ECE657A Assignment № 2

He Bing, 20848700 , b29he@uwaterloo.ca Feb 2020

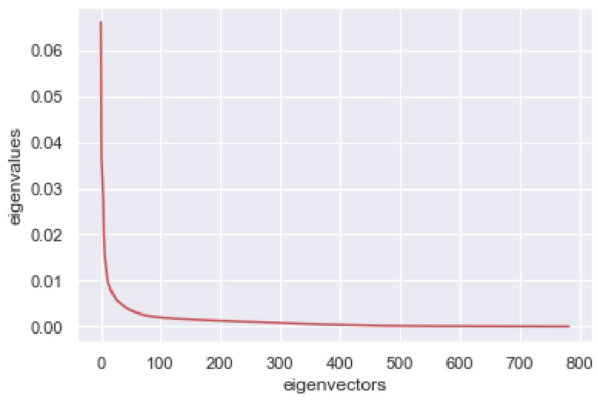
PLEASE SEE THE JUPYTER NOTEBOOK PDF AND CODE IN THE END OF THE REPORT!!!!

Problem 2

Principal Component Analysis (PCA)

Plot the scree plot and visually discuss which cut-off is good

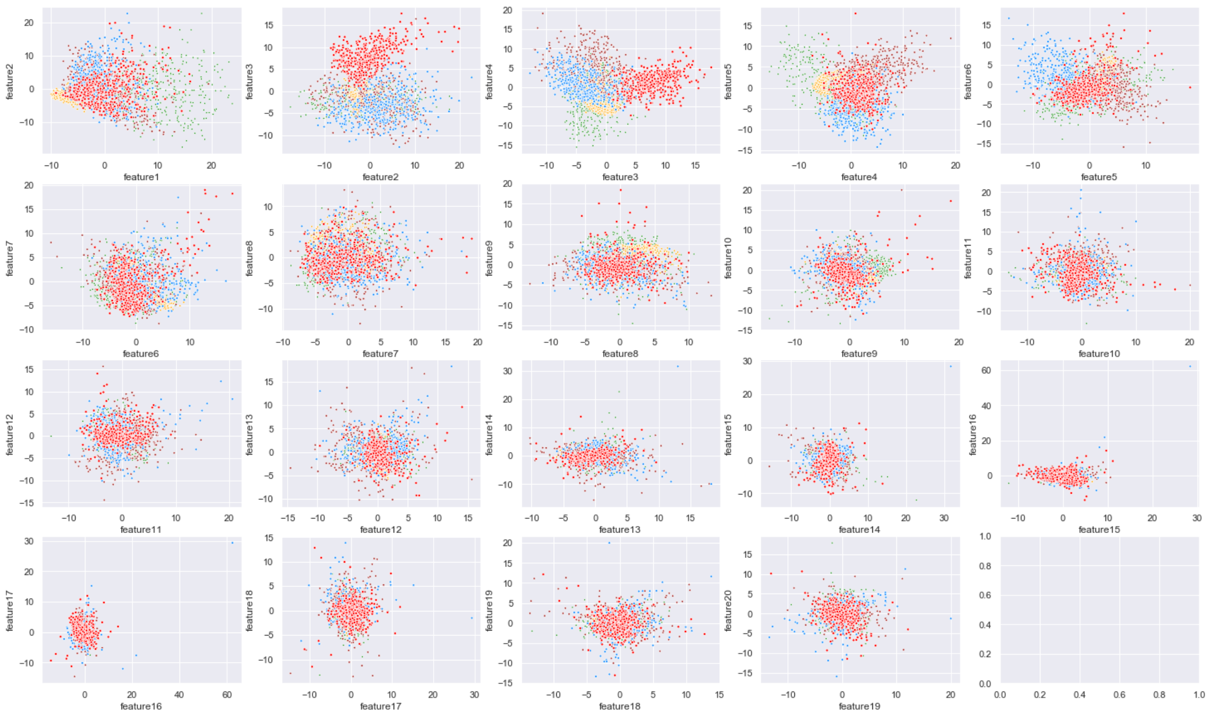
Figure 1: Scree plot of PCA



The cut-off can be around at 10-20. We can see the eigenvalues drop sharply when the index of eigenvector grows. Generally speaking, the first K components take up most part of all the features under PCA. And the eigenvector that is after 50's eigenvalue is very near to 0. So actually, they contribute a very little part to the classification.

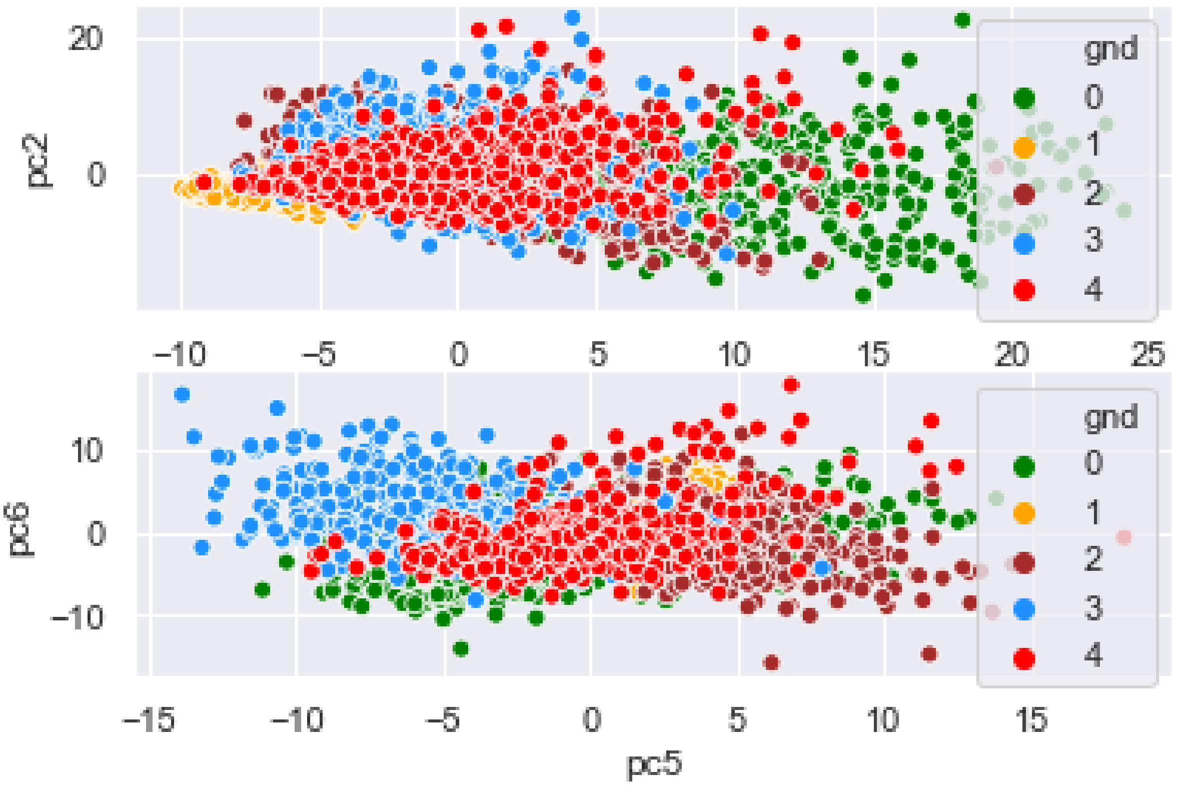
Scatter plot of the projected data with the top 20 eigenvalues

Figure 2: scatter plot of PCA



After observation, we found that the classification after fea.6 vs fea.7 is very vogue. So the feature after 7 can be cut off. The result we draw from this picture is very close to that of above. The more clear the subplot's classification is, the greater eigenvalue is. But we cannot have a accurately have a conclusion of which point is best to cut in the scree plot.

fea.1 and fea.2 vs fea.5 and fea.6



We cannot tell which picture has a better classification towards the points, but 0 can be split from other points in the first plot, 3 can be separated in the second picture. Although the principle component in PCA takes up the most part, we cannot do a great job on classifying just according to the first feature and the second feature. More features are beneficial to classification. So we can use PCA and set n_components as 7 to get a good performance as good as all features and save a lot of time.

implement

PCA

Listing 1: PCA

```
time_start = time.perf_counter()
eigValueU , eigVectorU = linalg.eigh(np.dot(data.T, data))
indexU = eigValueU.argsort()[::-1]
eigValueU = eigValueU[indexU]
eigVectorU = eigVectorU[:,indexU ]
X_pca = eigVectorU.T.dot(data.T).T
time_over = time.perf_counter()
print("running time: "str(time_over-time_start))
print(X_pca)
```

dual PCA using SVD

Listing 2: svd

```python
def my_svd(X):
    n, m = X.shape
    eigValueU , eigVectorU = linalg.eigh(np.dot(X, X.T))
    eigValueV , eigVectorV = linalg.eigh(np.dot(X.T, X))
    indexU = eigValueU.argsort()[::-1]
    indexV = eigValueV.argsort()[::-1]
    eigValueU = eigValueU[indexU]
    eigVectorU = eigVectorU[:,indexU ]
    eigValueV = eigValueV[indexV]
    eigVectorV = eigVectorV[:,indexV]
    if n>m:
        sigma=np.sqrt(eigValueU)
    else:
        sigma=np.sqrt(eigValueV)
    return eigVectorU, sigma ,eigVectorV.T
```

Listing 3: dual PCA via SVD

```python
time_start = time.perf_counter()
U,s,VT = my_svd(data.T)
time_over = time.perf_counter()
print("svd running time: "+str(time_over-time_start))
time_start = time.perf_counter()
X_dual_PCA = np.diag(s).dot(VT)[:784].T
time_over = time.perf_counter()
print("dual pca running time: "+str(time_over-time_start))
print(X_dual_PCA)
```

Time comparing

The running time of PCA is 0.35 s, svd is 2.63 s and dual PCA is 0.39 s. For this data set, the number of dimension is 784 (28 * 28), in the mean time, the number of samples is 2066. Only when the number of samples far greater than that of dimension, dual PCA will be faster than PCA. Because the running time of dual PCA depends on the number of the samples.

Prove that PCA is the best linear method for reconstruction

Suppose the line that make classification represented as:

$$\boldsymbol{L} = \boldsymbol{b} + \alpha \boldsymbol{v} \tag{1}$$

assume $\|v\| = 1$ for convenience.
And each instance x_i is associated with a point on the line $\overset{\wedge}{x_i} = \boldsymbol{b} + \alpha_i \boldsymbol{v}$.
And reconstruction error can be written to:

$$R = \sum_{i=1}^{m} \|\boldsymbol{x_i} - \overset{\wedge}{\boldsymbol{xi}}\|^2 \tag{2}$$

Our goal is to minimize the reconstruction error. We can rewrite equation (2) into:

$$R = \sum_{i=1}^{m} \|\boldsymbol{x_i} - (\boldsymbol{b} + \alpha_i \boldsymbol{v})\|^2 \tag{3}$$

We write the gradient of R wrt. a_i and set it to 0.

$$\frac{\partial R}{\partial a_i} = 2\|\boldsymbol{v}\|^2 a_i + 2\boldsymbol{v}\boldsymbol{x}_i + 2\boldsymbol{b}\boldsymbol{v} = 0 \tag{4}$$

We draw from equation(4):

$$\alpha_i = \boldsymbol{v}(\boldsymbol{x}_i - \boldsymbol{b}) \tag{5}$$

We write the gradient of R wrt. $\boldsymbol{b}$ and set it to 0.

$$\frac{\partial R}{\partial \boldsymbol{b}} = 2m\boldsymbol{b} - 2\sum_{i=1}^{m} \boldsymbol{x}_i + 2\left(\sum_{i=1}^{m} \alpha_i\right)\boldsymbol{v} = 0 \tag{6}$$

$$\sum_{i=1}^{m} \alpha_i = \boldsymbol{v}^T \left(\sum_{i=1}^{m} \boldsymbol{x}_i - m\boldsymbol{b}\right) \tag{7}$$

By plugging (7) into (6), we get:

$$\boldsymbol{v}^T \left(\sum_{i=1}^{m} \boldsymbol{x}_i - m\boldsymbol{b}\right)\boldsymbol{v} = \left(\sum_{i=1}^{m} \boldsymbol{x}_i - m\boldsymbol{b}\right) \tag{8}$$

This is satisfied when $\left(\sum_{i=1}^{m} \boldsymbol{x}_i - m\boldsymbol{b}\right) = 0$, which means:

$$\boldsymbol{b} = \frac{1}{m}\sum_{i=1}^{m} \boldsymbol{x}_i \tag{9}$$

Substituting equation (5) into the optimization problem, we get a new optimization problem:

$$max_v \sum_{i=1}^{m} \boldsymbol{v}^T (\boldsymbol{x}_i - \boldsymbol{b})(\boldsymbol{x}_i - \boldsymbol{b})^T \boldsymbol{v} (s.t. \|v\|^2 = 1) \tag{10}$$

The Lagrangian is:

$$L(\boldsymbol{v}, \lambda) = \sum_{i=1}^{m} \boldsymbol{v}^T (\boldsymbol{x}_i - \boldsymbol{b})(\boldsymbol{x}_i - \boldsymbol{b})^T \boldsymbol{v} + \lambda + \lambda\|v\|^2 \tag{11}$$

Let $S = \sum_{i=1}^{m} (\boldsymbol{x}_i - \boldsymbol{b})(\boldsymbol{x}_i - \boldsymbol{b})^T$, which we call it scatter matrix.

$$\frac{\partial L}{\partial \boldsymbol{v}} = 2S\boldsymbol{v} - 2\lambda\boldsymbol{v} = 0 \tag{12}$$

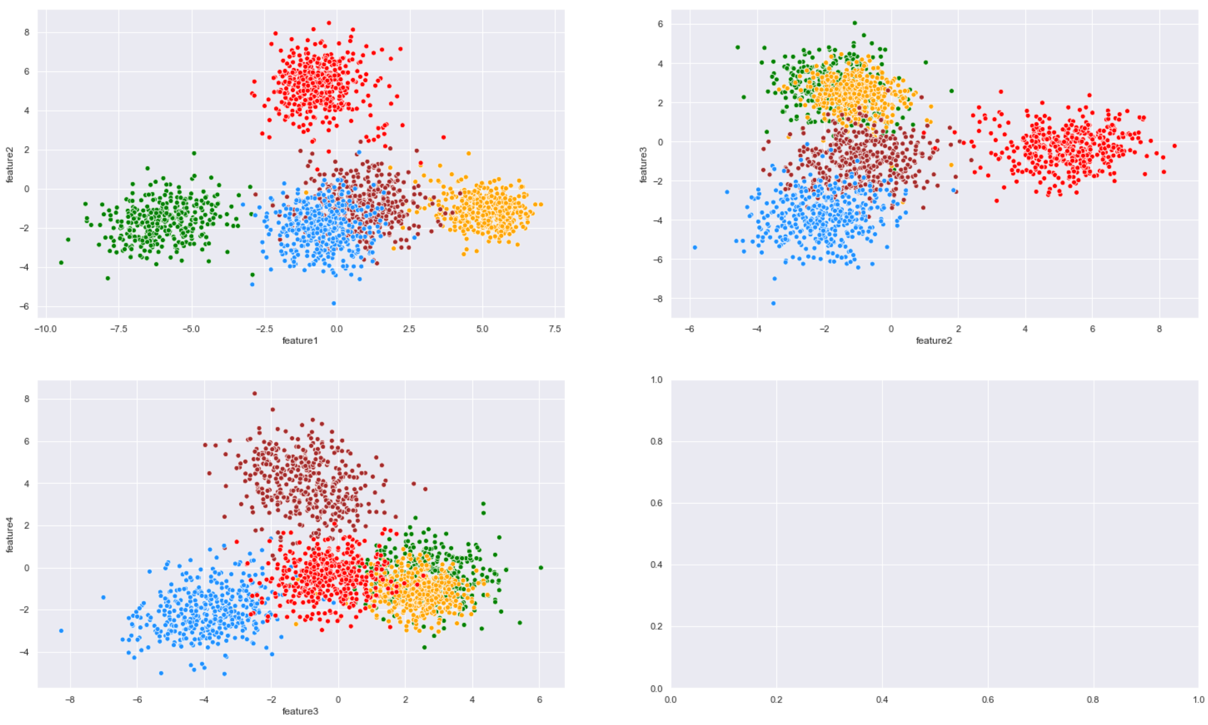
$$S\boldsymbol{v} = \lambda\boldsymbol{v} \tag{13}$$

We can know from equation (13) that when $\boldsymbol{v}$ is the eigenvector of S, we can get the minimized reconstruction error. And in the mean time, the $\boldsymbol{v}$ in PCA is the eigenvector of $X^T X$, it is the same as the S matrix we said above. So PCA is the best linear method.

FDA

The plots

Figure 3: LDA



Analysis

Plot 1 (feature 1 vs. feature 2) can separate 4 ,0 and 1 perfectly. Plot 2 (feature 2 vs. feature 3) can separate 3 and 4 perfectly. Plot 3 (feature 3 vs. feature 4) can separate 2 ,3 and 4 perfectly. But each of them is indispensable for the classifying.

Compare the results of the LDA with the results obtained by using PCA.

Firstly, LDA can only take n-1(n is the number of categories) features, so there are only 4 features extracted.The pictures of LDA look more clear than those of PCA. PCA is not a classifying method, but a dimension-reducing method. But in the mean time, LDA is a supervised classifying method. This is why LDA has a more clear classification than PCA.

Theoretical Question

The optimization of PCA is to maximize $tr(U^TSU)$ subject to $U^TU = I$, but in the mean time, LDA is to maximize $\frac{tr(U^TS_BU)}{tr(U^TS_WU)}$ subject toU^TS_WU. The matrix of U in PCA is the eigenvector of $S = X^TX$, but that of LDA is the eigenvector of $S_w^{-1}S_B$. And from the pictures, we can see that the scatter plots of LDA is far clearer than those of PCA, this is because PCA is actually not a classifying method, but a method to reduce the dimensions. But LDA will consider about the label of all the data, which is a supervised classification method.

question2

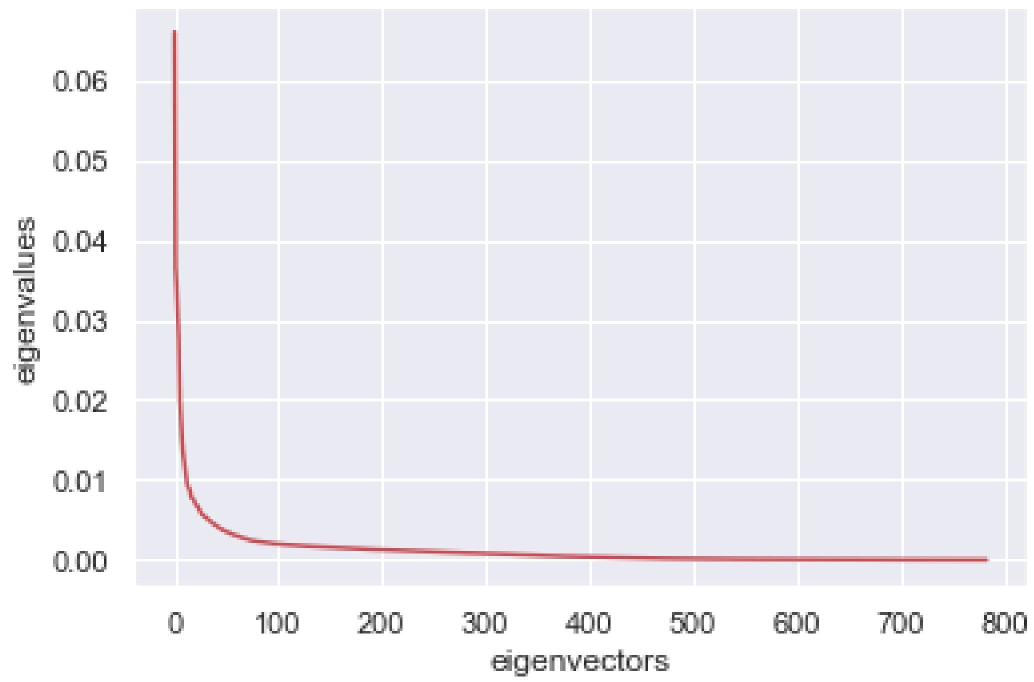
February 28, 2020

[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
```

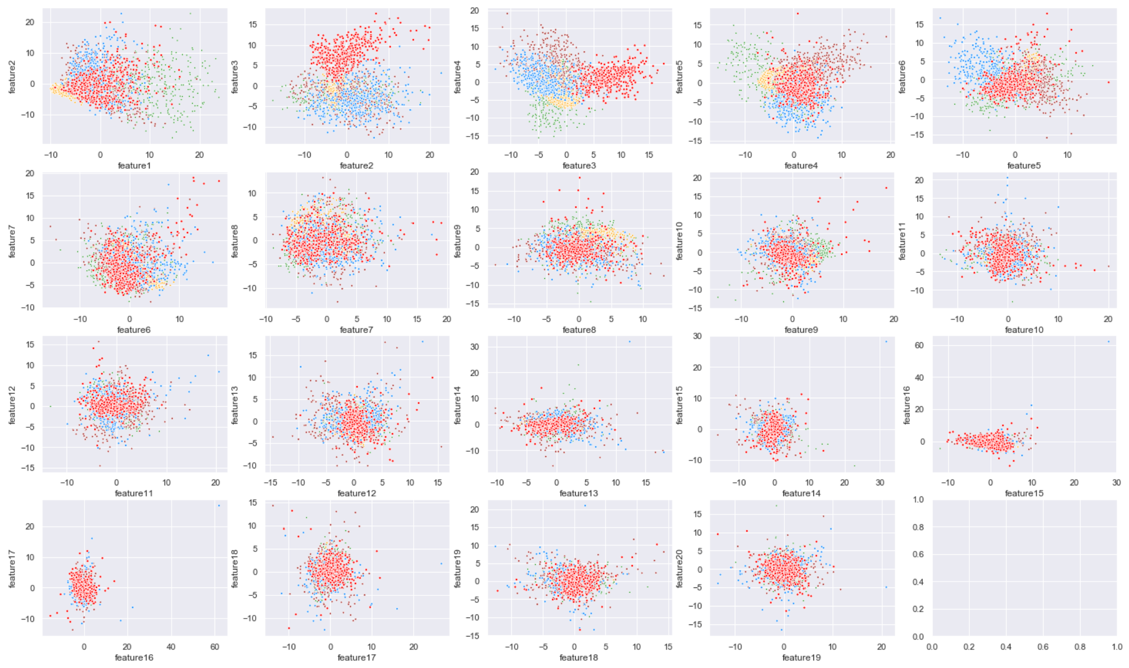
[2]:
```
handWritten = pd.read_csv("DataB.csv", sep=',')
handWritten.drop(columns=[handWritten.columns[0]],axis=1,inplace=True)
V = ['fea.'+str(i+1) for i in range(784)]
R = 'gnd'
VR = V +[R]
SS = StandardScaler()
handWritten[V] = SS.fit_transform(handWritten[V])
```

[3]:
```
from sklearn.decomposition import PCA
pca=PCA().fit(handWritten[V])
y=pca.explained_variance_ratio_
x=pca.components_
plt.xticks = [range(0,20,1)]
plt.xlabel('eigenvectors')
plt.ylabel('eigenvalues')
plt.plot(y,'r')
```

[3]:
```
[<matplotlib.lines.Line2D at 0x129b4af90>]
```

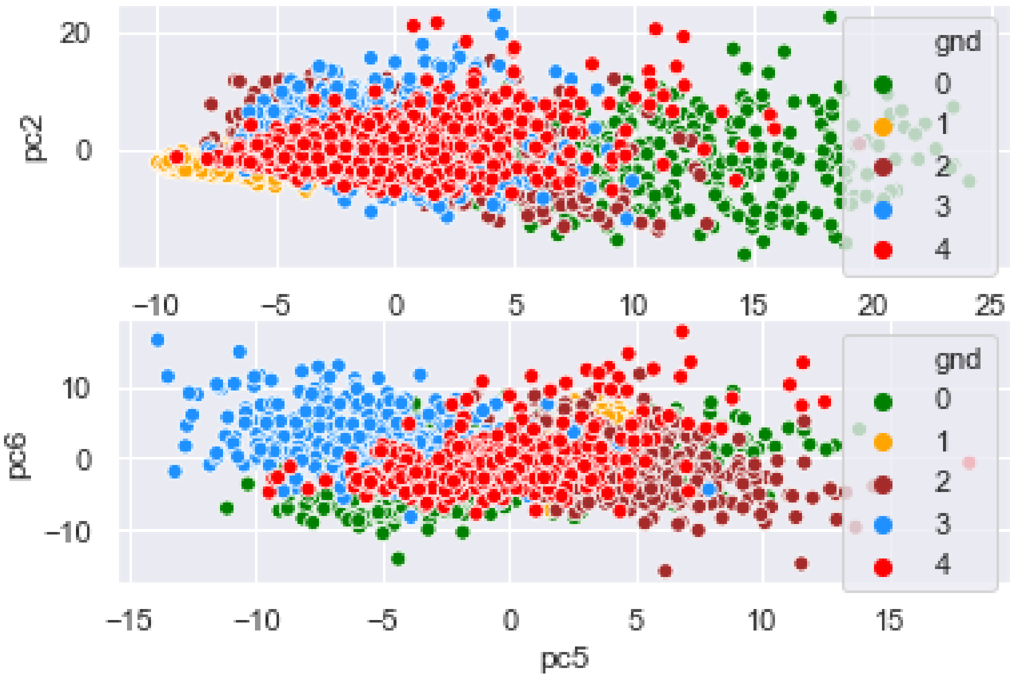


[4]:
```
transformer=PCA(n_components=20)
Data = pd.DataFrame(transformer.fit_transform(handWritten[V]))
Data.columns = ['feature'+str(i+1) for i in range(20)]
Data['gnd'] = handWritten['gnd']
fig, axes =plt.subplots(4,5,figsize=(25,15))
for j in range(19):
    sns.scatterplot(x="feature"+str(j+1), y="feature"+str(j+2),
                    ax=axes[int(j/5)][j%5],size='gnd' ,sizes=(5,10),legend =␣
 ↪False ,hue="gnd",␣
 ↪data=Data,palette=['green','orange','brown','dodgerblue','red'])
```



[5]:
```
pca=PCA(n_components=6).fit(handWritten.values)
X1=pd.DataFrame(pca.transform(handWritten))
X1.columns=['pc'+str(i+1) for i in range(6)]
X1['gnd']=handWritten['gnd']
fig, axes =plt.subplots(2,1)
sns.scatterplot(x="pc1", y="pc2",hue="gnd",ax=axes[0],␣
 ↪data=X1,palette=['green','orange','brown','dodgerblue','red'])
sns.scatterplot(x="pc5", y="pc6",hue="gnd",ax=axes[1],␣
 ↪data=X1,palette=['green','orange','brown','dodgerblue','red'])
```

[5]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1247bc390>
```



```
[6]: #svd implementation
     import numpy.linalg as linalg
     def my_svd(X):
         n, m = X.shape
         eigValueU , eigVectorU = linalg.eigh(np.dot(X, X.T))
         eigValueV , eigVectorV = linalg.eigh(np.dot(X.T, X))
         indexU = eigValueU.argsort()[::-1]
         indexV = eigValueV.argsort()[::-1]
         eigValueU = eigValueU[indexU]
         eigVectorU = eigVectorU[:,indexU ]
         eigValueV = eigValueV[indexV]
         eigVectorV = eigVectorV[:,indexV]
         if n>m:
             sigma=np.sqrt(eigValueU)
         else:
             sigma=np.sqrt(eigValueV)
         return eigVectorU, sigma ,eigVectorV.T
```

```
[7]: import time
     from statistics import mean
```

```
[8]: data = handWritten[V].values
     #pca
     time_start = time.perf_counter()
```

```
eigValueU , eigVectorU = linalg.eigh(np.dot(data.T, data))
indexU = eigValueU.argsort()[::-1]
eigValueU = eigValueU[indexU]
eigVectorU = eigVectorU[:,indexU ]
X_pca = eigVectorU.T.dot(data.T).T
time_over = time.perf_counter()
print("running time: "+str(time_over-time_start))
print(X_pca)
```

```
running time: 0.3689922300000035
[[ 9.97069222e+00  6.18172201e+00 -4.99286326e+00 … -2.84664097e-02
   6.97997657e-02  7.44883255e-02]
 [ 1.14159998e+01  6.94158705e+00 -5.06302886e+00 … -2.28147473e-01
   4.55949595e-02  6.14775845e-02]
 [ 3.69011918e+00  4.69309729e+00 -2.90865640e+00 …  1.15182625e-01
  -5.60241248e-03  3.14112641e-02]
 …
 [-3.49421529e-01  9.33681056e-01  8.10744188e+00 … -6.93521885e-02
  -8.49495279e-02 -1.65756817e-02]
 [-3.11526327e+00  2.09047425e+00  6.27251911e+00 …  1.26995790e-01
   1.46191708e-02  2.07556239e-03]
 [-5.64409375e+00 -2.46166632e-01  4.14018317e+00 …  1.57632818e-02
  -1.77113186e-02 -1.11643749e-02]]
```

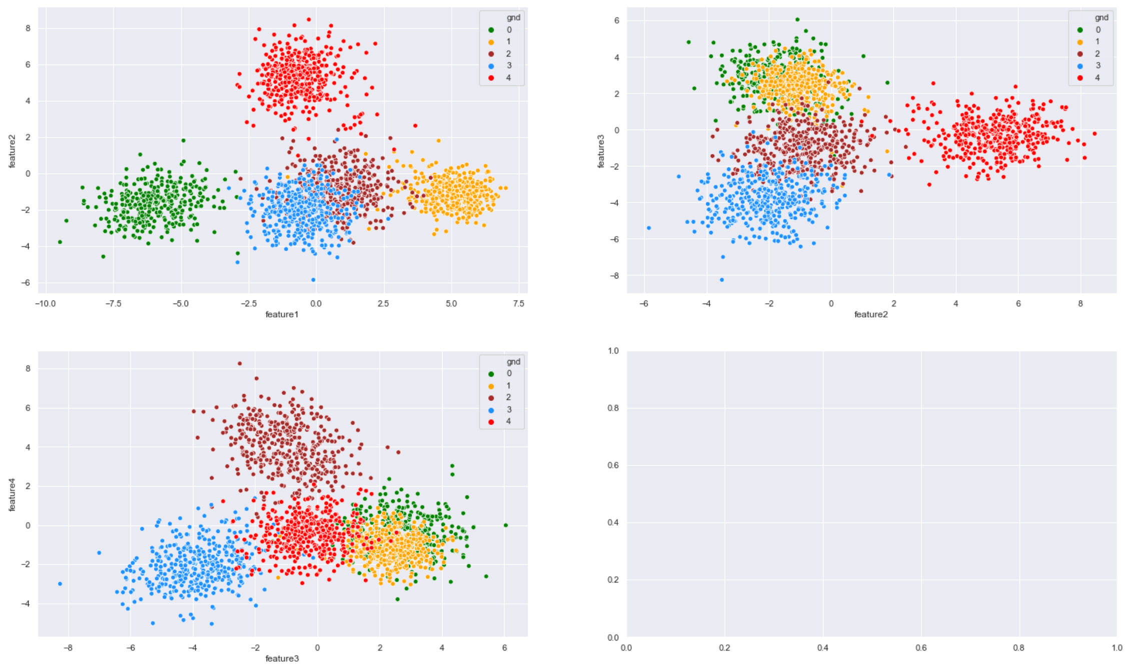
dual pca via SVD

[9]:
```
#dual_pca
time_start = time.perf_counter()
U,s,VT = my_svd(data.T)
time_over = time.perf_counter()
print("svd running time: "+str(time_over-time_start))
time_start = time.perf_counter()
X_dual_PCA = np.diag(s).dot(VT)[:784].T
time_over = time.perf_counter()
print("dual pca running time: "+str(time_over-time_start))
print(X_dual_PCA)
```

```
/usr/local/lib/python3.7/site-packages/ipykernel_launcher.py:16: RuntimeWarning:
invalid value encountered in sqrt
  app.launch_new_instance()
```

```
svd running time: 3.444919376999998
dual pca running time: 0.5355880999999982
[[ 9.97069222e+00 -6.18172201e+00  4.99286326e+00 …  2.84664097e-02
  -6.97997657e-02  7.44883255e-02]
 [ 1.14159998e+01 -6.94158705e+00  5.06302886e+00 …  2.28147473e-01
  -4.55949595e-02  6.14775845e-02]
 [ 3.69011918e+00 -4.69309729e+00  2.90865640e+00 … -1.15182625e-01
   5.60241248e-03  3.14112641e-02]
…
[-3.49421529e-01 -9.33681056e-01 -8.10744188e+00 …  6.93521885e-02
  8.49495279e-02 -1.65756817e-02]
[-3.11526327e+00 -2.09047425e+00 -6.27251911e+00 … -1.26995790e-01
 -1.46191708e-02  2.07556239e-03]
[-5.64409375e+00  2.46166632e-01 -4.14018317e+00 … -1.57632818e-02
  1.77113186e-02 -1.11643749e-02]]
```

[10]:
```python
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
transformer=LinearDiscriminantAnalysis()
Data = pd.DataFrame(transformer.fit_transform(handWritten[V],handWritten[R]))
Data.columns = ['feature'+str(i+1) for i in range(4)]
Data['gnd'] = handWritten['gnd']
fig, axes =plt.subplots(2,2,figsize=(25,15))
for j in range(3):
    sns.scatterplot(x="feature"+str(j+1), y="feature"+str(j+2),
                    ax=axes[int(j/2)][j%2],legend = 'full' ,hue="gnd",␣
 ↪data=Data,palette=['green','orange','brown','dodgerblue','red'])
```


gnd
feature1
feature2
feature3
feature4