ECON 425 HW9 Solutions

March 5, 2024

Due Thu, March 14, 6pm in Bruinlearn

Problem 1 (PC1 is the direction of maximal variation) Prove that the first principal component w_1 of the centered data $X \in \mathbb{R}^{n\times k}$ is the direction of maximum variance in the sense that

$$w_1 = \arg\max_{w\in\mathbb{R}^k} \widehat{\mathrm{Var}}\left[w'x\right] \text{ s.t. } \|w\| = 1,$$

where $\|\cdot\|$ is the Euclidean norm and

$$\widehat{\mathrm{Var}}\left[w'x\right] = \frac{1}{n}\sum_{i=1}^{n}(w'x_i)^2.$$

Hint. Set the derivative of the Lagrangian to zero and use this equality to prove that the Lagrange multiplier is the largest eigenvalue of the sample covariance matrix $\hat{\Sigma} = \frac{1}{n}X'X$.

Solution

Rewrite $(w'x_i)^2 = w'x_i \cdot x_i'w$ and $\sum_{i=1}^{n}(w'x_i)^2 = w'X'Xw$. Also rewrite the constraint $\|w\|^2 = w'w = 1$ and set up the Lagrangian

$$\mathcal{L}(w,\lambda) = w'\frac{1}{n}X'Xw + \lambda\left(1 - w'w\right).$$

Taking the derivative w.r.t. w yields

$$\frac{\partial\mathcal{L}(w,\lambda)}{\partial w} = 2w'\frac{1}{n}X'X - 2\lambda w'.$$

Transposing and setting to zero yields

$$\frac{1}{n}X'Xw = \lambda w,$$

i.e. w is an eigenvector of the sample covariance matrix $\hat{\Sigma} = \frac{1}{n}X'X$ corresponding to some eigenvalue λ. Pre-multiplying this equality by w' shows that we can rewrite the objective function as

$$\widehat{\mathrm{Var}}\left[w'x\right] = w'\frac{1}{n}X'Xw = \lambda w'w = \lambda.$$

Hence, the objective function is maximized by picking λ to be the largest eigenvalue, and the optimal w is the corresponding eigenvector, which is exactly the first principal component.

Problem 2 (K-means clustering)

In this problem, you will perform K-means clustering manually, with $K = 2$, on a small example with $n = 6$ observations and $p = 2$ features. The data matrix is

$$X = \begin{pmatrix} 1 & 4 \\ 1 & 3 \\ 0 & 4 \\ 5 & 1 \\ 6 & 2 \\ 4 & 0 \end{pmatrix}$$

(a) Plot the observations.

(b) Randomly assign a cluster label to each observation. You can use the np.random.choice() function to do this. Report the cluster labels for each observation.

(c) Compute the centroid for each cluster.

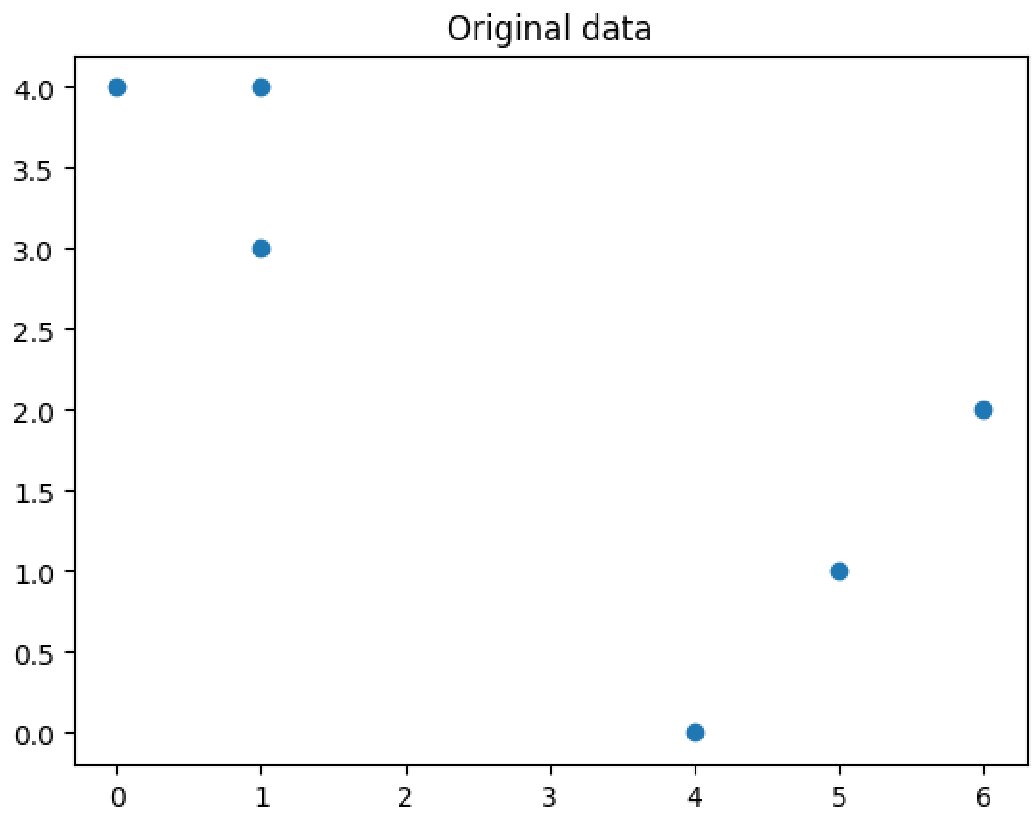
(d) Assign each observation to the centroid to which it is closest, in terms of Euclidean distance. Report the cluster labels for each observation.

(e) Repeat (c) and (d) until the answers obtained stop changing.

(f) In your plot from (a), color the observations according to the cluster labels obtained.

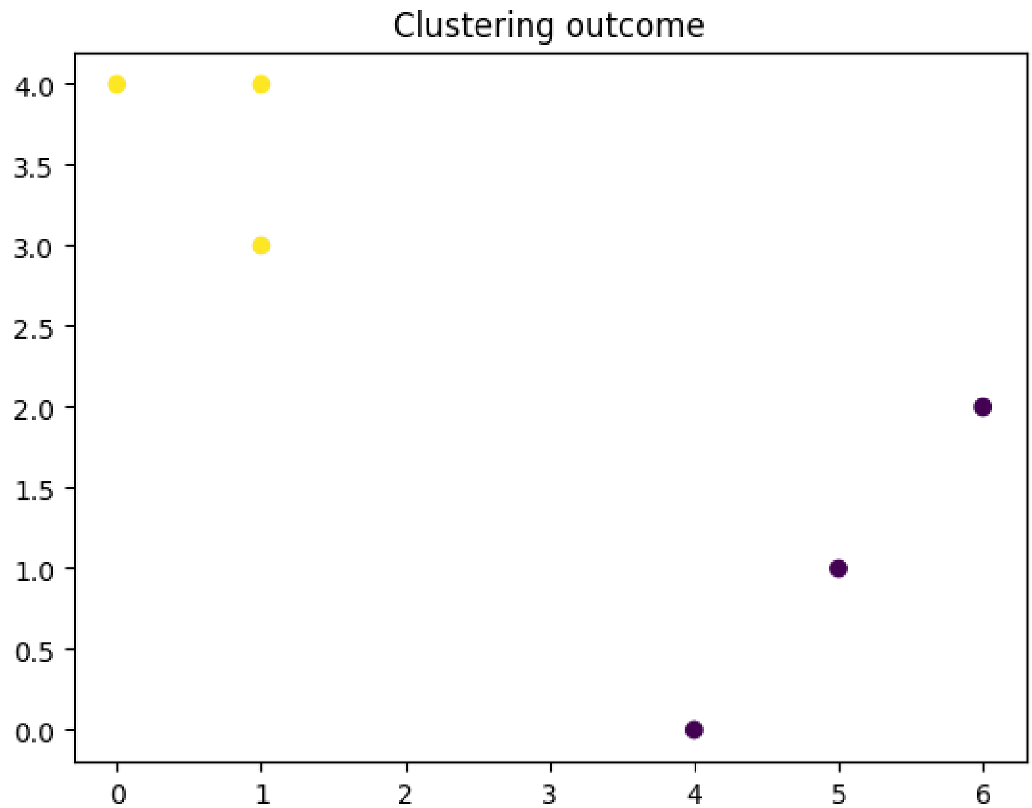
[45]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

X = pd.DataFrame([[1,4], [1,3], [0,4], [5,1], [6,2], [4,0]], columns=['x1',␣
 ↪'x2'])
n, k = X.shape

# (a)
plt.figure()
plt.scatter(X['x1'], X['x2'])
plt.title('Original data')
plt.show()

# (b)
cluster_labels = [0, 1]
np.random.seed(0)
clusters = np.random.choice(cluster_labels, size=n, replace=True)
print('Initial cluster assignment:', clusters)

# (c), (d), (e)
clusters_prev = np.empty((2,))
i = 0
while not np.array_equal(clusters_prev, clusters):
    i += 1
    print('=== Iteration', i, '===')
```

```
    centroids = pd.DataFrame([X[clusters==0].mean(axis=0), X[clusters==1].
↪mean(axis=0)])
    print('Centroids:', centroids)

    dist0 = np.linalg.norm(X - centroids.loc[0], axis=1)
    dist1 = np.linalg.norm(X - centroids.loc[1], axis=1)
    clusters_prev = clusters
    clusters = np.array(1*(dist1 <= dist0))
    print('Clusters:', clusters)

# (f)
plt.figure()
plt.scatter(X['x1'], X['x2'], c=clusters)
plt.title('Clustering outcome')
plt.show()
```




```
Initial cluster assignment: [0 1 1 0 1 1]
=== Iteration 1 ===
Centroids:       x1    x2
0   3.00   2.50
1   2.75   2.25
```

```
Clusters: [1 1 1 0 0 1]
=== Iteration 2 ===
Centroids:     x1    x2
0  5.5  1.50
1  1.5  2.75
Clusters: [1 1 1 0 0 0]
=== Iteration 3 ===
Centroids:          x1        x2
0  5.000000  1.000000
1  0.666667  3.666667
Clusters: [1 1 1 0 0 0]
```



Problem 3 (PCA on Olivetti faces)

Use the Olivetti faces dataset available through sklearn to do the following.

(a) Fetch and load the data with the fetch_olivetti_faces method from sklearn.datasets.

(b) Demean each face in the data set (no need to divide by standard deviation as every dimension is a number between a fixed range representing a pixel).

(c) Compute and display the first 9 *eigenfaces*. The k-th eigenface of a given face is an image based on the first k principal components only.

(d) Any given face in the data set can be represented as a linear combination of the eigenfaces. For any face in the data set, show how it progresses as we combine $1, 51, 101, \ldots$ eigenfaces, until the full image is recovered.

Solution

[46]:
```python
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import fetch_olivetti_faces
from sklearn.decomposition import PCA
```

0.1 Set plotting parameters and function

[47]:
```python
IMAGE_SHAPE = (64, 64)
def plot_gallery(title, images, n_row=3, n_col=3):
    plt.figure(figsize=(2. * n_col, 2.26 * n_row))
    plt.suptitle(title, size=16)
    for i, comp in enumerate(images):
        plt.subplot(n_row, n_col, i + 1)
        vmax = max(comp.max(), -comp.min())
        plt.imshow(comp.reshape(IMAGE_SHAPE), cmap=plt.cm.gray,
                   interpolation='nearest',
                   vmin=-vmax, vmax=vmax)
        plt.xticks(())
        plt.yticks(())
    plt.subplots_adjust(0.01, 0.05, 0.99, 0.93, 0.04, 0.)
```

[48]:
```python
N_SUBPLOTS = 9
```

0.2 Set up data

[49]:
```python
dataset = fetch_olivetti_faces(data_home = '/Users/franguri/Library/
↪CloudStorage/Dropbox/_TEACHING_/Econ 425 ML UCLA Winter 2024/
↪econ425_homework_codes')
faces = dataset['data']
n_faces, n_pixels = faces.shape
```

[50]:
```python
centered_faces = faces - faces.mean(axis=0)
```

0.3 Run PCA and extract eigenfaces

[51]:
```python
pca = PCA()
pca.fit(centered_faces)
eigenfaces = pca.components_
eigenweights = pca.transform(centered_faces)
```

[52]:
```
plt.gcf().clear()
plot_gallery("First 9 Eigenfaces", eigenfaces[:N_SUBPLOTS])
plt.show()
```

```
<Figure size 640x480 with 0 Axes>
```

0.4 Construct and plot progressing faces

[53]:
```
progressing_idx = list(range(1, n_faces, n_faces // (N_SUBPLOTS-1)))
progressing_idx.append(n_faces)
```

[54]:
```
face_idx = np.random.randint(n_faces)
progressing_faces = np.zeros((N_SUBPLOTS, n_pixels))
for i, n_eigenfaces in enumerate(progressing_idx):
    progressing_faces[i,:] = eigenweights[face_idx, :n_eigenfaces].reshape(1,␣
 ↪n_eigenfaces) \
                                                                              .
 ↪dot(eigenfaces[:n_eigenfaces, :])
```

[55]:
```
plt.gcf().clear()
plot_gallery("Progressing Faces", progressing_faces)
plt.show()
```

```
<Figure size 640x480 with 0 Axes>
```