Unsupervised Learning

In supervised machine learning models are trained using labeled data under the supervision of training data. But there may be many cases in which we do not have labeled data and need to find the hidden patterns from the given dataset. So, to solve such types of cases in machine learning, we need unsupervised learning techniques.

Definition :

Unsupervised learning is a type of machine learning in which models are trained using an unlabeled dataset and are allowed to act on that data without any supervision.

Goal :

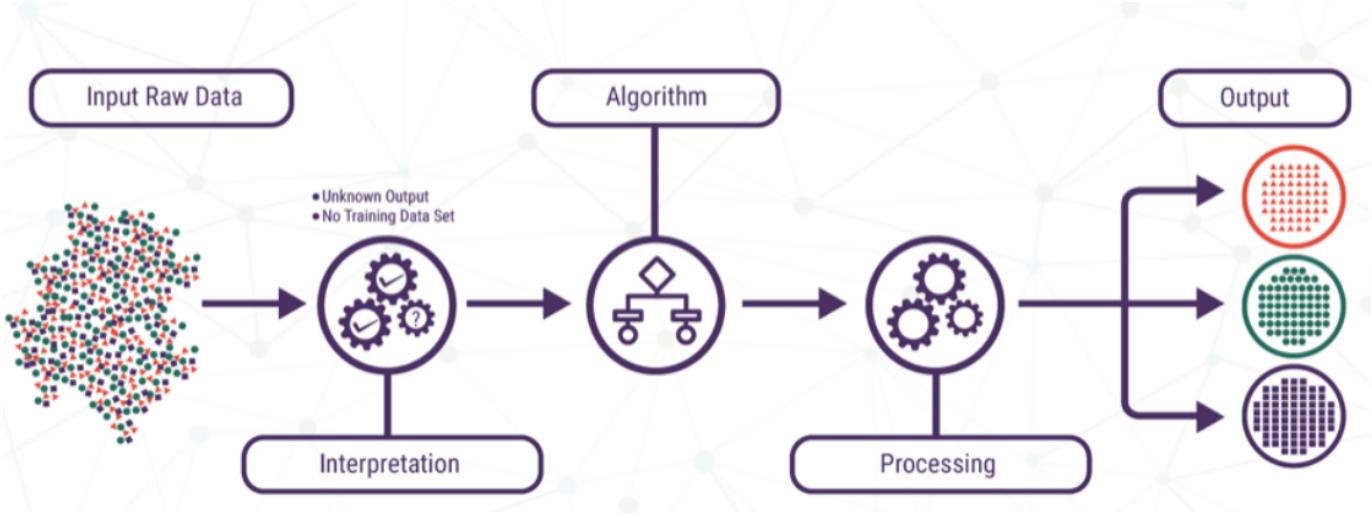
To find the underlying structure of the dataset,
group that data according to similarities,
and represent that dataset in a compressed format.

Why :

- Unsupervised learning is helpful for finding useful insights from the data.
- It is much similar as a human learns to think by their own experiences, which makes it closer to the real AI.

- It works on unlabeled and uncategorized data which make unsupervised learning more important.
- In real-world, we do not always have input data with the corresponding output so to solve such cases, we need unsupervised learning.

Working :



Pros :

- Use **for more complex tasks** as compared to supervised learning because, in unsupervised learning, we don't have labeled input data.
- It is preferable as it is **easy to get unlabeled data** in comparison to labeled data.

Cons :

- It is intrinsically **more difficult** than supervised learning as it does not have corresponding output.
- The result of the unsupervised learning algorithm might be **less accurate** as input data is not labeled, and algorithms **do not know the exact output** in advance.

Clustering

What is clustering ?

A way of grouping the data points into different clusters, consisting of similar data points. The objects with the possible similarities remain in a group that has less or no similarities with another group.

Clustering Analysis (“data segmentation”) is an exploratory method for identifying homogeneous groups (“clusters”) of records
Similar records should belong to the same cluster
Dissimilar records should belong to different clusters

Clustering = Grouping “**Similar**” things together !

Why clustering ?

To **understand/discover** structure in data
Summarize data point by their “**Cluster center**”

Business Questions

Types of Pages are there on the Web ?
Types of Customers are there in my Market ?
Types of People are there on Social Network ?
Types of Email in my Index ?
Types of Genes the Human genome has ?

Uses :

- Market Segmentation
- Statistical data analysis
- Social network analysis
- Image segmentation
- Recommendation engine
- Anomaly detection

Methods :

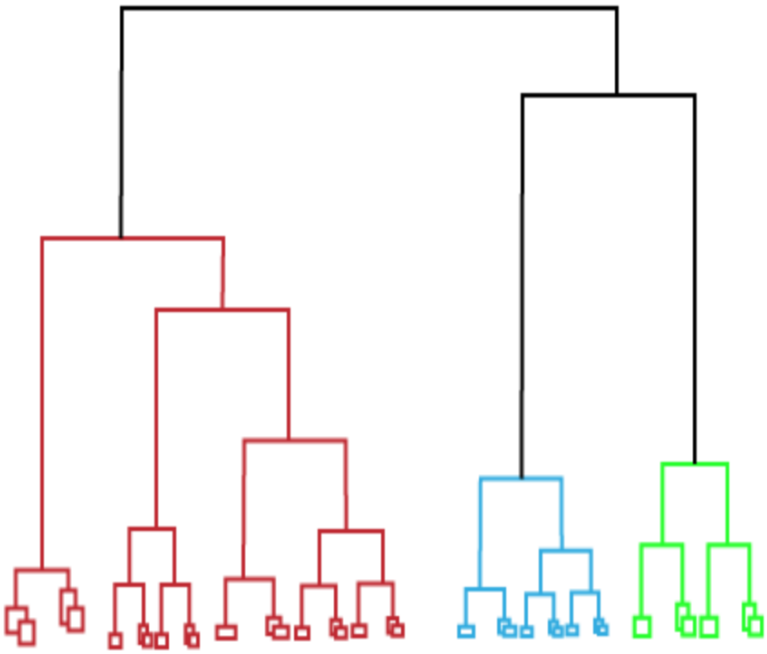
1. **Hierarchical Clustering (Agglomerative)**
2. **Partitioning Clustering (K-Means Clustering)**
3. **Density-Based Clustering (DBSCAN)**
4. **Distribution Model-Based Clustering (EM Clustering)**
5. **Fuzzy Clustering (Fuzzy C-means)**

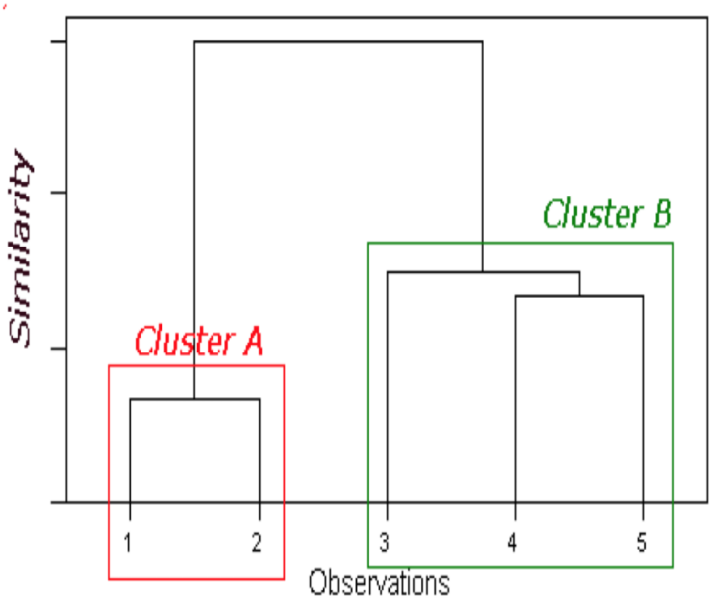
Hierarchical Clustering (Agglomerative Clustering) Bottom-Up

Merge similar points together

The Agglomerative hierarchical algorithm performs the bottom-up hierarchical clustering.

In this, each data point is treated as a single cluster at the outset and then successively merged(**distance**). The cluster hierarchy can be represented as a tree-structure.





The dataset is divided into clusters to create a tree-like structure, which is called a **dendrogram**
In the dendrogram plot,
The *y-axis* shows the *Euclidean distances* between the data points,
and the *x-axis* shows *all the data points* of the given dataset

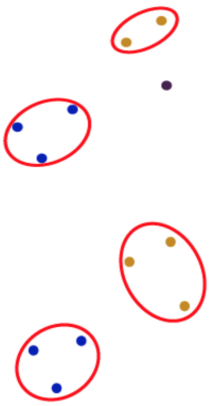
Working :

Step-1: Create each data point as a single cluster. Let's say there are N data points, so the number of clusters will also be N.

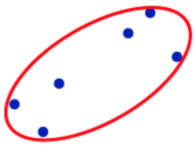
Step-2: Take two closest data points or clusters and merge them to form one cluster. So, there will now be N-1 clusters.

Step-3: Again, take the two closest clusters and merge them together to form one cluster. There will be N-2 clusters.

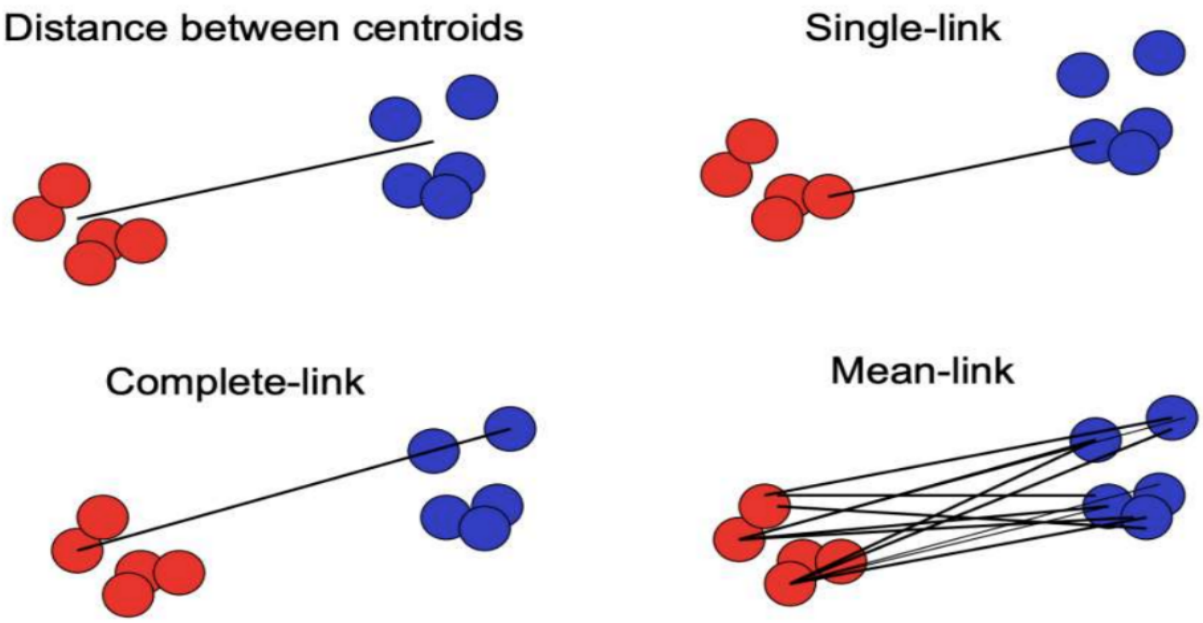
Step-4: Repeat Step 3 until only one cluster left. So, we will get the following clusters. Consider the following images:



Step-5: Once all the clusters are combined into one big cluster, develop the dendrogram to divide the clusters as per the problem.



All Linkage methods : (Measure for the distance between two clusters)



Centroid Linkage: It is the distance between the centroid of the clusters is calculated.

Single Linkage: It is the Shortest Distance between the closest points of the clusters.

Complete Linkage: It is the farthest distance between the two points of two different clusters.

Average Linkage(mean): The distance between each pair of datasets is added up and then divided by the total number of datasets to calculate the average distance between two clusters.

Drawbacks :

- ➢ It isn't suitable for big dataset, has high computational complexity.
- ➢ Need metric for merging the clusters(Linkage) that affects the clustering result.
- ➢ Sensitive to noise.

Note : we can see Divisive Clustering here

Code :

```
## Perform Clustering for the crime data and identify the number of clusters formed and draw inferences.
# Import the libraries
        import pandas as pd
        import numpy as np
        import matplotlib.pyplot as plt
        import seaborn as sns
        import scipy.cluster.hierarchy as sch
        from sklearn.cluster import AgglomerativeClustering
        from sklearn.preprocessing import MinMaxScaler
# Import the data set
        df=pd.read_csv("D:\\DATA SCIENCE\\Clustering\\crime_data.csv")
        df.head()
        df.info()
        df.describe()
## Using Hierarchical clustering
        scaler=MinMaxScaler()
        df_norm=scaler.fit_transform(df.iloc[:,1:])
        df_norm=pd.DataFrame(df_norm,columns=df.iloc[:,1:].columns)
        df_norm.head()
#create dendrogram
        dend=sch.dendrogram(sch.linkage(df_norm,method='average'))
        dend=sch.dendrogram(sch.linkage(df_norm,method='single'))
        dend=sch.dendrogram(sch.linkage(df_norm,method='complete'))
        dend=sch.dendrogram(sch.linkage(df_norm,method='centroid'))
#create cluster  (k is subjective)
        hc1=AgglomerativeClustering(n_clusters=3,affinity='euclidean',linkage='complete')
        #choosing no. of clusters depends upon domain
        y_hc1=hc1.fit_predict(df_norm)
#cluster1
        cluster1=pd.DataFrame(y_hc1,columns=['cluster1'])
        df_norm['h_clusterid1']=hc1.labels_
        df_norm.head()
        df_norm.groupby(['h_clusterid1']).count()
#create cluster
        hc2=AgglomerativeClustering(n_clusters=3,affinity='manhattan',linkage='complete')
        #choosing no. of clusters depends upon domain
        y_hc2=hc2.fit_predict(df_norm)
#cluster2
        cluster2=pd.DataFrame(y_hc2,columns=['cluster2'])
```

```
        df_norm['h_clusterid2']=hc2.labels_
        df_norm.head()
        df_norm.groupby(['h_clusterid2']).count()
#create cluster
        hc3=AgglomerativeClustering(n_clusters=3,affinity='minkowski',linkage='complete')
        #choosing no. of clusters depends upon domain
        y_hc3=hc3.fit_predict(df_norm)

        df_norm['h_clusterid3']=hc3.labels_
        df_norm.head()
        df_norm.groupby(['h_clusterid3']).count()
#create cluster
        hc4=AgglomerativeClustering(n_clusters=3,affinity='hamming',linkage='complete')
        #choosing no. of clusters depends upon domain
        y_hc4=hc4.fit_predict(df_norm)
        df_norm['h_clusterid4']=hc3.labels_
        df_norm.head()
        df_norm.groupby(['h_clusterid4']).count()
```

Partitioning Clustering (K-Means Clustering) Non-hierarchical

It is a type of clustering that divides the data into non-hierarchical groups. It is also known as the **centroid-based method**. The most common example of partitioning clustering is the **K-Mean Clustering Algorithm.** K-Means Clustering is an unsupervised learning algorithm that is used to solve the clustering problems in machine learning or data science.

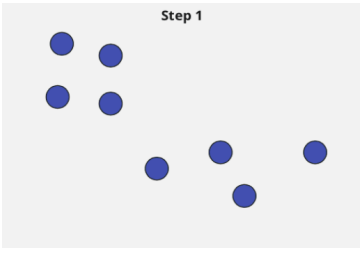
What is K-Means Algorithm?

K-Means Clustering is an Unsupervised Learning algorithm, which groups the unlabeled dataset into different clusters. Here K defines the number of predefined clusters that need to be created in the process, as if K=2, there will be two clusters, and for K=3, there will be three clusters, and so on.

Definition :

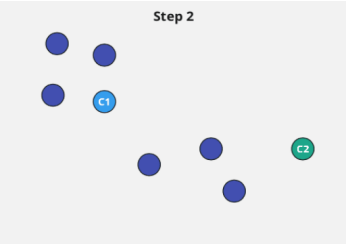
It is an iterative algorithm that divides the unlabeled dataset into k different clusters in such a way that each dataset belongs to only one group that has similar properties.

Working :

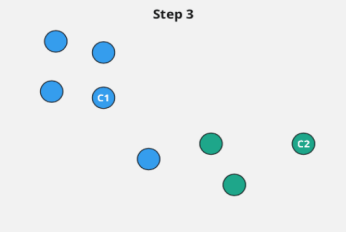


Step 1 Here we are having a few data points, which we want to cluster. So we would start by picking the number of clusters we want to have for this case. Let us select 2 for this instance. And then randomly selecting a point considering it to be the centroid of the cluster.

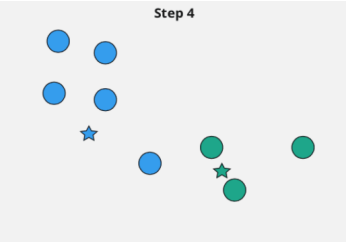
Step 2 We have successfully marked the centers of these clusters. Now we will be marking all the points with respective colors on the basis of the distance they have from the centroid.



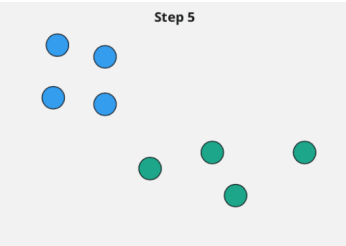
Step 3 After marking all the data points, we will now be computing the centroid of this cluster again. We are doing it because initially, we had picked the centroid randomly. Then to remove errors, if any, we are doing it. The centroid of the cluster is computed by finding a point within the cluster that would be equidistant from all the data points.



Step 4 Now since we have computed the centroid again and we know it is not the same as it was before so we would iterate the process again and would find the points nearest to this centroid for each cluster.

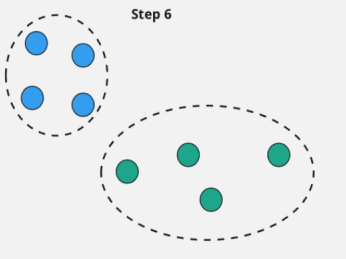


Step 5 Now we have got the result again. One may ask when shall we stop the iteration of this finding the centroid and then placing the data points accordingly? Well, you have to do it till the time when the position of the centroids don't change.



Step 6 We marked the two clusters.

In this case, it was easy, so we were able to get the results in 2 iterations only.



How To Evaluate Clusters

Two criteria for evaluating clusters.

1. Inertia
2. Dunn Index

Inertia : It calculates the sum of distances of all the entities present in the cluster.

Dunn Index : It is the ratio of the minimum inter cluster distance to the maximum of intra cluster distance. So more will be the value of the dunn index better would be the clusters in terms of being separable.

Note :

1. **Intra** cluster distance is handled by inertia, and that is the distance between the data points which are inside one cluster.
2. **Inter** cluster distance means the distance between 2 different clusters.

K-Means++

This algorithm ensures a smarter initialization of the centroids and improves the quality of the clustering. Apart from initialization, the rest of the algorithm is the same as the standard K-means algorithm. That is, K-means++ is the standard K-means algorithm coupled with a smarter initialization of the centroids.

K-Mean	K-Mean++
Takes less time to implement. Randomly chooses two centroids.	Takes more time to implement. Choose one centroid at random and another on the basis of the square of the distance from the first one.

How to identify the best “K” ?

1. Elbow Method
2. Silhouette Method

Elbow Method : It is to calculate the sum of the distances from data points to centroids and aims at minimizing the sum to an optimal value.

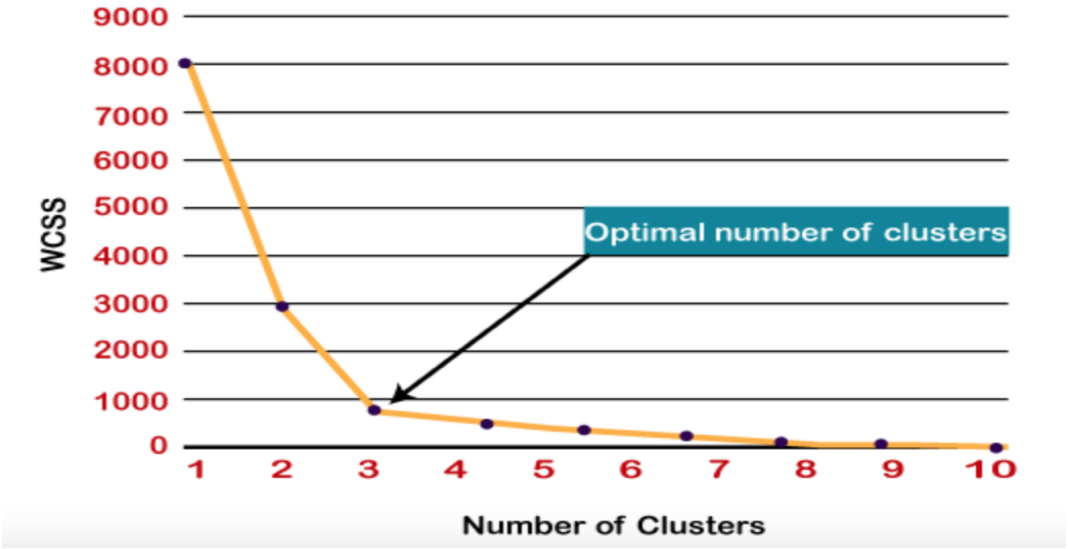
Silhouette Method : The silhouette value measures how similar a point is to its own cluster (cohesion) compared to other clusters (separation).

Elbow Method

The Elbow method is one of the most popular ways to find the optimal number of clusters. This method uses the concept of WCSS value. **WCSS** stands for **Within Cluster Sum of Squares**, which defines the total variations within a cluster. $\text{WCSS}= \sum_{P_i \text{ in Cluster}_1} \text{distance}(P_i\ C_1)^2 + \sum_{P_i \text{ in Cluster}_2} \text{distance}(P_i\ C_2)^2 + \sum_{P_i \text{ in CLuster}_3} \text{distance}(P_i\ C_3)^2$

To find the optimal value of clusters, the elbow method follows the below steps:

- It executes the K-means clustering on a given dataset for different K values (ranges from 1-10).
- For each value of K, calculate the WCSS value.
- Plots a curve between calculated WCSS values and the number of clusters K.
- The sharp point of bend or a point of the plot looks like an arm, then that point is considered as the best value of K.



Since the graph shows the sharp bend, which looks like an elbow, hence it is known as the elbow method.

Note : If we choose the number of clusters equal to the data points, then the value of WCSS becomes zero, and that will be the endpoint of the plot.

Drawbacks :

- ➢ K-Mean Algorithm required to specify the no. of clusters a priory(the K value).
- ➢ Sensitivity to Noise.
- ➢ K-Mean does not perform well on finding Non-convex/Non-Spherical shapes of the clusters.

Code :

```
# Import the Libraries
        import numpy as np
        import pandas as pd
        import matplotlib.pyplot as plt
        from sklearn.cluster import KMeans
        from sklearn.preprocessing import StandardScaler
# Import data set
        univ=pd.read_csv("Universities.csv")
        univ.head()
# standardized the data
        scaler=StandardScaler()
        std_univ_df=scaler.fit_transform(univ.iloc[:,1:])
# how to find the no. of optimal clusters
# the k-mean algorithm aims to choose centroids that minimize the inertia, or within-cluster
sum-of-square criterion:
        wcss=[]
        for i in range(1,11):
           kmeans=KMeans(n_clusters=i,random_state=0)
           kmeans.fit(std_univ_df)
           wcss.append(kmeans.inertia_)
        wcss
        plt.plot(range(1,11),wcss)
        plt.title("Elbow Method");plt.xlabel("Number of clusters");plt.ylabel("WCSS")
# Build cluster algorithm
        from sklearn.cluster import KMeans
        clusters_new=KMeans(4,random_state=42)
        clusters_new.fit(std_univ_df)
        clusters_new.labels_
# Assigning clusters to the data set
        univ['clusterid-new']=clusters_new.labels_
        univ.head()
        clusters_new.cluster_centers_   # These are the standardized values
univ.groupby(['clusterid-new']).agg(['mean'])
```

Density-Based Clustering (DBSCAN)

Density-Based Clustering refers to unsupervised learning methods that identify distinctive groups/clusters in the data, based on the idea that a cluster in a data space is a contiguous region of high point density, separated from other such clusters by contiguous regions of low point density.
K-Means (distance between points),
Affinity propagation (graph distance),
Mean-shift (distance between points),
DBSCAN (distance between nearest points),
Gaussian mixtures (Mahalanobis distance to centers),
Spectral clustering (graph distance) etc.

All clustering methods use the same approach i.e. first we calculate similarities and then we use it to cluster the data points into groups or batches. Here we will focus on **Density-based spatial clustering of applications with noise** (DBSCAN) clustering methods.

The ***DBSCAN algorithm*** is based on this intuitive notion of "clusters" and "noise". The key idea is that for each point of a cluster, the neighborhood of a given radius has to contain at least a minimum number of points.

Why DBSCAN ?

Partitioning methods (K-means) and hierarchical clustering are suitable only for compact and well-separated clusters. Moreover, they are also severely affected by the presence of noise and outliers in the data.

Real life data may contain irregularities, like:

- Clusters can be of
  arbitrary shape such as those shown in the figure above.
- Data may contain noise.

Parameters of DBSCAN

Eps(ε) : It defines the neighborhood around a data point i.e. if the distance between two points is lower or equal to 'eps' then they are considered as neighbors. If the eps value is chosen too small then large parts of the data will be considered as outliers. If it is chosen very large then the clusters will merge and the majority of the data points will be in the same clusters. One way to find the eps value is based on the ***k-distance graph***.

MinPts(n): Minimum number of neighbors (data points) within eps radius. Larger the dataset, the larger value of MinPts must be chosen. As a general rule, the minimum MinPts can be derived from the number of dimensions D in the dataset as, MinPts >= D+1. The minimum value of MinPts must be chosen at least 3.

Three types of data points :

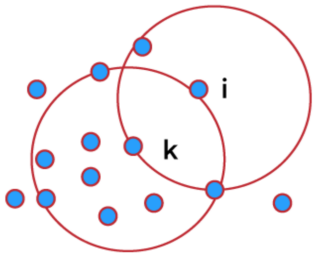
Core Point : A point is a core point if it has more than MinPts points within eps.

Border Point : A point which has fewer than MinPts within eps but it is in the neighborhood of a core point.

Noise or outlier: A point which is not a core point or border point.

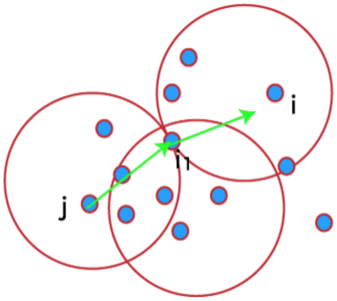
Directly density reachable :

A point i is considered as the directly density reachable from a point k with respect to Eps, MinPts if i belongs to NEps(k)



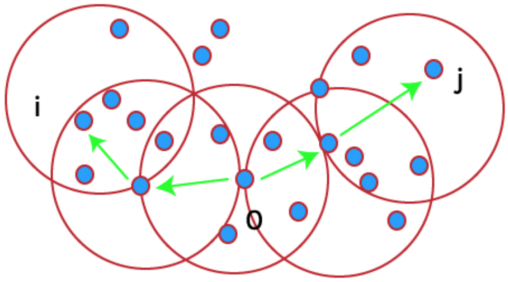
Density reachable :

A point denoted by i is a density reachable from a point j with respect to Eps, MinPts if there is a sequence chain of a point i1,...., in, i1 = j, pn = i such that i_i + 1 is directly density reachable from i_i.

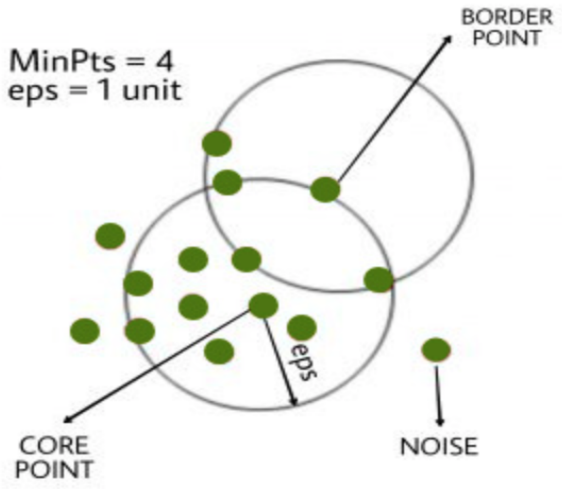


Density connected :

A point i refers to density connected to a point j with respect to Eps, MinPts if there is a point o such that both i and j are considered as density reachable from o with respect to Eps and MinPts.



Working :



Step 1: Find all the neighbor points within eps and identify the core points or visited with more than MinPts neighbors.

Step 2: For each core point if it is not already assigned to a cluster, create a new cluster.

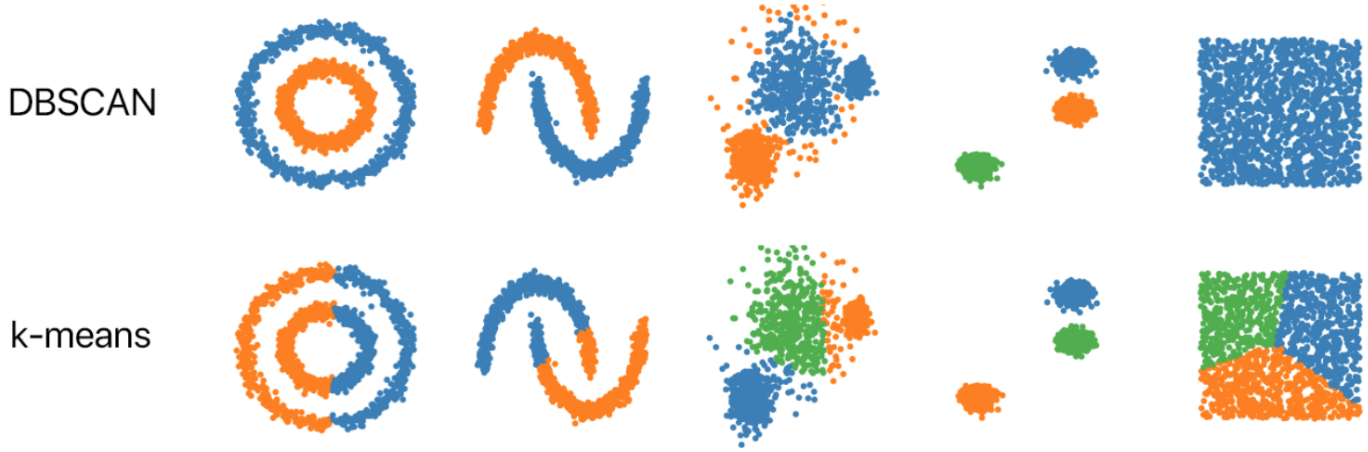
Step 3: Find recursively all its density connected points and assign them to the same cluster as the core point.

Step 4: Iterate through the remaining unvisited points in the dataset. Those points that do not belong to any cluster are noise.

Metrics for measuring DBSCAN's Performance :

Silhouette Score : The silhouette score is calculated utilizing the mean intra- cluster distance between points, AND the mean nearest-cluster distance. For instance, a cluster with a lot of data points very close to each other (high density) AND is far away from the next nearest cluster (suggesting the cluster is very unique in comparison to the next closest cluster), will have a strong silhouette score. A silhouette score ranges from -1 to 1, with -1 being the worst score possible and 1 being the best score. Silhouette scores of 0 suggest overlapping clusters.

K-means Clustering	**DBSCAN**
Distance based clustering	Density based clustering
Every observation becomes a part of some cluster eventually	Clearly separates outliers and clusters observations in high density areas
Build clusters that have a shape of a hypersphere	Build clusters that have an arbitrary shape or clusters within clusters.
Sensitive to outliers	Robust to outliers
Require no. of clusters as input	Doesn't require no. of clusters as input



DBSCAN Pros :

- Identifies randomly shaped clusters
- doesn't necessitate to know the number of clusters in the data previously (as opposed to K-means)
- Handles noise

DBSCAN Cons :

- Datasets with varying densities are problematic
- Input parameters (x and MinPts) may be difficult to determine
- Computational complexity — when the dimensionality is high, it takes O(n2)

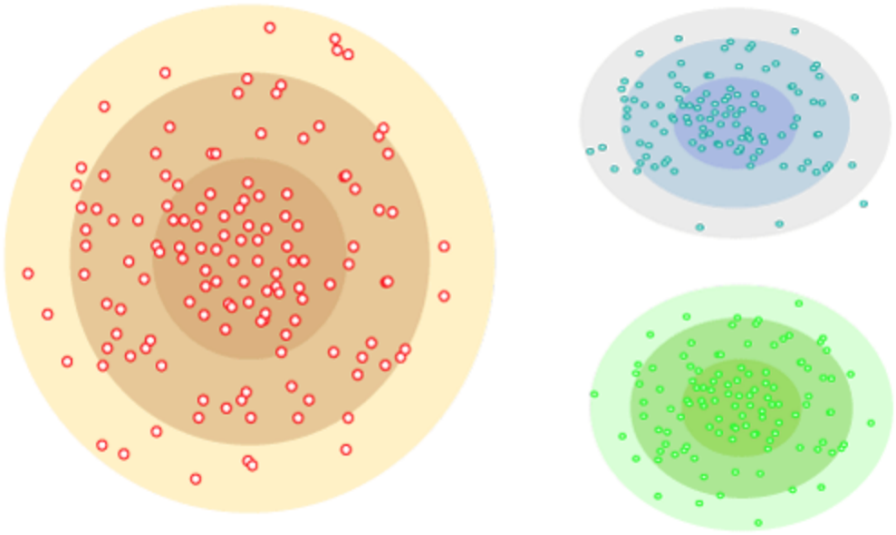
Code :

```
#Import the libraries
        from sklearn.cluster import DBSCAN
        from sklearn.preprocessing import StandardScaler
        import numpy as np
        import pandas as pd
        import matplotlib.pyplot as plt
# Import .csv file and convert it to a DataFrame object
        # from google.colab import drive
        # drive.mount('/content/drive')
        df = pd.read_csv("/content/drive/MyDrive/jupyter notebook
        workspace/Clustering/DBSCAN/Wholesale customers data.csv");
        df.head()
        print(df.info())
        df.drop(['Channel','Region'],axis=1,inplace=True)
        array=df.values
        array
# data standardization
        stscaler = StandardScaler().fit(array)
        X = stscaler.transform(array)
        X
        dbscan = DBSCAN(eps=0.8, min_samples=6)
        dbscan.fit(X)
#Noisy samples are given the label -1.
        dbscan.labels_
        cl=pd.DataFrame(dbscan.labels_,columns=['cluster'])
        cl
pd.concat([df,cl],axis=1)
```

Distribution Model-Based Clustering (EMClustering algorithm)

In the distribution model-based clustering method, the data is divided based on the probability of how a dataset belongs to a particular distribution.

The grouping is done by assuming some distributions, commonly **Gaussian Distribution**.The example of this type is the **Expectation-Maximization Clustering algorithm** that uses Gaussian Mixture Models (GMM).



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.

Here ***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.

Working :

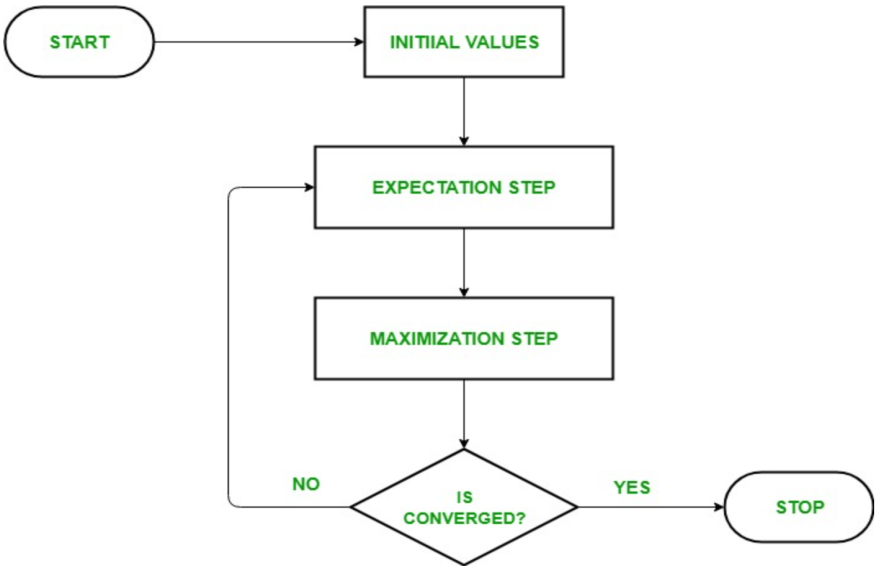
Step 1: Initially, a set of initial values of the parameters are considered. A set of incomplete observed data is given to the system with the assumption that the observed data comes from a specific model.

Step 2: 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.

Step 3: 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**.**

Step 4: 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 :



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.

Pros :

- 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.

Cons :

- 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).

Hidden Markov Models Simplified here

Fuzzy Clustering (Fuzzy C-means)

Fuzzy clustering is a type of soft method in which a data object may belong to more than one group or cluster. Each dataset has a set of membership coefficients, which depend on the degree of membership to be in a cluster. Fuzzy C-means algorithm is an example of this type of clustering; it is sometimes also known as the Fuzzy k-means algorithm.

Applications of Clustering

- **In Identification of Cancer Cells:** It divides the cancerous and non-cancerous data sets into different groups.
- **In Search Engines:** The search result appears based on the closest object to the search query. It does it by grouping similar data objects in one group that is far from the other dissimilar objects. The accurate result of a query depends on the quality of the clustering algorithm used.
- **Customer Segmentation:** It is used in market research to segment the customers based on their choice and preferences.
- **In Biology:** It is used in the biology stream to classify different species of plants and animals using the image recognition technique.
- **In Land Use:** It is used in identifying the area of similar lands used in the GIS database. This can be very useful to find what purpose the particular land should be used for.

Association Rules

Study of “what goes with what"

Also called **market basket analysis**,**affinity analysis**
Origin: Study of customer transaction databases to determine dependencies between purchases of different items

Market basket analysis :

One basket tells you about what one customer purchased at one time.
A loyalty card makes it possible to tie together purchases by a single customer (or household) over time.

What Is Association Rule Mining?

Association rule learning is a type of unsupervised learning technique that checks for the dependency of one data item on another data item and maps accordingly so that it can be more profitable. It tries to find some interesting relations or associations among the variables of dataset. It is based on different rules to discover the interesting relations between variables in the database.

The association rule learning is one of the very important concepts of machine learning and it is employed in **Market Basket analysis, Web usage mining, continuous production, etc.** Here market basket analysis is a technique used by the various big retailers to discover the associations between items. We can understand it by taking an example of a supermarket, as in a supermarket, all products that are purchased together are put together.

For example, if a customer buys bread, he most likely can also buy butter, eggs, or milk, so these products are stored within a shelf or mostly nearby.

Rule form

Antecedent → Consequent (support, confidence] And lift
(support and confidence are user defined measures of interestingness)

Examples

• buys(x, "computer") buys(x, "financial management software") [0.5%, 60%]
• age(x, "30..39") ^ income(x, "42...48K") — buys(x, "car")[1%,75%]

Association rule types

- **Actionable Rules**
  contain high-quality, actionable information
- **Trivial Rules**
  information already well-known by those familiar with the business
- **Inexplicable Rules**
  no explanation and do not suggest action

Note : Trivial and Inexplicable Rules occur most often

Ex. of Actionable Rule

1. Wal-Mart customers who purchase Barbie dolls have a 60% likelihood of also purchasing one of three types of candy bars [Forbes, Sept 8, 1997]
2. Customers who purchase maintenance agreements are very likely to purchase large appliances (Linoff and Berry experience)
1. When a new hardware store opens, one of the most commonly sold items is toilet bowl cleaners (Linoff and Berry experience)

How does Association Rule Learning work?

Association rule learning works on the concept of If and Else Statement, such as if A then B.

Here the If element is called **antecedent**, and then statement is called as **Consequent**. These types of relationships where we can find out some association or relation between two items is known *as single cardinality*. It is all about creating rules, and if the number of items increases, then cardinality also increases accordingly. So, to measure the associations between thousands of data items, there are several metrics. These metrics are given below:

- **Support**
- **Confidence**
- **Lift**

Support :

Criterion for “Support": Support of an item % (or number) of transactions in which antecedent (IF) and consequent (THEN) appear in the data

Support **=** $\#\ transactions\ with\ both\ antecedent\ \&\ consequent\ item\ sets / \#\ Total\ transactions$

Confidence :

Confidence: % of antecedent (IF) transactions that also have the consequent (THEN) item set

Confidence **=**
$\#\ transactions\ with\ both\ antecedent\ \&\ consequent\ item\ sets / \#\ transactions\ with\ antecedent\ item\ set$

Lift :

Lift ratio **=** confidence/(benchmark confidence)

Benchmark assumes independence between antecedent and consequence:
P(antecedent & consequent) = P(antecedent) x P(consequent)
Benchmark confidence: P(C|A) = P(C&A) / P(A) = P(C) x P(A) / P(A) = P(C)
Interpreting Lift
Lift > 1 indicates a rule that is useful in finding consequent items sets (i.e., more useful than selecting transactions randomly)

- If **Lift= 1**: The probability of occurrence of antecedent and consequent is independent of each other.
- **Lift>1**: It determines the degree to which the two itemsets are dependent on each other.
- **Lift<1**: It tells us that one item is a substitute for other items, which means one item has a negative effect on another.

Interpretation revisited

- Lift ratio shows how effective the rule is in finding consequents vs. random (useful if finding particular consequents is important)
- Confidence shows the rate at which consequents will be found (useful in learning costs of promotion)
- Support measures overall impact (% transactions affected)

Types of Association Rule Learning

Association rule learning can be divided into three types of algorithms:

1. **Apriori Algorithm**

This algorithm uses frequent datasets to generate association rules. It is designed to work on the databases that contain transactions. This algorithm uses a breadth-first search and Hash Tree to calculate the itemset efficiently.

For K products …

1. Set minimum support criterion
2. Generate list of one-item sets that meet the support criterion
3. Use list of one-item sets to generate list of two-item sets that meet support criterion
4. Use list of two-item sets to generate list of three-item sets that meet support criterion
5. Continue up through k-item sets

It is mainly used for market basket analysis and helps to understand the products that can be bought together. It can also be used in the healthcare field to find drug reactions for patients.

2. **Eclat Algorithm**

The Eclat algorithm stands for **Equivalence Class Transformation**. This algorithm uses a depth-first search technique to find frequent itemsets in a transaction database. It performs faster than Apriori Algorithm.

3. **F-P Growth Algorithm**

The F-P growth algorithm stands for **Frequent Pattern**, and it is the improved version of the Apriori Algorithm. It represents the database in the form of a tree structure that is known as a frequent pattern or tree. The purpose of this frequent tree is to extract the most frequent patterns.

Applications of Association Rule Learning

- **Market Basket Analysis:** It is one of the popular examples and applications of association rule mining. This technique is commonly used by big retailers to determine the association between items.
- **Medical Diagnosis:** With the help of association rules, patients can be cured easily, as it helps in identifying the probability of illness for a particular disease.

- **Protein Sequence:** The association rules help in determining the synthesis of artificial Proteins.
- It is also used for Catalog **Design** and **Loss-leader Analysis** and many more other applications.

Code :

```
# import important libraries
import pandas as pd
from mlxtend.frequent_patterns import apriori,association_rules
from mlxtend.preprocessing import TransactionEncoder

titanic=pd.read_csv("Association Rules/Titanic.csv")
titanic.head()

titanic.info()

titanic.describe()

## Pre-processing

#As the data is not in transaction formation We are using transaction Encoder
TE=TransactionEncoder()

dataset=TE.fit(titanic).transform(titanic)
dataset=dataset.astype('int')
print(dataset)

df1=pd.get_dummies(titanic) # encoding using pandas
df1.tail()

#data=pd.DataFrame(dataset,columns=df1.columns)

## Apriori algorithm

frequent_itemsets=apriori(df1,min_support=0.1,use_colnames=True)
frequent_itemsets

rules=association_rules(frequent_itemsets,metric='lift',min_threshold=0.7)
rules

#####  An leverage value of 0 indicates independence. Range will be [-1 1]
A high conviction value means that the consequent is highly depending on the antecedent and
range [0 inf]

rules.sort_values('lift',ascending = False)[0:20]

rules[rules.lift>1]
```

Recommendation System

Recommender systems can forecast user ratings, even before they have provided one, making them an effective tool. The recommendation system can wisely select which filters to apply to a particular user's specific situation. It facilitates marketers to maximize conversions and average order value.
Mainly, a recommendation system processes data through four phases as follows-

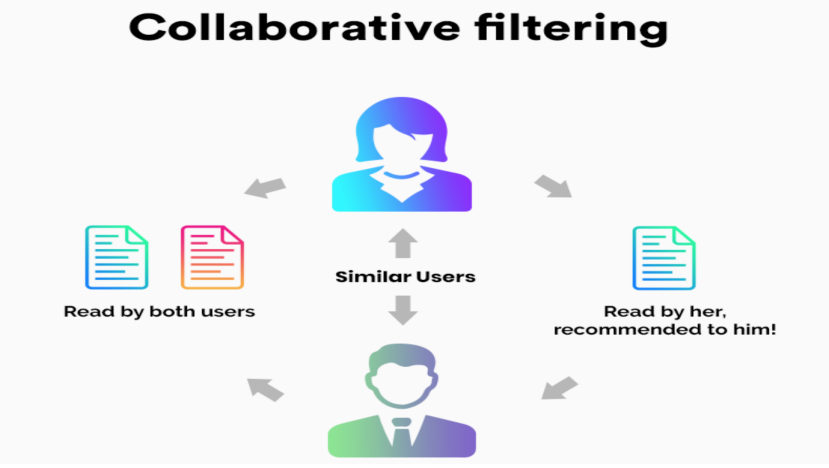
Collection : Data collected can be explicit (ratings and comments on products) or implicit (page views, order history, etc.).
Storing : The type of data used to create recommendations can help you decide the kind of storage you should use- NoSQL database, object storage, or standard SQL database.
Analyzing : The recommender system finds items with similar user engagement data after analysis.
Filtering : This is the last step where data gets filtered to access the relevant information required to provide recommendations to the user. To enable this, you will need to choose an algorithm suiting the recommendation system.

Types Of Recommendation Systems

There are three main types of recommendation systems –

1. Collaborative Filtering

The collaborative filtering method is based on gathering and analyzing data on user's behavior. This includes the user's online activities and predicting what they will like based on the similarity with other users.



For example,
if user A likes Apple, Banana, and Mango while user B likes Apple, Banana, and Jackfruit, they have similar interests. So, it is highly likely that A would like Jackfruit and B would enjoy Mango. This is how collaborative filtering takes place.

Two kinds of collaborative filtering techniques used are:

- User-User collaborative filtering
- Item-Item collaborative filtering

One of the main advantages of this recommendation system is that it can recommend complex items precisely without understanding the object itself. There is no reliance on machine analyzable content.

Advantages:

- No need for domain knowledge because embedding is learned automatically.
- Capture inherent subtle characteristics.

Disadvantages:

- Cannot handle fresh items due to cold start problems.
- Hard to add any new features that may improve quality of model

How to address Cold Start?

Approaches to address cold start with new users:

1. Popular items (get quick reaction of the users)
2. Demographically relevant items
3. Browsing history
4. Secondary source of data --- social network,subscription
5. Netflix - start with rating a few movies

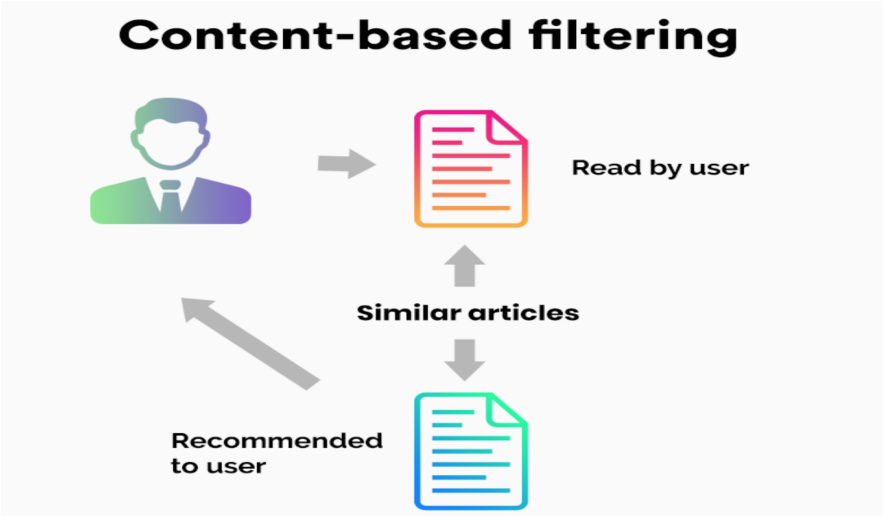
Approaches to address cold start with new items:

1. Recommend to random users/ or some selective users based on certain criteria
2. How about offering the product to influential people in the social network

Note : we can see Collaborative Filtering using Matrix Factorization here

2. Content-Based Filtering

Content-based filtering methods are based on the description of a product and a profile of the user's preferred choices. In this recommendation system, products are described using keywords, and a user profile is built to express the kind of item this user likes.



For instance,
if a user likes to watch movies such as Iron Man, the recommender system recommends movies of the superhero genre or films describing Tony Stark.
The central assumption of content-based filtering is that you will also like a similar item if you like a particular item.

Advantages:

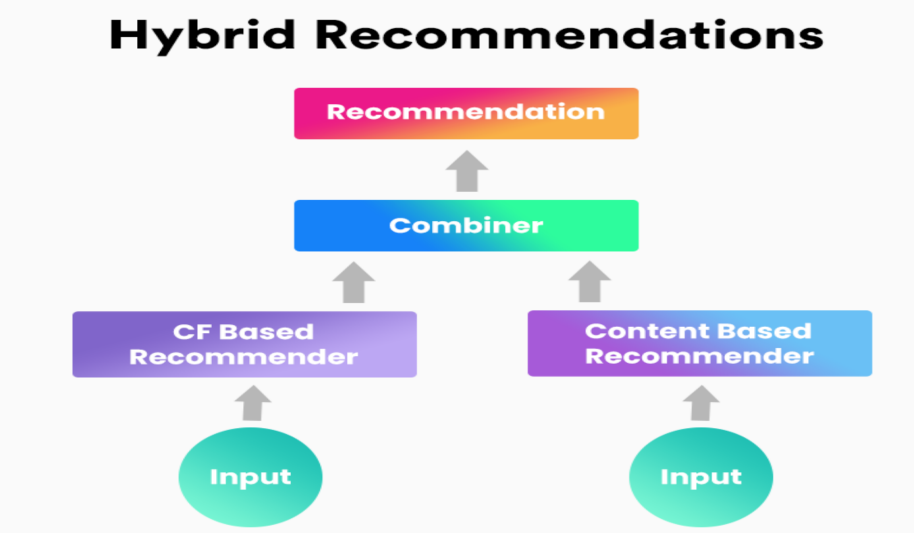
- No need for data on other users when applying to similar users.
- Able to recommend to users with unique tastes.
- Able to recommend new & popular items
- Explanations for recommended items.

Disadvantages:

- Finding the appropriate feature is hard.
- Doesn't recommend items outside the user profile.

3. Hybrid Recommendation Systems

In hybrid recommendation systems, products are recommended using both content-based and collaborative filtering simultaneously to suggest a broader range of products to customers. This recommendation system is up-and-coming and is said to provide more accurate recommendations than other recommender systems.



Netflix is an excellent case in point of a hybrid recommendation system. It makes recommendations by juxtaposing users' watching and searching habits and finding similar users on that platform. This way, Netflix uses collaborative filtering.

By recommending such shows/movies that share similar traits with those rated highly by the user, Netflix uses content-based filtering. They can also veto the common issues in recommendation systems, such as cold start and data insufficiency issues.

Similarity Measures

To determine the degree of similarity, most recommendation systems rely on one or more of the following:

- cosine
- dot product
- Euclidean distance

Cosine

This is simply the cosine of the angle between the two vectors, s(q,x)=cos(q,x)

$$similarity(A, B) = cos(\theta) = \frac{A \cdot B}{\|A\| \|B\|}$$

Dot Product

The dot product of two vectors is s(q,x)=⟨q,x⟩=∑i=1dqixi
. It is also given by s(q,x)=‖x‖‖q‖cos(q,x)
(the cosine of the angle multiplied by the product of norms).
Thus, if the embeddings are normalized, then dot-product and cosine coincide.

Euclidean distance

This is the usual distance in Euclidean space, s(q,x)=‖q−x‖=[∑i=1d(qi−xi)^2]^1/2.
A smaller distance means higher similarity. Note that when the embeddings are normalized, the squared Euclidean distance coincides with dot-product (and cosine) up to a constant, since in that case 1/2‖q−x‖^2=1−⟨q,x⟩

Code :

```
#import libraries
import pandas as pd
import numpy as np

data=pd.read_csv("Recommendation systems/Movie.csv")
data.tail()
data.info()

# no. of unique users in the data set
len(data.userId.unique())

# no. of unique movies
len(data.movie.unique())
```

```
user_data=data.pivot(index='userId',columns='movie',values='rating').reset_index(drop=True)
user_data

user_data.index=data.userId.unique()  # original unique userId as index
user_data

# import this NaN with 0 values
user_data.fillna(0,inplace=True)
user_data

#calculate cosine similarity between users
from sklearn.metrics import pairwise_distances
from scipy.spatial.distance import cosine,correlation

user_sim=1-pairwise_distances(user_data.values,metric='cosine')
user_sim

#store the result in data frame
user_sim_df=pd.DataFrame(user_sim)
user_sim_df

# set the index and column name to user ID
user_sim_df.index=data.userId.unique()
user_sim_df.columns=data.userId.unique()
user_sim_df

np.fill_diagonal(user_sim,0)
user_sim_df

#Most similar User
user_sim_df.idxmax(axis=1)[0:]

data[(data['userId']==6) | (data['userId']==168)]

user_1=data[data['userId']==6]
user_1

user_2=data[data['userId']==168]
user_2

pd.merge(user_1,user_2,on='movie',how='outer')

#### User ID 6 will recommended to user ID 168 to watch 'Grumpier Old Men' and 'Sabrina ' movies
```

Dimensionality Reduction

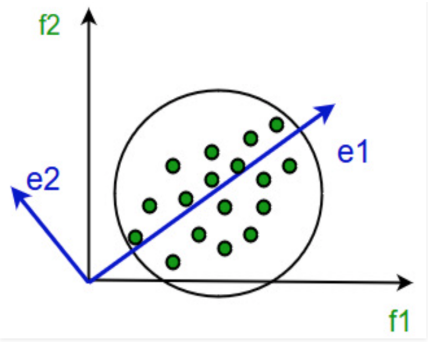
PCA

PCA is ..

- A backbone of modern data analysis.
- A black box that is widely used but poorly understood.

It works on a condition that while the data in a higher dimensional space is mapped to data in a lower dimension space, the variance of the data in the lower dimensional space should be maximum.

Use PCA to :
1) Identify relation between columns
2) Reduce # columns
3) Visualize multi-dim to low-dim(in 2D)



- It extracts a low dimensional set of features from a high dimensional data set with a motive to capture as much information as possible.
- With fewer variables, visualization also becomes much more meaningful.
- PCA is more useful when dealing with 3 or higher dimensional data.
- PCA is a linear orthogonal transformation that transforms the data to a new coordinate system such that the greatest variance by any projection of the data comes to lie on the first coordinate (called the first principal component), the second greatest variance on the second coordinate, and so on ....

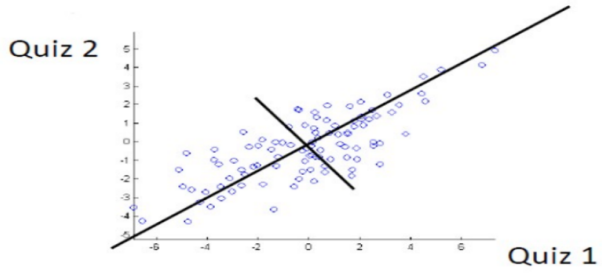
Example: TV and Movies
TV and Movies are almost always 2-D, even though the subjects are 3-D.

This is OK. The 3rd dimension doesn't usually add much to the story. Things still look believable without it. People look like people, things look like things, even when they have no depth and are flat on a screen.

A movie camera takes 3-D information and flattens it to 2-D without too much loss of information.

How to compress the data loosing the least amount of information?

Main idea: high variance = lots of information



Note : Here Quiz1 ant Quiz 2 are features

PCA⇨

Input	Output
p measurements	p principal components (= p weighted averages of original measurements)
Correlated	Uncorrelated Ordered by variance Keep top principal components; drop rest

PCA Weights :

Univ	SAT	Top10	Accept	SFRatio	Expenses	GradRate
Brown	1310	89	22	13	22704	94
CalTech	1415	100	25	6	63575	81
CMU	1260	62	59	9	25026	72
Columbia	1310	76	24	12	31510	88
Cornell	1280	83	33	13	21864	90
Dartmouth	1340	89	23	10	32162	95
Duke	1315	90	30	12	31585	95
Georgetown	1255	74	24	12	20126	92
Harvard	1400	91	14	11	39525	97
JohnsHopkins	1305	75	44	7	58691	87
MIT	1380	94	30	10	34870	91
Northwestern	1260	85	39	11	28052	89
NotreDame	1255	81	42	13	15122	94
PennState	1081	38	54	18	10185	80

PC_1	PC_2	PC_3	PC_4	PC_5	PC_6

The i^{th} principal component is a weighted average of original measurements / columns:

$$PC_i = a_{i1}X_1 + a_{i2}X_2 + a_{i3}X_3.... + a_{in}X_n$$

Weights (a_{ij}) are chosen such that:

1. PCs are ordered by their variance (PC_1 has largest variance, followed by PC_2, PC_3, and so on)
2. Pairs of PCs have correlation = 0
3. For each PC, sum of squared weights = 1 (Unit Vector)

What do we need

1) Covariance Matrix
2) Eigenvectors and eigenvalues of Covariance matrix

Variance :

- A measure of the spread of the data in a data set with mean

$$\sigma^2 = \frac{\sum_{i=1}^{n} (X_i - \overline{X})^2}{(n-1)}$$

- Variance is claimed to be the original statistical measure of spread of data.

Covariance :

- Variance – measure of the deviation from the mean for points in one dimension, e.g., heights
- Covariance – a measure of how much each of the dimensions varies from the mean with respect to each other.
- Covariance is measured between 2 dimensions to see if there is a relationship between the 2 dimensions, e.g., number of hours studied and grade obtained.
- The covariance between one dimension and itself is the variance

$$\text{var}(X) = \frac{\sum_{i=1}^{n} (X_i - \overline{X})(X_i - \overline{X})}{(n-1)}$$

$$\text{cov}(X, Y) = \frac{\sum_{i=1}^{n} (X_i - \overline{X})(Y_i - \overline{Y})}{(n-1)}$$

What is the interpretation of covariance calculations?

Say you have a 2-dimensional data set
X: number of hours studied for a subject
Y: marks obtained in that subject
And assume the covariance value (between X and Y) is: 104.53

What does this value mean?

Exact value is not as important as its sign.

- A **positive value** of covariance indicates that both dimensions increase or decrease together, e.g., as the number of hours studied increases, the grades in that subject also increase.
- A **negative value** indicates while one increases the other decreases, or vice-versa, e.g., active social life Vs. performance in ECE Dept.
- If **covariance is zero**: the two dimensions are independent of each other, e.g., heights of students vs. grades obtained in a subject.

Why bother with calculating (expensive) covariance when we could just plot the 2 values to see their relationship?
Covariance calculations are used to find relationships between dimensions in high dimensional data sets (usually greater than 3) where visualization is difficult.

Eigenvalue Problem
The eigenvalue problem is any problem having the following form:

$$A.V=\lambda V$$

A: m x m matrix
v: m x 1 non-zero vector
λ: scalar
Any value of λ for which this equation has a solution is called the **eigenvalue** of A and the vector v which corresponds to this value is called the **eigenvector** of A.

$$\begin{bmatrix} 2 & 3 \\ 2 & 1 \end{bmatrix} \times \begin{bmatrix} 3 \\ 2 \end{bmatrix} = \begin{bmatrix} 12 \\ 8 \end{bmatrix} = 4 \times \begin{bmatrix} 3 \\ 2 \end{bmatrix}$$

$$A \cdot \quad v \qquad = \lambda \cdot \quad v$$

Therefore, (3,2) is eigenvector of matrix A
AND 4 is eigenvalue of matrix A

It involves the following steps:

- Construct the covariance matrix of the data.
- Compute the eigenvectors of this matrix.
- Eigenvectors corresponding to the largest eigenvalues are used to reconstruct a large fraction of variance of the original data.

Hence, we are left with a lesser number of eigenvectors, and there might have been some data loss in the process. But, the most important variances should be retained by the remaining eigenvectors.

Advantages

- It helps in data compression, and hence reduced storage space.
- It reduces computation time.
- It also helps remove redundant features, if any.

Disadvantages

- It may lead to some amount of data loss.
- PCA tends to find linear correlations between variables, which is sometimes undesirable.
- PCA fails in cases where mean and covariance are not enough to define datasets.
- We may not know how many principal components to keep- in practice, some thumb rules are applied.

Code :

```
# import the libraries
import pandas as pd
import numpy as np
```

```
from sklearn.decomposition import PCA
import matplotlib.pyplot as plt
from sklearn.preprocessing import scale

uni = pd.read_csv("Universities.csv")
uni.describe()
uni.head()

# Considering only numerical data
uni.data = uni.iloc[:,1:]
uni.data.head()
# Converting into numpy array
UNI = uni.data.values
UNI

# Normalizing the numerical data
uni_normal = scale(UNI)
uni_normal

pca = PCA()
pca_values = pca.fit_transform(uni_normal)
pca_values

pca = PCA(n_components = 6)
pca_values = pca.fit_transform(uni_normal)

# The amount of variance that each PCA explains is
var = pca.explained_variance_ratio_
var

# Cumulative variance
var1 = np.cumsum(np.round(var,decimals = 4)*100)
var1

pca.components_

# Variance plot for PCA components obtained
plt.plot(var1,color="red")

pca_values[:,0:1]

# plot between PCA1 and PCA2
x = pca_values[:,0:1]
y = pca_values[:,1:2]
#z = pca_values[:2:3]
plt.scatter(x,y)

finalDf = pd.concat([pd.DataFrame(pca_values[:,0:2],columns=['pc1','pc2']), uni[['Univ']]], axis = 1)

import seaborn as sns
sns.scatterplot(data=finalDf,x='pc1',y='pc2',hue='Univ')
```

t-SNE

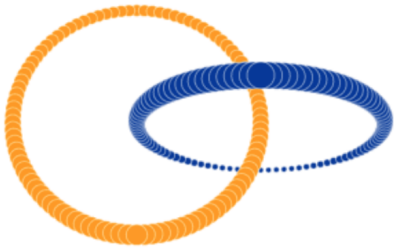
T-distributed Stochastic Neighbor Embedding (t-SNE) is a nonlinear dimensionality reduction technique well-suited for embedding high-dimensional data for visualization in a low-dimensional space of two or three dimensions.

t-SNE has become so popular because :

- it was demonstrated to be useful in many situations,
- it's incredibly flexible, and
- can often find structure where other dimensionality-reduction algorithms cannot.

t-SNE is a **non-linear** dimensionality reduction algorithm used for exploring high-dimensional data

linear structures ???
Can we apply PCA for nonlinear structured data? No



How t-SNE WORKS ?

The t-SNE algorithm calculates a similarity measure between pairs of instances in the high dimensional space and in the low dimensional space. It then tries to optimize these two similarity measures using a cost function.

The similarities between points in the high dimensional space

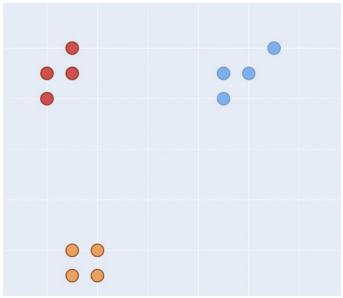
Think of a bunch of data points scattered on a 2D space. For each data point (xi), center a Gaussian distribution over that point. Then we measure the density of all points (xj) under that Gaussian distribution. Then renormalize for all points.

This gives us a set of probabilities (Pij) for all points. Those probabilities are proportional to the similarities. All that means is, if data points x1 and x2 have equal values under this gaussian circle

then their proportions and similarities are equal and hence you have local similarities in the structure of this high dimensional space.

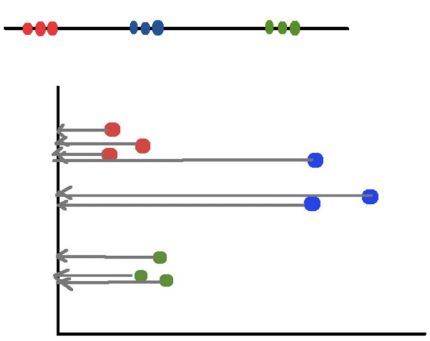
The Gaussian distribution or circle can be manipulated using what's called **perplexity**, which influences the variance of the distribution (circle size) and essentially the number of nearest neighbors

Probability Distribution



It has 3 different classes, and we can easily distinguish them from each other. The first part of the algorithm is to create a probability distribution that represents similarities between neighbors.
“ similarity of datapoint xj to datapoint Xi is the conditional probability p(j|i), that Xi would pick X; as its neighbor“.

We want to reduce the 2D plot into a 1D plot while maintaining clear boundaries between the clusters.



Simply projecting the data on to an axis is a poor approach to dimensionality reduction because we lose a substantial amount of information

A step in the t-SNE algorithm involves measuring the distance from one point to every other point. Instead of working with the distances directly, them to a probability distribution.

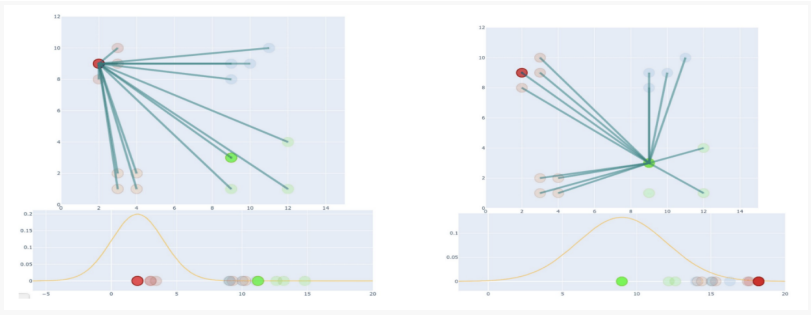
Mathematically, We write the equation for a normal distribution as follow :

$$P(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-(x-\mu)^2/(2\sigma^2)}$$

Drop everything before the exponent and use another point instead of the mean,

$$p_{j|i} = \frac{\exp\left(-\|x_i - x_j\|^2/2\sigma_i^2\right)}{\sum_{k\neq i}\exp\left(-\|x_i - x_k\|^2/2\sigma_i^2\right)},$$

If we take two points and try to calculate conditional probability between them then values of p_{ilj} and p_{jl. i} will be different:



The reason for that is because they are coming from two different distributions. Which one should we pick for the calculation then?

$$p_{ij} = \frac{p_{j|i} + p_{i|j}}{2N}$$

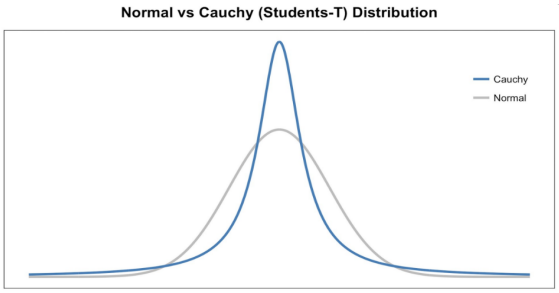
Create low dimensional space :

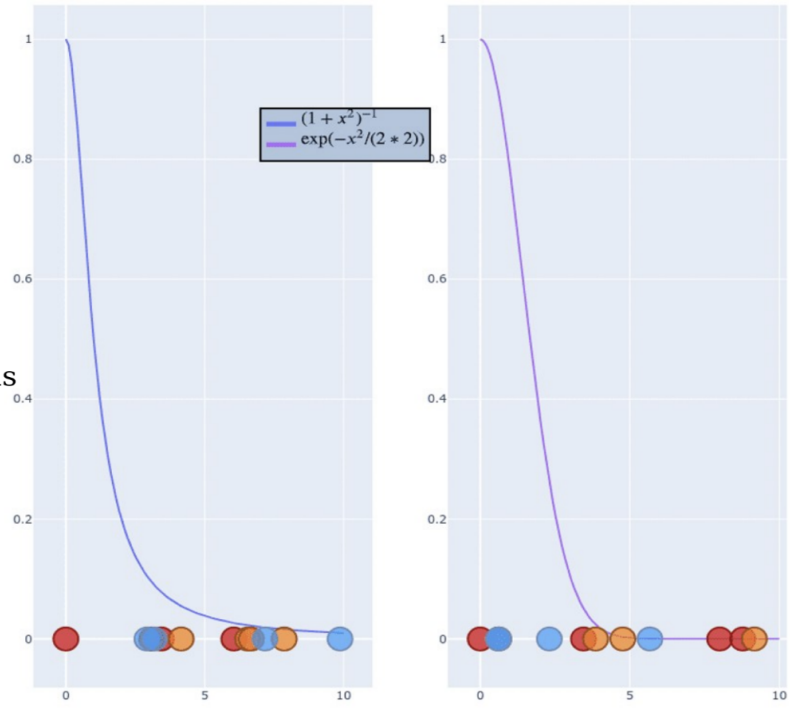
The next part of t-SNE is to create low-dimensional space with the same number of points as in the original space.
Points should be spread randomly on a new space. **The goal of this algorithm is to find similar probability distribution in low dimensional space**. The most obvious choice for the new distribution would be to use Gaussian again.
But the problem with Gaussian is that it has a “short tail” and because of that it creates a **crowding problem**. To solve that we're going to use **Student t-distribution** with a single degree of freedom

$$q_{ij} = \frac{\exp(-\|y_i - y_j\|^2 / 2\sigma_i^2)}{\sum_{k \neq l} \exp(-\|y_k - y_l\|^2 / 2\sigma_i^2)} \quad \Rightarrow \quad q_{ij} = \frac{(1 + \|y_i - y_j\|^2)^{-1}}{\sum_{k \neq l} (1 + \|y_k - y_l\|^2)^{-1}}$$

Instead of using a Gaussian distribution you use a Student t-distribution with one degree of freedom, which is also known as the **Cauchy distribution** .

This gives us a second set of probabilities (Qij) in the low dimensional space. As you can see the Student t-distribution has heavier tails than the normal distribution. The heavy tails allow for better modeling of far apart distances





Student’s t :
Low Dimensions

Gaussian : High Dimensions

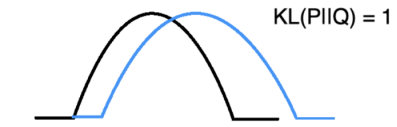
The last step is that we want these set of probabilities from the low dimensional space (Qij) to reflect those of the high dimensional space (Pij) as best as possible.

We want the two map structures to be similar. We measure the difference between the probability distributions of the two-dimensional spaces using Kullback-Liebler divergence (KL). KL is an asymmetrical approach that efficiently compares large Pij and Qij values.

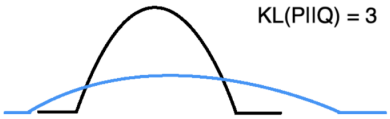
Finally, we use gradient descent to minimize our KL cost function.

Kullback-Liebler divergence

We make use of something called the Kullback-Leiber divergence. The KL divergence is a measure of how different one probability distribution is from a second.



The lower the value of the KL divergence, the closer two distributions are to one another. A KL divergence of 0 implies that the two distributions in question are identical



t-SNE in ML

t-SNE could be used to investigate, learn, or evaluate segmentation.
Oftentimes we select the number of segments prior to modelling or iterate after results.

t-SNE can often show clear separation in the data.
This can be used prior to using your segmentation model to select a cluster number or after to evaluate

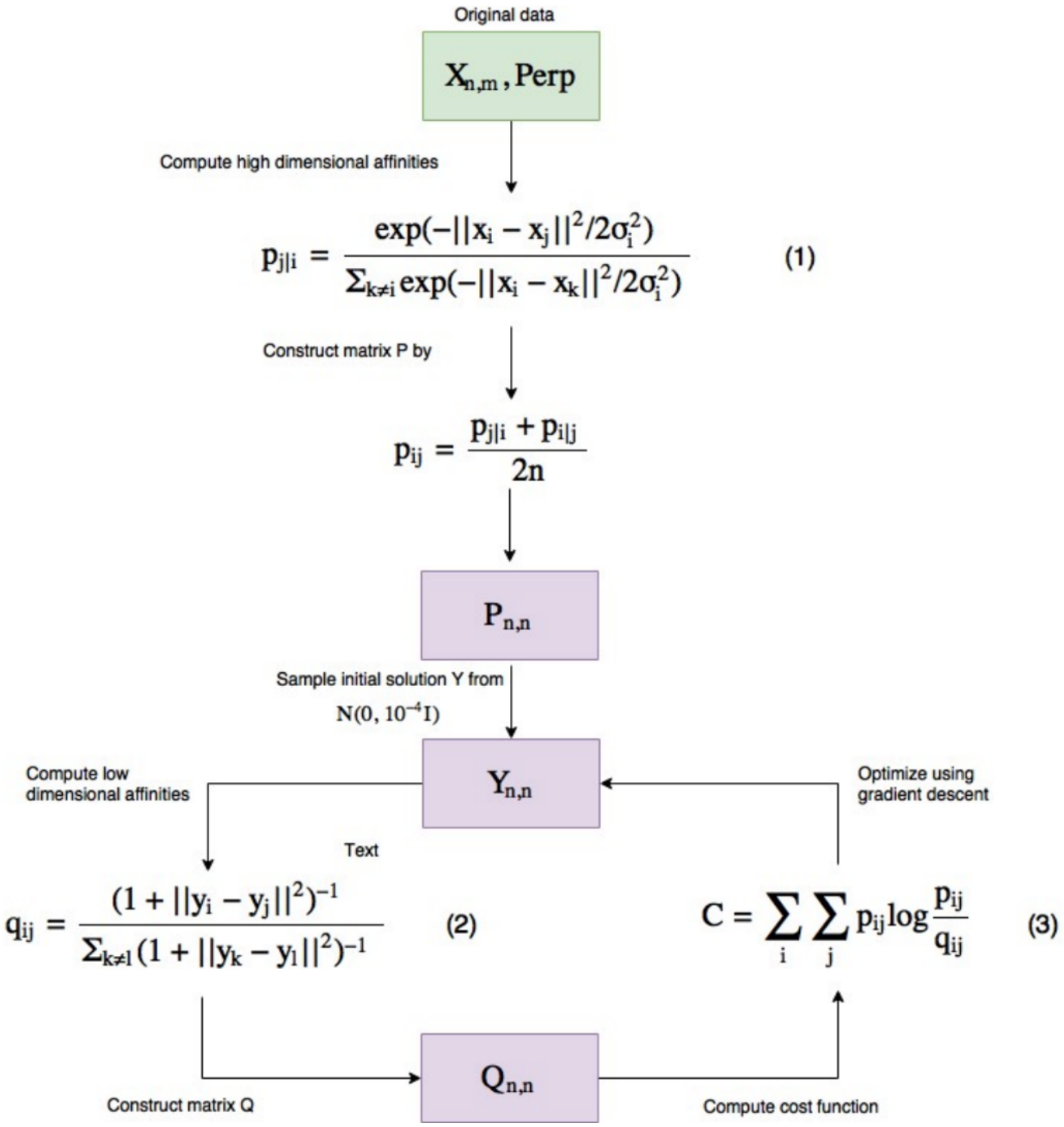
Applications

t-SNE in areas like climate research,
- computer security,
- bioinformatics,
- cancer research, etc.

t-SNE could be used on high dimensional data and then the output of those dimensions then become inputs to some other classification model

Drawbacks of t-SNE

Problems with t-SNE arise when intrinsic dimensions are higher i.e. more than 2-3 dimensions.
t-SNE has the tendency to get stuck in local optima like other gradient descent based algorithms.
The basic t-SNE algorithm is slow due to nearest neighbor search queries.



Code :

```
# Import libraries
from pandas import read_csv
from sklearn.manifold import TSNE
from bioinfokit.visuz import cluster

df=read_csv("/content/drive/MyDrive/jupyter notebook workspace/t-SNE/TSNE_data.csv")
df.head()

df.info()

array=df.values
```

```
#spit-out validation data
X=array[:,1:]
Y=array[:,0]

#TSNE algorithm
data_tsne=TSNE(n_components=2).fit_transform(X)

data_tsne

#TSNE Visualization
cluster.tsneplot(score=data_tsne)

# get list of categories
color_class=df['diagnosis'].to_numpy()
cluster.tsneplot(score=data_tsne,colorlist=color_class,legendpos='upper right',legendanchor=(1.15, 1))
#plot will be in default directory
```

References :

Book : Machine Learning with Python Cookbook by Chris Albon

Book : Understanding Machine Learning: From Theory to Algorithms

https://www.javatpoint.com/unsupervised-machine-learning

https://dataaspirant.com/unsupervised-learning-algorithms/#t-1610384863259

https://www.javatpoint.com/clustering-in-machine-learning

https://scikit-learn.org/stable/index.html#

https://www.kdnuggets.com/2020/04/dbscan-clustering-algorithm-machine-learning.html

https://medium.com/mlearning-ai/what-are-the-types-of-recommendation-systems-3487cbafa7c9