Advanced Python for Neuroscientists
Lecture 4: Unsupervised learning

Summer 2021

Princeton Neuroscience Institute
Instructors: Yisi Zhang & Tyler Giallanza

July 15, 2021

Recap

Lecture 1

- Learning problems

Lecture 2

- linear regression
- Variable selection
- Shrinkage

Lecture 3

- Classifiers
- Cross-validation
- Error analysis

Outline

- Principal component analysis
- K-means clustering
- Hierarchical clustering

4.1 Principal component analysis

Recall that in linear regression, we are trying to find the coefficients that best describe y as a linear combination of X.

$$y = \beta_0 + X\beta.$$

In PCA, we do not have a response, and our goal is finding a set of orthogonal unit vectors V that can reconstruct X as linear combinations of V.

$$X = \mu + V\lambda$$

.

4.1 Principal component analysis

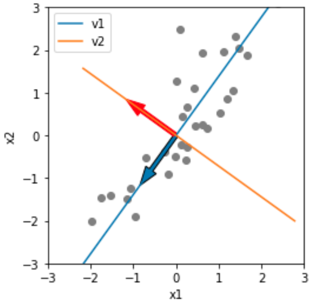
Using least squares, we are solving for V s.t. $\sum_{i=1}^{N} ||(x_i - \mu - V\lambda_i||^2$ is minimized.
One can show that the solution is the singular value decomposition (SVD) of $N \times p$ matrix X (centered):

$$X = USV^T,$$

where U is an $N \times p$ orthogonal matrix $(U^T U = I_p)$, whose columns u_j are called the *left singular vectors*; V is a $p \times p$ orthogonal matrix $(V^T V = I_p)$ with columns v_j called the *right singular vectors*; S is a $p \times p$ diagonal matrix, with elements $s_1 \geq s_2 \geq ... \geq s_p \geq 0$.

4.1 Principal component analysis

The column vectors of V, called loadings are the unit vectors in the original X space ($v_1, v_2, ..., v_q \in \mathbb{R}^p$).



4.1 Principal component analysis

Exercise: Find loadings

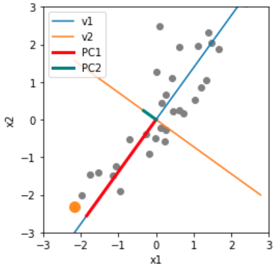
```
import numpy as np
import matplotlib.pyplot as plt
x1 = np.random.normal(0,1,30) # make a 2d X
x2 = x1 + np.random.normal(0,1,30)
X = np.vstack((x1,x2)).T
Xmean = np.mean(X,axis=0)
Xc = X - Xmean # center the data
u, s, vh = np.linalg.svd(Xc, full_matrices=False)
plt.scatter(x1, x2, c='gray')
plt.arrow(0,0,vh[0,0],vh[0,1],width=0.1)
plt.arrow(0,0,vh[1,0],vh[1,1],width=0.1,color='red')
```

4.1 Principal component analysis

For a given data point x_i, it can be written as a linear combination of V:

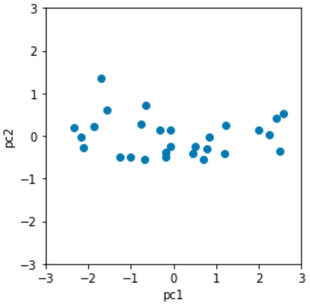
$$x_i = u_{i1}s_1 v_1^T + u_{i2}s_2 v_2^T + \dots + u_{ip}s_p v_p^T.$$

Thus, $u_{ij}s_j$ is the projection of x_i onto the axis v_j, also called the ith principal component (PC) or score.



4.1 Principal component analysis

We can then plot the principal components in the coordinate of the PCs (use $v_1, v_2, \ldots$ as coordinates). Plots in the PC space are often used to visualize patterns in the data. It is obvious that we can essentially reduce the 2d data to 1d in this case.



4.1 Principal component analysis

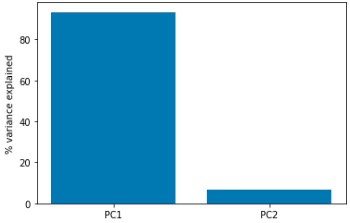
Exercise: Plot principal components

```
pc = u*s
fig = plt.figure()
plt.scatter(pc[:,0], pc[:,1])
plt.xlabel('pc1')
plt.ylabel('pc2')
```

4.1 Principal component analysis

The variance of the first principal component is thus $\mathrm{Var}(Xv_1) = v_1^T X^T X v_1 = s_1^2$, and so on.
We can calculate the percentage of variance each component explained as

$$\%var_i = \frac{s_i^2}{\sum_{j=1}^{p} s_j^2}$$



4.1 Principal component analysis

Exercise: Check our results with sklearn

```
from sklearn.decomposition import PCA
pca = PCA(n_components=2)
pca.fit(X)
print(pca.explained_variance_ratio_)
print(var_explained)
```

4.2 K-means clustering

K-means clustering partitions a data into K non-overlapping clusters with the *within cluster variation* minimized:

$$\min_{C_1,\ldots,C_k} \sum_{k=1}^{K} W(C_k),$$

$$W(C_k) = \frac{1}{|C_k|} \sum_{i,i' \in C_k} \sum_{j=1}^{p} (x_{ij} - x_{i'j})^2,$$

where $|C_k|$ denotes the number of observations within the cluster and p the dimension of the features.

4.2 K-means clustering

K-means algorithm

- 1. Randomly assign a number from 1 to K to each of the observations.
- 2. Iterate until cluster assignment stops changing
  (a) Computer cluster centroid.
  (b) Assign each observation to the cluster whose centroid is closest.

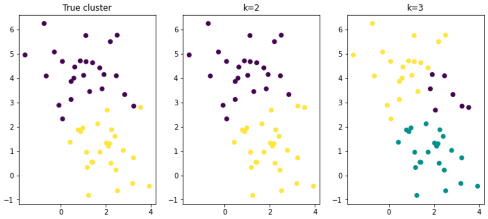
4.2 K-means clustering

Exercise: Write a k-means code

```
import numpy as np
def kmeans(x, k, maxiter = 1000):
    n,p = x.shape
    c = np.random.choice(k, n)
    assign_finish = False
    niter = 0
    while (not assign_finish) or niter<maxiter:
        c_last = c
        cent=[np.mean(x[c==i,:], axis=0) for i in range(k)]
        kdist = [np.sum((x-cent[i])**2, axis=1) for i in range(k)]
        kdist = np.stack(kdist,axis=0)
        c = np.argmin(kdist,axis=0)
        assign_finish = not(any(c!=c_last)) ...
```

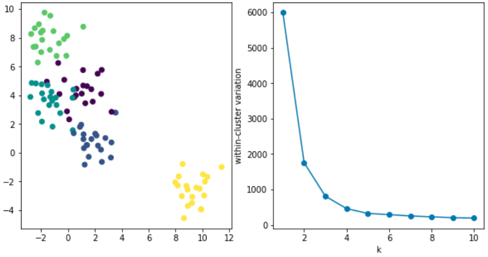
4.2 K-means clustering

How to choose K?



4.2 K-means clustering

One way is using the elbow shape of the within-cluster variation vs. k plot to choose the k where the error starts to diminish. This method may not be obvious sometimes.

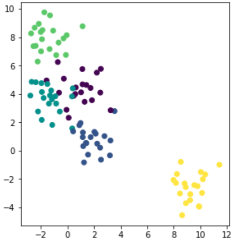


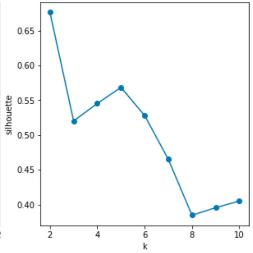
4.2 K-means clustering

A more sensitive measure is the silhouette metric, which measures how similar a point is to its own cluster compared to other clusters.

$$s(i) = \frac{b(i) - a(i)}{max\{a(i), b(i)\}},$$

where $a(i)$ is the average within-cluster distance and $b(i)$ is the average cross-cluster distance for the ith data point.





4.2 K-means clustering

Exercise: "Elbow" plot

```
from sklearn.cluster import KMeans
# define a function to calculate within-cluster variation
def kmeans_WCV(x,kmax):
    wcv = []
    for k in range(1,kmax+1):
        kmeans = KMeans(n_clusters = k).fit(x)
        cent = kmeans.cluster_centers_
        c = kmeans.labels_
        kdist = [2*np.sum((x[c==i,:]-cent[i,:])**2) for i in range(k)]
        wcv.append(np.sum(kdist))
    return wcv
```

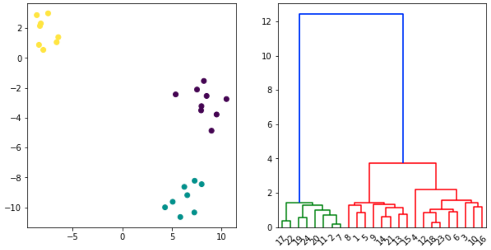
4.2 K-means clustering

Exercise: Silhouette plot

```
from sklearn.metrics import silhouette_score
sil = []
kmax = 10
for k in range(2, kmax+1):
   kmeans = KMeans(n_clusters=k).fit(X)
   labels = kmeans.labels_
   sil.append(silhouette_score(X, labels, metric='euclidean'))
```

4.3 Hierarchical clustering

Hierarchical clustering is an alternative approach that does not require that we manually choose the number of clusters K. It also gives us a tree-based representation of the data, called a dendrogram. We cut the dendrogram to obtain clusters.



4.3 Hierarchical clustering

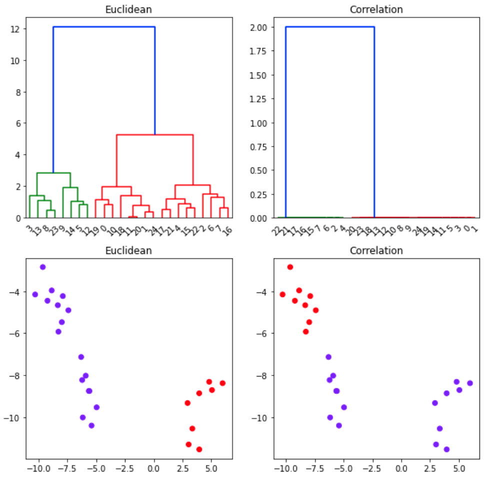
There are two strategies for hierarchical clustering: *agglomerative* (bottom-up) and *divisive* (top-down).
The way the dissimilarity between two groups of points (linkage) is determined affects the clustering result. There are four most common types:

Linkage	**Description**
Complete	Largest dissimilarity between two clusters.
Single	Smallest dissimilarity between two clusters.
Average	Mean inter-cluster dissimilarity.
Centroid	Dissimilarity between the centroids of two clusters.

4.3 Hierarchical clustering

The choice of dissimilarity measure has a strong effect on the dendrogram.
The most common ones are either *Euclidean* or *correlation*-based distance.
The correlation-based distance focuses on the shape of observation profiles rather than the magnitudes.

4.3 Hierarchical clustering



4.3 Hierarchical clustering

Exercise: Hierarchical clustering

```
from scipy.cluster import hierarchy
from sklearn.cluster import AgglomerativeClustering
# get linkage, specify method and distance metric
Z = hierarchy.linkage(X, 'complete', 'euclidean')
plt.subplot(1,2,1) # plot dendrogram
dn = hierarchy.dendrogram(Z)
# get cluster based on cut of dendrogram
cluster = AgglomerativeClustering(n_clusters=2,
affinity='euclidean', linkage='complete')
cluster.fit_predict(X)
plt.subplot(1,2,2)
plt.scatter(X[:,0],X[:,1], c=cluster.labels_)
```

Homework

- Make sure you understand all the exercises above
- Run through the codes here that should replicate all the figures https://github.com/yisiszhang/AdvancedPython/blob/main/colab/Lecture4.ipynb
- Make high-dimensional blob data and plot in the PC space the first 2 PCs.
- Apply clustering to the data.