K-means and hierarchical clustering with PYTHON

Objective: The objective of today's exercise is to understand how the unsupervised learning methods k-means clustering and hierarchical clustering work. Upon completing the exercise you should also understand how the choice of number of clusters, distance metrics and linkage functions can impact the solutions obtained and further be able to interpret dendrograms and measures of cluster validity.

Material: Lecture notes *"Introduction to Machine Learning and Data Mining"* as well as the files provided from DTU Learn.

In previous exercises we considered supervised learning, i.e., we were given both input data $\boldsymbol{X}$ and output values $\boldsymbol{y}$. In classification, the outputs were discrete variables, and in regression, the outputs were continuous.

We now move on to unsupervised learning where we are only provided input data $\boldsymbol{X}$. The aim is here to find common patterns in the data such as groups of observations that are similar in some sense. In this exercise we will consider two clustering approaches for unsupervised learning: k-means clustering and hierarchical clustering.

10.1 k-means clustering

In this part of the exercise we will investigate k-means clustering. In k-means each of the data points are assigned to the cluster in closest proximity according to some measure of distance between cluster centers and data points. When the distance is given by the squared Euclidean distance the centers are also denoted centroids. Once the data points have been assigned, each cluster center is updated to be placed at the center of the data points that are assigned to the cluster. This continues iteratively, usually until the assignment of data points to centers no longer change or until a maximal number of iterations is reached.

10.1.1 You can load the `Data/synth1.mat` data set file into Python using the `loadmat` function. The script `ex10_1_1.py` clusters the data into $K = 4$ clusters using the k-means algorithm. Notice how the script makes a scatter plot of the data and the clustering using the `clusterplot` function in the toolbox.

Script details:

- *In Python, you can use the function* `k_means()` *from the package* `sklearn.cluster` *to compute k-means clustering. Import the function and type* `help(k_means)` *to learn how to use the function.*
- *The function can be called as* `centroids, clusters, inertia = k_means(X,K);` *where K is the number of clusters.*
- *Type* `clusterplot(X,clusters,centroids,y)` *to plot the data and the clustering.*
- *Type* `help(clusterplot)` *to learn more about how to use the clustering plot tool in the toolbox.*
- *To be more robust against different initial conditions, you can run multiple iterations of clustering with different initial centroid seeds, see the* `'n_init'` *parameter of* `k_means()`.

Does the clustering coincide with the true classes? Try running your code several times (with `n_init` set to 1) to show that the algorithm can fail if the initial conditions are poor. Try also the data sets `synth2`, `synth3`, and `synth4`.

For supervised learning we evaluated the model performance in terms of the error rate and accuracy. This however requires that we can match the estimated outcome to the true underlying classes, provided they are known. Even when we know the true underlying classes, we do not in

general know which cluster corresponds to which class. Thus, by some means the true classes and the estimated clusters must be related to each other. One way of relating the two is to find out which cluster best matches each class such that each class is assigned one of the clusters and then calculate the error rate based on these clusters.

10.1.2 Is the above definition of the classification error rate for clustering reasonable if we extract the same number of clusters as classes in the data? Does the definition of the classification error make sense when the number of extracted clusters are different from the true underlying classes?

Rather than using the error rate we will consider the supervised measures of cluster validity, in particular Rand Statistic, Jaccard coefficient and normalized mutual information (NMI). Carefully review these measures in the book and make sure you understand how they are calculated.

10.1.3 The script `ex10_1_3.py` repeats exercise 10.1.1, but this time perform k-means clustering for $K = 1, \ldots, 10$ clusters. For each value of K the three cluster validity measures mentioned above are computed. Notice how the script plots the cluster validity measures as a function of K.

Script details:

- *It is a good exercise that write your own code for computing the cluster validity measures. You can look at simple example in the toolbox (function* `clusterval()`*).*
- *You can find many additional cluster validity measures in* `sklearn.metrics.cluster` *package. The script* `ex10_1_3.py` *shows how to use some of them (completeness score, homogeneity score, v-measure-score and others). You can find more details in* `sklearn` *package documentation.*

How can the cluster validity measures be used to select the best number of clusters?

What happens when more than four clusters are used to model the data?

10.1.4 For supervised learning we used cross-validation to evaluate performance and estimate the number of parameters in our models, i.e., the number of clusters.

Let us assume that we split the data into a training and a test set, train the k-means model on the training set and evaluate how well the model accounts for the test data. Consider evaluating the clustering by summing the distance of test points to the closest estimated cluster center obtained. What will happen with this training and test error as we increase the number of clusters?

K-means clustering has many different applications, one of which is data compression. A data set can be compressed by performing k-means clustering and then representing each data object by its cluster center. Thus, the only data that need to be stored is the K cluster centers and the N cluster indices.

10.1.5 We will consider a subset of the wild faces data described in [2]. You can load the wildfaces data set from the `Data/wildfaces.mat` file with the `loadmat` function, see the script `ex10_1_5.py`. Each data object is a $40 \cdot 40 \cdot 3 = 4800$ dimensional vector, corresponding to a 3-color 40×40 pixels image. The script computes a k-means clustering of the data with $K = 10$ clusters. Plot a few random images from the data set as well as their corresponding cluster centroids to see how they are represented.

Script details:

- *You can plot an image by the command* `imshow(np.reshape(X[k,:],(c,x,y)).T)` *which reshapes an image vector to a 3-dimensional array and plots it. Similarly, you can plot the cluster centroids.*

- *Running k-means on a large data set can be slow. If you type* `k_means(X, K, verbose=True, max_iter=X, n_init=S)` *it will provide information about the iterations as they run.* `n_init` *as before constrains the number of initial repeated centroid seeds. Type* `help(k_means)` *to read more about these options.*

How well is the data represented by the cluster centroids? Are you able to recognize the faces in the compressed representation? What happens if you increase or decrease the number of clusters?

10.1.6 Modify the script `ex10_1_5.py` to repeat the previous exercise but with the digits data set. You can load the digits data set from the file `Data/digits`. Each data object is a $16 \cdot 16 = 256$ dimensional vector, corresponding to a gray scale 16×16 pixels image.

Script details:

- *You can change the color map to black-on-white grey-scale by adding paramter* `cmap=cm.binary` *to* `imshow()` *function.*

Why does running k-means with $K = 10$ not give you 10 clusters corresponding to the 10 digits 0–9? How many clusters do you need to visually represent the 10 different digits? Are there any digits that the clustering algorithm seems to confuse more than others?

10.2 Hierarchical clustering

We will in this part of the exercise consider hierarchical clustering based on the functions from the package `scipy.cluster.hierarchy`. Function `linkage()` forms a sample to sample distance matrix according to a given distance metric, and creates the linkages between data points forming the hierarchical cluster tree. Function `dendrogram` creates a plot of the generated tree. Function `fcluster` extracts the cluster from linkage matrix wrt given criterion. Use `help` for the three function and see how they are used and inspect what distance metrics and linkage functions are implemented.

10.2.1 Inspect and run the script `ex10_2_1.py`. The script loads the data set from the file `Data/synth1` and partitions dthe data using hierarchical clustering with single linkage using the Euclidean distance measure. Notice how the script is used to cluster the data into 4 clusters by cutting off dendrogram at a threshold and plots a dendrogram and a scatter plot of the clusters.

Script details:

- *The function* `linkage()` *computes the hierarchical clustering, resulting in a matrix representing the hierarchy of clusterings. Type* `help(linkage)` *to learn how to use it.*
- *The second parameter of* `linkage()` *can be used to select the method used in the hierarchical clustering procedure.*
- *The third parameter of* `linkage()` *can be used to change the distance measure.*
- *You can e.g. type* `Z = linkage(X, method='single', metric='euclidean)` *to use single linkage with the Euclidean distance measure.*
- *To compute a clustering, you can use the function* `fcluster()`*. For example, type* `cls = fcluster(Z, c` *to get a maximum of 4 clusters. Type* `help(fcluster)` *to learn more about what this function does.*
- *To plot a dendrogram, you can use the* `dendrogram()` *function.*
- *Again, you can use the function* `clusterplot()` *from the toolbox to plot a scatter plot of the clustering.*

Try changing the linkage method and see how it changes the dendrogram. Try running your code several times to see if it generates exactly the same dendrogram each time. Try also the data sets `synth2`, `synth3`, and `synth4`, and choose suitable distance measures for these data sets.

10.3 Old Faithful geyser data

Old Faithful is a famous geyser located in Yellow Stone national park in the US. The Old Faithful geyser dataset described in [1, 3] consists of $N = 272$ observations of two variables, namely the duration of each eruption in minutes (duration) and the waiting time between eruptions also in minutes (waiting).

10.3.1 Load the Old Faithful data in the file `load Data/faithful.mat`. Analyze the data by `k-means` visually inspect the labeling of the data points.

What happens if you increase K beyond the obvious $K = 2$? Try to run the program a couple of times (resulting in different initial conditions). Do you see the same solution every time? The scaling of the variables can seriously affect the results we get in clustering. Discuss whether it is more reasonable to normalize the Old faithful data set? Try normalizing the data and see whether the results of running `k-means` change.

10.3.2 Run hierarchical clustering of the Old Faithful data using the "single" and "ward" linkage, with and without normalizing the data. Do you find support for a two cluster model from the structure of the dendrograms?

10.4 Extra Challenge

10.4.1 Try clustering some of the data sets you have analyzed in the previous exercises such as the Iris data and the wine data using k-means and hierarchical clustering.

References

[1] A Azzalini and AW Bowman. A look at some data on the old faithful geyser. *Applied Statistics*, pages 357–365, 1990.

[2] Tamara L Berg, Alexander C Berg, Jaety Edwards, and DA Forsyth. Who's in the picture. *Advances in neural information processing systems*, 17:137–144, 2005.

[3] Wolfgang Härdle. *Smoothing techniques: with implementation in S*. Springer, 1991.