Homework 2

Solutions

1. (Correlation coefficient) If the correlation coefficient is one, then the covariance equals the product of the standard deviations $\text{Cov}(\tilde{x}[1], \tilde{x}[2]) = \sigma_{\tilde{x}[1]}\sigma_{\tilde{x}[2]}$. The covariance matrix equals

$$\Sigma_{\tilde{x}} = \begin{bmatrix} \sigma^2_{\tilde{x}[1]} & \sigma_{\tilde{x}[1]}\sigma_{\tilde{x}[2]} \\ \sigma_{\tilde{x}[1]}\sigma_{\tilde{x}[2]} & \sigma^2_{\tilde{x}[2]} \end{bmatrix} \tag{1}$$

$$= \begin{bmatrix} \sigma_{\tilde{x}[1]} \\ \sigma_{\tilde{x}[2]} \end{bmatrix} \begin{bmatrix} \sigma_{\tilde{x}[1]} & \sigma_{\tilde{x}[2]} \end{bmatrix}. \tag{2}$$

The matrix is low rank. The first eigenvalue equals $\sqrt{\sigma^2_{\tilde{x}[1]} + \sigma^2_{\tilde{x}[2]}}$ and the second equals zero. The variance of the second principal component therefore equals zero. Intuitively, both components are linearly dependent, so the data lie along a line. The variance along the line is the variance of the first principal component. The variance orthogonal to the line, which is the variance of the second principal component, is zero because all the points are on the line.

2. (Not centering) We have

$$\text{E}(\tilde{x}\tilde{x}^T) = \text{E}\left[(c(\tilde{x}) + \mu)(c(\tilde{x}) + \mu)^T\right] \tag{3}$$

$$= \text{E}\left[c(\tilde{x})c(\tilde{x})^T\right] + \text{E}\left[c(\tilde{x})\mu^T\right] + \text{E}\left[\mu c(\tilde{x})^T\right] + \text{E}(\mu\mu^T) \tag{4}$$

$$= \Sigma_{\tilde{x}} + \mu\mu^T \tag{5}$$

$$= I + \mu\mu^T. \tag{6}$$

Let $u_1 := \mu / ||\mu||_2$, and let $u_2, \ldots, u_d$ orthonormal vectors orthogonal to u_1, we have

$$\begin{bmatrix} u_1 & u_2 & \cdots & u_d \end{bmatrix} \begin{bmatrix} u_1 & u_2 & \cdots & u_d \end{bmatrix}^T = I, \tag{7}$$

because it is an orthonormal set, so the matrix is orthogonal. This implies

$$\text{E}(\tilde{x}\tilde{x}^T) = I + \mu\mu^T \tag{8}$$

$$= \begin{bmatrix} u_1 & u_2 & \cdots & u_d \end{bmatrix} \begin{bmatrix} ||\mu||_2^2 + 1 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 \\ \cdots & \cdots & \cdots & \cdots \\ 0 & 0 & \cdots & 1 \end{bmatrix} \begin{bmatrix} u_1 & u_2 & \cdots & u_d \end{bmatrix}^T \tag{9}$$

so the first eigenvalue equals $||\mu||_2^2 + 1$ and the corresponding eigenvector is collinear with the mean.

3. (Financial data)

   (a) The first princpal direction is shown below.

AAPL	AMZN	MSFT	GOOG	XOM	APC
0.0546	0.8679	0.0367	0.4827	0.0079	0.0098
CVX	C	GS	JPM	AET	JNJ
0.0139	0.0124	0.0534	0.0207	0.0086	0.0133
DGX	SPY	XLF	SSO	SDS	USO
0.0120	0.0544	0.0050	0.0442	-0.0169	0.0015

The second principal direction is shown below.

AAPL	AMZN	MSFT	GOOG	XOM	APC
-0.0419	0.4950	-0.0268	-0.8511	-0.0280	-0.0092
CVX	C	GS	JPM	AET	JNJ
-0.0288	-0.0259	-0.1150	-0.0371	-0.0280	-0.0411
DGX	SPY	XLF	SSO	SDS	USO
-0.0113	-0.0695	-0.0091	-0.0567	0.0214	-0.0004

As can be seen above, in both vectors the largest two are amzn and goog. This is due to the fact that they have the largest variance. To see this, we show the sample standard deviations below.

AAPL	AMZN	MSFT	GOOG	XOM	APC
2.1289	19.4385	1.0525	13.2065	0.7770	1.0380
CVX	C	GS	JPM	AET	JNJ
1.2916	0.8195	3.0984	1.1657	1.7488	1.2298
DGX	SPY	XLF	SSO	SDS	USO
1.1167	1.7223	0.2518	1.3955	0.5511	0.1823

This is somewhat expected since amzn and goog have the largest share prices in the group. To see this, we show the prices from the last day in the data set below.

AAPL	AMZN	MSFT	GOOG	XOM	APC
219.2650	1944.3000	113.0815	1186.8700	83.9688	63.4848
CVX	C	GS	JPM	AET	JNJ
118.2641	73.7602	236.4426	116.8560	204.6858	141.0816
DGX	SPY	XLF	SSO	SDS	USO
106.7525	290.5603	28.6738	128.6087	32.5735	14.8000

(b) Below we give the first principal direction.

AAPL	AMZN	MSFT	GOOG	XOM	APC
0.1952	0.1913	0.2539	0.2512	0.2020	0.1510
CVX	C	GS	JPM	AET	JNJ
0.2034	0.2533	0.2631	0.2734	0.1117	0.1792
DGX	SPY	XLF	SSO	SDS	USO
0.1450	0.3275	0.2953	0.3266	-0.3240	0.1136

Next we give the second principal direction.

AAPL	AMZN	MSFT	GOOG	XOM	APC
-0.1949	-0.2193	-0.2164	-0.1860	0.3803	0.4622
CVX	C	GS	JPM	AET	JNJ
0.4167	0.0270	0.0202	0.0086	-0.0830	-0.0700
DGX	SPY	XLF	SSO	SDS	USO
-0.1958	-0.0656	-0.0083	-0.0644	0.0745	0.4850

The first principal direction is an average of all the stocks (except sds). This represents the trend in the market (as a whole). The reason that sds has a negative coefficient is that it is designed to move in the opposite direction of the market. The second principal component appears to group financial and oil stocks together, and computes the difference in their returns against the technology and health care stocks. Note that the PCA algorithm did not know about the meanings of the stocks, so these relationships were extracted from the data.

(c) The portfolio standard deviation is computed by

$$\sqrt{\alpha^T \Sigma \alpha} \approx 4309.94$$

(d) Since $\tilde{y}$ is normally distributed with mean 879.782454 and standard deviation 4309.94 we obtain

$$\Pr(\tilde{y} \leq -1000) = 0.3764$$

This isn't as startling as it may appear, since the value of our portfolio (as of the last day in the dataset) is about 856755 dollars.

Q4 Solutions

February 27, 2020

[1]:
```
import numpy as np
from sklearn.datasets import fetch_olivetti_faces
import plot_tools
```

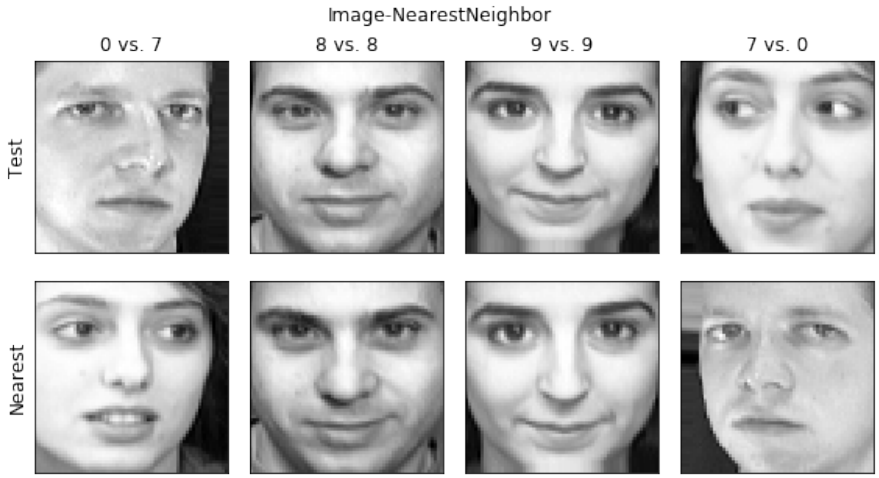
[3]:
```
def compute_nearest_neighbors(train, testImage, V_T = None):

    train_matrix = train[0];
    for i in range(1, len(train)):
        train_matrix = np.vstack([train_matrix, train[i]]);
    train_matrix = train_matrix.astype(float);
    if V_T is not None:
        train_matrix = np.dot(train_matrix, np.transpose(V_T));
        testImage = np.dot(V_T, testImage);

    distance_array = [np.linalg.norm(np.array(train_matrix[i,:]) - np.
 ↪array(testImage)) for i in range(train_matrix.shape[0])];

    return int(np.argmin(np.array(distance_array)))
```

0.1 4(a)

[4]:
```
test_idx = [1,  87,  94, 78]

data = fetch_olivetti_faces()
targets = data.target
data = data.images.reshape((len(data.images), -1))

train_idx = np.array(list(set(list(range(data.shape[0]))) - set(test_idx) ) )

train_set = data[train_idx ]
y_train = targets[train_idx]
test_set = data[np.array(test_idx)]
y_test = targets[ np.array(test_idx)]


imgs = []
estLabels = []
```

```
for i in range(test_set.shape[0]):
    testImage = test_set[i, :]
    nnIdx = compute_nearest_neighbors(train_set, testImage)
    imgs.extend( [testImage, train_set[nnIdx,:]] )
    estLabels.append(y_train[nnIdx])


row_titles = ['Test','Nearest']
col_titles = ['%d vs. %d'%(i,j) for i,j in zip(y_test, estLabels)]
plot_tools.plot_image_grid(imgs,
                "Image-NearestNeighbor",
                    (64,64),␣
 ↪len(test_set),2,True,row_titles=row_titles,col_titles=col_titles)
```

```
downloading Olivetti faces from https://ndownloader.figshare.com/files/5976027
to /Users/sreyas/scikit_learn_data
```



0.2 4(b)

[8]:
```
import matplotlib.pyplot as plt
```

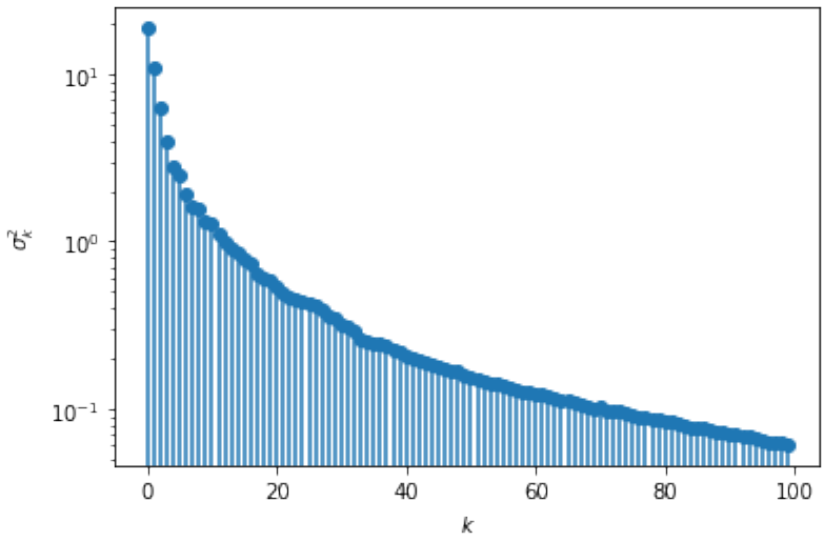
[42]:
```
cov_mat = np.cov(train_set, rowvar = False)
```

[43]:
```
eigvals, eigvecs = np.linalg.eig(cov_mat)
```

[44]:
```
eigvecs = np.real(eigvecs)
```

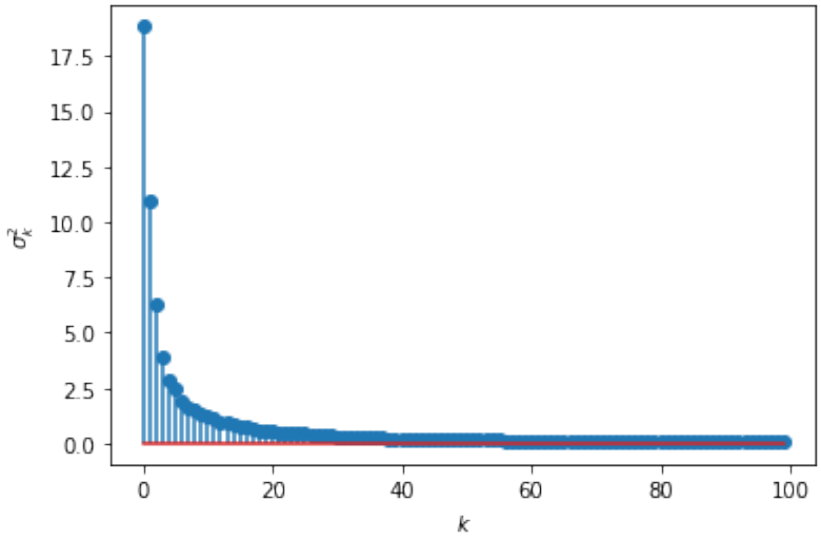
[52]:
```
plt.stem(eigvals[0: 100], use_line_collection = True)
# plt.xlim([0, 100])
plt.yscale('log')
plt.xlabel(r'$k$')
plt.ylabel(r'$\sigma_k^2$')
plt.savefig('variance_plot_log.pdf', bbox_inches='tight')
```

```
/anaconda3/envs/denoising/lib/python3.7/site-packages/numpy/core/numeric.py:538:
ComplexWarning: Casting complex values to real discards the imaginary part
  return array(a, dtype, copy=False, order=order)
```



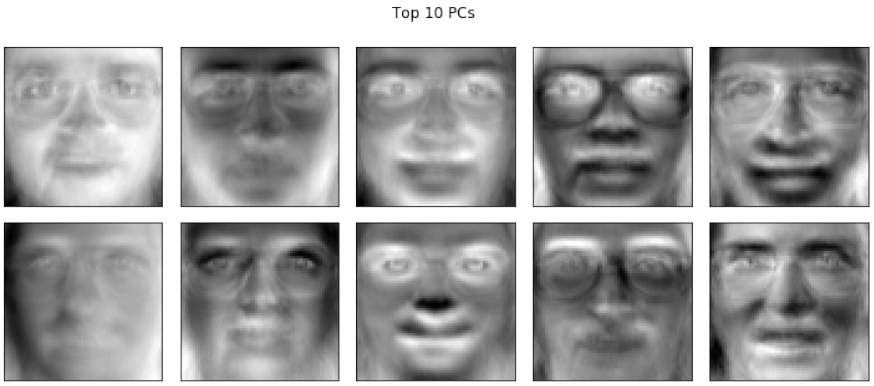
[54]:
```
plt.stem(eigvals[0: 100], use_line_collection = True)
plt.xlabel(r'$k$')
plt.ylabel(r'$\sigma_k^2$')
plt.savefig('variance_plot.pdf', bbox_inches='tight')
```

```
/anaconda3/envs/denoising/lib/python3.7/site-packages/numpy/core/numeric.py:538:
ComplexWarning: Casting complex values to real discards the imaginary part
  return array(a, dtype, copy=False, order=order)
```



0.2.1 4(c)

[46]:
```
plot_tools.plot_image_grid(eigvecs[:, :10].T,
                  "Top 10 PCs",
                      (64,64), 5,2,True)
```



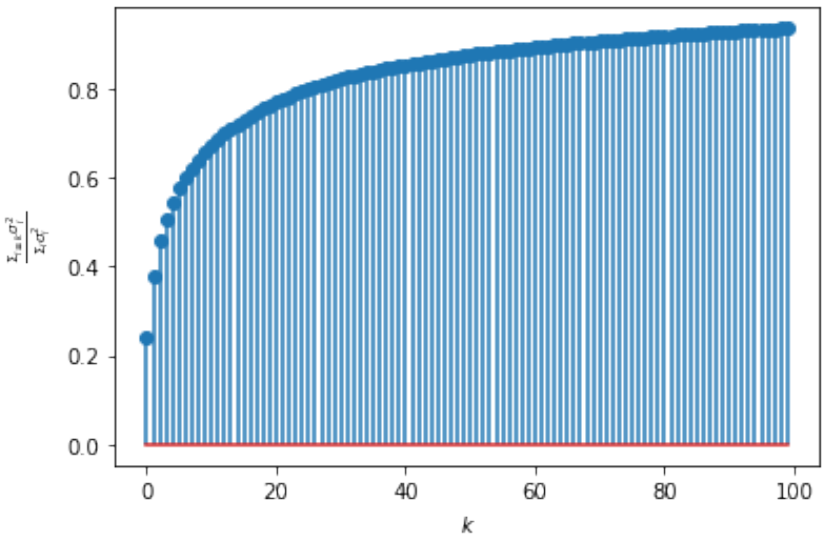
0.2.2 4(d)

Let's look at percentage of variance explained

[50]:
```
explained_variance = np.cumsum(eigvals[0: 100])/np.sum(eigvals)
```

[51]:
```
plt.stem(explained_variance, use_line_collection = True)

plt.xlabel(r'$k$')
plt.ylabel(r'$\frac{ \Sigma_{i \leq k}\sigma_i^2 } { \Sigma_{i}\sigma_i^2 }$')
plt.savefig('variance_explained_plot.pdf', bbox_inches='tight')
```

```
/anaconda3/envs/denoising/lib/python3.7/site-packages/numpy/core/numeric.py:538:
ComplexWarning: Casting complex values to real discards the imaginary part
  return array(a, dtype, copy=False, order=order)
```



Let's put a threshold at 75%

[55]:
```
thresh = 0.75
```

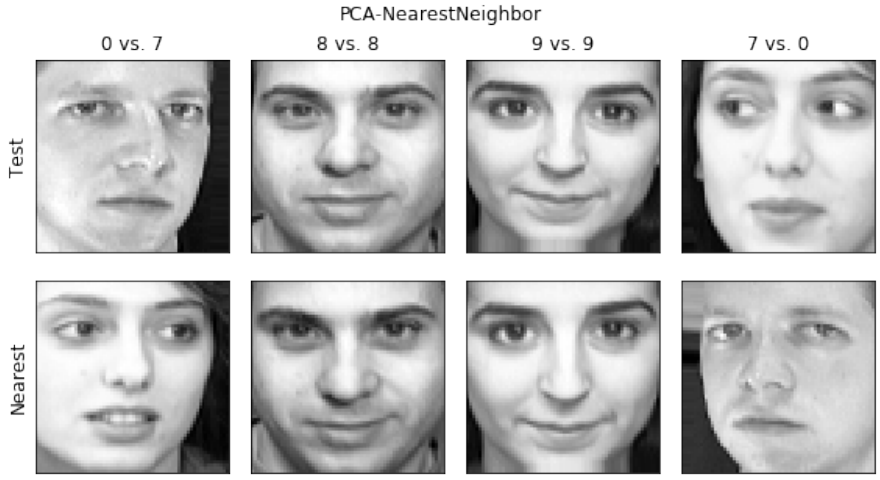
[56]:
```
k_thresh = np.sum( explained_variance < thresh )
```

[57]:
```
k_thresh
```

[57]:
```
18
```

This is not the only way to pick. One could also look at the plot of singular values (not in log scale) and see that there's a dramatic drop about k = 10 and pick this. Any k is ok as long as the student explains how they picked it.

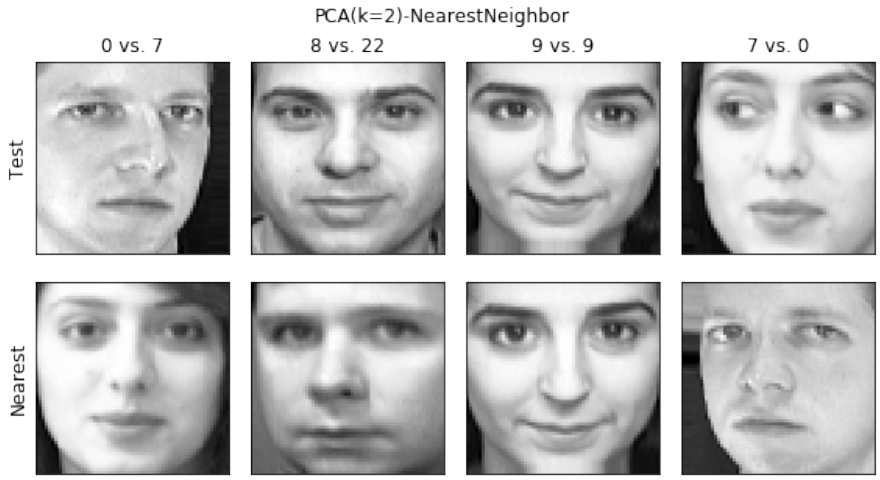
[59]:
```
imgs = []
estLabels = []
for i in range(test_set.shape[0]):
    testImage = test_set[i, :]
    nnIdx = compute_nearest_neighbors(train_set, testImage, eigvecs[:, :
 ↪k_thresh].T)
    imgs.extend( [testImage, train_set[nnIdx,:]] )
    estLabels.append(y_train[nnIdx])


row_titles = ['Test','Nearest']
col_titles = ['%d vs. %d'%(i,j) for i,j in zip(y_test, estLabels)]
plot_tools.plot_image_grid(imgs,
                "PCA-NearestNeighbor",
                    (64,64),␣
 ↪len(test_set),2,True,row_titles=row_titles,col_titles=col_titles)
```



We don't see any change in results here, but the result could with k. For example for $k = 2$:

[61]:
```
imgs = []
estLabels = []
for i in range(test_set.shape[0]):
    testImage = test_set[i, :]
    nnIdx = compute_nearest_neighbors(train_set, testImage, eigvecs[:, :2].T)
    imgs.extend( [testImage, train_set[nnIdx,:]] )
    estLabels.append(y_train[nnIdx])
```

```
row_titles = ['Test','Nearest']
col_titles = ['%d vs. %d'%(i,j) for i,j in zip(y_test, estLabels)]
plot_tools.plot_image_grid(imgs,
                "PCA(k=2)-NearestNeighbor",
                    (64,64),␣
↪len(test_set),2,True,row_titles=row_titles,col_titles=col_titles)
```



0.2.3 4(e)

In this particular experiment, the results does not become better from an accuracy point of view. However, you achieve a massive reduction in computation - now you are computing distance in $\mathcal{R}^{18}$ instead of $\mathcal{R}^{4096}$.

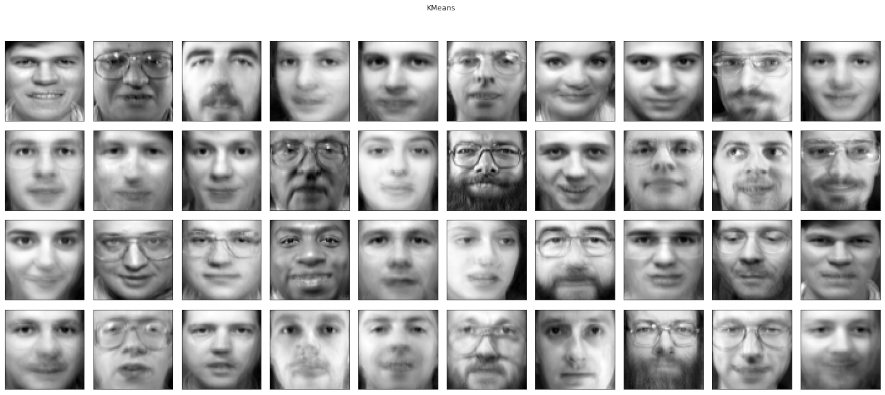
In general, projecting to a small number of top principal components might help you concentrate on the signal and ignore unwanted noise fluctations. For example, consider $x_1 = x + \eta_1$ and $x_2 = x + \eta_2$ where η_1 and η_2 are two noise realizations. The distance between x_1 and x_2 is influenced by noise. However, if the both x_1 and x_2 are projected to top k principal directions (assuming that the underlying signal is low dimensional), then the influence of noise will be much less.

0.2.4 4(f)

[62]:
```
from sklearn.cluster import KMeans
```

[63]:
```
kmeans = KMeans(n_clusters=40)
kmeans.fit(data)
```

[64]:
```
plot_tools.plot_image_grid(kmeans.cluster_centers_,
                 "KMeans",
                     (64,64), 10, 4 ,True)
```



[ ]: