ECE657A Assignment № 2

He Bing, 20848700 , b29he@uwaterloo.ca

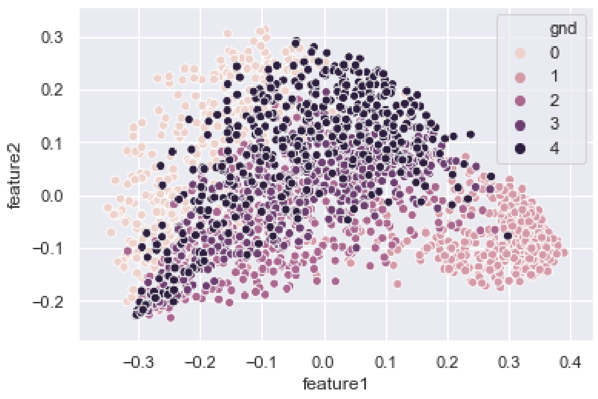
Feb 2020

PLEASE SEE THE JUPYTER NOTEBOOK PDF AND CODE IN THE END OF THE REPORT!!!!

Question3

The pictures are listed below

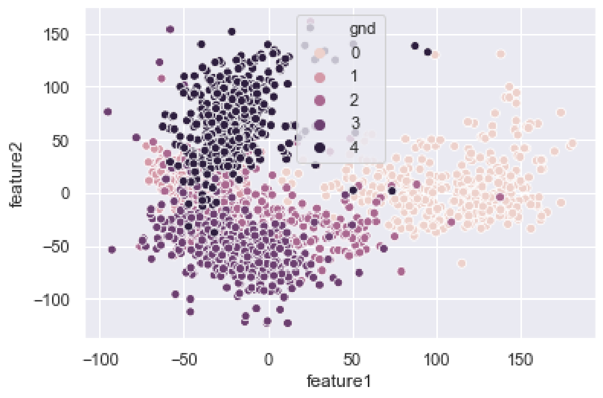
Figure 1: kernel_PCA



kernel_pca

Kernel PCA works better to separate digits 1 and others. It can be seen that kernel PCA has better performance on hand-written data set classification. Kernel PCA use kernel function to extend the dimension of the data to solve non-linear problems.
If the data is linearly inseparable, that is to say, they cannot be separated within current dimension. What we should do under this circumstance is to map the data into higher dimensional space. This is basically what kernel PCA does.
Kernel function looks like $K(x_i, x_j) = \Phi(x_i)^T \Phi(x_j)$
And there are many different kinds of kernel functions, and what we use in this problem is 'rnf'.
The kernel function is $K(x, x') = exp(-\frac{\|x-x'\|^2}{2\sigma^2})$. The value of kernel function is the new value of the point in the extended dimension.
However, kernel PCA runs far faster than the other 4 models, which is 1.144 s.

Figure 2: Isomap



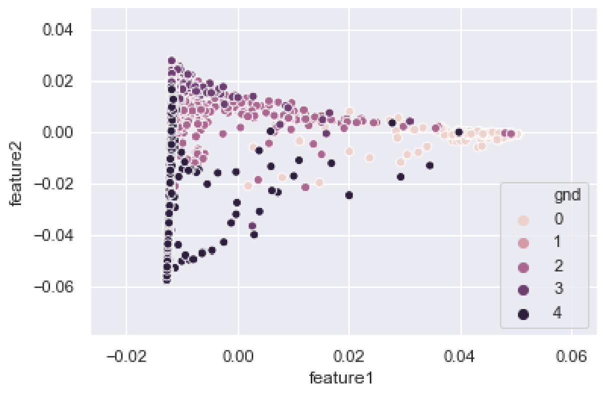
Isomap

0, 4 and 2 can be separated clearly in isomap model. Digit 1 is covered by 3 and 4, which cannot be seen in the plot. Isomap is a non-linear method to reduce the dimension of "Swiss Roll"-like data. For two arbitrary points on non-linear manifold, their Euclidean distance may not accurately reflect their real similarity. So we need isomap to solve this problem.

In isomap, we define a constant names e, and for each data point, we find all its neighbors which satisfy this condition: $d_X(i,j) \leq e$. Then we construct a graph on the manifold between i and j if $d_X(i,j) \leq e$ and in the mean time, find the shortest path between them names $d_G(i,j)$. Finally, we apply MDS on $d_G(i,j)$.

It is obvious that isomap has better performance than kernel PCA with rbf kernel. But its executing time is also a lot longer than kernel PCA.

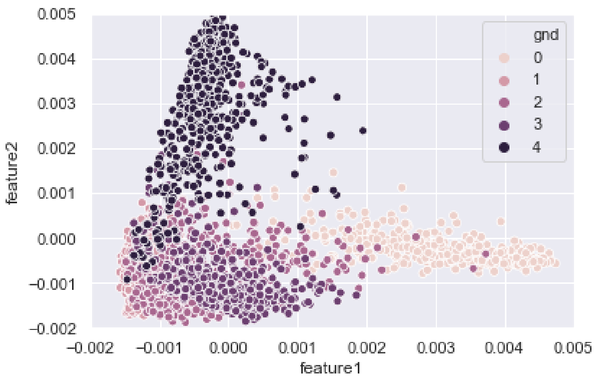
Figure 3: LLE



LLE

LLE reform the points into a triangle. We cannot clearly see points of 1.(Because they are covered by the other points.) The classification is not very clear from the 2-components plot. However, from the higher dimension graphs, things turned to be different.(Details can be seen in jupyter notebook file) Points of same category are clustered closely in random higher space. The LLE algorithm is based on geometric intuition. For each data point i in p dims, we find K nearest neighbors N(i) in Euclidean distance. Then compute a local principle component plane to the points in the neighborhood, minimizing $\sum_i |x_i - \sum_{j \in N(i)} W_{ij} x_j|^2$, which is also called cost function. Weights W_{ij} satisfy $\sum_j W_{ij} = 1$. Finally, we find the points y_i in a lower dimension space to minimize $\sum_i |y_i - \sum_{j \in N(i)} W_{ij} y_j|^2$ with W_{ij} fixed.

Figure 4: Laplacian Eigenmap



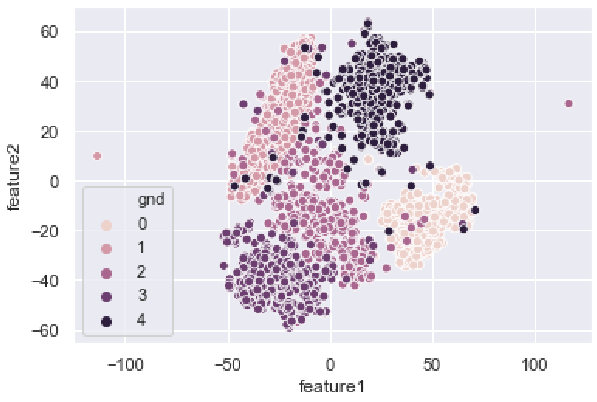
Laplacian Eigenmap

Laplancian Eigenmap can separate 0 and 4 from other digits properly compared with other models. But points of 1, 2 and 3 are mixed together. Laplacian embedding is based on the eigenvectors and eigenvalues of the the generalized eigenvalue system.

$$Lx = \lambda Dx \tag{1}$$

where L is the Laplacian matrix L = D S, S is a weight matrix that encodes the connections between nodes in a graph, and D is a diagonal matrix whose entries contain the row or column sums of S.

Figure 5: t-SNE



t-SNE

From the 2D pictrues of all the 5 models, t-SNE is the best method that can split all the points clearly. But in the mean time, it also cost the far longer time compared with other methods.
t-SNE is a state-of-art classification method, the main idea of it is as follow: we place the points on a line randomly. And make movement to all the points slightly every time. And the moving distance based on all the other points, if they belong to one category, then they move closer, but if they belong to different categories, they move farther. We add all the distance together and get one final value. And after many times' movement, the points are split very clearly.

question3

February 28, 2020

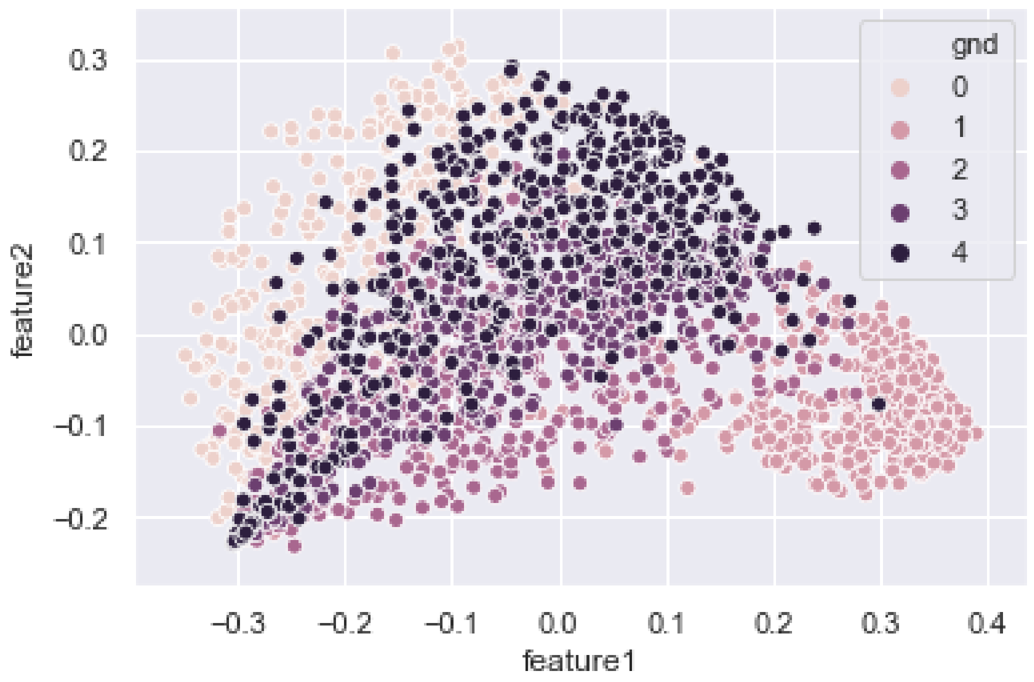
[1]:
```
import numpy as np
import pandas as pd
import random
import seaborn as sns; sns.set()
from sklearn import neighbors
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
import sklearn.preprocessing
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler
```

[2]:
```
handWritten = pd.read_csv("DataB.csv", sep=',')
handWritten.drop(columns=[handWritten.columns[0]],axis=1,inplace=True)
V = ['fea.'+str(i+1) for i in range(784)]
R = 'gnd'
VR = V +[R]
SS = StandardScaler()
handWritten[V] = SS.fit_transform(handWritten[V])
```

[3]:
```
import time
```

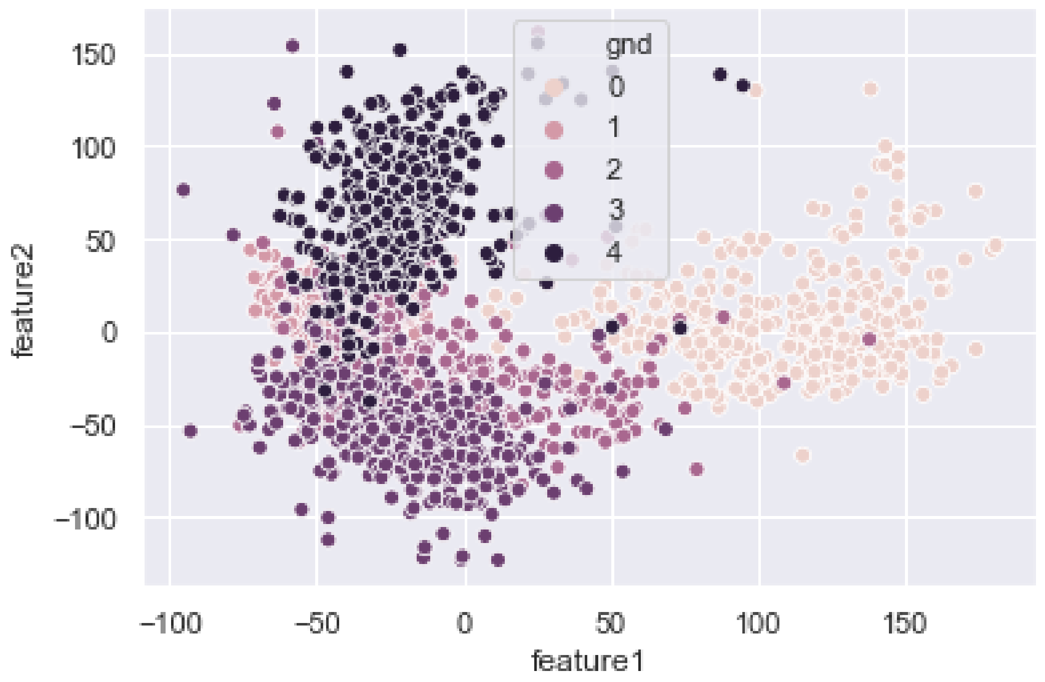
[4]:
```
from sklearn.decomposition import KernelPCA
start = time.time()
transformer = KernelPCA(n_components=2, kernel='rbf')
kernelPCAData = pd.DataFrame(transformer.fit_transform(handWritten[V]))
kernelPCAData.columns = ['feature1','feature2']
kernelPCAData['gnd'] = handWritten['gnd']
g = sns.scatterplot(x="feature1", y="feature2",
                    hue="gnd", data=kernelPCAData ,legend = 'full')
end = time.time()
print(str(end - start))
```

```
0.956899881362915
```



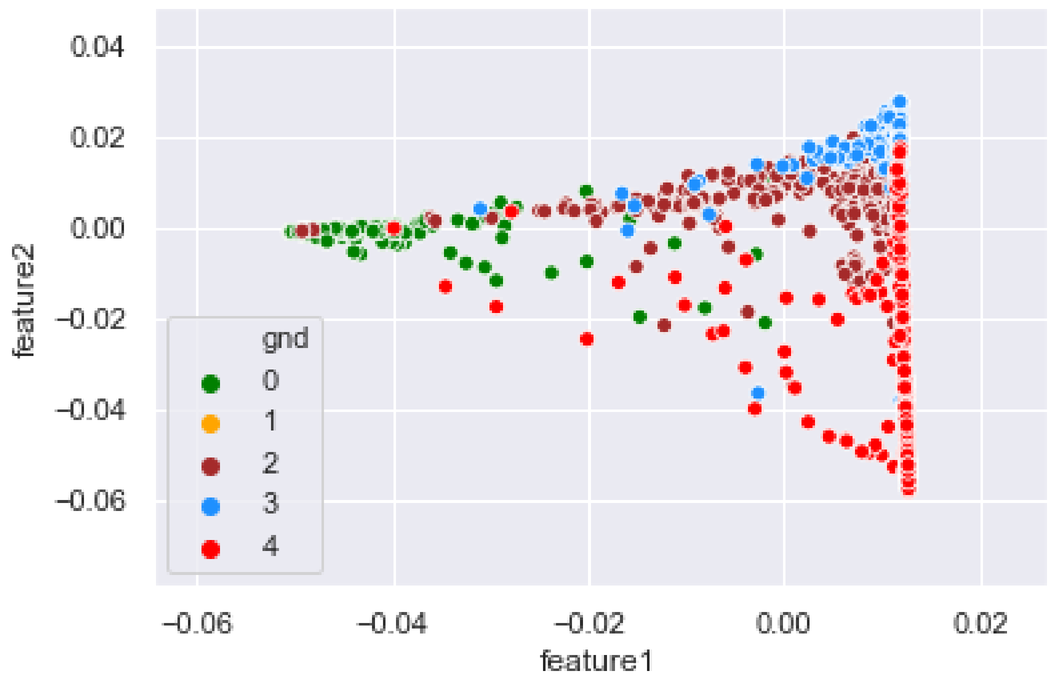
```
[5]: from sklearn.manifold import Isomap
     start = time.time()
     transformer = Isomap(n_components=2)
     IsomapData = pd.DataFrame(transformer.fit_transform(handWritten[V]))
     IsomapData.columns = ['feature1','feature2']
     IsomapData['gnd'] = handWritten['gnd']
     g = sns.scatterplot(x="feature1", y="feature2",
                          hue="gnd", data=IsomapData,legend = 'full')
     end = time.time()
     print(str(end - start))
```

```
14.923115015029907
```



```
[6]: from sklearn.manifold import LocallyLinearEmbedding
     start = time.time()
     transformer = LocallyLinearEmbedding(n_components=2, random_state = 42)
     LLEData = pd.DataFrame(transformer.fit_transform(handWritten[V]))
     LLEData.columns = ['feature1','feature2']
     LLEData['gnd'] = handWritten['gnd']
     g = sns.scatterplot(x="feature1", y="feature2",
                          hue="gnd", data=LLEData,legend =␣
      ↪'full',palette=['green','orange','brown','dodgerblue','red'])
     end = time.time()
     print(str(end - start))
```

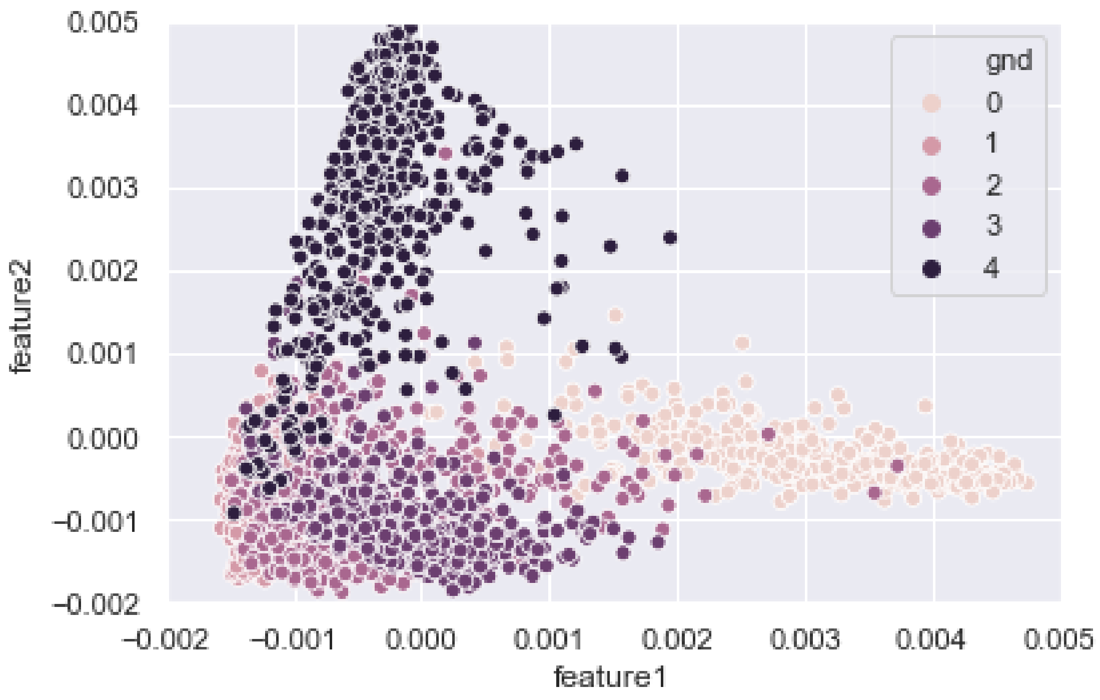
```
10.226557970046997
```



I added some plots to see if LLE can separate the points in higher dimensions.

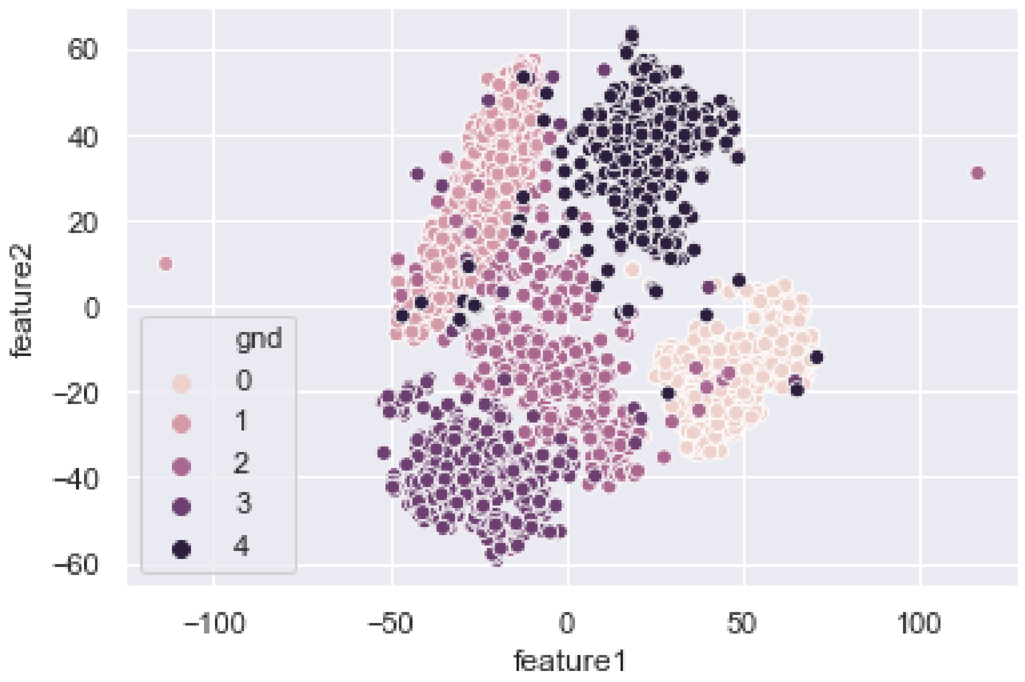
```
[7]: from sklearn.manifold import SpectralEmbedding
     start = time.time()
     transformer = SpectralEmbedding(n_components=2)
     SEData = pd.DataFrame(transformer.fit_transform(handWritten[V]))
     SEData.columns = ['feature1','feature2']
     SEData['gnd'] = handWritten['gnd']
     g = sns.scatterplot(x="feature1", y="feature2",
                          hue="gnd", data=SEData,legend = 'full')
     g.set(xlim=(-0.002,0.005))
     g.set(ylim=(-0.002,0.005))
     end = time.time()
     print(str(end - start))
```

```
11.882560968399048
```



[8]:
```
from sklearn.manifold import TSNE
start = time.time()
transformer = TSNE(n_components=2, random_state = 42)
TSNEData = pd.DataFrame(transformer.fit_transform(handWritten[V]))
TSNEData.columns = ['feature1','feature2']
TSNEData['gnd'] = handWritten['gnd']
g = sns.scatterplot(x="feature1", y="feature2",
                     hue="gnd", data=TSNEData,legend = 'full')
end = time.time()
print(str(end - start))
```

```
60.011224031448364
```


feature2
feature1
gnd
0
1
2
3
4
60
40
20
0
−20
−40
−60
−100
−50
0
50
100

```
[ ]:
```