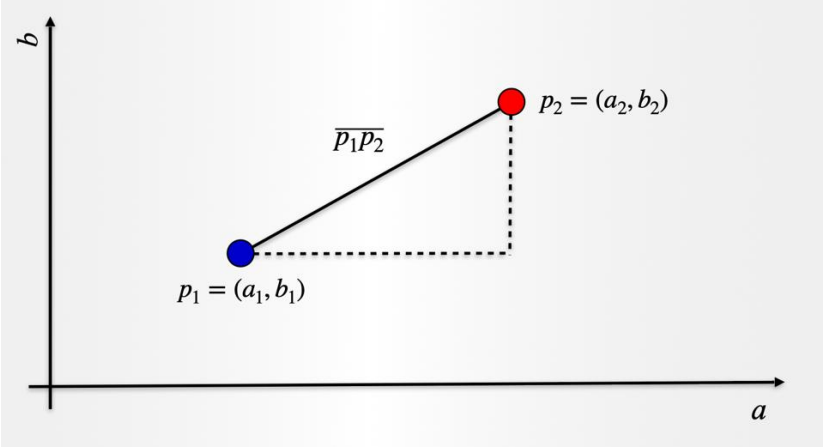
Machine Learning
Lab 7

K-means - what does it mean?

We mentioned that we are interested in finding out commonalities among our data observations. One way to determine that commonality or similarity is through a measure of distance among the data points. The shorter the distance, the more similar the observations are. There are different ways in which we can measure that distance and one that is very familiar to a lot of people is the Euclidean distance. That's right! The same one we are taught when learning the Pythagorean theorem. Let us take a look and consider two data observations over two attributes ***a*** and ***b***. Point ***p1*** has coordinates ***(a1, b1)*** and point ***p2 = (a2, b2)***.



The distance is given by:

$$\overline{p_1p_2} = \sqrt{(a_2 - a_1)^2 + (b_2 - b_1)^2}$$

The expression above can be extended to more than 2 attributes and the distance can be measured between any two points. For a dataset with ***n*** observations, we assume there are ***k*** groups or clusters, and our aim is to determine which observation corresponds to any of those ***k*** groups. This is an important point to emphasise: the algorithm will not give us the number of clusters, instead we need to define the number ***k*** in advance. We may be able to run the algorithm with different values for ***k*** and determine the best possible solution.

In a nutshell, k-means clustering tries to minimise the distances between the observations that belong to a cluster and maximise the distance between the different clusters. In that way, we have cohesion between the observations that belong to a group, while observations that belong to a different group are kept further apart. Please note that k-means is exhaustive in the sense that every single observation in the dataset will be forced to be part of one of the ***k*** clusters assumed.

It should now be clear where the ***k*** in k-means comes from, but what about the "means" part? Well, it turns out that as part of the algorithm we are also looking to identify the centre for each cluster. We call this a centroid, and as we assign observations to one cluster or the

other, we update the position of the cluster centroid. This is done by taking the mean (average if you will) of all the data points that have been included in that cluster. Easy!

A recipe for k-means
The recipe for k-means is quite straightforward.

1. Decide how many clusters you want, i.e. choose k
2. Randomly assign a centroid to each of the k clusters
3. Calculate the distance of all observation to each of the k centroids
4. Assign observations to the closest centroid
5. Find the new location of the centroid by taking the mean of all the observations in each cluster
6. Repeat steps 3-5 until the centroids do not change position

Take a look at the representation below where the steps are depicted in a schematic way for a 2-dimensional space. The same steps can be applied to more dimensions (i.e. more features or attributes). For simplicity, in the schematic we only show the distance measured to the closest centroid, but in practice all distances need to be considered.

Load data

Scikit-learn is a good choice and it has a very nice implementation for K-means. In this case we will show how k-means can be implemented in a couple of lines of code using the well-known Iris dataset. We can load it directly from Scikit-learn and we will shuffle the data to ensure the points are not listed in any particular order.

```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.colors import ListedColormap
%matplotlib inline

from sklearn.cluster import KMeans
from sklearn import datasets
from sklearn.utils import shuffle

# import some data to play with
iris = datasets.load_iris()
X = iris.data
y = iris.target
names = iris.feature_names
X, y = shuffle(X, y, random_state=42)
```

Apply k-means clustering

We can call the KMeans implementation to instantiate a model and fit it. The parameter *n_clusters* is the number of clusters ***k***. In the example below we request 3 clusters:

```
model = KMeans(n_clusters=3, random_state=42)
iris_kmeans = model.fit(X)
```

Check clustering results

We can look at the labels that the algorithm has provided as follows:

```
iris_kmeans.labels_
```

In order to make a comparison, let us reorder the labels:

```
y = np.choose(y, [1, 2, 0]).astype(int)
y
```

We can check how many observations were correctly assigned. We do this with the help of a confusion matrix:

```
from sklearn.metrics import confusion_matrix
conf_matrix=confusion_matrix(y, iris_kmeans.labels_)

fig, ax = plt.subplots(figsize=(7.5, 7.5))
ax.matshow(conf_matrix, cmap=plt.cm.Blues, alpha=0.3)
for i in range(conf_matrix.shape[0]):
    for j in range(conf_matrix.shape[1]):
        ax.text(x=j, y=i,s=conf_matrix[i, j], va='center',
                ha='center', size='xx-large')

plt.xlabel('Predictions', fontsize=18)
plt.ylabel('Actuals', fontsize=18)
plt.title('Confusion Matrix', fontsize=18)
plt.show()
```

Let us look at the location of the final clusters:

```
iris_kmeans.cluster_centers_
```

And we can look at some 3D graphics. In this case we will plot the following features:

- Petal width
- Sepal length
- Petal length

```
customcmap = ListedColormap(["crimson", "mediumblue", "darkmagenta"])

fig = plt.figure(figsize=(20, 10))
ax1 = fig.add_subplot(1, 2, 1, projection='3d')
ax1.scatter(X[:, 3], X[:, 0], X[:, 2],
            c=iris_kmeans.labels_.astype(float),
           edgecolor="k", s=150, cmap=customcmap)
ax1.view_init(20, -50)
ax1.set_xlabel(names[3], fontsize=12)
ax1.set_ylabel(names[0], fontsize=12)
ax1.set_zlabel(names[2], fontsize=12)
ax1.set_title("K-Means Clusters for the Iris Dataset", fontsize=12)

ax2 = fig.add_subplot(1, 2, 2, projection='3d')

for label, name in enumerate(['virginica','setosa','versicolor']):
    ax2.text3D(
        X[y == label, 3].mean(),
        X[y == label, 0].mean(),
        X[y == label, 2].mean() + 2,
        name,
        horizontalalignment="center",
        bbox=dict(alpha=0.2, edgecolor="w", facecolor="w"),
    )

ax2.scatter(X[:, 3], X[:, 0], X[:, 2],
            c=y, edgecolor="k", s=150,
            cmap=customcmap)
ax2.view_init(20, -50)
ax2.set_xlabel(names[3], fontsize=12)
ax2.set_ylabel(names[0], fontsize=12)
ax2.set_zlabel(names[2], fontsize=12)
ax2.set_title("Actual Labels for the Iris Dataset", fontsize=12)
fig.show()
```

Activity

Hierarchical clustering is also a commonly used unsupervised learning method for clustering data points. The algorithm builds clusters by measuring the dissimilarities between data. Unsupervised learning means that a model does not have to be trained, and we do not need a "target" variable. This method can be used on any data to visualize and interpret the relationship between individual data points.

Consider the following data points, please use euclidean distance and the Ward linkage method, and visualize it using a dendrogram:

x = [4, 5, 10, 4, 3, 11, 14 , 6, 10, 12]

y = [21, 19, 24, 17, 16, 25, 24, 22, 21, 21]