X))? Ideally, `X_hat` should be similar to `X` .

Mark all answers that are true.

`NOTE:` To visualize the data, make a scatter plot.

```
In [266…  df_s1 = pd.read_csv("synthetic1.csv")

          # Use X for PCA
          X = df_s1.values
          print("Shape of X:", X.shape)

          df_s1.head()
          ax = df_s1.plot.scatter(x = 0, y = 1)
```

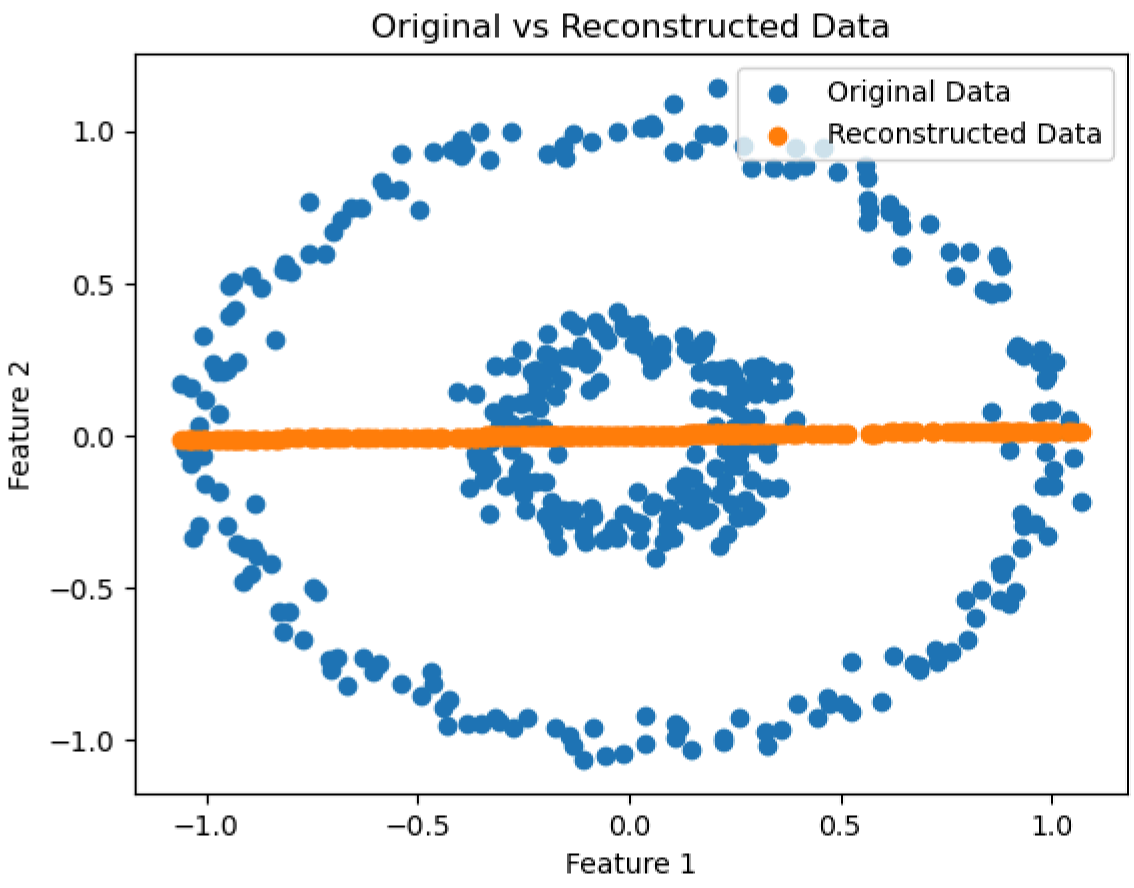
```
Shape of X: (400, 2)
```



In [269…

```
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA

pca = PCA(n_components=1)

X_transformed = pca.fit_transform(X)

# Inverse transform to reconstruct original data
X_hat = pca.inverse_transform(X_transformed)

print("Shape of X:", X.shape)
print("Shape of X_hat:", X_hat.shape)

# original and reconstructed plots
plt.scatter(X[:, 0], X[:, 1], label='Original Data')
plt.scatter(X_hat[:, 0], X_hat[:, 1], label='Reconstructed Data')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.legend()
plt.title('Original vs Reconstructed Data')
plt.show()
```

```
Shape of X: (400, 2)
Shape of X_hat: (400, 2)
```



Problem 5. Which feature in `cars.csv` has the variance with the highest numerical value?

```
In [44]: df_cars = pd.read_csv("cars.csv")
         df_cars.head()
```

Out[44]:

	Acceleration	Cylinders	Displacement	Horsepower	Miles_per_Gallon	Weight_in_lbs
0	12.0	8	307.0	130.0	18.0	3504
1	11.5	8	350.0	165.0	15.0	3693
2	11.0	8	318.0	150.0	18.0	3436
3	12.0	8	304.0	150.0	16.0	3433
4	10.5	8	302.0	140.0	17.0	3449

```
In [47]: variances = df_cars.var()
         highest_variance_feature = variances.idxmax()
         highest_variance_feature
```

```
Out[47]: 'Weight_in_lbs'
```

Problem 6. Perform PCA on the `cars.csv` dataset, and find 6 principal components. The data can be transformed by PCA into 6 dimensions.
Now, perform the inverse transform on the transformed data (e.g., `X_hat = inverse_transform(transform(X))`).

Is the approximation `X_hat` equal to the original data `X`, and why?

`NOTE:` To check if they are same, we can just subtract the two vectors and find the absolute sum. Round the sum to 2 decimal place and check if its 0.0. If this sum is zero, we'll know that they are same.

In [271…
```
df_cars = pd.read_csv("cars.csv")
X = df_cars.values
```

In [274…
```
# PCA with 6 principal components
pca = PCA(n_components=6)
X_pca = pca.fit_transform(X)

# inverse transform to reconstruct the original data
X_hat = pca.inverse_transform(X_pca)

# Check if X_hat is equal to the original data X
absolute_sum = np.sum(np.abs(X - X_hat))
if round(absolute_sum, 2) == 0.0:
    print("X_hat is equal to the original data X.")
else:
    print("X_hat is not equal to the original data X.")
```

```
X_hat is equal to the original data X.
```

Problem 7. Import the data in the `cars.csv` dataset.

`Part 1` Find the number of principle components required to explain Y% (Y is given to you in Canvas) of the variance.

`Part 2` Standardize the original data using `StandardScaler`. Now find the number of principle components required to explain Y% of the variance.

Does standardizing the data change the number of principal components required to explain Y% of the variance in Parts I and II? If so, please elaborate on why. If not, does this result match your expectations?

In [275…
```
df_cars = pd.read_csv("cars.csv")
X = df_cars.values
df_cars.head()
```

Out[275…

	Acceleration	Cylinders	Displacement	Horsepower	Miles_per_Gallon	Weight_in_lbs
0	12.0	8	307.0	130.0	18.0	3504
1	11.5	8	350.0	165.0	15.0	3693
2	11.0	8	318.0	150.0	18.0	3436
3	12.0	8	304.0	150.0	16.0	3433
4	10.5	8	302.0	140.0	17.0	3449

In [276…

```
#Part-1: Find the number of principal components required to explain 99% of the var

from sklearn.decomposition import PCA
embedding = PCA()
embedding.fit(X)

embedding = PCA(n_components = 0.99)
X_low = embedding.fit_transform(X)
embedding.n_components_, embedding.explained_variance_ratio_.sum()
```

Out[276…

```
(1, 0.9975535062891329)
```

In [61]:

```
#Part-2
from sklearn.decomposition import PCA

scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

embedding = PCA()
embedding.fit(X_scaled)

embedding = PCA(n_components = 0.99)
X_low = embedding.fit_transform(X_scaled)
embedding.n_components_, embedding.explained_variance_ratio_.sum()
```

Out[61]:

```
(5, 0.9939530000311947)
```

In [282…

```
#item 2 Q1, Q2
def longest_substring(s):
    max_length = 0
    current_length = 0
    stack = []

    for char in s:
        if char == '{' or char == '[' or char == '(':
            stack.append(char)
            current_length +=1
        elif char == '}' or char == ']' or char == ')':
            if stack and ((char == '}' and stack[-1] == '{') or
                          (char == ']' and stack [-1] == '[') or
                          (char == ')' and stack [-1] == '(')):
                stack.pop()
                current_length += 1
            else:
                max_length = max(max_length, current_length)
                current_length = 0
                stack = []
        else:
            current_length += 1
    max_length = max(max_length, current_length)
    return max_length

s2 = 'egc[]aGMgzROGwVvc[{(Y{}{})[(){}V]wJX()}](Kf[F]())m)()l)(W){}DgrFPRMSSr[B]X(y{
longest_substring(s2)
```

Out[282…

```
95
```

In [158…

```
import numpy as np
from sklearn.decomposition import PCA

def num_components_for_variance(data, k):
    pca = PCA()
    pca.fit(data)
    cumulative_variance = np.cumsum(pca.explained_variance_ratio_)
    num_components = np.argmax(cumulative_variance >= k) + 1
    return num_components

# Generate some random data
num_samples = 2000
num_features = 40
random_data = np.random.rand(num_samples, num_features)

# Set the desired minimum variance (e.g., 0.9 for 90% variance)
k = 0.9

required_components = num_components_for_variance(random_data, k)
required_components
```

Out[158…

```
35
```