Data Analysis & Visualization

LVC 3: Unsupervised Learning

The world of machine learning is full of diverse problems, and there are techniques appropriate to resolve each problem to an accurate level. But at a major level, these techniques can be grouped into two categories - **Supervised Learning and Unsupervised Learning techniques**. In a supervised learning technique, there has to be labeled data to train the algorithm. The algorithm is given an opportunity to supervise the data and gain some sense of patterns existing in it. The ultimate goal of such an algorithm is to be accurate, and generic when applied to an **unseen / out-of-sample** data point. All such problems come under the domain of supervised learning. Examples of supervised learning problems are predicting the **price of a house**, **forecasting the stock price, classifying a product as defective or not**, etc.

However, there are a lot of problems where we cannot avail of a target label in the data. This is because the very nature of the problem is different and it does not require labeling. **Unsupervised Learning** techniques are used to solve such problems, where the presence of a target feature is not required. This technique is used to create groups with some common patterns among the given data points. The created groups are called **clusters** of the data. The data points in a cluster are similar to each other, while those in different clusters are dissimilar to each other. Below are a few examples where unsupervised learning is used:

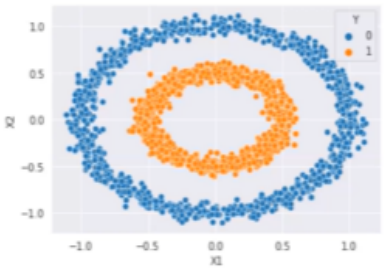
1. **Customer Segmentation:** Given a set of customer data containing features describing each customer, the goal is to create different customer groups called clusters, such that the customers in each group show similar characteristics. Such customer segments can be created using clustering techniques from unsupervised learning, and the business can then implement targeted strategies to maximize revenue from each group of customers based on their profile.

2. **Blood Group:** Given a set of patients' blood sample data, the goal is to create groups of people with similar profiles. This would be useful in a scenario where there was no prior

knowledge about the blood group available. After creating these groups or clusters, it is possible that each cluster has a similar blood group, and this may be a more useful, targeted way to identify potential blood donors from a large population for example, among other potential uses.

While creating clusters in the data, it would be a great help if we already knew the possible number of clusters that exist in the data. To do so, we can use the **data visualization** techniques we learned in the first lecture. Visual inspection may give us an idea about the number of clusters present by identifying those regions where lots of data points seem to be grouped together as possible clusters in the dataset.

For example, looking at the below graph, it is fairly straightforward to understand that there are two clusters in the data, the **outer ellipse** and the **inner one**. A visual inspection like this can help a lot in then deploying the right method to segregate the different clusters from the data because in two dimensions, our eyes are still the best tool we have to identify potential clusters and groupings in scatter plots.



The above data is 2-dimensional**,** and hence it is easy for us to visually inspect the number of clusters in the plot. But in the real world, it is not always possible to get 2-dimensional data for every problem; rather the data tends to be high-dimensional due to the abundance of features and information we are able to extract for each instance in our dataset in today's digitally interconnected world. For high-dimensional data, it is not possible to visually inspect scatter plots and identify the potential clusters in the data. In such cases, clustering algorithms are necessary.

Before discussing clustering algorithms, **let's first understand how we measure the similarity between data points,** in order to identify whether two points belong to the same cluster or not.

How do you find the similarity between two data points?

Understanding similarity in a general context is easy stuff for humans. But in the context of clustering, every data point is just represented by a certain set of numbers or a vector. Understanding whether two data points are similar to each other or not then requires mathematical intervention. The most suitable quantity to measure the similarity between two points is the **distance between them**. In a multi-dimensional feature space, distance would seem to be a more appropriate way of understanding similarity between points. **Points that are closer to each other in terms of some distance metric can be considered more similar** to each other than they are to other points. A small distance between points is one way of saying there's a small difference or a high similarity in the dimensions of those data points being measured.

Now, there are many kinds of distance metrics used to measure the similarity between two points. The most common ones are as follows:

1. **Euclidean Distance:** Euclidean distance is the straight-line distance between two points in vector space (feature space). It is the **most widely used** distance measurement method. In Euclidean geometry, it it represents the shortest-possible distance between two points. Mathematically, it can be given as follows:

$$d(x^{(i)} - x^{(j)}) = \sqrt{(x_1^{(i)} - x_1^{(j)})^2 + (x_2^{(i)} - x_2^{(j)})^2 + ... + (x_p^{(i)} - x_p^{(j)})^2}$$

   Here, $d(x^{(i)} - x^{(j)})$ is the distance between the two points, $x^{(i)}$ and $x^{(j)}$, where each of the points have p dimensions. This method is also sometimes known as the L_2 norm, since it relies on a second-order power (the square) of the difference in magnitudes.

2. **Manhattan Distance:** Manhattan Distance is the sum of absolute differences between points across all dimensions. Mathematically, it can be calculated as follows:

$$d(x^{(i)} - x^{(j)}) = |x_1^{(i)} - x_1^{(j)}| + |x_2^{(i)} - x_2^{(j)}| + ... + |x_p^{(i)} - x_p^{(j)}|$$

   This method is also known as the L_1 norm, since it only relies on the first-order power of the magnitude of differences.

3. **Maximum Distance:** The maximum distance calculates the distance along all the dimensions between two points and then picks the maximum one.

$$d(x^{(i)} - x^{(j)}) = max_{k=1,2,...p} |x_k^{(i)} - x_k^{(j)}|$$

The above methods provide different ways to compute the distance between any two points in a specific feature space. There is no **order of superiority** between these distances - with respect to clustering algorithms, one can generally be used in place of another without any bias or priority. When computing the distance between two points, there is a possibility that two or more features may belong to different units. For example: we may be measuring both the height (in feet) and the weight (in pounds) of a person as two of our features / columns in the dataset, and these two features obviously have very different scales. The coordinate of a single point may consist of multiple such features with widely varying scales of measurement for each feature. Those scales of measurement obviously depend on the units that each feature is measured in. However, the distance computing methods above do not take any information about these units into account, and merely use the magnitude of the feature in their calculation. Therefore, if there is a scale difference among the different features, normalization of the features needs to be done in order to bring the features to a common scale. **Normalization or scaling is a critical pre-requisite not just for clustering but for any distance-based methods in machine learning**, because the relative distance between points is highly dependent on the scale of each feature.

Let us now learn about the different methods used to create clusters. We will discuss four clustering algorithms as part of this summary:

1. K-means Clustering
2. Gaussian Mixture Models
3. Hierarchical Clustering
4. DBSCAN

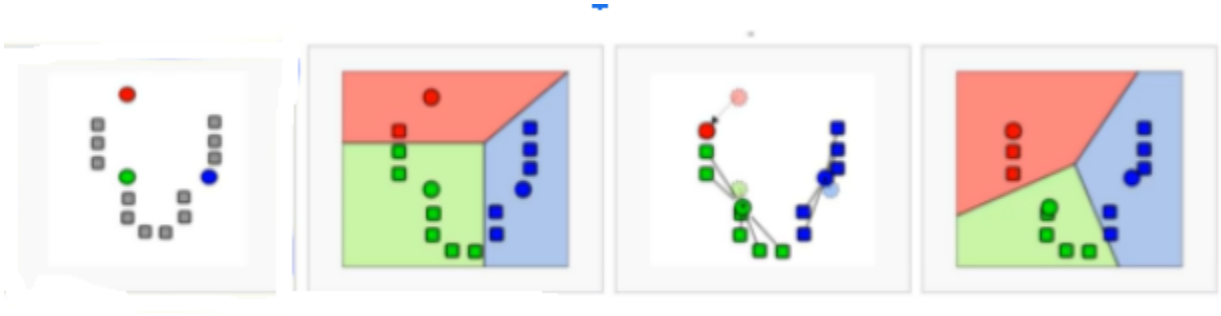
Let us get into the details of these four algorithms:

1. **K-means Clustering:** In K-means Clustering, we create K clusters from the given data. We start by deciding on a fixed number of clusters we expect to find in the data, denoted by K.

Data visualization techniques like PCA and t-SNE, and other methods like elbow plots can help give us an idea about a suitable value for K.

K-means Clustering minimizes the sum of the pairwise distances between data points within each cluster, also known as the **Within-Group Sum of Squares** (WGSS). This is equivalent to minimizing the sum of the distances to the cluster means. The lower the value of WGSS, the better we consider the quality of the clusters to be.

Let us understand the working of K-Means with the help of the below diagram. We will assume that there are three existing clusters in the data, therefore, we choose K=3. We start with three



random centroids or means shown in the first diagram as the red, green, and blue dots. Now, each data point is assigned to one of the three centroids depending on which centroid is closest to a particular point. In the next iteration, the centroids or means are revised to get a new set of three centroids. They are computed by taking the mean the of data points that are there in the existing clusters from the last iteration. This time multiple points will change their clusters from the last iteration depending on which of the new centroids are closest to a particular point. This process is repeated till there is no change in the means of the clusters and the clusters can be considered to be stabilized.

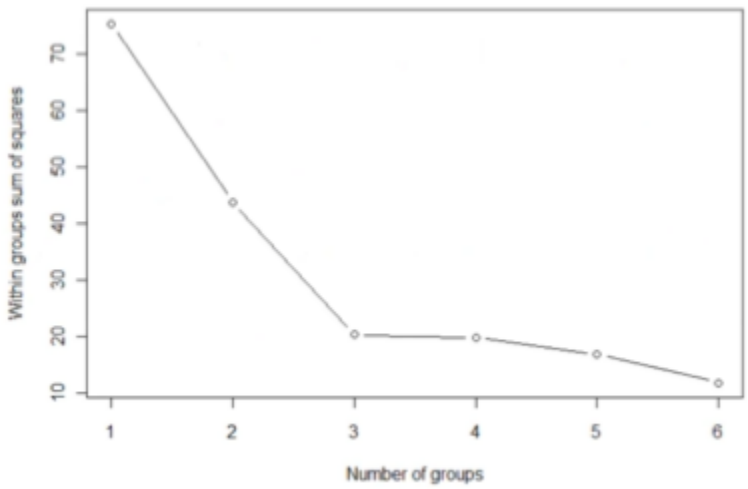
Remarks:

1. The means of the clusters may not be actual data points from the data.
2. K-means Clustering leads to spherically-shaped clusters of similar radii since the distance of each data point within a cluster to the cluster's centroid is being minimized.

Now that we understand the working principle, let us look at a good way of selecting the number of clusters K, before running the K-means algorithm.

It begins with finding the WGSS value for different values of K (the number of clusters). The plot of the within-groups sum of squares against the number of clusters is called the **elbow plot.** The below plot shows a sample elbow plot. To find the most appropriate number of clusters present in the data, we pick the value of K after the last big drop in WGSS. The value of WGSS will decrease with an increase in K, but in general, we only prefer a limited number of clusters for interpreting and data-driven decision-making. Hence, we don't necessarily choose a large K value with an extremely small WGSS; we rather choose the value of K with a significant drop of WGSS, beyond which subsequent drops in the value of WGSS seem small or insignificant.

According to the figure below, WGSS has its last big drop at K=3. Beyond that, the drops in WGSS are only minimal. Hence, we can choose K=3 while running the K-Means algorithm.



Now, let's look at some of the drawbacks of the K-means algorithm.

Drawbacks of K-means Clustering

1. Depending on the starting points chosen randomly in the beginning, the final clusters may change. There are more sophisticated initialization methods available instead of choosing starting points randomly, in order to avoid this issue and get better clusters.
2. The second drawback of the K-means algorithm is that it does hard clustering, i.e., any individual point is associated with only one cluster. Sometimes, we would prefer a "softer" approach to clustering, where we simply get the probabilities of belonging to each cluster for each point. K-means Clustering however provides no scope for this idea.

We know from basic statistics that the mean of the data gets highly affected by outliers. Since K-means Clustering uses means to find centroids at each iteration, outliers impact the final clusters. In the presence of outliers, the change in mean in every iteration will be much more than in comparison to when there are no outliers in the data.

A robust alternative is to use another method, known as **Partitioning Around Medoids (PAM)** or **K-medoids Clustering**, to avoid the impact of outliers. PAM uses medoids, which are the medians or **actual observations** from the data, as cluster centers. PAM is more robust than K-means Clustering when there are outliers in the data and it also gives a representative object (medoid) for each cluster that helps in interpreting the final clusters well.

Gaussian Mixture Models

The Gaussian Mixture Model (GMM) is another clustering algorithm in unsupervised learning. It allows a single data point to be in multiple clusters with some corresponding **probabilities**. For example, a certain data point X can belong to two clusters A and B with probabilities, say 86% and 14%. The higher the probability of a point belonging to a cluster, of course, the more likely it is to be in that cluster.

GMM is a probabilistic model that assumes all the data points are generated from a mixture of a finite number of Gaussian distributions with unknown parameters. Hence, the distribution of samples within each cluster is modeled by a Gaussian. Parameter estimates for Gaussian distributions, i.e., the mean and variance for each cluster, are usually found using the Expectation-Maximization (EM) algorithm.

Some of the key features of this method are as follows:

1. To understand the advantage of this method, let us take an example of a patient going to a hospital for treatment. Assume that the doctor is employing a data-driven approach to diagnosis with the features and symptoms of the patient as the dimensions of the data. The doctor may know about the clusters of diseases they would expect to see but found that this patient's data shows the patient to be somewhat in between these clusters of diseases. In that case, a probabilistic approach like this may help the doctor decide on the right treatment for

this particular patient.

2. The optimal number of clusters in GMM can be determined in a statistically sound way, for example, using the Bayesian Information Criterion.

3. Unlike K-means, GMM allows for **ellipsoidal-shaped clusters** that help identify more elongated clusters in the data, than would be possible to detect by an algorithm looking for more spherically-shaped groups.

4. For K-means to deal with outliers, we were taking the actual data points as initial clusters and the problem of high variance in the mean was resolved. But in GMM, this is not going to work. For GMM, the solution needs to be to remove the outliers before beginning the process of clustering itself.

5. The disadvantage associated with GMM is that the involvement of the probability component with each point in the dataset makes the method more mathematical and computationally heavy.

Hierarchical Clustering

There are two types of hierarchical clustering:

1. **Agglomerative Clustering:** In this method, we consider all the points to be individual clusters and then start combining clusters until we have one big cluster containing all the observations.
2. **Divisive Clustering:** In this method, we start with the whole group of observations and split them into clusters until each observation is an individual cluster.

To merge or split clusters, we need to understand how to find the distance between two clusters. There are various methods to find the distance between clusters. These methods are known as linkages, as they link different clusters. Some of the most common linkages are:

1. **Complete Linkage:** It creates perfect but small clusters by merging very small clusters. It uses the **maximum distance** between the two clusters to measure the distance between them. In the below diagram, it can be observed that small, but compact and clear, clusters are created using the complete linkage.

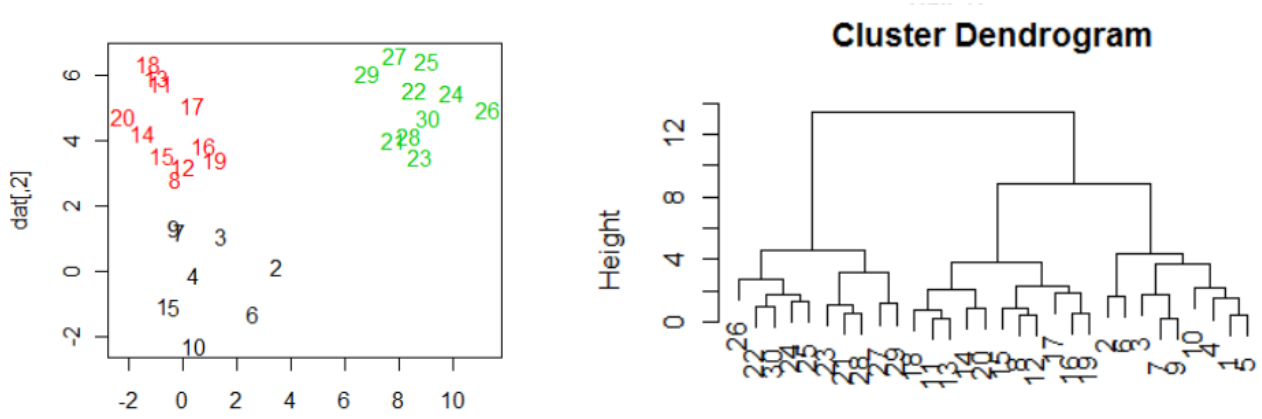
2. **Single Linkage:** In this method, we use the **minimum distance** to find the distance between clusters. This method allows us to find thin and long clusters. In the below figure, it can be seen that a chain of blue points forms a single cluster.

3. **Average Linkage:** In this method, the distance between two clusters is defined as the average of distances between all pairs of points, where each pair is made up of one point from each cluster. This method gives similar results to K-means Clustering. The below figure shows the average linkage method.

Now, let us understand how to choose the right number of clusters in hierarchical clustering. The first thing to understand is that there is **no strict rule** for this. So, the estimated count may vary a little, and some other value of cluster count may work better.

To estimate the number of clusters, we will take the help of a plot called a **Dendrogram**. A dendrogram is a tree-like structure that represents hierarchical clusters existing in the given data. It follows the hierarchy from top to bottom or bottom to top. It is used to represent the clusters and also to find the number of clusters.

The below figure shows 2-dimensional data on the left side and the corresponding dendrogram for agglomerative clustering, on the right side. In the dendrogram, the X-axis shows the observations, the Y-axis shows the distance between the merged clusters, and the horizontal line shows the merging of two clusters.



To find the number of clusters, a preferred way is to cut the dendrogram, using a horizontal line, where the vertical drop is maximum. The number of points at which the horizontal line cuts the dendrogram can be taken as the number of clusters. However, as mentioned earlier, there is no strict rule followed for this.

Now, let's look into the pros and cons of using the hierarchical clustering algorithm.

The **Pros** of hierarchical clustering are as follows:

- It shows all the possible linkages between the clusters as it is presented by a well-organized plot, the **Dendrogram**.

- It gives us **more understanding** of the data due to visual presentation.

- There is no need to set the number of clusters in the beginning. It helps to solve clustering for all the possible number of clusters 1, 2,..., n at once, and we can choose the desired number of clusters later.

The **Cons** of hierarchical clustering are as follows:

- It becomes very hard to interpret when the number of data points is very high, and hence the count of clusters is also high. The higher the observation count the more complex it is to interpret the dendrogram.

DBSCAN

It is a comparatively new method that is understood to be the generalization of **single linkage hierarchical clustering**. It uses certain parameters, which are explained below:

a. **minPts:** The minimum number of points (a threshold) clustered together for a region to be considered dense or a cluster.
b. **eps (ϵ):** A distance measure that will be used to locate the points in the neighborhood of any point.

Below are the steps followed in this algorithm:

a. In the beginning, the algorithm starts by randomly picking up a point in the dataset. The radius of the circle is ϵ.
b. If there are at least **"minPts"** points within the circle radius of **"ϵ"** then the point is considered a **core point**, otherwise, **non-core point**.
c. Once we identify all the core and non-core points, We randomly pick a core point and assign it to a cluster. The core points that are within the radius of ϵ are all added to the cluster. This continues until there is no core point in the radius of eps. Initially, it adds only the core points to the cluster.
d. When the cluster reaches a non-core point and if it falls in the eps range, it will be added to the cluster but can't be used to extend the cluster.
e. Once a cluster is completely formed, then a new core point will be chosen to form another cluster and the process continues.
f. There will be some points that will not be a part of any cluster and they will be treated as outliers.

One of the major benefits of this method is it handles the **outliers** existing in the data all by itself.

There are many such clustering algorithms, but how do we measure the quality of the generated clusters? To measure the quality of clusters, we use a **Silhouette Plot**.

Quality of Clustering - Silhouette Plot:

The Silhouette Plot displays the Silhouette score for each cluster, which is a metric used to measure how similar an object is to its own cluster in comparison to other clusters.

The Silhouette score ranges from -1 to 1, where:

- 1 means clusters are well apart from each other and clearly distinguished.
- 0 means clusters are indifferent, or we can say that the distance between clusters is not significant.
- -1 means clusters are assigned in the wrong way.
- The higher the value of the Silhouette score, thr better the quality of clusters created. Generally, **0.5 is used as a cut-off** value to say that the data points are well clustered.

To calculate the Silhouette score, let us go through the following steps:

For all the points $x^{(i)}$, let us compute:

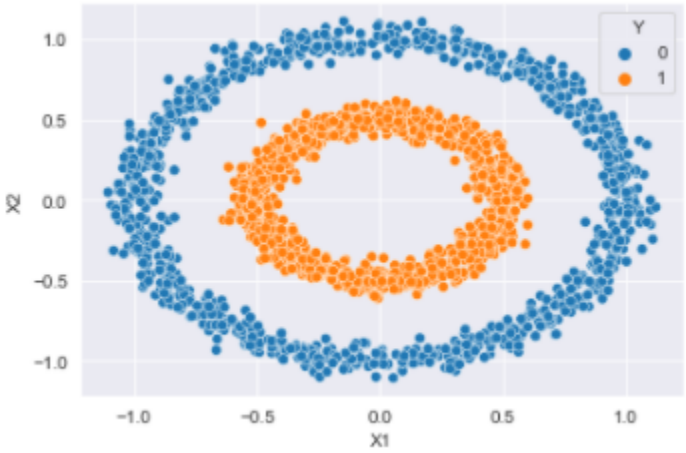
a. The average dissimilarity between $x^{(i)}$ and all other points in its cluster. This is denoted as a($x^{(i)}$).

b. The average distance between $x^{(i)}$ and the nearest cluster to which it does not belong. It is denoted as $b(x^{(i)})$.

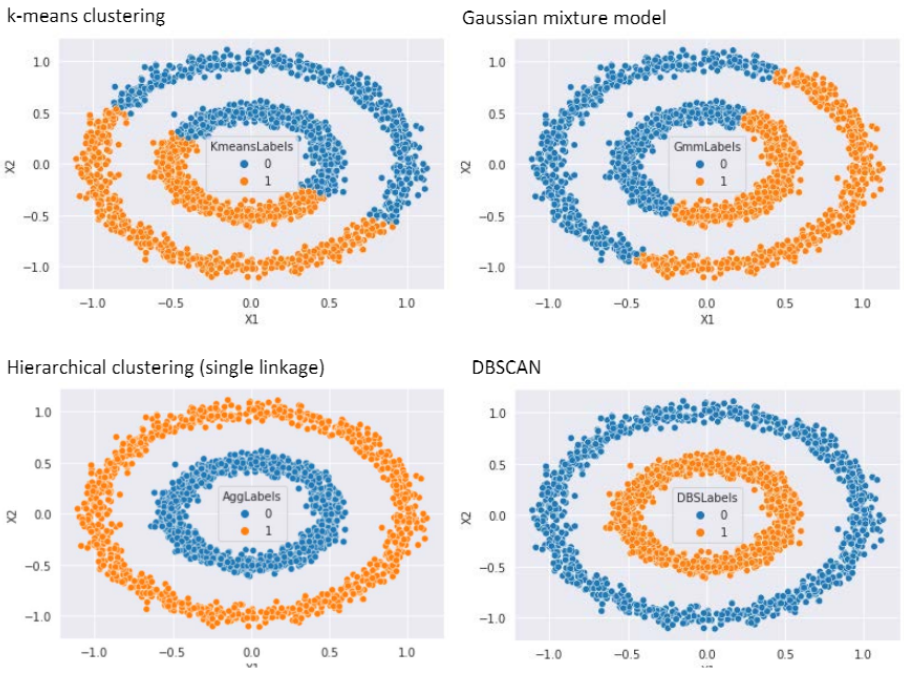
c. Calculate the Silhouette score, denoted by $s(x^{(i)})$, using the below formula:

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

After understanding this quantitative measure, let us apply these methods to a single problem and see their performance and suitability. The below scatter plot shows two concentric circles in a toy dataset, each belonging to a different class. Let's apply the clustering algorithms we have learned, and see how well each clustering algorithm performs in terms of identifying the underlying pattern of concentric circles.



To do so, all the existing algorithms are applied to the toy dataset, to solve the task. The below figure shows that the **Hierarchical** (except for the opposite labeling of clusters) and **DBSCAN** algorithms are able to identify the clusters correctly while **K-means** and **GMM** did not identify the underlying pattern of concentric circles.



The reason K-means and GMM algorithms could not find the clusters is because they are good at finding spherical and ellipsoidal-shaped clusters, respectively. However, the task required here is to find a cluster inside the shape of another cluster.

Community Detection

In today's world, we are sometimes required to apply clustering algorithms along with network analysis, to identify certain "communities" within large networks. Community detection is hence an important part of network analysis. The objective of community detection is to detect subsets of nodes that are more densely connected to each other in the network than outside the community.

The clustering techniques discussed so far can be used to determine subsets of points that are 'close' to each other given a pairwise distance or similarity measure defined by the network. To compare the similarity, it needs to associate a distance between the nodes. Below are a few measures useful for measuring the pairwise distance between nodes of a network:

1. **Geodesic Distance:** It is the shortest distance between two nodes of a network in terms of the connections between them.
2. Number of different neighbors
3. Correlation between adjacency matrix columns.

Among the above distance measures for nodes, the **Geodesic Distance** is the most common one.

There are a few intuitions that help with clustering in the network, for example:

1. **Betweenness Centrality:** It is associated with the edges through which most paths go through. Intercommunity edges have a large value of edge betweenness because many shortest paths connecting vertices of different communities will pass through them. In a network, if these edges are removed, then the remaining nodes will form the requisite clusters.

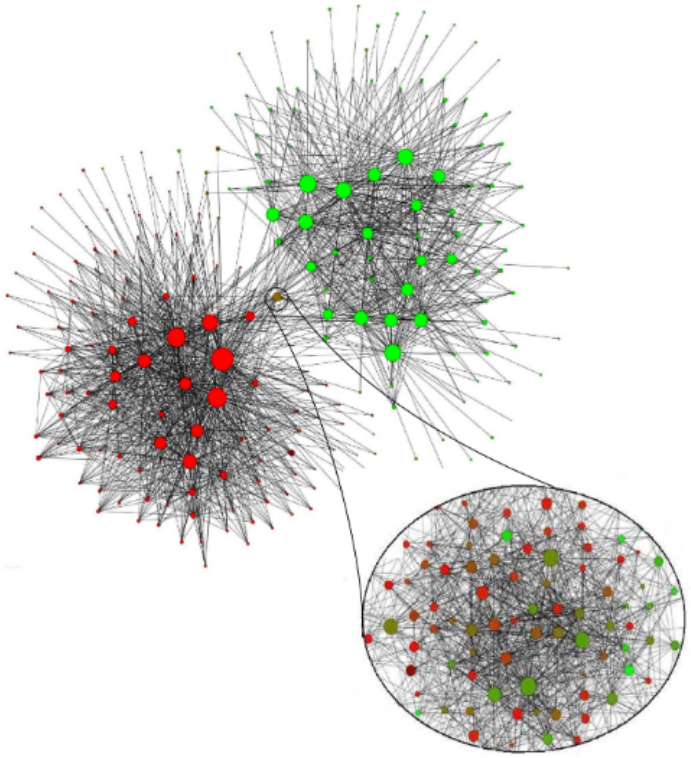
2. **Modularity maximization:** Modularity is the most popular quality function, i.e., a function that assigns a number, as a quality measure, to each partition of a network. Networks with a high modularity score will have many connections within a community but only a few pointing outwards to other communities.

To perform community detection in networks, two common methods are:

1. **The Algorithm of Girvan and Newman:** This method is similar to **divisive hierarchical clustering,** where we iteratively remove the edges with the highest centrality. This method is useful but very slow in practice.

2. **The Louvain Method:** It is similar to **agglomerative hierarchical clustering**, where it uses modularity as a measure of similarity. It iteratively merges pairs of clusters that are connected by more edges than expected if the edges were randomly distributed. It is extremely fast and provides a decomposition of networks into communities for different levels of the organization.

The below figure shows the application of the Louvain method on a **Belgian mobile phone network** with 2M customers.



We observe that the Louvain method created two clear communities (clusters) of French-speaking people (highlighted by the red color) and Dutch-speaking people (highlighted by the green color). The connections between the two communities are those people who speak both languages.

References

- For Clustering:
    - Chapter 14 in T.Hastie, R.TIbshirani, & J. Friedman, The elements of statistical learning: Data mining, inference, and prediction. Springer, 2009.
- For community detection in networks:
    - V.D. Blondel, et al. Fast unfolding of communities in large networks. Journal of Statistical Mechanics: Theory and experiment 10, 2008.
    - S.Fortunato. community detection in the graph. Physics Reports 486, 2010.
    - Lecture notes on Laplacian and spectral clustering (prominent method not discussed in this module) by T . Roughgarden & G. Valiant: http://web.stanford.edu/class/cs168/1/111/pdf
- For Clustering images - https://www.geeksforgeeks.org/clustering-in-machine-learning/