MATH-GA 2840 HW#1

Yifei(Fahy) Gao yg1753

1. (Centering) To analyze what happens if we apply PCA without centering, let $\tilde{x}$ be a d -dimensional vector with mean $\mu \in \mathbb{R}^d$ and covariance matrix $\Sigma_{\tilde{x}}$ equal to the identity matrix. If we compute the eigendecomposition of the matrix $\mathrm{E}\left(\tilde{x}\tilde{x}^T\right)$ what is the value of the largest eigenvalue? What is the direction of the corresponding eigenvector?

Since from the notes:

$$\Sigma\tilde{x} = \text{identity matrix}$$

Then

$$var(\tilde{x}[i]) = 1, \text{and } covar(\tilde{x}[i], \tilde{x}[j]) = 0 \text{ from regular PCA}$$

Since PCA does subtract mean for centering, so for $\mathrm{E}\left(\tilde{x}\tilde{x}^T\right)$, we have to get the mean back: $\tilde{x} = c(\tilde{x}) + \mathrm{E}(\tilde{x}) = c(\tilde{x}) + \mu$

Since $\mathrm{E}\left(c(\tilde{x})c(\tilde{x})^T\right) = \mathrm{E}\left((\tilde{x}-\mu)(\tilde{x}-\mu)^T\right) = \mathrm{E}\left(\tilde{x}\tilde{x}^T\right) - \mathrm{E}\left(\tilde{x}\mu^T\right) - \mathrm{E}\left(\mu\tilde{x}^T\right) + \mathrm{E}\left(\mu\mu^T\right) = \mathrm{E}\left(\tilde{x}\tilde{x}^T\right) - 2\mathrm{E}\left(\tilde{x}\mu^T\right) + \mathrm{E}\left(\tilde{u}\mu^T\right) = \mathrm{E}\left(\tilde{x}\tilde{x}^T\right) - \mu\mu^T$

Then, $\mathrm{E}\left(\tilde{x}\tilde{x}^T\right) = \mathrm{E}\left(c(\tilde{x})c(\tilde{x})^T\right) + \mu\mu^T = 1 + \mu\mu^T$

In other words,

$$\mathrm{E}\left(\tilde{x}\tilde{x}^T\right) = \begin{bmatrix} \sigma_1^2+\mu_1^2 & covar(\tilde{x}[1],\tilde{x}[2])+\mu_1\mu_2 & \cdots & covar(\tilde{x}[1],\tilde{x}[d])+\mu_1\mu_d \\ covar(\tilde{x}[1],\tilde{x}[2])+\mu_1\mu_2 & \sigma_2^2+\mu_2^2 & \cdots & covar(\tilde{x}[2],\tilde{x}[d])+\mu_2\mu_d \\ \vdots & \vdots & \ddots & \vdots \\ covar(\tilde{x}[1],\tilde{x}[d])+\mu_1\mu_d & covar(\tilde{x}[1],\tilde{x}[2])+\mu_1\mu_2 & \cdots & \sigma_d^2+\mu_d^2 \end{bmatrix} = \begin{bmatrix} 1+\mu_1^2 & 0+\mu_1\mu_2 & \cdots & 0+\mu_1\mu_d \\ 0+\mu_1\mu_2 & 1+\mu_2^2 & \cdots & 0+\mu_2\mu_d \\ \vdots & \vdots & \ddots & \vdots \\ 0+\mu_1\mu_d & 0+\mu_1\mu_2 & \cdots & 1+\mu_d^2 \end{bmatrix}$$

And let us do the eigendecomposition of $\mathrm{E}\left(\tilde{x}\tilde{x}^T\right)$:

Note that we only need to find the eigenvalues of uu^T, since both identity matrix and uu^T are both symmetric matrices and the eigenvalue of identity matrix is 1 intuitively. Then:

let $A = uu^T$

$$Au = uu^Tu = u\left(u^Tu\right) = \left(u^Tu\right)u = \lambda u$$

Then

$$\lambda = u^Tu = \|\mu\|^2$$

and all others are equal to 0 since A is an outer-product of u on itself.

So by spectral theorem the final eigenvalue will be: $1+\|\mu_1\|^2$ and we can easily see the first μ_1 points in the direction in which the data has maximum variance which is the $span(\mu)$.

2. (Low-rank covariance matrix) Let $\tilde{x}$ be a d -dimensional vector, show that if its covariance matrix $\Sigma_{\tilde{x}}$ is rank $r < d$, then $\tilde{x}$ is restricted to a hyperplane of dimension r, in the sense that it belongs to the hyperplane with probability one. Recall that by Chebyshev's inequality, for any positive constant $c > 0$ and any random variable $\tilde{a}$ with bounded variance,

$$\mathrm{P}\left((\tilde{a} - \mathrm{E}(\tilde{a}))^2 \geq c\right) \leq \frac{\mathrm{Var}(\tilde{a})}{c}$$

Since the covariance matrix $\Sigma_{\tilde{x}}$ is rank r s.t. $r < d$, the dimension of $\tilde{x}$, then intuitively the eigen-decomposition of this covariance matrix will be:

$$A = \begin{bmatrix} u_1 & u_2 & \cdots & u_d \end{bmatrix} \begin{bmatrix} \lambda_1 & 0 & \cdots & 0 \\ 0 & \lambda_2 & \cdots & 0 \\ & & \cdots & \lambda_r \\ 0 & 0 & \cdots & 0 \\ 0 & 0 & \cdots & 0 \\ 0 & 0 & \cdots & 0 \end{bmatrix} \begin{bmatrix} u_1 & u_2 & \cdots & u_d \end{bmatrix}^T$$

And since $Var(\tilde{x}) = v^T\tilde{x}$ for any vect v, and a hyperplane through origin can be expressed as $v^T\tilde{x} = 0$ where $c(\tilde{x}) = \tilde{x} - \mathrm{E}(\tilde{x})$for some vectors v and dimension d-1.

Then for all the dimention t that $r < t \leq d$,

$$Var(\tilde{x[i]}) = v^T\tilde{x[i]} = 0 \text{ for i} \in \text{t}$$

then $\tilde{x}$ has bounded variance and by satisfying the requirement of the Chebyshev's inequality, then for all c>0:

$$\mathrm{P}\left((\tilde{x} - \mathrm{E}(\tilde{x}))^2 \geq c\right) = \mathrm{P}\left((v^T\tilde{x} - v^T\mathrm{E}(\tilde{x})^2 \geq c\right) \leq \frac{Var(\tilde{x})}{c} = \frac{0}{c} = 0$$

Then

$$\mathrm{P}((v^T\tilde{x} == v^T\mathrm{E}(\tilde{x})^2) = 1$$

Therefore, there are d-r numbers of v, s.t. v are linear independent and the hyperplane characterized as $v^Tc(\tilde{x})$ have r elements and:

$$ker(v) = r$$

Then we can say that the feasible value of x is a hyperplane of dimention r and $\tilde{x}$ must be restricted to this hyperplane.

3. (Dimensionality reduction) Data containing a large number of features can be difficult to analyze and process. The goal of dimensionality-reduction techniques is to embed the data points in a low-dimensional space where they can be described with a small number of variables. Here we study linear dimensionality reduction, where the lower-dimensional representation is obtained by computing the inner products of each data point with a small number of basis vectors. We will show that PCA provides an optimal choice for the linear transformation.

(a) Our proof will be based in the following linear-algebraic result. Let $A \in \mathbb{R}^{d\times d}$ be a symmetric matrix, and $u_1, \ldots, u_k$ be its first k eigenvectors. For any set of k orthonormal vectors $v_1, \ldots, v_k$

$$\sum_{i=1}^{k} u_i^T A u_i \geq \sum_{i=1}^{k} v_i^T A v_i$$

Show that this follows from the spectral theorem using induction.

Let us first consider the case when k is equal to 1, then the left hand of the inequality will be $u_1^T A u_1$.

Since A is a symmetric matrix so all eigenvalues and eigenvectors are all real, then by spectral theorem:

$$A = \begin{bmatrix} u_1 & u_2 & \cdots & u_d \end{bmatrix} \begin{bmatrix} \lambda_1 & 0 & \cdots & 0 \\ 0 & \lambda_2 & \cdots & 0 \\ & & \cdots & \\ 0 & 0 & \cdots & \lambda_d \end{bmatrix} \begin{bmatrix} u_1 & u_2 & \cdots & u_d \end{bmatrix}^T$$

since by spectral theorem$\lambda_1 = \max x^T A x$ and $u_1 = \arg\max_{\|x\|_2=1} x^T A x$, then

$\lambda_1 x^T = \max x^T A$ and $\lambda_1 u^T = \max u^T A$

Then, $u_1^T A u_1 = \lambda_1 u_1^T u_1 = \lambda_1 \geq v_1^T A v_1$ where v is any set of k orthonormal vectors $v_1, \ldots, v_k$, since λ_1 is the $\max Var(x) = v^T x$ for any vector v.

Let us assume the $\sum_{i=1}^{k} u_i^T A u_i \geq \sum_{i=1}^{k} v_i^T A v_i$ is true,

then for k+1:

$$\sum_{i=1}^{k+1} u_i^T A u_i = \sum_{i=1}^{k} u_i^T A u_i + u_{k+1}^T A u_{k+1}$$

Except u_1, the largest variance will be λ_2 because

$$\lambda_k = \max_{\|x\|_2=1, x\perp u_1,\ldots,u_{k-1}} x^T A x, \quad 2 \leq k \leq d \tag{1}$$

by spectral theorem.

Then same for λ_{k+1} as in:

Since $\sum_{i=1}^{k} u_i^T A u_i = \sum_{i=1}^{k} \lambda_i u^T u = \sum_{i=1}^{k} \lambda_i$,

Then:

$$\sum_{i=1}^{k} u_i^T A u_i + u_{k+1}^T A u_{k+1} = \lambda_1 + \lambda_2 + \cdots + \lambda_k + \lambda_{k+1} \geq v_1^T A v_1 + \cdots + v_{k+1}^T A v_{k+1} = \sum_{i=1}^{k+1} v_i^T A v_i$$

(b) Let us consider an arbitrary dataset $X = \{x_1, \ldots, x_n\}$ of d -dimensional points. For a fixed dimension $k < d$, and any set of k orthonormal vectors $v_1, \ldots, v_k$ show that the sum of the sample variances of the k first principal components,

$$\sum_{i=1}^{k} \text{var}(\text{pc}[i]) \geq \sum_{i=1}^{k} \sigma_{Xv_i}^2$$

where $\sigma_{X_{v_i}}^2$ is the sample variance of $\{v_i^T x_1, \ldots, v_i^T x_n\}$. This establishes that projecting onto the principal directions is the linear transformation that preserves the most variance in the data.

Since from the notes (54-56):

$$\begin{aligned} \text{Var}(pc[i]) &= u_i^T \Sigma_{\tilde{x}} u_i \\ &= \lambda_i u_i^T u_i \\ &= \lambda_i \end{aligned}$$

Then $\sum_{i=1}^{k} \text{Var}(pc[i]) = \sum_{i=1}^{k} \lambda_i$ and we know from note (30) that $\sigma_{X_v}^2 = v^T \Sigma_X v$ as sample variance, then:

the inequality will be

$$\sum_{i=1}^{k} \lambda_i \geq \sum_{i=1}^{k} v_i^T \Sigma_X v_i$$

For k=1,

$$\lambda_1 \geq v^T X v = Var(v^T X) \text{ for any vector v in } R^d$$

Then for $\sum_{i=1}^{k} \lambda_1 \geq \sum_{i=1}^{k} v^T A v$ for any vector v in R^d is true,

By (1) in part a, then we know $\lambda_k + 1 \geq v_{k+1}^T X v_{k+1}$ then:

$$\sum_{i=1}^{k+1} \lambda_1 \geq \sum_{i=1}^{k+1} v^T A v$$

Therefore,

$$\sum_{i=1}^{k} \text{var}(\text{pc}[i]) = \sum_{i=1}^{k} \lambda_i \geq \sum_{i=1}^{k} v_i^T \Sigma_X v_i = \sum_{i=1}^{k} \sigma_{Xv_i}^2$$

4. (Nearest neighbors in lower dimensions) To illustrate a possible use of PCA-based dimensionality reduction, we consider the problem of face classification. The nearest-neighbor algorithm is a classical method to perform classification. When applied on d -dimensional points, it requires computing distances in d dimensions, which can. be very computationally expensive if d is large. Here you will use PCA to perform nearest neighbors in a lower-dimensional space. You will find the relevant code in the folder faces.

Create a new file in which to write the code for the remaining questions. Include all plots in your submitted homework.

(a)

Complete the compute_nearest_neighbors () function in nearest_neighbors.py that finds the image in the training data that is closest to a given test image. Include the generated images in your submitted homework. In this question, do not apply dimensionality reduction.

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

def compute_nearest_neighbors(train_matrix, testImage):
    dist=np.sqrt(((train_matrix-testImage)**2).sum(axis=1))
    idx_of_closest_point_in_train_matrix=np.argmin(dist)
    #print(np.argmin(dist))
            ## fill your code here. feel free to modify the signature of the function if required.
    return idx_of_closest_point_in_train_matrix
```

In [2]:
```
test_idx = [1,  87,  94, 78] #use this

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]):
    #print(i)
    testImage = test_set[i, :]
    nnIdx = compute_nearest_neighbors(train_set, testImage)
    imgs.extend( [testImage, train_set[nnIdx,:]] )
    estLabels.append(y_train[nnIdx])
```

In [3]:
```
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-4a",
                (64,64), len(test_set),2,True,row_titles=row_titles,col_titles=col_titles)
```

(b)

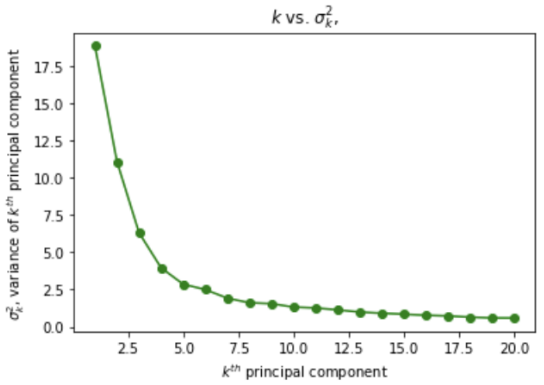
Generate a plot of k vs. σ_k^2, where σ_k^2 is the variance of the k th principal component of the data (e.g., σ_1^2 is the largest variance). You can limit the x axis to a reasonable number.

In [4]:
```
import matplotlib.pyplot as plt
```

In [5]:
```
#do the PCA
data = fetch_olivetti_faces()
data = data.images.reshape((len(data.images), -1))
cov=np.cov(data.T)

train_eigenval,train_eigenvec= np.linalg.eig(cov)
```

In [15]:
```
plt.plot(np.arange(1,21,1), train_eigenval[:20],"go-")
plt.title("$k$ vs. $\sigma_{k}^{2},$")
plt.xlabel("$k^{th}$ principal component")
plt.ylabel("$\sigma_{k}^{2}$, variance of $k^{th}$ principal component");
plt.savefig("k_vs_variacne_4b.png")
```

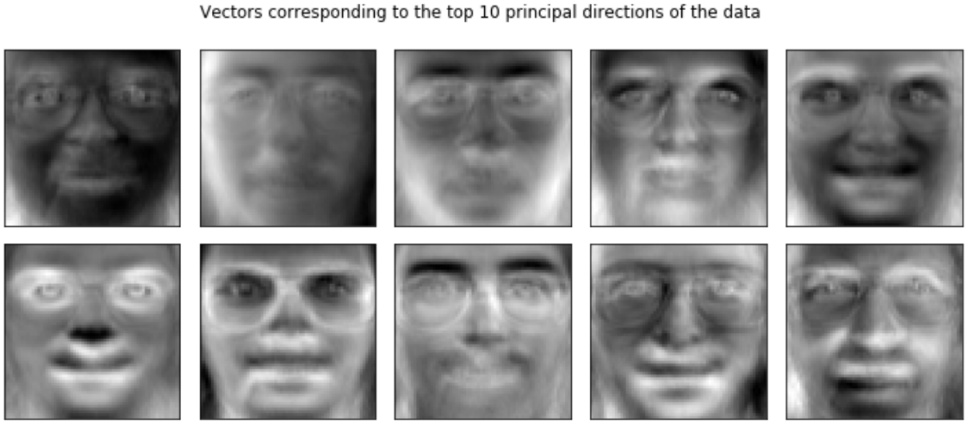
```
/Users/herculesgao/opt/anaconda3/lib/python3.7/site-packages/numpy/core/_asarray.py:85: ComplexWarning: Casting compl
ex values to real discards the imaginary part
  return array(a, dtype, copy=False, order=order)
```



(c)

Plot (using plot_image_grid() in plot_tools.py) the vectors corresponding to the top 10 principal directions of the data. Your principal direction vectors should be elements of $\mathbb{R}^{4096}$ (i.e., they should represent images).

In [7]:
```
plot_tools.plot_image_grid(train_eigenvec.real.T[:10], "Vectors corresponding to the top 10 principal directions of the
                           image_shape=(64,64),n_col=5, n_row=2, bycol=0, row_titles=None,col_titles=None)
```



d)

Use the variance of principal directions plot to determine a relatively small number k of principal components that explains the training data reasonably well. Project the training data and the test data onto the first k principal components, and run nearest neighbors for each test image in this lower dimensional space. Include your choice for k, and the plots of your nearest neighbor results in your submitted homework document. You should use the code from nearest_neighbors .py to generate your image plots.

In [8]:
```
from sklearn.decomposition import PCA
```

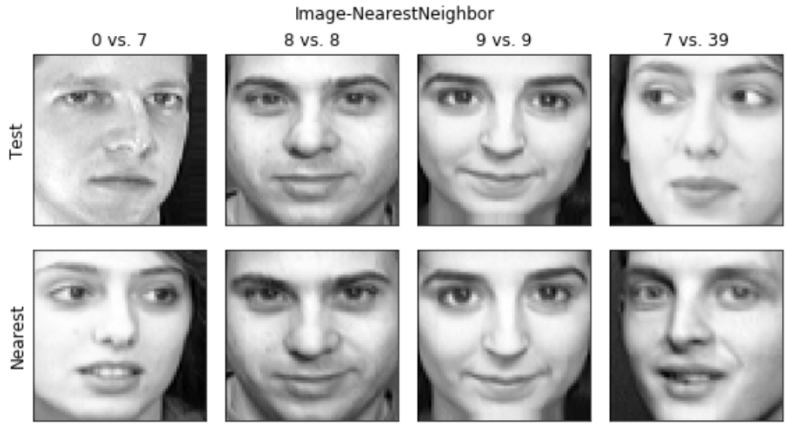
In [9]:
```
def project_k(main_set,k):
    pca=PCA(k)
    pca.fit_transform(data)
    top_n_princ=pca.components_[:k,:]
    main_set_proj=np.matmul(top_n_princ,main_set.T)
    return np.transpose(main_set_proj)
```

In [10]:
```
train_project=project_k(train_set,10)
test_project=project_k(test_set,10)

nnIdx_total =[]

for i in range(test_project.shape[0]):
    testImage= test_project[i,:]
    nnIdx = compute_nearest_neighbors(train_project, testImage)
    nnIdx_total.append(nnIdx)
nnIdx_total
```

Out[10]:
```
[77, 81, 93, 392]
```

In [11]:
```
imgs = []
estLabels = []
for i in range(test_set.shape[0]):
    testImage = test_set[i, :]

    nnIdx = nnIdx_total[i]

    imgs.extend( [testImage, train_set[nnIdx,:]] )
    estLabels.append(y_train[nnIdx])
```

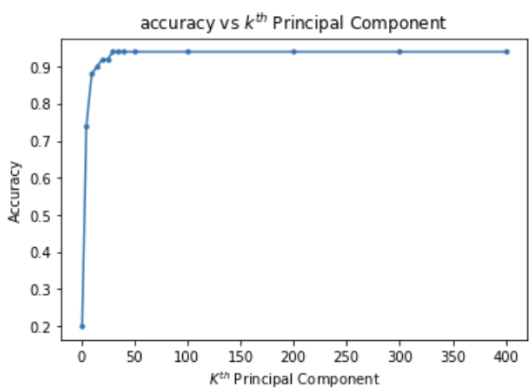
In [12]:
```
row_titles = ['Test','Nearest']
col_titles = ['%d vs. %d'%(i,j) for i,j in zip(y_test, estLabels)]
plot_tools.plot_image_grid(imgs,
                "Image-NearestNeighbor",
                (64,64), len(test_set),2,True,row_titles=row_titles,col_titles=col_titles)
```



e)

Let's explicitly study how accuracy changes with the number of principal components k used to project before computing nearest neighbhor. We will divide our dataset of 400 elements into 350 for training and 50 for test and compute the classification accuracy of these 50 with different values of k. The indices of 50 test elements and the minimum set of k values are specified as comments at the end of nearest_neighbors.py. Please feel free to include more k values. Generate a plot of accuracy vs k

In [13]:
```
test_idx = [207, 209, 396,  10,  15, 334, 101, 286, 255, 305,  37,  38,  97,
            331, 227, 347,  45, 105, 151,  65, 265, 217,  19, 238,  56, 378,
            3, 316, 246,  69, 179, 303, 250, 103, 337, 145, 183, 236,  71,
            354, 395, 281,  81, 350, 301, 381,  67, 297, 205, 358] #use this test indices
k_values_4e = [1, 5, 10, 15, 20, 25, 30, 35, 40, 50, 100, 200, 300, 400]  #use atleast these many k values. can add to


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)]
accuracy_total = []

for k in k_values_4e :

    train_project=project_k(train_set,k )
    test_project=project_k(test_set,k )


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


    accuracy = np.sum(np.equal(estLabels,y_test))/len(y_test)
    accuracy_total.append(accuracy)


plt.plot(k_values_4e,accuracy_total,'.-')
plt.title("accuracy vs $k^{th}$ Principal Component")
plt.xlabel("$K^{th}$ Principal Component")
plt.ylabel("Accuracy");
plt.savefig("accuracy_vs_kth_principal_component_4e.png")
```



f)

Repeat the analysis above, but now with a train and test dataset that is corrupted by Gaussian noise with standard deviation $150/255$. Sample code for corrupting the data is provided in nearest_neighbors.py. Do you observe a similar pattern as in the previous part? Can you explain what you are observing?

f)

Repeat the analysis above, but now with a train and test dataset that is corrupted by Gaussian noise with standard deviation $150/255$. Sample code for corrupting the data is provided in nearest_neighbors.py. Do you observe a similar pattern as in the previous part? Can you explain what you are observing?

In [14]:
```
train_set_no = train_set + np.random.randn(*train_set.shape)*(150/255)
test_set_no = test_set + np.random.randn(*test_set.shape)*(150/255)

accuracy_total = []

for k in k_values_4e:

    train_project=project_k(train_set_no ,k )
    test_project=project_k(test_set_no ,k )

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


    accuracy = np.sum(np.equal(estLabels,y_test))/len(y_test)
    accuracy_total.append(accuracy)


plt.plot(k_values_4e,accuracy_total,'.-')
plt.title("accuracy vs $k^{th}$ Principal Component")
plt.xlabel("$K^{th}$ Principal Component")
plt.ylabel("Accuracy");
plt.savefig("accuracy_with_noisy_vs_kth_principal_component_4f.png")
```