Q1: PCA

(a) Principle Component and Transformed space

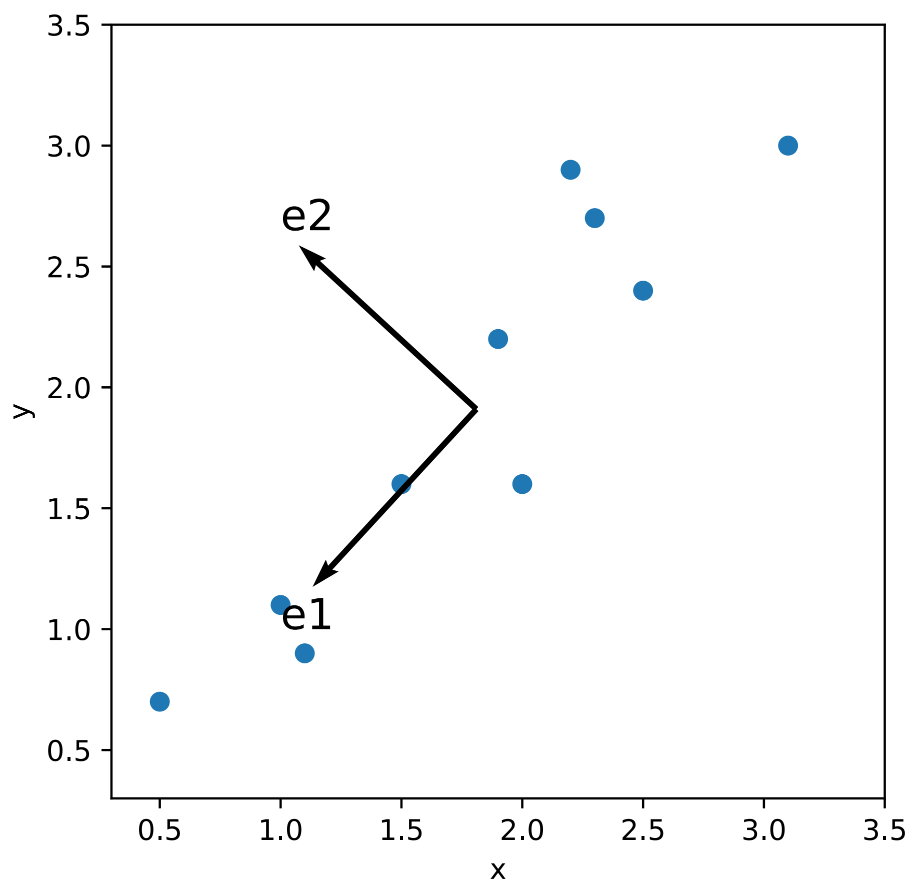
1.

In [115…
```
import numpy as np
import matplotlib.pyplot as plt
```

In [116…
```
data = np.loadtxt('q1adata.txt')
mean_a = np.mean(data,axis = 0)
D = data - mean_a
n = len(data)
cov_a = D.T@D
v,d = np.linalg.eig(cov_a)
idx = v.argsort()[::-1]
v = v[idx]
d = d[:,idx]
print("eigenvalues: ",v)
print("eigenvector: ",d)
```

```
eigenvalues:  [11.55624941  0.44175059]
eigenvector:  [[-0.6778734  -0.73517866]
 [-0.73517866  0.6778734 ]]
```

In [117…
```
fig,ax=plt.subplots(figsize = (5,5))
ax.set_aspect('equal', adjustable='box')
ax.scatter(data[:,0],data[:,1])
e1 = d[:,0]
e2 = d[:,1]
a = ax.quiver(mean_a[0], mean_a[1], e1[0], e1[1], angles='xy', scale_units='xy', scale=1)
b = ax.quiver(mean_a[0], mean_a[1], e2[0], e2[1], angles='xy', scale_units='xy', scale=1)
ax.set_xlabel("x")
ax.set_ylabel("y")
ax.set_xlim(0.3,3.5)
ax.set_ylim(0.3,3.5)
ax.text(1,2.65,"e2",size = 15)
ax.text(1,1,"e1",size = 15)
plt.show()
```

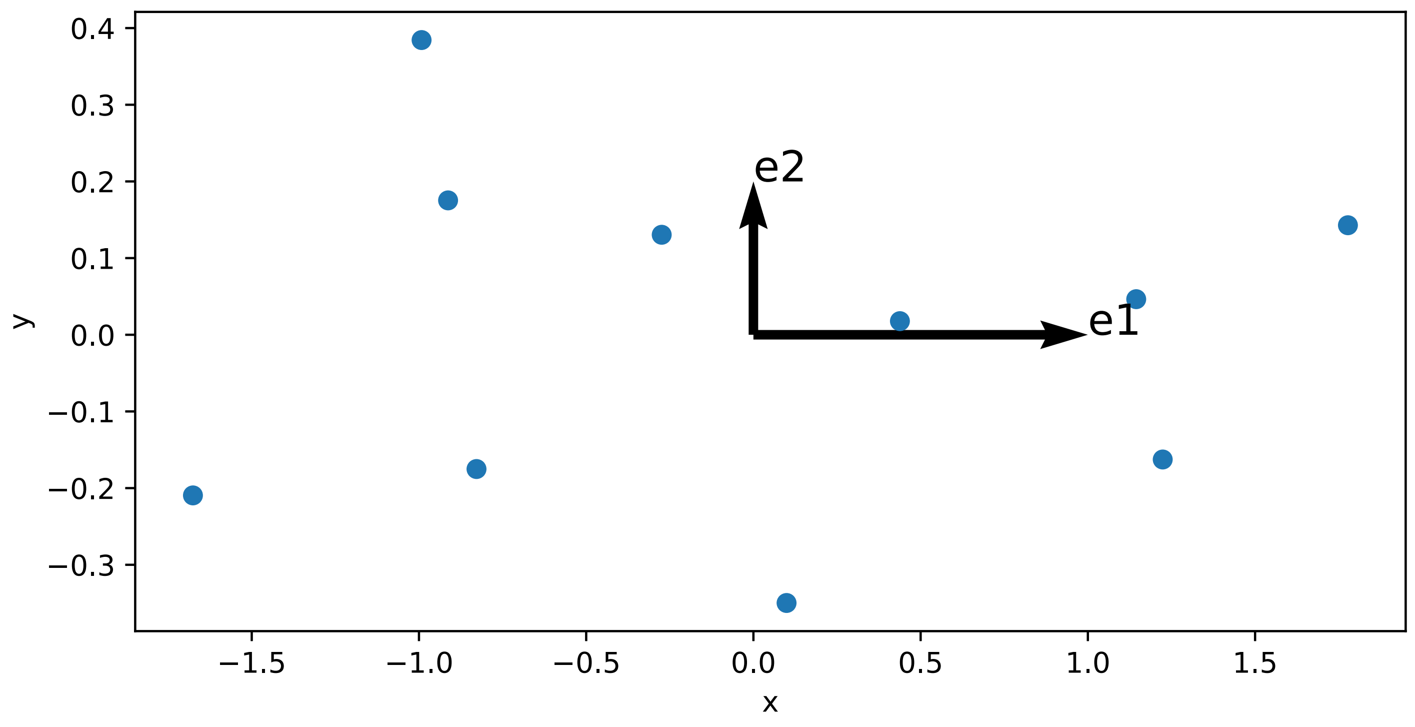


2.

In [118…
```
A = D@d
A
```

Out[118…
```
array([[-0.82797019, -0.17511531],
       [ 1.77758033,  0.14285723],
       [-0.99219749,  0.38437499],
       [-0.27421042,  0.13041721],
       [-1.67580142, -0.20949846],
       [-0.9129491 ,  0.17528244],
       [ 0.09910944, -0.3498247 ],
       [ 1.14457216,  0.04641726],
       [ 0.43804614,  0.01776463],
       [ 1.22382056, -0.16267529]])
```

In [119…
```
fig,ax=plt.subplots(figsize = (8,4))
a = ax.quiver(0, 0, 1, 0, angles='xy', scale_units='xy', scale=1)
b = ax.quiver(0, 0, 0, 0.2, angles='xy', scale_units='xy', scale=1)
ax.text(1,0,"e1",size = 15)
ax.text(0,0.2,"e2",size = 15)
plt.scatter(A[:,0],A[:,1])
plt.xlabel("x")
plt.ylabel("y")
plt.show()
```



3.

In [120…
```
distance = np.max(A[:,0])-np.min(A[:,0])
distance
```

Out[120…
```
3.453381743924969
```

I think e1 is the optimal one-dimentional representation of the data because it can capture more information of the data. And the range of data is 3.453.

(b)

1.

In [121…
```
data_b = np.loadtxt('q1bdata.txt')
mean_b = np.mean(data_b,axis = 0)
D_b = data_b - mean_b
n = len(data_b)
cov_b = D_b@D_b.T
v,d = np.linalg.eig(cov_b)
idx = v.argsort()[::-1]
v = v[idx]
d = d[:,idx]
print("covariance: ",cov_b)
print("eigenvalues: ",v)
print("eigenvector: ",d)
```

```
covariance:  [[ 69.875 -18.875 -26.375 -24.625]
 [-18.875 121.375 -53.125 -49.375]
 [-26.375 -53.125  98.375 -18.875]
 [-24.625 -49.375 -18.875  92.875]]
eigenvalues:  [175.55118219 114.31705238  92.63176543   0.        ]
eigenvector:  [[-0.06628148  0.04124587 -0.86249959 -0.5       ]
 [-0.79038331 -0.06822502  0.34733208 -0.5       ]
 [ 0.47285044 -0.69123739  0.22046165 -0.5       ]
 [ 0.38381435  0.71821654  0.29470586 -0.5       ]]
```

There are three non-zero eigenvalues which means there are three effective eigenvectors.

2.

```
In [122…
u = D_b.T@d[:,0:3]
# normalize u
u = u/np.linalg.norm(u,axis=0)
Projection = D_b@u
print("projection: ",Projection)
```

```
projection:  [[ -0.8782013    0.44099733  -8.3011616 ]
 [-10.47224127  -0.72945617   3.34291139]
 [  6.26506632  -7.39065157   2.12184196]
 [  5.08537624   7.67911041   2.83640825]]
```

3.

```
In [123…
omega = mean_b.reshape(1,-1) + Projection@u.T
mse = np.mean((data_b-omega)**2,axis = 1)
print("MSE: ",mse)
```

```
MSE:  [6.31380655e-31 2.30252020e-30 3.01077587e-30 3.12689931e-30]
```

4.

```
In [124…
u2 = D_b.T@d[:,0:2]
# normalize u
u2 = u2/np.linalg.norm(u2,axis=0)
Projection2 = D_b@u2
print("projection2: ",Projection2)
omega2 = mean_b.reshape(1,-1) + Projection2@u2.T
mse2 = np.mean((data_b-omega2)**2,axis = 1)
print("MSE: ",mse2)
```

```
projection2:  [[ -0.8782013    0.44099733]
 [-10.47224127  -0.72945617]
 [  6.26506632  -7.39065157]
 [  5.08537624   7.67911041]]
MSE:  [3.62680441 0.58816087 0.2369586  0.4234322 ]
```

5.

```
In [125…
Y = np.array([1,3,0,3,-2,2,4,1,3,0,-2,0,1,1,-3,0,1,-2,-3])
```

```
In [126…
ED = np.linalg.norm(Y-omega,axis = 1)
```

```
In [127…
ED
```

```
Out[127…
array([12.92284798,  5.65685425, 16.79285562, 15.77973384])
```

The second sample is the most similar one to this new vector.

6.

```
In [128…
ED2 = np.linalg.norm(Y-data_b,axis = 1)
ED2
```

```
Out[128…
array([12.92284798,  5.65685425, 16.79285562, 15.77973384])
```

This result is the same as part(5). Although there are 19 dimensions in original space, there are only three informative eigenvectors, we can reduce dimensions to 3 without loss.

Q2: Eigenface with PCA

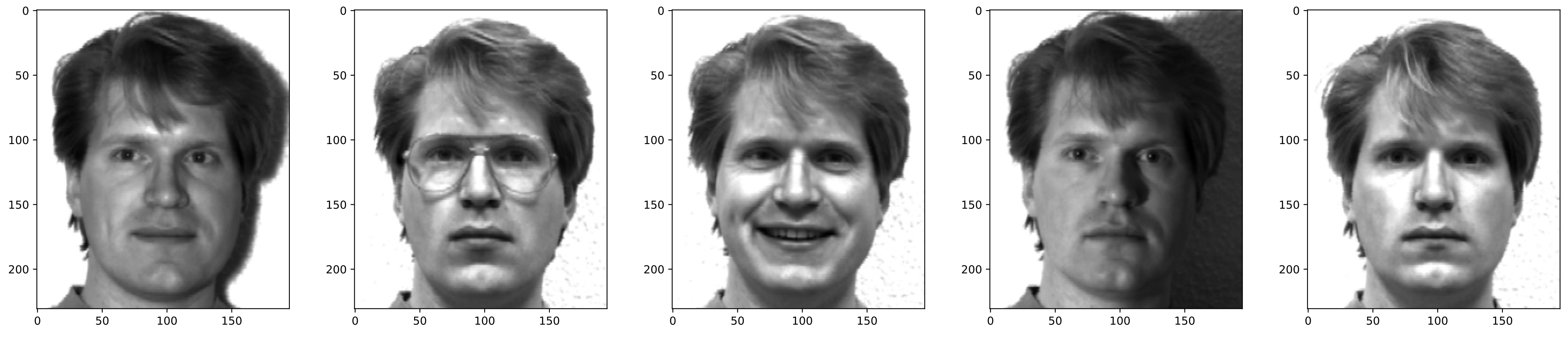
(a)

```
In [129…
import scipy.io
from PIL import Image
mat = scipy.io.loadmat("data-1.mat")
print(mat.keys())
```

```
dict_keys(['__header__', '__version__', '__globals__', 'Y', 'X', 'testimages', 'trainimages'])
```

```
In [130…
X = mat["X"]
```

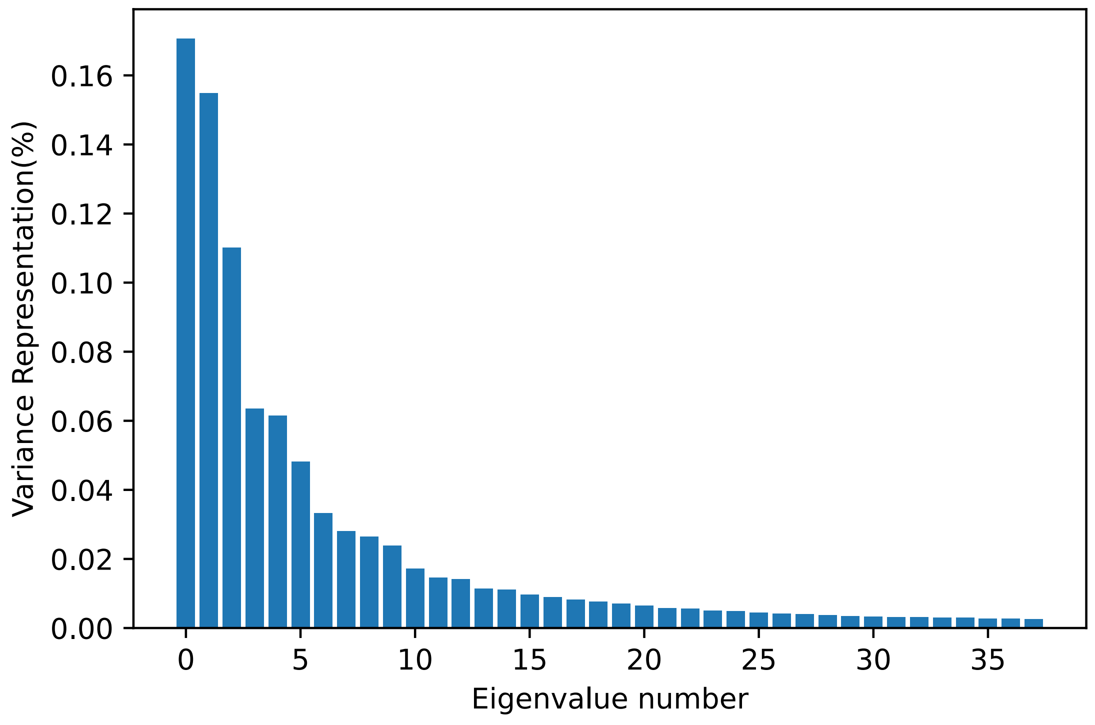
In [131…
```
fig,axs = plt.subplots(1,5,figsize = (25,5))
for i in range(5):
    face = np.transpose(X[:,i].reshape(195,231))
    axs[i].imshow(face,cmap = "gray")
```



(b)

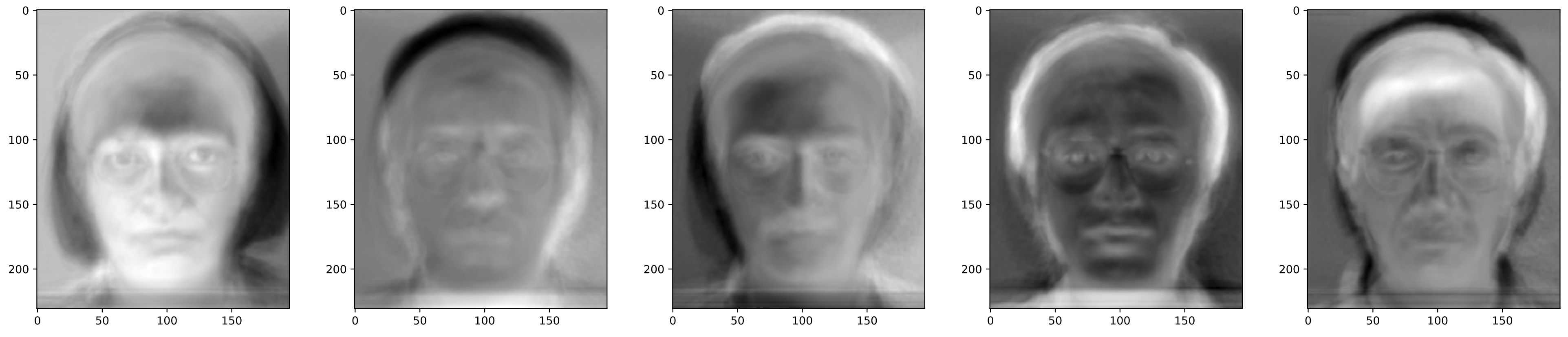
In [132…
```
from sklearn.decomposition import PCA
mean = np.mean(X,axis = 1)
D = X-mean.reshape(-1,1)
pca = PCA(n_components=0.9)
pca.fit(X.T)
print(pca.explained_variance_ratio_)
m = np.arange(0,pca.n_components_,1)
plt.bar(m,pca.explained_variance_ratio_)
plt.xlabel("Eigenvalue number")
plt.ylabel("Variance Representation(%)")
plt.show()
```

```
[0.17061649 0.15477326 0.11013432 0.06354261 0.06141484 0.0481875
 0.03331124 0.02809504 0.02639302 0.02379812 0.01709908 0.01454142
 0.01411703 0.01143241 0.01108751 0.00968342 0.00885608 0.00816275
 0.00755732 0.00697576 0.00642891 0.00572675 0.00558187 0.00507357
 0.00489485 0.00441013 0.00412782 0.00402231 0.00376886 0.00347413
 0.0032588  0.00317372 0.00307058 0.00302325 0.00292907 0.00265777
 0.00262715 0.00258159]
```



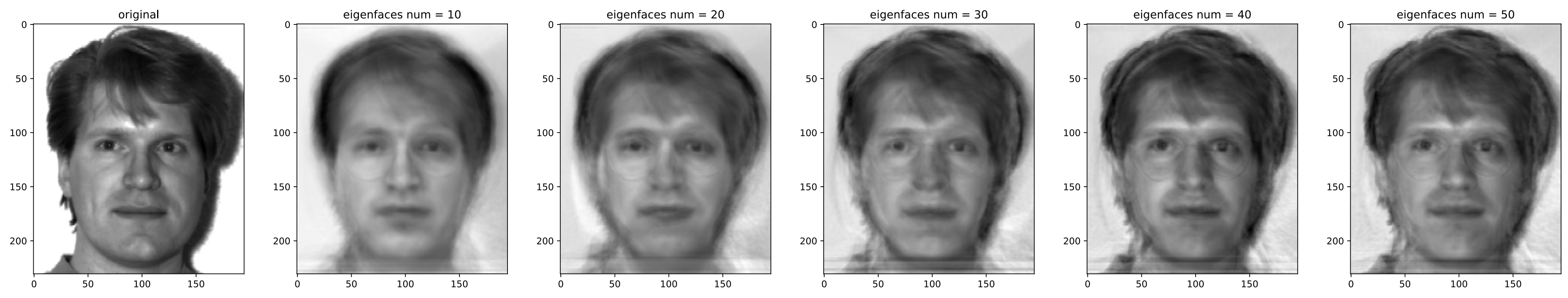
In [133…
```
A =  pca.transform(X.T)
u = D@A
```

In [134…
```
fig,axs = plt.subplots(1,5,figsize = (25,5))
for i in range(5):
    face = np.transpose(u[:,i].reshape(195,231))
    axs[i].imshow(face,cmap = "gray")
```



(c)

```
In [135…
fig,axs = plt.subplots(1,6,figsize = (30,5))
face = np.transpose(X[:,0].reshape(195,231))
axs[0].imshow(face,cmap = "gray")
axs[0].set_title("original")
eigen_num = [10,20,30,40,50]
for i in eigen_num:
    pca = PCA(n_components = i)
    pca.fit(X.T)
    A =  pca.transform(X.T)
    approx = pca.inverse_transform(A)
    face = np.transpose(approx.T[:,0].reshape(195,231))
    axs[eigen_num.index(i)+1].imshow(face,cmap = "gray")
    axs[eigen_num.index(i)+1].set_title(f"eigenfaces num = {i}")
```



(d)

```
In [136…
from sklearn.neighbors import KNeighborsClassifier
train = mat['trainimages'].ravel()-1
test = mat['testimages'].ravel()-1
Y = mat['Y'].ravel()
```

```
In [137…
pca = PCA(n_components = 30)
A1 = pca.fit_transform(X[:, train].T)
train_X = pca.inverse_transform(A1)
A2 = pca.fit_transform(X[:, test].T)
test_X = pca.inverse_transform(A2)
```

```
In [138…
score_30 = KNeighborsClassifier(n_neighbors=1).fit(train_X, Y[train]).score(test_X, Y[test])
score_original = KNeighborsClassifier(n_neighbors=1).fit(X[:,train].T, Y[train]).score(X[:,test].T, Y[test])
print("accuracy of reduced dimension space: ",score_30)
print("accuracy of original dimension space: ",score_original)
```

```
accuracy of reduced dimension space:  0.8
accuracy of original dimension space:  0.8333333333333334
```

Yes, this is what I expected. We reduce dimensions from 45045 to 30, but the accuracy only decreases 3.33%, which means these 30 dimensions capture most information of the data.