Introduction to clustering, Hierarchical: AGNES, DIANA, Partitional: K-means clustering, K-Mode Clustering, Self-Organizing Map, Expectation Maximization, Gaussian Mixture Models

Hierarchical Methods

A hierarchical clustering method works by grouping data objects into a tree of clusters. Hierarchical clustering methods can be further classified as either agglomerative or divisive, depending on whether the hierarchical decomposition is formed in a bottom-up (merging) or top-down (splitting) fashion. The quality of a pure hierarchical clustering method suffers from its inability to perform adjustment once a merge or split decision has been executed. That is, if a particular merge or split decision later turns out to have been a poor choice, the method cannot backtrack and correct it. Recent studies have emphasized the integration of hierarchical agglomeration with iterative relocation methods**.**

Agglomerative and Divisive Hierarchical Clustering

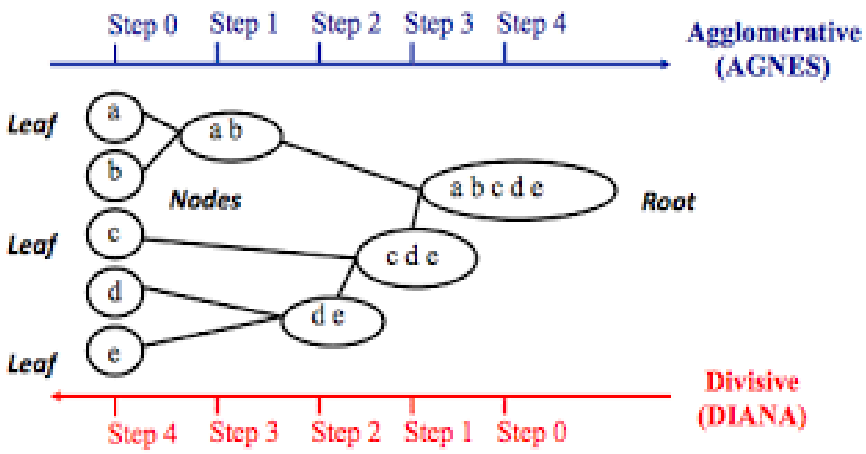
In general, there are two types of hierarchical clustering methods:

1) Agglomerative hierarchical clustering: This bottom-up strategy starts by placing each object in its own cluster and then merges these atomic clusters into larger and larger clusters, until all of the objects are in a single cluster or until certain termination conditions are satisfied. Most hierarchical clustering methods belong to this category. They differ only in their definition of inter-cluster similarity.

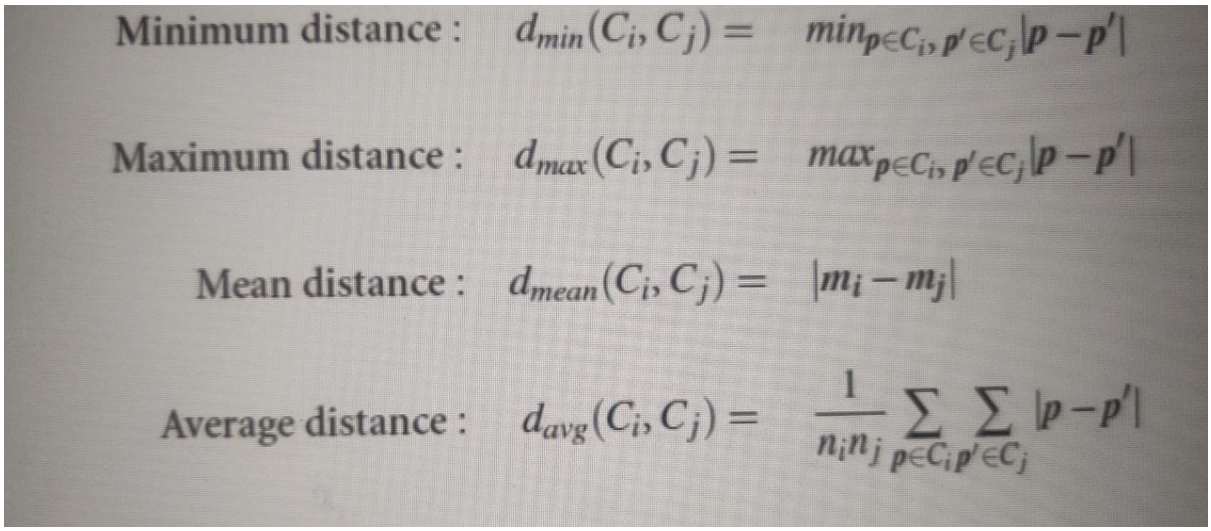
2)Divisive hierarchical clustering: This top-down strategy does the reverse of agglomerative hierarchical clustering by starting with all objects in one cluster. It subdivides the cluster into smaller and smaller pieces, until each object forms a cluster on its own or until it satisfies certain termination conditions, such as a desired number of clusters is obtained or the diameter of each cluster is within a certain threshold.

Agglomerative versus divisive hierarchical clustering.

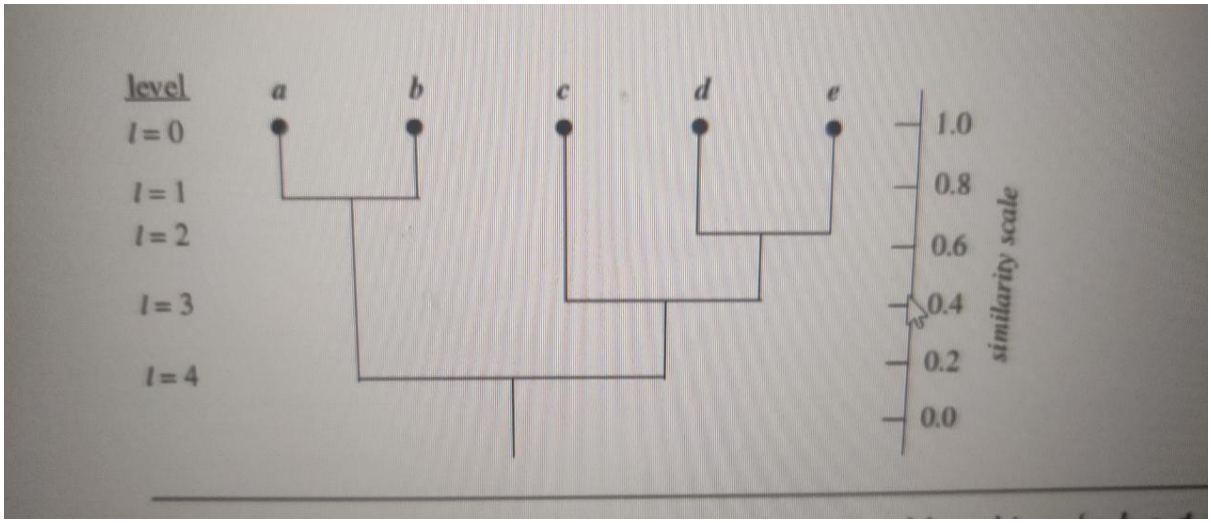
The below figure shows the application of AGNES (AGglomerative NESting), an agglomerative hierarchical clustering method, and DIANA (DIvisive ANAlysis), a divisive hierarchical clustering method, to a data set of five objects, {a, b, c, d, e}. Initially, AGNES places each object into a cluster of its own. The clusters are then merged step-by-step according to some criterion. For example, clusters C1 and C2 may be merged if an object in C1 and an object in C2 form the minimum Euclidean distance between any two objects from different clusters. This is a single-linkage approach in that each cluster is represented by all of the objects in the cluster, and the similarity between two clusters is measured by the similarity of the closest pair of data points belonging to different clusters. The cluster merging process repeats until all of the objects are eventually merged to form one cluster. In DIANA, all of the objects are used to form one initial cluster. The cluster is split according to some principle, such as the maximum Euclidean distance between the closest neighboring objects in the cluster. The cluster splitting process repeats until, eventually, each new cluster contains only a single object.



In either agglomerative or divisive hierarchical clustering, the user can specify the desired number of clusters as a termination condition. A tree structure called a dendrogram is commonly used to represent the process of hierarchical clustering. It shows how objects are grouped together step by step. Figure 7.7 shows a dendrogram for the five objects presented in Figure 7.6, where l = 0 shows the five objects as singleton clusters at level 0. At l = 1, objects a and b are grouped together to form the first cluster, and they stay together at all subsequent levels. We can also use a vertical axis to show the similarity scale between clusters. For example, when the similarity of two groups of objects, {a, b} and {c, d, e}, is roughly 0.16, they are merged together to form a single cluster. Four widely used measures for distance between clusters are as follows, where |p-p'| is the distance between two objects or points, p and p0; mi is the mean for cluster, Ci; and ni is the number of objects in Ci.



When an algorithm uses the minimum distance, dmin(Ci, Cj), to measure the distance between clusters, it is sometimes called a nearest-neighbor clustering algorithm. oreover, if the clustering process is terminated when the distance between nearest clusters exceeds an arbitrary threshold, it is called a single-linkage algorithm. If we view the data points as nodes of a graph, with edges forming a path between the nodes in a cluster, then the merging of two clusters, Ci and Cj, corresponds to adding an edge between



K-Means Clustering

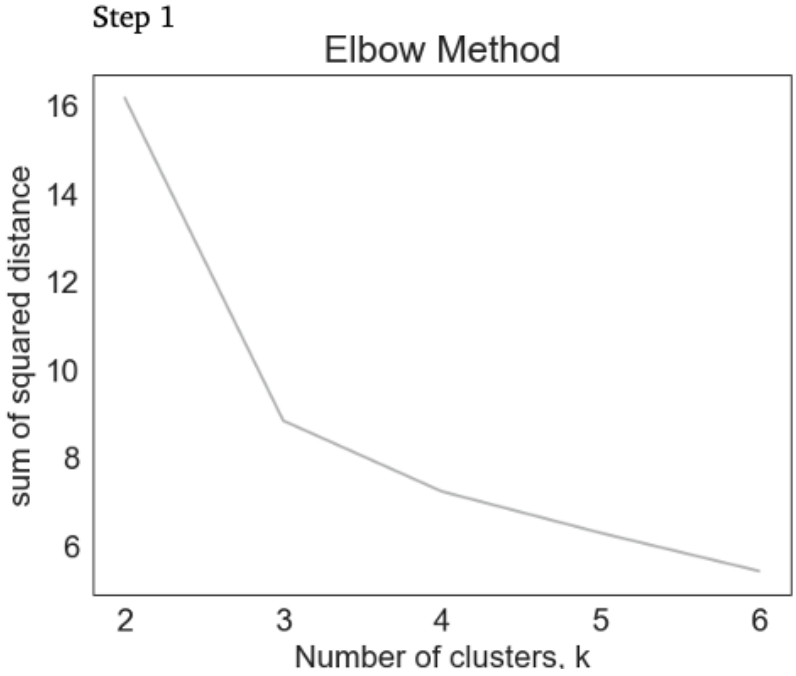
K-Means is easily the most popular clustering algorithm due to its simplicity. Ultimately, it assumes that the closer data points are to each other, the more similar they are. The process is as follows:

1. Choose the number of clusters *K*
2. Randomly establish the initial position for each centroid
3. Assign each data point to the nearest centroid using the distance measure
4. Reset each centroid location using the mean of the data points assigned to each cluster
5. Return to step 3 and repeat until no single data point changes cluster

How do you find the optimal number of clusters, *K*?

One of the most difficult steps in clustering is to determine the optimal number of clusters, *K,* to group the data, and there is no 'right' answer. The most common approach is known as 'the elbow method'. Essentially, it runs K-Means clustering on the whole dataset for various values of *K* and calculates the overall sum of squared errors (SSE) for each *K.*

It is important to plot the results. The plot resembles an arm, and as the name suggests, the value of *K* where the steepness changes the most (i.e. the elbow) is considered the optimal number of clusters. **The objective is not to find the *K* that minimizes total squared distance, but rather the *K* that results in diminishing returns when *K* is increased.**



In our example, the elbow method returns a value of 3 as the optimal number of clusters (and we know this to be correct based on the actual data).

What is the distance measure?

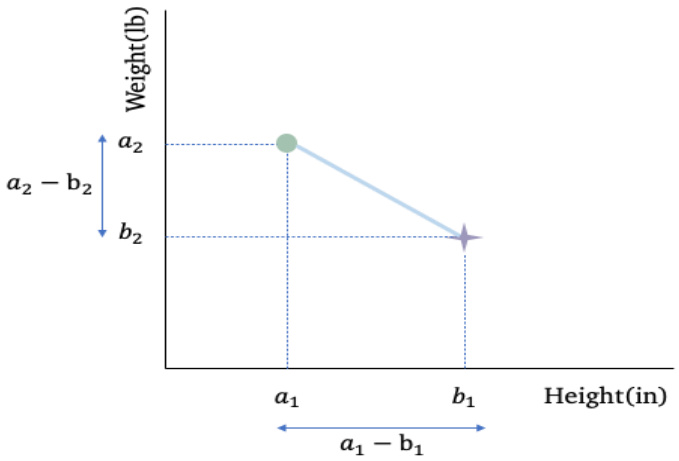
The K-Means algorithm uses the **Euclidean Distance Measure**. This means that the measure of distance around each cluster center is 'circular'. Said differently, **the importance of each dimension is equal, hence the term 'circular'**. The distance can be defined as

$$d_{a,b} = \sqrt{\sum_{j=1}^{J}(a_j - b_j)^2}$$

where *J* represents the number of dimensions.

In our example, we apply this formula to two dimensions:

$$d_{a,b} = \sqrt{(a_1 - b_1)^2 + (a_2 - b_2)^2}$$



If we compute the distance between the Height and Weight of Derrick Henry (DH) versus the Heights and Weights of the RB and TE centroids, we see that there is a much shorter distance to the TE centroid and therefore, he is classified as a TE.

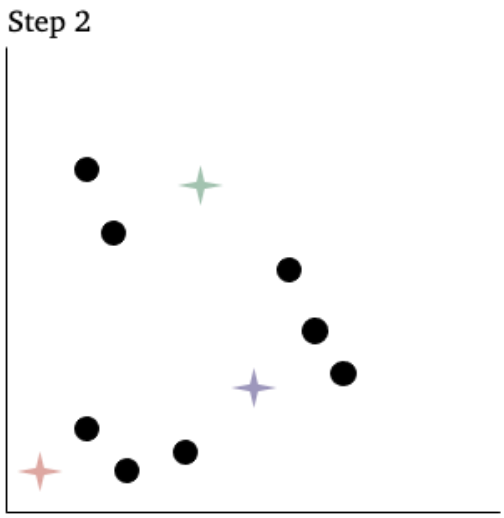
$$d_{DH,\ RB\ centroid} = \sqrt{(75 - 71)^2 + (247 - 213)^2} = 34$$

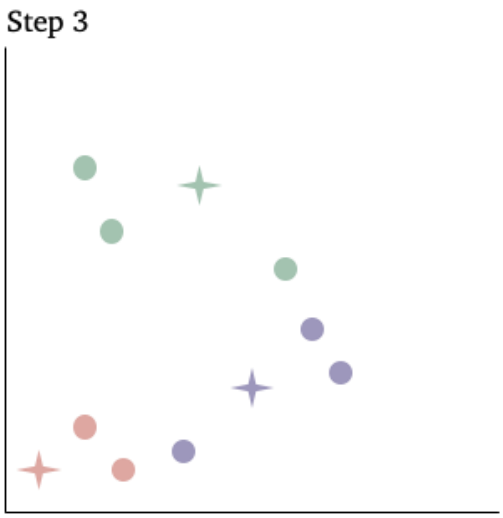
$$d_{DH,\ TE\ centroid} = \sqrt{(75 - 76)^2 + (247 - 251)^2} = 4$$

How do you reset each centroid location?

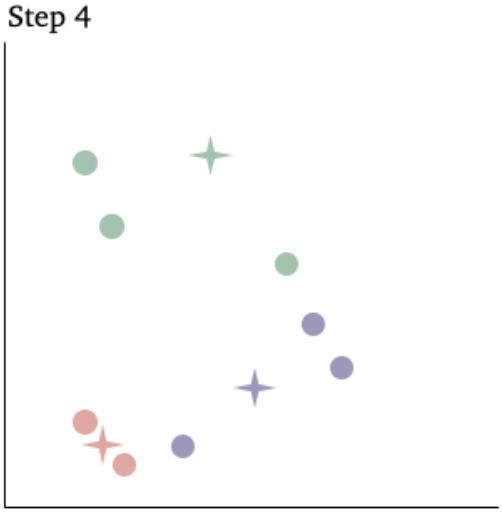
Having assigned each data point to a cluster, we simply calculate the arithmetic mean of the data points for each dimension in each cluster (i.e. the mean of all player Heights and the mean of all player Weights for each position). This is then the new position of each centroid. The use of the mean is where the algorithm gets its name: K-Means.

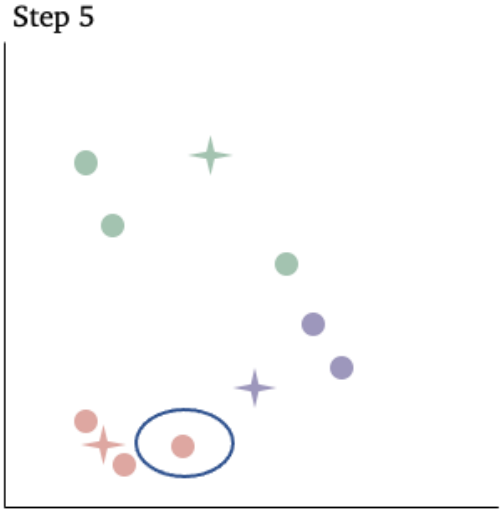
The algorithm is implemented directly in the sklearn.cluster.KMeans package. However, we will go through a small example to show what happens after step 1 (determining the optimal number of clusters).



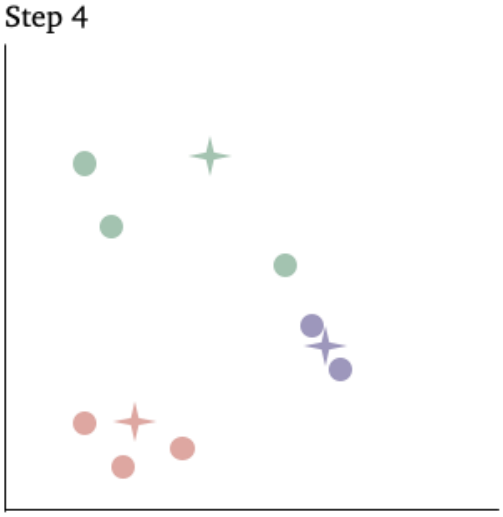


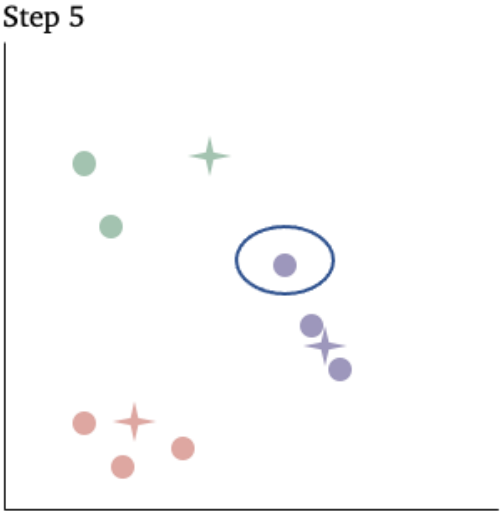
To start, in step 2, each centroid is randomly initialized. In the second image, step 3, the data points are assigned to the centroid that is closest using the distance measure.



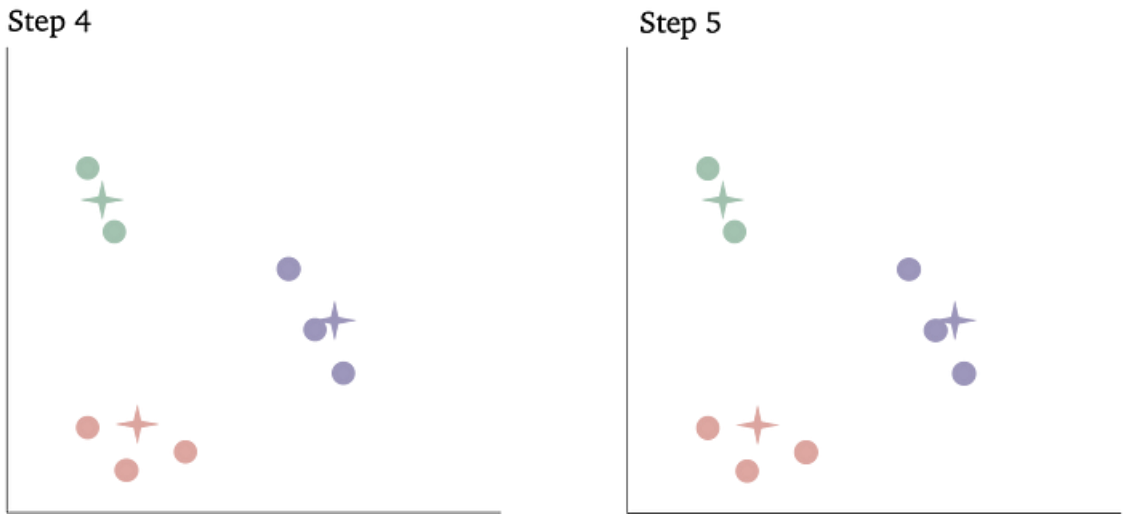


Next, each centroid location is reset based on the mean. In the second image, the data points are reassigned to the centroid that now has the least distance to the data point.



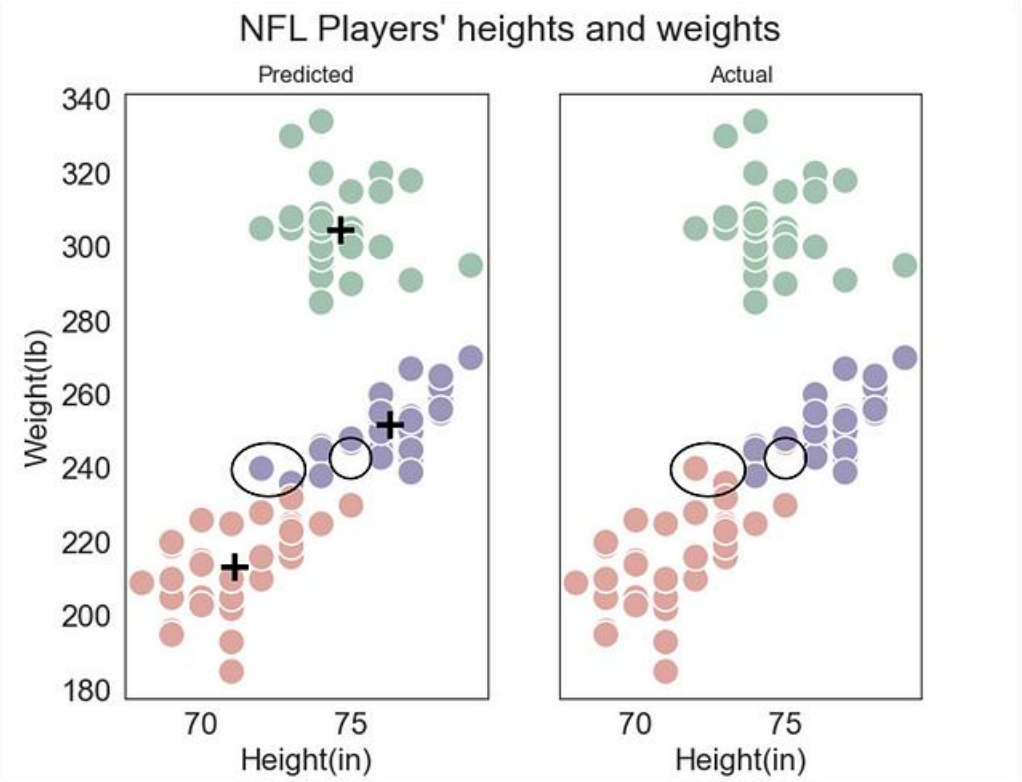


The process above is repeated



The process is finished since no data point changes clusters with the new centroid locations

We ran the algorithm on the data set in Python and the results were encouraging.



Results: The clusters correspond to the actual data clusters with only three players being misclassified. We are able to easily evaluate the clustering since we already know each players' position. All DTs were clustered into one group successfully, but that is not surprising given that we could see that by eye. What is impressive is that the majority of the RBs and TEs were clustered correctly despite appearing to be one continuous group of data.

Advantages of K-Means:

- Simple to understand
- Very quick (all that is being computed is the distance between each point and cluster center)
- Easy to implement

Disadvantages of K-Means:

- Must choose *K* manually
- Depends on initial centroid locations
- Potential to misrepresent centroid positions due to outliers

KModes clustering :

It is basically a collection of objects based on similarity and dissimilarity between them. **KModes clustering** is one of the unsupervised machine learning algorithmsthat is used to cluster **categorical variables.**

How Does the KModes Algorithm Work?

Unlike Hierarchial clustering methods, we need to upfront specify the K.

1. Pick K observations at random and use them as leaders/clusters
2. Calculate the dissimilarities and assign each observation to its closest cluster
3. Define new modes for the clusters
4. Repeat 2–3 steps until there are is no re-assignment required

I hope you got the basic idea of the KModes algorithm by now. So let us quickly take an example to illustrate the working step by step.

Example: Imagine we have a dataset that has the information about hair color, eye color, and skin color of persons. We aim to group them based on the available information(maybe we want to suggest some styling ideas)

Hair color, eye color, and skin color are all categorical variables. Below is how our dataset looks like.

person	hair color	eye color	skin color
P1	blonde	amber	fair
P2	brunette	gray	brown
P3	red	green	brown
P4	black	hazel	brown
P5	brunette	amber	fair
P6	black	gray	brown
P7	red	green	fair
P8	black	hazel	fair

Alright, we have the sample data now. Let us proceed by defining the number of clusters(K)=3

Step 1: Pick K observations at random and use them as leaders/clusters

I am choosing P1, P7, P8 as leaders/clusters

Leaders			
P1	blonde	amber	fair
P7	red	green	fair
P8	black	hazel	fair

person	hair color	eye color	skin color
P1	blonde	amber	fair
P2	brunette	gray	brown
P3	red	green	brown
P4	black	hazel	brown
P5	brunette	amber	fair
P6	black	gray	brown
P7	red	green	fair
P8	black	hazel	fair

Leaders and Observations

Step 2: Calculate the dissimilarities(no. of mismatches) and assign each observation to its closest cluster

Iteratively compare the cluster data points to each of the observations. Similar data points give 0, dissimilar data points give 1.

Leaders			
P1	blonde	amber	fair
P7	red	green	fair
P8	black	hazel	fair

person	hair color	eye color	skin color
P1	blonde	amber	fair
P2	brunette	gray	brown
P3	red	green	brown
P4	black	hazel	brown
P5	brunette	amber	fair
P6	black	gray	brown
P7	red	green	fair
P8	black	hazel	fair

Comparing leader/Cluster P1 to the observation P1 gives 0 dissimilarities.

Leaders			
P1	blonde	amber	fair
P7	red	green	fair
P8	black	hazel	fair

person	hair color	eye color	skin color
P1	blonde	amber	fair
P2	brunette	gray	brown
P3	red	green	brown
P4	black	hazel	brown
P5	brunette	amber	fair
P6	black	gray	brown
P7	red	green	fair
P8	black	hazel	fair

Comparing leader/cluster P1 to the observation P2 gives 3(1+1+1) dissimilarities.

Likewise, calculate all the dissimilarities and put them in a matrix as shown below and assign the observations to their closest cluster(cluster that has the least dissimilarity)

	Cluster 1 (P1)	Cluster 2 (P7)	Cluster 3 (P8)	Cluster
P1	0 ✓	2	2	Cluster 1
P2	3 ✓	3	3	Cluster 1
P3	3	1 ✓	3	Cluster 2
P4	3	3	1 ✓	Cluster 3
P5	1 ✓	2	2	Cluster 1
P6	3	3	2 ✓	Cluster 3
P7	2	0 ✓	2	Cluster 2
P8	2	2	0 ✓	Cluster 3

Dissimilarity matrix

After step 2, the observations P1, P2, P5 are assigned to cluster 1; P3, P7 are assigned to Cluster 2; and P4, P6, P8 are assigned to cluster 3.

Note: If all the clusters have the same dissimilarity with an observation, assign to any cluster randomly. In our case, the observation P2 has 3 dissimilarities with all the leaders. I randomly assigned it to Cluster 1.

Step 3: Define new modes for the clusters

Mode is simply the **most observed value**.

Mark the observations according to the cluster they belong to. Observations of Cluster 1 are marked in Yellow, Cluster 2 are marked in Brick red, and Cluster 3 are marked in Purple.

person	hair color	eye color	skin color
P1	blonde	amber	fair
P2	brunette	gray	brown
P3	red	green	brown
P4	black	hazel	brown
P5	brunette	amber	fair
P6	black	gray	brown
P7	red	green	fair
P8	black	hazel	fair

Looking for Modes (Image by author)

Considering one cluster at a time, for each feature, look for the Mode and update the new leaders.

Explanation: Cluster 1 observations(P1, P2, P5) has brunette as the most observed hair color, amber as the most observed eye color, and fair as the most observed skin color.

Note: If you observe the same occurrence of values, take the mode randomly. In our case, the observations of Cluster 3(P3, P7) have one occurrence of brown, fair skin color. I randomly chose brown as the mode.

Below are our new leaders after the update.

New Leaders			
	hair color	eye color	skin color
Cluster 1	brunette	amber	fair
Cluster 2	red	green	fair
Cluster 3	black	hazel	brown

Obtained new leaders

Repeat steps 2–4

After obtaining the new leaders, again calculate the dissimilarities between the observations and the newly obtained leaders.

New Leaders			
	hair color	**eye color**	**skin color**
Cluster 1	brunette	amber	fair
Cluster 2	red	green	fair
Cluster 3	black	hazel	brown

person	**hair color**	**eye color**	**skin color**
P1	blonde	amber	fair
P2	brunette	gray	brown
P3	red	green	brown
P4	black	hazel	brown
P5	brunette	amber	fair
P6	black	gray	brown
P7	red	green	fair
P8	black	hazel	fair

Comparing Cluster 1 to the observation P2 gives 2 dissimilarities.

Likewise, calculate all the dissimilarities and put them in a matrix. Assign each observation to its closest cluster.

	Cluster 1	**Cluster 2**	**Cluster 3**	**Cluster**
P1	1 ✓	2	3	Cluster 1
P2	2 ✓	3	2	Cluster 1
P3	3	1 ✓	2	Cluster 2
P4	3	3	0 ✓	Cluster 3
P5	0 ✓	2	3	Cluster 1
P6	3	3	1 ✓	Cluster 3
P7	2	0 ✓	3	Cluster 2
P8	2	2	1 ✓	Cluster 3

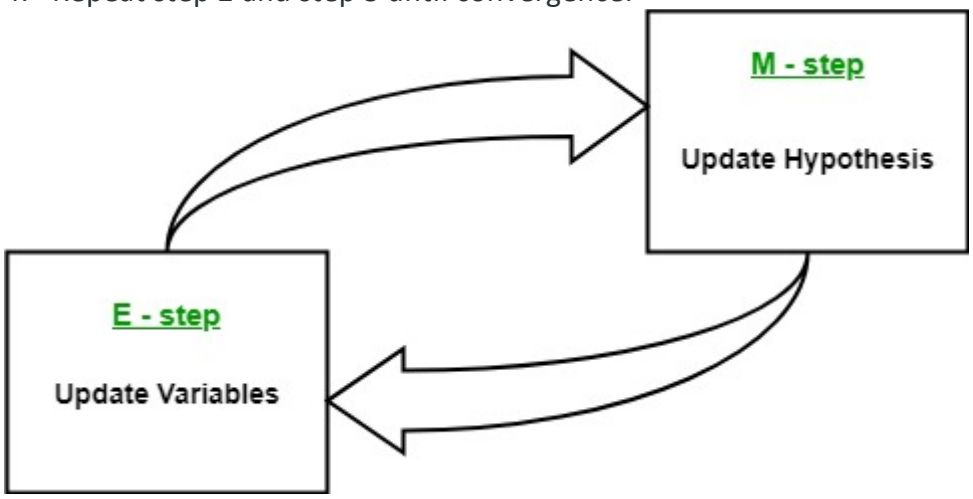
Expectation -maximization algorithm:

In the real-world applications of machine learning, it is very common that there are many relevant features available for learning but only a small subset of them are observable. So, for the variables which are sometimes observable and sometimes not, then we can use the instances when that variable is visible is observed for the purpose of learning and then predict its value in the instances when it is not observable.

On the other hand, ***Expectation-Maximization algorithm*** can be used for the latent variables (variables that are not directly observable and are actually inferred from the values of the other observed variables) too in order to predict their values with the condition that the general form of probability distribution governing those latent variables is known to us. This algorithm is actually at the base of many unsupervised clustering algorithms in the field of machine learning.
It was explained, proposed and given its name in a paper published in 1977 by Arthur Dempster, Nan Laird, and Donald Rubin. It is used to find the *local maximum likelihood parameters* of a statistical model in the cases where latent variables are involved and the data is missing or incomplete.

Algorithm:

1. Given a set of incomplete data, consider a set of starting parameters.
2. **Expectation step (E – step):** Using the observed available data of the dataset, estimate (guess) the values of the missing data.
3. **Maximization step (M – step):** Complete data generated after the expectation (E) step is used in order to update the parameters.
4. Repeat step 2 and step 3 until convergence.

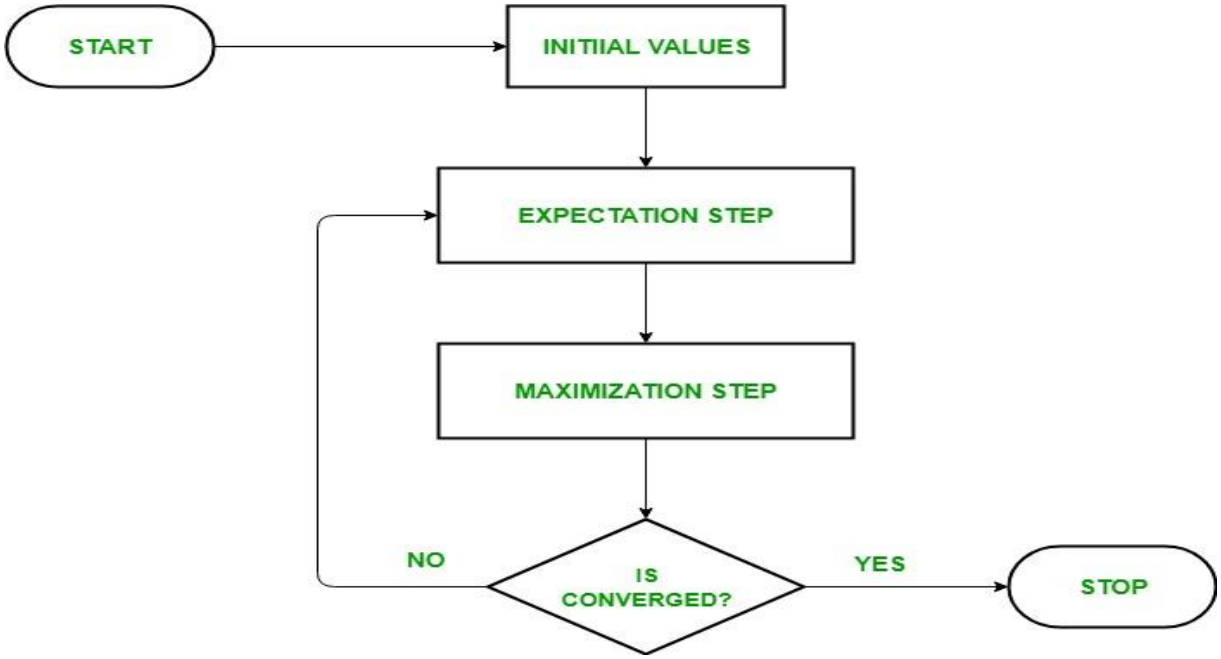


The essence of Expectation-Maximization algorithm is to use the available observed data of the dataset to estimate the missing data and then using that data to update the values of the parameters. Let us understand the EM algorithm in detail.

- Initially, a set of initial values of the parameters are considered. A set of incomplete observed data is given to the system with the assumption that the observed data comes from a specific model.
- The next step is known as “Expectation” – step or *E-step*. In this step, we use the observed data in order to estimate or guess the values of the missing or incomplete data. It is basically used to update the variables.

- The next step is known as “Maximization”-step or *M-step*. In this step, we use the complete data generated in the preceding “Expectation” – step in order to update the values of the parameters. It is basically used to update the hypothesis.
- Now, in the fourth step, it is checked whether the values are converging or not, if yes, then stop otherwise repeat *step-2* and *step-3* i.e. “Expectation” – step and “Maximization” – step until the convergence occurs.

Flow chart for EM algorithm –



Usage of EM algorithm –

- It can be used to fill the missing data in a sample.
- It can be used as the basis of unsupervised learning of clusters.
- It can be used for the purpose of estimating the parameters of Hidden Markov Model (HMM).
- It can be used for discovering the values of latent variables.

Advantages of EM algorithm –

- It is always guaranteed that likelihood will increase with each iteration.
- The E-step and M-step are often pretty easy for many problems in terms of implementation.
- Solutions to the M-steps often exist in the closed form.

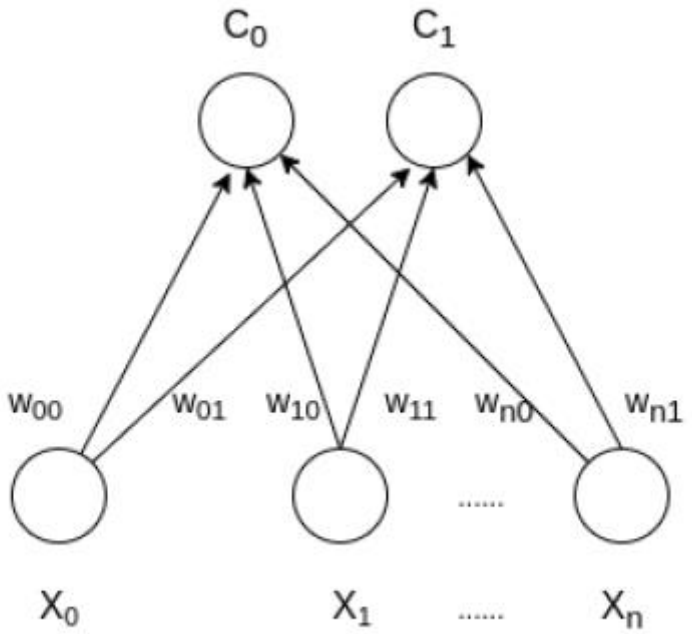
Disadvantages of EM algorithm –

- It has slow convergence.
- It makes convergence to the local optima only.
- It requires both the probabilities, forward and backward (numerical optimization requires only forward probability).

Self Organizing Maps – Kohonen Maps:

Self Organizing Map (or Kohonen Map or SOM) is a type of Artificial Neural Network which is also inspired by biological models of neural systems from the 1970s. It follows an unsupervised learning approach and trained its network through a competitive learning algorithm. SOM is used for clustering and mapping (or dimensionality reduction) techniques to map multidimensional data onto lower-dimensional which allows people to reduce

complex problems for easy interpretation. SOM has two layers, one is the Input layer and the other one is the Output layer.
The architecture of the Self Organizing Map with two clusters and n input features of any sample is given below:



How do SOM works?
Let's say an input data of size (m, n) where m is the number of training examples and n is the number of features in each example. First, it initializes the weights of size (n, C) where C is the number of clusters. Then iterating over the input data, for each training example, it updates the winning vector (weight vector with the shortest distance (e.g Euclidean distance) from training example). Weight updation rule is given by :

$w_{ij} = w_{ij}(old) + alpha(t) * (x_i^k - w_{ij}(old))$
where alpha is a learning rate at time t, j denotes the winning vector, i denotes the i^{th} feature of training example and k denotes the k^{th} training example from the input data. After training the SOM network, trained weights are used for clustering new examples. A new example falls in the cluster of winning vectors.

Gaussian Mixture Models

The Gaussian Mixture Model is an Expectation-Maximization (EM) algorithm with data points that are assumed to have a Gaussian (Normal) distribution. It is commonly described as a more sophisticated version of K-Means. It requires two parameters, the mean and the covariance, to describe the position and shape of each cluster.

The model is based on the multivariate Gaussian distribution which is:

$$N(x : \mu, \Sigma) = \frac{1}{(2\pi\Sigma)^{1/2}} exp\left(-\frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu)\right)$$

where for each cluster,

x = vector of a data point
μ = vector of the mean of each cluster
Σ = covariance matrix between each dimension

GMM requires a five step process:

1. Choose the number of clusters, *K*
2. Randomly establish the initial parameters for each centroid (ie the mean, covariance and weighting)
3. Calculate the likelihood that the data set has come from this set of clusters with the current parameters
4. Maximize the likelihood by changing the parameters (cluster proportions, centers and spread around the centers)
5. Return to step 3 and repeat until the change in the parameters is less than a pre-programmed amount or a fixed number of iterations have been completed.

What is the Expectation-Maximization algorithm?

EM starts by calculating the likelihood that a given data point comes from the mixture of Gaussian distributions with the current set of parameters. It then maximizes the likelihood by changing the parameters.

What is the likelihood and how is it maximized?

The likelihood function is based on the probability that a data point comes from the weighted collection of Gaussian clusters:

$$P(x_i) = \sum_{k=1}^{K} \pi_k N(x_i : \mu_k, \Sigma_k)$$

where

π = weight of each cluster
K = number of clusters

Of course, the weights will sum to 1.

The likelihood is then the probability of the whole data set of *N* data points coming from the set of clusters with these parameters.

$$L = \prod_{i=1}^{N} P(x_i)$$

Expectation: without assigning a data point to an individual cluster, we calculate the probability that the parameters of each cluster will generate each data point. This means re-calculating for each iteration a new value of *P(x)* as given above.

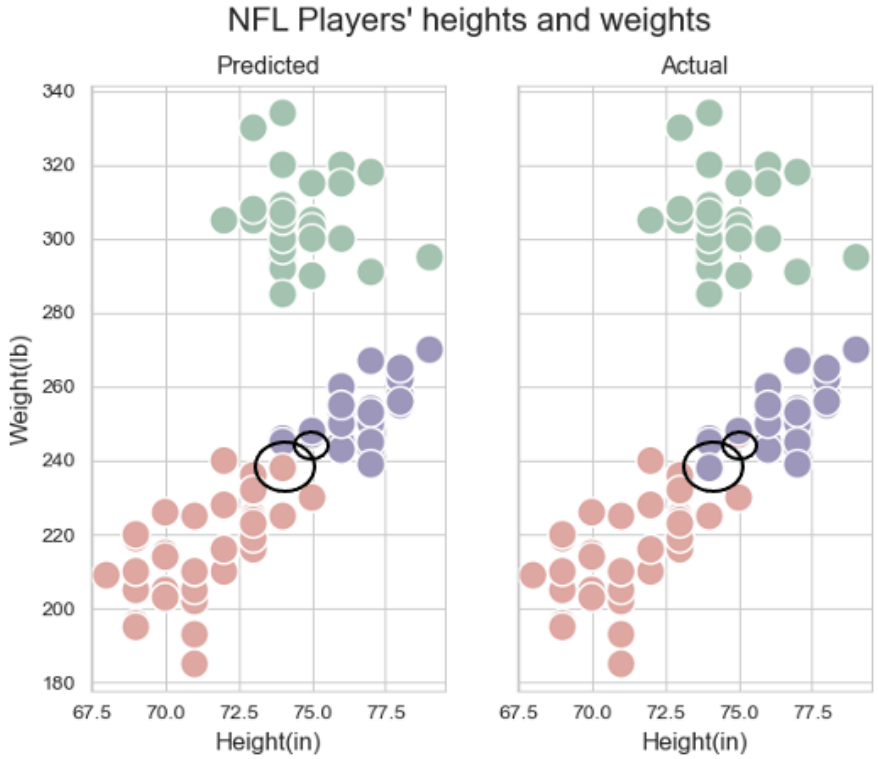
Maximization: we need to update the weight, mean and covariance for each cluster respectively, using the following formulae:

$$\hat{\pi}_k = \frac{1}{N} \sum_{i=1}^{N} \frac{\pi_k N(x_i : \mu_k, \Sigma_k)}{P(x_i)}$$

$$\hat{\mu}_k = \frac{1}{N} \sum_{i=1}^{N} \frac{x_i N(x_i : \mu_k, \Sigma_k)}{P(x_i)}$$

$$\hat{\Sigma}_k = \frac{1}{N} \sum_{i=1}^{N} \frac{(x_i - \mu_k)(x_i - \mu_k)^T N(x_i : \mu_k, \Sigma_k)}{P(x_i)}$$

Running the algorithm on our data set gave slightly better results than K-Means.



GMM successfully clustered the data into three groups; however, it mis-assigned one RB and one TE.

Results: All DTs were again clustered into one group successfully while Derrick Henry was again misclassified as a Tight End and a Tight End was misclassified as a Running Back.

Advantages of GMM:

- can analyze more complex and mixed data
- can handle outliers more easily

Disadvantages of GMM:

- more difficult to directly interpret results
- does not directly assign data points to clusters