Academy of Mathematics and Systems Science
Chinese Academy of Sciences
Lecturer: Xiayang Li
Personal Pages: https://xiayangli2301.github.io

Apr. 11, 2024
Lecture Notes

Machine Learning in Neuroscience
Lecture: Summary on Unsupervised Learning

1 Principal Components Analysis

1.1 Theory Review

- **Purpose**. The purpose is to obtain a lower-dimensional representation of the data. (Why?)
- **How to do**. To find a low-dimensional representation of a dataset that contains as much as possible of the variation.
- **The (first) principle component**. The first principal component of a set of features $X_1, X_2, \cdots, X_p$ is the normalized linear combination of the features

$$Z_1 = \phi_{11}X_1 + \phi_{21}X_2 + \cdots + \phi_{p1}X_p \tag{1}$$

has the largest variables.

Given an $n \times p$ data set $\boldsymbol{X}$, we assume that each of the variables in $\boldsymbol{X}$ has been centered to have mean zero (Along the column, and usually have standard deviation one). We can transform the problem of finding the first principal component vector into the following optimization problem

$$\underset{\phi_{1i},\ldots,\phi_{pi}}{\text{maximize}} \left\{ \frac{1}{n}\sum_{i=1}^{n}\left(\sum_{j=1}^{p}\phi_{ji}x_{ij}\right)^2 \right\} \text{ subject to } \sum_{j=1}^{p}\phi_{ji}^2 = 1. \tag{2}$$

Problem (2) can be solved via an **eigen decomposition**.

The second principal component is also the linear combination of $X_1, \cdots, X_p$ that has maximal variance out of all linear combinations that are uncorrelated wih Z_1

$$Z_2 = \phi_{12}X_1 + \phi_{22}X_2 + \cdots + \phi_{p2}X_p \tag{3}$$

Z_2 is uncorrelated with $Z_1 \Leftrightarrow$ The direction ϕ_2 to be orthogonal to the direction ϕ_1.

1.2 Practice

We illustrate the use of PCA on the **USArrests dataset** (https://www.kaggle.com/datasets/halimedogan/usarrests)

In the USArrests dataset, each column represents the following:

- Murder: The murder rate per 100,000 residents.
- Assault: The assault rate per 100,000 residents.
- UrbanPop: The percentage of urban population.
- Rape: The rape rate per 100,000 residents.

These columns provide crime rate data for the 50 states of the United States in 1973, as well as the percentage of urban population.

First, we need to import the relevant libraries.

Warm Reminder: Before starting any code related tasks today, please make sure to install the four libraries `matplotlib`, `numpy`, `scipy`, and `scikit-learn`, which are essential for plotting, numerical computations, and machine learning.

```
import scipy.io as scio
from scipy import stats
from sklearn.decomposition import PCA
import numpy as np
import matplotlib.pyplot as plt
```

Then we load the USArrests dataset. We can first observe the original form of the data. We can calculate some common statistical measures such as variance and mean.

```
data  = scio.loadmat('./USArrests_data.mat')
X = data['X']
print(np.mean(X,axis=0))
print(np.var(X,axis=0))
```

Then we use Principal Component Analysis to find all principal components, and output the explained variance and ratio. `pca.explained_variance_` means the amount of variance explained by each of the selected components.

```
standardX = stats.zscore(X, ddof = 1)
pca = PCA(n_components=4)
pca.fit(standardX)

print(pca.components_)

print(pca.explained_variance_)
```

```
print(pca.explained_variance_ratio_*100)
```

Question 1

What's the relationship among the coordinates of the principal component vectors? Check it.

We plot a Pareto chart to determine how many principal components should be retained to explain the majority of the variance in the dataset. The `data.cumsum()` would calculate the cumulative variance sum of principal components.

```
def Pareto_analysis(data):
    p = data.cumsum()/data.sum()
    print(p)
    key = np.where(p>0.95)[0][0]
    key_product = data[0:key+1]

    x = np.arange(1,1+key+1,1)
    plt.bar(x,key_product)
    plt.ylabel('Variance Explained (%)')
    plt.xlabel('Principal Component')
    plt.plot(x,p[0:key+1]*100, marker='o',color='r',label='cumulative sum')
    plt.legend()
    plt.xticks(x)
```

From the Pareto chart we get above, we find that 3 components are enough. Next, we visualize the data in the principal component space. `pca.transform` is a method in the Principal Component Analysis (PCA) module of `scikit-learn`. In the PCA model, the `transform` method is used to project the original data into the new principal component space. Specifically, it converts the original data into coordinates in the principal component space, facilitating dimensionality reduction or feature extraction.

```
print(pca.transform(standardX))
scores = pca.transform(standardX)
plt.figure()
plt.plot(scores[:,0], scores[:,1],'+')
plt.ylabel('2nd Principal Component')
plt.xlabel('1st Principal Component')
# add text to each dots
for i in range(X.shape[0]):
    plt.text(scores[i,0]+0.1, scores[i,1]+0.1,data['states'][i][0][0],
```

```
    fontsize=6)
```

If we want to visualize the three-dimensional principal component space, run the following code.

```
import matplotlib.pyplot as plt
import numpy as np

fig = plt.figure(figsize=(20,8))
ax = fig.add_subplot(111, projection='3d')
ax.scatter(scores[:,0], scores[:,1], scores[:,2])

for i in range(X.shape[0]):
    ax.text(scores[i,0], scores[i,1], scores[i,2], data['states'][i][0][0],
    size=6)

ax.set_xlabel('1st Principal Component')
ax.set_ylabel('2nd Principal Component')
ax.set_zlabel('3rd Principal Component')

plt.show()
```

Last, we will plot the key visualization of Principal Component Analysis (PCA) called a **Biplot**. The biplot is a two-dimensional graphical representation that simultaneously displays the distribution of samples and features in the principal component space. In the biplot, **the projection of samples** in the principal component space is represented as a scatter plot, while **the direction and length of features** represent their contributions and correlations in the principal component space.

Specifically, a biplot typically includes the following elements:

1. **Sample Projection**: The projection of samples in the principal component space, represented as a scatter plot. Each point represents a sample, and its position is determined by the sample's principal component scores.
2. **Feature Vectors**: The direction and length of features represent their contributions and correlations in the principal component space. They are usually represented by arrows, where the direction of the arrow indicates the loading of the feature on the principal component, and the length of the arrow indicates the importance of the feature.
3. **Axes**: Typically, there are two axes representing the first and second principal components in the principal component space.

Biplot visualization helps us understand the structure of the data, discover similarities and differences between samples, and identify which features contribute most to the variability in the data.

```
def biplot(scores, coefs, name):
    fig, ax = plt.subplots()
    for i in range(X.shape[1]):
        ax.arrow(0, 0, coefs[0, i], coefs[1, i])
        ax.text(coefs[0, i], coefs[1, i], name[i][0])
    xs = scores[:,0]
    ys = scores[:,1]
    scalex = 1.0/(xs.max() - xs.min())
    scaley = 1.0/(ys.max() - ys.min())
    ax.scatter(xs * scalex, ys * scaley,s = 8)

    ax.set_xlabel('Component 1')
    ax.set_ylabel('Component 2')
biplot(scores, pca.components_, data['var_names'][0])
```

Question 2

How can we obtain the directions of the feature vectors? Check it.

2 K-Means Clustering

2.1 Theory Review

- **Purpose.** K-means clustering is a simple and elegant approach for partitioning a data set into K distinct, non-overlapping clusters.
- **How to do.** Specify the desired number of clusters K, and then assign each observation to exactly one of the K clusters. We denote the kth cluster as C_k, and the within-cluster variation for cluster C_k is a measure $W(C_k)$, e.g., the most common choice involves squared Euclidean distance.

$$\begin{aligned} &\min_{C_1,\cdots,C_k} \{\sum_{k=1}^{K} W(C_k)\} \\ &W(C_k) = \frac{1}{|C_k|} \sum_{i,i' \in C_k} \sum_{j=1}^{p} (x_{ij} - x_{i'j})^2 \end{aligned} \tag{4}$$

Algorithm: K-Means Clustering.

1. Randomly assign a number, from 1 to K, to each of the observations. These serve as initial cluster assignments for the observations.
2. Iterate until the cluster assignments stop changing:
   (a) For each of the K clusters, compute the cluster *centroid*. The kth cluster centroid is the vector of the p feature means for the observations in the kth cluster.

(b) Assign each observation to the cluster whose centroid is closest. (where closest is defned using Euclidean distance).

- **Note:** K-means algorithm finds a local rather than a global optimum, the results obtained will depend on the initial cluster.

Question 3

What are the similarities and differences between K-Means and PCA?

2.2 Practice

In this section, we also use the **USArrests dataset.** As usual, we load the dataset first and pocessing the data.

```
# import module needed
import numpy as np
import scipy
from scipy import io
from sklearn.cluster import KMeans

Data = scipy.io.loadmat("./USArrests_data.mat")
X = data['X']
standardX = stats.zscore(X, ddof = 1)
```

We fit the data using the built-in module `KMeans` from `sklearn.cluster`. For more information, you can check: https://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html.

The meanings of some parameters:

- **n_clusters**: int, default=8

  The number of clusters to form as well as the number of centroids to generate.

- **n_init:** 'auto' or int, default='auto' Number of times the k-means algorithm is run with different centroid seeds. The final results is the best output of n_init consecutive runs in terms of inertia.

- **random_state**: random_stateint, RandomState instance or None, default=None Determines random number generation for centroid initialization. Use an int to make the randomness deterministic.

```
# Set the options for the KMeans algorithm
k = 3
options = {'max_iter': 1000}

# Create an instance of the KMeans class and fit the data
kmeans = KMeans(n_clusters=k, n_init=100, random_state=2).fit(standardX)
```

Next, we visualize the clustering results.

```
# Get the label, cluster centers
label= kmeans.labels_
centers = kmeans.cluster_centers_
for i in range(k):
    print('Cluster Centroid %.i'%i,centers[i,:])

plt.figure()
plt.title("K-Means K=3")
plt.scatter(standardX[label == 0, 0], standardX[label == 0, 1], color='r')
plt.scatter(standardX[label == 1, 0], standardX[label == 1, 1], color='y')
plt.scatter(standardX[label == 2, 0], standardX[label == 2, 1], color='g')
plt.scatter(centers[:, 0], centers[:, 1], marker='o', s=100, linestyle=':',
            edgecolor='k', linewidth=2)
plt.legend(['Cluster 1','Cluster 2','Cluster 3','Centroids'])
plt.show()
```

Exercise 1

1. Try to change the value of the `n_init` parameter and observe the outcomes of the execution.
2. Try to change the value of the `n_clusters` parameter and observe the outcomes of the execution.

Next, let's calculate the "*probability*" that each city belongs to different categories. The term "*probability*" here is not in a strict sense. It is reflected by the distance.

```
#calculate membership of each point
dist = np.zeros([data['X'].shape[0],k])
mem_  = np.zeros([data['X'].shape[0],k])
membership  = np.zeros([data['X'].shape[0],k])
for i in range (data['X'].shape[0]):
    for j in range (k):
```

```
        dist[i,j]=np.sqrt(np.sum(np.square(standardX[i,:] - centers[j,:])))
        mem_ [i,j]= 1/dist[i,j]
for i in range (data['X'].shape[0]):
    for j in range (k):
        membership[i,j] = mem_ [i,j]/np.sum(mem_[i,:])
```

Learn by yourself. In order to estimate the "probability" of each city belonging to different clusters, traditional clustering algorithms like K-means do not directly provide probability values. These algorithms assign each observation (in this case, cities) to a specific cluster without probabilistic assignments.

However, if we employ a **probabilistic clustering** method like **Gaussian Mixture Model (GMM)**, we can obtain the probabilities of each city belonging to each cluster or cluster components. This is commonly referred to as posterior probabilities. GMM provides a soft assignment, allowing us to assess the likelihood of an observation belonging to different clusters based on the estimated probabilities.

Question 4

The number of clusters, often represented by K, is indeed a key hyperparameter in K-Means clustering algorithm. Then, how to evaluate different clustering results?

Last we want to evaluate the different clustering results. Compute the mean Silhouette Coefficient of all samples.(For more information, see https://scikit-learn.org/stable/modules/generated/sklearn.metrics.silhouette_score.html, or https://en.wikipedia.org/wiki/Silhouette_(clustering))

Introductions: Silhouette

- The Silhouette Coefficient is calculated using the mean intra-cluster distance a and the mean nearest-cluster distance b for each sample. The Silhouette Coefficient for a sample is $(b - a)/\max\{a, b\}$. To clarify, b is the distance between a sample and the nearest cluster that the sample is not a part of. Note that Silhouette Coefficient is only defined if number of labels is $2 \geqslant n\_labels \geqslant n\_samples - 1$.
- This function returns the mean Silhouette Coefficient over all samples.
- The best value is 1 and the worst value is -1. Values near 0 indicate overlapping clusters. Negative values generally indicate that a sample has been assigned to the wrong cluster, as a different cluster is more similar.

```
from sklearn.metrics import silhouette_samples, silhouette_score
silhouette_avg = silhouette_score(standardX, result)
print('silhouette score = ',silhouette_avg)
silhouette_values = silhouette_samples(standardX, result)
```

We show the Silhouette Coefficient for each sample.

```
#Plot the silhouette values
import matplotlib.pyplot as plt
y_lower = 10
fig, ax1 = plt.subplots()
ax1.set_xlim([-0.1, 1])
ax1.set_ylim([0, len(standardX) + (k + 1) * 10])
for i in range(k):
    ith_cluster_silhouette_values = silhouette_values[result == i]
    ith_cluster_silhouette_values.sort()
    size_cluster_i = ith_cluster_silhouette_values.shape[0]
    y_upper = y_lower + size_cluster_i
    color = plt.colormaps.get_cmap("Spectral")(float(i) / k)
    ax1.fill_betweenx(np.arange(y_lower, y_upper),
    ith_cluster_silhouette_values,facecolor=color, edgecolor=color, alpha=0.5)
    ax1.text(-0.05, y_lower + 0.5 * size_cluster_i, str(i))
    y_lower =y_upper+10
ax1.set_title("Silhouette plot for the k-means clustering of USArrests dataset")
ax1.set_xlabel("Silhouette coefficient values")
ax1.set_ylabel("Cluster label")
ax1.axvline(x=silhouette_avg, color="red", linestyle="--")
ax1.set_yticks([])  # Clear the yaxis labels / ticks
plt.show()
```

3 Hierarchical clustering

3.1 Theory Review

- **Purpose.** Hierarchical clustering is an alternative approach which does not require that we commit to a particular choice of K,i.e., the number of clusters.
- **How to do.** Computer the **dissimilarity** measure between each pair of observations, and **fuse** the most similar two clusters. **Iterate** this procedure and get a tree-based representation of the observations, i.e., the **dendrogram**.

  Algorithm: Hierarchical Clustering

1. Begin with n observations and a measure of all the $\binom{n}{2}$ pairwise **dissimilarities**. Treat each observation as its own cluster.
2. For $i = n, n-1, ..., 2$:
   (a) Examine all pairwise inter-cluster dissimilarities among the i clusters and identify the pair of clusters that are most similar). **Fuse** these two clusters. The dissimilarity between these two clusters indicates the height in the **dendrogram** at which the fusion should be placed.
   (b) Compute the new pairwise inter-cluster dissimilarities among the $i-1$ remaining clusters.

- **Note.** There are two key question left.
  1. How to define **the dissimilarity measure** between two samples?
  2. How to define the dissimilarity **between two clusters**?

  For the first question, we mainly focus on the measure we choose. We can use the `scipy.spatial.distance.pdist` module from the `Scipy` library. Euclidean distance, Minkowski distance and Manhattan distance are most common. For more information, see https://docs.scipy.org/doc/scipy/reference/generated/scipy.spatial.distance.pdist.html.

  For the second question, we mainly focus on the measure **between clusters**. We develop the notion of ***linkage***, the four more common types of linkage are **complete, average, single and centroid**. For more information, you can refer to https://docs.scipy.org/doc/scipy/reference/generated/scipy.cluster.hierarchy.linkage.html.

3.2 Practice

In this section, we also use **USArrests dataset.** This algorithm program is relatively simple and some details we have provided before.

```
from scipy.cluster.hierarchy import dendrogram, linkage,fcluster
from scipy.spatial.distance import pdist
import scipy.io as scio
from scipy import stats
from matplotlib import pyplot as plt

dataFile = "USArrests_data.mat"
data = scio.loadmat(dataFile)
standard_data = stats.zscore(data['X'])
Y = pdist(standard_data,metric='cityblock')
# complete
```

```
Z1 = linkage(Y, 'complete')
fig = plt.figure(figsize=(10, 6))
dn = dendrogram(Z1)
plt.show()
```

After that, we can get a dendrogram. We go on to form flat clusters from the hierarchical clustering. This is also referred to as ***pruning operation***. We use the module `scipy.cluster.hierarchy.fcluster` to realize it. It is necessary to select an appropriate pruning strategy. For more information, see https://docs.scipy.org/doc/scipy/reference/generated/scipy.cluster.hierarchy.fcluster.html.

```
Z3 = fcluster(Z1, t=3, criterion='maxclust',depth=2)

print(Z3) # to see the results
```

We use the criterion `maxclust` to form three clusters.

Exercise 2

Try to change different distance, different type of linkage, or different pruning criterion.

Exercise 3

Test the three unsupervised learning methods we reviewed today by changing the dataset. (We also provide the Iris Dataset in the appendix)