Data Mining

Moustafa ElBadry

A thesis submitted in fulfillment of the requirements for
the degree of Bachelor of Arts in Mathematics

Department of Mathematics and Computer Science
Whitman College
2016

Abstract

Data mining is the art and science of discovering new, useful and fruitful relationships in data. This paper investigates unsupervised learning models of data mining. It sheds the light on various techniques used to find meaningful patterns in data, and analyze each technique to highlight its advantages and limitations. Furthermore, the paper gives detailed overview and analysis for the algorithms used in unsupervised learning models.

Acknowledgements

This paper wouldn't be possible without the continuous help and support from my thesis adviser Professor Hundley and from Professor Balof. Thank you for your constructive feedback and guidance that gave me the opportunity to learn, develop new ideas and complete this work. Thanks to the Mathematics and Computer Science Department for continuously providing students with the opportunity to challenge themselves, learn and grow on both a personal and professional level. Finally, thanks to Katie Gillespie for her contributions to editing this paper.

Contents

List of Figures

Chapter 1

Introduction

Data mining is the science of converting a large amount of non-meaningful data into meaningful structures or outputs that can be used to make predictions. Data mining has two main forms: supervised learning and unsupervised learning. Supervised learning is concerned with using data to derive a function that best approximates a desired output from input data vectors. In other words, supervised learning aims to predict a target value y_i given features $x_{i,j}$. The diagram below illustrates the idea:

$$\begin{array}{rcl} x_{11}, x_{12}, x_{13}, ..., x_{1n} & \Longrightarrow & y_1 \\ x_{21}, x_{22}, x_{23}, ..., x_{2n} & \Longrightarrow & y_2 \\ x_{31}, x_{32}, x_{33}, ..., x_{3n} & \Longrightarrow & y_3 \\ . & \Longrightarrow & \\ . & \Longrightarrow & \\ . & \Longrightarrow & \\ x_{m1}, x_{m2}, x_{m3}, ..., x_{mn} & \Longrightarrow & y_m \end{array}$$

Unsupervised learning is concerned with understanding patterns of data. There is no predetermined output or approximations that we are trying to match the input data objects to. In other words, we are trying to describe how $x_1, x_2, x_3, ..., x_n$ relate to each other. The diagram below illustrates the idea of unsupervised learning:

Unsupervised learning algorithms classify data into clusters that best fit how the data relate to each other (See Figure 1.1 for unsupervised data clustering example). A **cluster** is defined as a group or category that contains objects which share similar characteristics. The center of a cluster is called

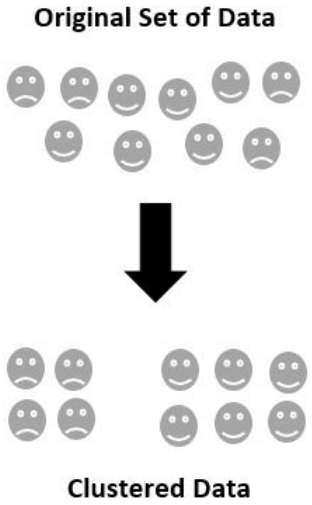

Figure 1.1: Unsupervised data clustering example

the **centroid**. The output clusters will vary depending on the characteristics we are using to classify our data points. In this paper we will focus on analyzing different unsupervised learning techniques, understanding how different unsupervised learning techniques relate to each other, and developing efficient algorithms for various unsupervised learning techniques.

Chapter 2

Partitioning Method

In this chapter we will explore the **k-means** unsupervised data mining method to partition numerical data into clusters. The k-means clustering method aims to partition n observations into k clusters such that each observation belongs to the cluster with the nearest mean. This will lead to partitioning the data into **Voronoi cells**. To understand the concept of Voronoi cells assume that $\{c^{(i)}\}_{i=1}^{k}$ are points in R^n. Therefore, $\{c^{(i)}\}_{i=1}^{k}$ form k Voronoi cells, where the jth cell is defined as the set of points that are closer to cell j than any other cluster:

$$V_j = \{x \in R^n | ||x - c^{(j)}|| \leq ||x - c^{(i)}||, i = 1, 2, \ldots, k\}$$

Note that the points $\{c^{(i)}\}_{i=1}^{k}$ are called cluster centers. Figure 2.1 shows a sample Voronoi diagram sketched by hand.

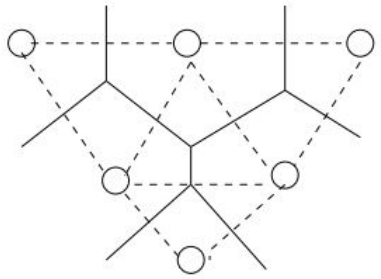

Figure 2.1: Sample Voronoi diagram sketched by hand

2.1 Partitioning Method(K-Means)

The partitioning method is a data mining technique that works only with numerical sets of data. The partitioning method tries to find patterns within a given sample of numerical data to analyze if those patterns can help us predict future outcomes. The partitioning method divides the given numerical data into clusters based on how far the data points are from each other in Euclidean n-space.

Euclidean distance is the length of a straight line drawn between two points in Euclidean space. For example, if x and y are two points in Euclidean n-space, then the distance between $\mathbf{x} = x_1, x_2, ..., x_n$ and $\mathbf{y} = y_1, y_2, ..., y_n$ is defined as

$$d(\mathbf{x}, \mathbf{y}) = \sqrt{(x_1 - y_1)^2 + ... + (x_n - y_n)^2}$$

The classic algorithm used to create clusters using the partitioning method is the k-means algorithm. Here is how the k-means algorithm works:

- The k-means algorithm aims to partition a given set of numerical data $\{x_1, ..., x_n\}$ into k clusters. The number of clusters k has to be defined before the algorithm runs.

- The k-means algorithm will select k points randomly from the given set of data points.

- After identifying the k points they will be considered as the initial clusters' centers, the k-means algorithm use the Euclidean distance formula defined above to calculate the distance between each point x_i in the given data sample and each k centroid.

- After calculating the distance between each data point x_i and each centroid, the data points are assigned to the centroids that they are closest to in terms of distance. The data points that belong to one centroid all together are called a cluster.

We want to chose centroids which create clusters that best fit the data. To do this we will analyze the Euclidean distance formula which we used to compute the distance between the centroids and the data points. Our goal is to find a point in our data sample that will minimize the distance between the points x_i in the sample and the centroid $\mathbf{c} = c_1, c_2, ...c_n$. The Euclidean

distance formula is defined as:

$$d(\mathbf{x}, \mathbf{c}) = \sqrt{(x_1 - c_1)^2 + ... + (x_n - c_n)^2} = \sqrt{\sum_{i}^{n} (\mathbf{x}_{i=1} - \mathbf{c}_i)^2}$$

To minimize this function we will take the first derivative and set it equal to 0. Note that the minimum of

$$\sqrt{\sum_{i=1}^{n} (\mathbf{x}_i - \mathbf{c}_i)^2}$$

is the same as the minimum of

$$\sum_{i}^{n} (\mathbf{x}_i - \mathbf{c}_i)^2$$

This is refereed to in Statistics as the **sum of squares of error**. The sum of squares error is defined as the sum of the squares of residuals. Residuals are the deviations of predicted values of data from the actual empirical values of data. The formula for the sum of squares error is defined by:

$$\sum_{i}^{n} (x_i - \overline{x})^2$$

where n is the number of observations x_i is the value of the ith observation and finally $\overline{x}$ is the mean of all the observations in the data sample.

Therefore, we will use

$$\sum_{i}^{n} (\mathbf{x}_i - \mathbf{c}_i)^2$$

in our computation for simplicity.

$$\frac{d}{dc} \sum_{i}^{n} (\mathbf{x}_i - \mathbf{c}_i)^2 =$$

$$\sum_{i}^{n} -2 \cdot (\mathbf{x}_i - \mathbf{c}_i)$$

Now we will equate it to 0, and solve for c_i.

$$(x_1 - c_1) + (x_2 - c_2) + ... + (x_n - c_n) = 0$$

$$x_1 - c_1 + x_2 - c_2 + ... + x_n - c_n = 0$$

$$x_1 + x_2 + ... + x_n - nc_n = 0$$

$$x_1 + x_2 + ... + x_n = nc_n$$

$$c_i = \frac{x_1 + x_2 + ... + x_n}{n} = \overline{\mathbf{x}}$$

Therefore, we can conclude that to minimize the distance between the points in a data set and the centroids, we should choose our centroids to be equal to the mean value of the data points in a given cluster.

To ensure that the k-means algorithm has partitioned our data into the best fit clusters, we carry out two more steps:

- We compute the mean value of the points in our initial k clusters, then we will choose the mean point in each cluster to be our new centroid.
- We compute the distance between all the points in our data samples and the new centroids, then assign our data points to the centroids that they are closest to.

We will repeat the last two steps until the mean value of the points in each cluster is as close as possible to the value of the centroid of that cluster.

Chapter 3

Hierarchical data mining

Hierarchical clustering is a method of data mining which seeks to build a hierarchy of clusters. One of the problems of the partitioning method using the k-means algorithm is that we needed to predefine the number of clusters into which we want our data will be divided. This approach can be problematic because it is not always clear what number of clusters will best fit the scale we are interested in. The hierarchical clustering solves this problem, because it shows the hierarchy of all the possible clusters on each scale. Therefore, if we look at the data produced by the divisive approach we can see the best fit number of clusters on each scale possible. The two main approaches of hierarchical clustering are agglomerative and divisive.

3.1 Distance Between Clusters

Before we dive into analyzing the agglomerative and divisive approaches we will first define three different methods we can use to measure the distance between two clusters. We will give a formal definition for the singleton link, the complete link and the average link methods. Assume that we have 2 clusters namely c_1 and c_2. We will define each cluster as the following: $c_1 = \{x_1, x_2, x_3, ..., x_n\}$ and $c_2 = \{y_1, y_2, y_3, ..., y_n\}$.

- **The singleton link** approach define the distance between c_1 and c_2 as

  $$D(c_1, c_2) = min_{x_i \in c_1, y_i \in c_2} D(x_i, y_i)$$

  where D stands for Euclidean distance. In other words, the singleton link approach looks for the two closest points where each point is in a

different cluster, and then uses this distance as the distance between the two clusters.

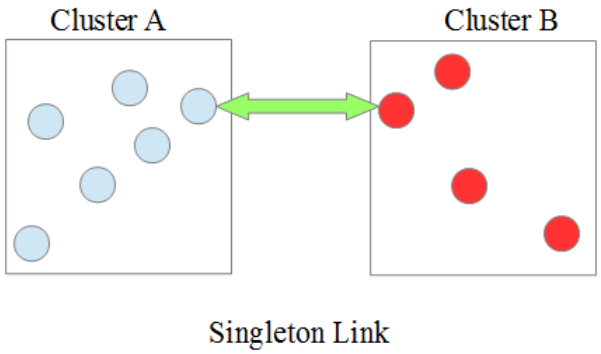

Figure 3.1: Illustrative diagram for the singleton link approach

- **The complete link** approach defines the distance between c_1 and c_2 as

$$D(c_1, c_2) = max_{x_i \in c_1, y_i \in c_2} D(x_i, y_i)$$

where D stands for distance. In other words, the complete link approach looks for two which are the furthest apart where each point is in a different cluster, and then uses this distance as the distance between the two clusters.

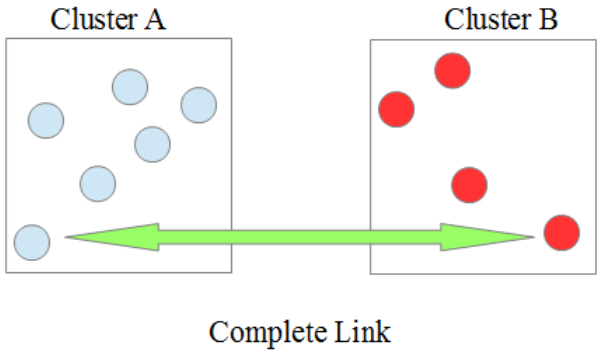

Figure 3.2: Illustrative diagram for the complete link approach

- **The average link** approach define the distance between c_1 and c_2 as

$$D(c_1, c_2) = \frac{1}{|c_1|}\frac{1}{|c_2|}\sum_{x_i \in c_1}\sum_{y_i \in c_2} D(x_i, y_i)$$

  where D stands for distance. In other words, the average link approach averages all pairwise distances and uses the result as the distance between the two clusters.

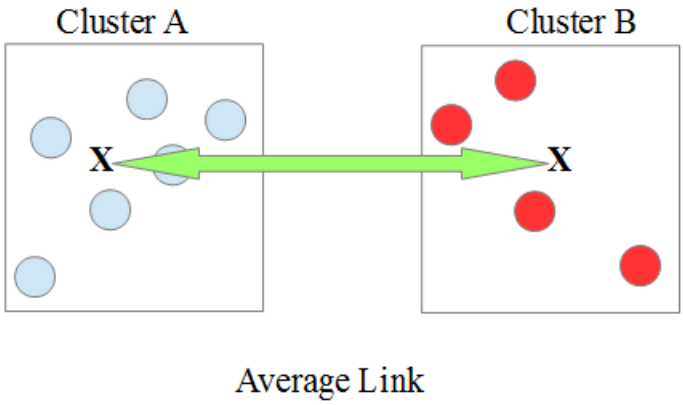

Figure 3.3: Illustrative diagram for the average link approach

3.2 Agglomerative Approach

Agglomerative clustering is also called the bottom up approach we start with each point as a single cluster and then keep merging clusters together until we end up with one cluster. The agglomerative approach will ensure that any nearby points will end up in the same cluster. Here is a simple explanation of how the agglomerative approach works:

- Each cluster begins with one data point $c_i = \{x_i\}$
- We apply the following algorithm recursively until one cluster remains:
  - Find a pair of clusters that is closest to each other

$$min_{i,j} D(c_i, c_j)$$

We will find the distance between the two clusters using the singleton link method which we explained earlier in the "Distance Between Clusters" section.

- After we finding the two points that are the closest to each other c_i and c_j, merge them into one cluster which we will call c_{i+j}.
- Now we will remove the clusters c_i and c_j from the collection and add the cluster c_{i+j}

3.3 Divisive Approach

Divisive approach is also called the top down approach we start with one cluster that contains all our objects or points and then divides the clusters iteratively until we end up with singleton clusters. Figure 3.1 illustrates how the divisive approach works in comparison with the agglomerative approach.

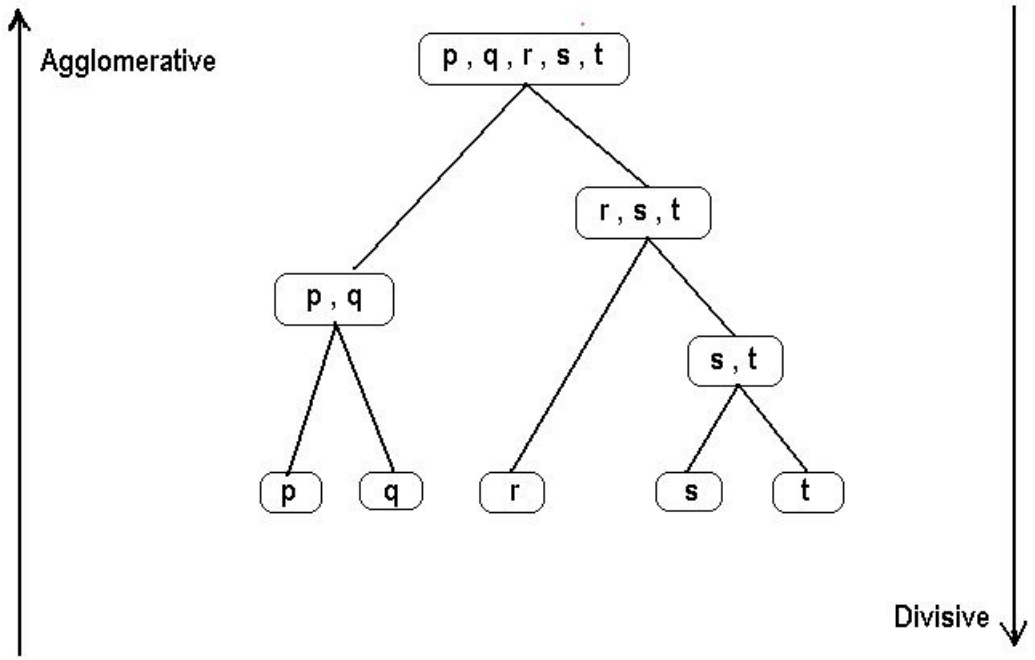

Figure 3.4: Divisive approach in comparison with the agglomerative approach[10]

Let's look at how the divisive approach works:

- First run the k-means algorithm with $k = 2$ on the single cluster that contains all of our data points $\{x_1, x_2, x_3, ..., x_n\}$. This will result in dividing our single cluster into 2 clusters.
- Run the k-means algorithms with $k = 2$ on each of the two clusters. This will result in dividing the 2 clusters into 4 clusters.
- Continue to run the k-means algorithm with $k = 2$ on each cluster until we end up with singleton clusters.

The downside about this method is that it uses the greedy approach, which means that at some levels of the hierarchy two points that are nearby each other can end up in different clusters. Once two points are in different clusters, they can't be grouped again in the same cluster which can possibly result in the loss of valuable data patterns.

3.4 Lance William Algorithm

The **Lance William algorithm** is concerned with finding an efficient and effective method for the agglomeration hierarchical clustering procedure. The Lance William algorithm aims to optimize the value of an objective function at each step of the clustering procedure to choose the next pair of clusters that will be merged together. The term "objective function" refers to a mathematical function which will be optimized. The objective function used by the Lance William algorithm can be different each time the algorithm runs, but the objective function should always reflect the researcher's interest. For instance, the researcher could define the objective function for the Lance William Algorithm to be the sum of squared errors. The **sum of squared errors** in Statistics is defined as the squared difference between each observation and its group mean. This function is used to measure the variation within each cluster.

$$Sum\ of\ Squared\ Errors\ (SSE) = \sum_{i=1}^{n}(x_i - \overline{x})^2$$

This approach of the Lance William Algorithm is called Ward's method. Ward's method is one of the many approaches that could be used with the Lance William Algorithm. The Lance William algorithm works as follows:

- Each data point $\{x_1, x_2, x_3, ..., x_n\}$ will be considered a cluster $\{c_1, c_2, c_3, ..., c_n\}$.
- The Lance William algorithm now will measure the distance between all the clusters. There are different ways to measure the distance between clusters. Among the most popular methods are singleton linkage, complete linkage, group linkage and Ward's method.
- When the Lance William algorithm identifies two clusters C_i and C_j to be the closest clusters, then the next step is to merge those two clusters into C_{i+j}, so they become one cluster.
- The next step is to update the distance between the cluster C_{i+j} and the remaining n clusters. In Lance and William's paper, they introduced a recursive formula that can be used to update the distance between C_{i+j} and the remaining n clusters:

$$D_{C_n,C_{i+j}} = \alpha_i D_{C_n,C_i} + \alpha_j D_{C_n,C_j} + \beta D_{C_i,C_j} + \gamma |D_{C_n,C_i} - D_{C_n,C_j}|$$

where the parameters α_i, α_j, β and γ are determined by the nature of the strategy used in the Lance William algorithm. Here is a table for the parameter values based on the distance measuring method used:

Method	α_i	α_j	β	γ
Singleton Linkage	0.5	0.5	0	-0.5
Complete Linkage	0.5	0.5	0	0.5
Group Linage	$\frac{n_i}{n_i+n_j}$	$\frac{n_j}{n_i+n_j}$	0	0
Ward	$\frac{n_i+n_k}{n_i+n_j+n_k}$	$\frac{n_j+n_k}{n_i+n_j+n_k}$	$\frac{-n_k}{n_i+n_j+n_k}$	0

The n_i, n_j and n_k in the table refer to the size of clusters C_i, C_j and C_k respectively.

We will show why the Lance William coefficients for the singleton linkage make sense. The coefficients for the rest of the methods can be verified using the same approach. First of all, note that our goal is to find the minimum distance between two clusters say C_{i+j} and C_k. We

are looking to minimize $D_{(C_{i+j},C_k)}$. Now if we use the Lance William recursive formula we get:

$$D_{C_k,C_{i+j}} = \alpha_i D_{C_k,C_i} + \alpha_j D_{C_k,C_j} + \beta D_{C_i,C_j} + \gamma |D_{C_k,C_i} - D_{C_k,C_j}|$$

$$D_{C_k,C_{i+j}} = 0.5 \cdot D_{C_k,C_i} + 0.5 \cdot D_{C_k,C_j} + 0 \cdot D_{C_i,C_j} - 0.5 \cdot |D_{C_k,C_i} - D_{C_k,C_j}|$$

$$D_{C_k,C_{i+j}} = 0.5 \cdot (D_{C_k,C_i} + D_{C_k,C_j} - |D_{C_k,C_i} - D_{C_k,C_j}|)$$

Note that D_{C_k,C_i} represent the distance between the cluster C_k and C_i, and D_{C_k,C_j} represent the distance between the cluster C_k and C_j. Since the two clusters C_i and C_j have been merged into the cluster C_{i+j}, then the distances D_{C_k,C_i} and D_{C_k,C_j} represent the maximum and the minimum distance between the cluster C_k and the cluster C_{i+j}. Assume that D_{C_k,C_i} represents the maximum distance, and that D_{C_k,C_j} represents the minimum distance. Therefore, we can write the Lance William recursive formula as:

$$D_{C_k,C_{i+j}} = 0.5 \cdot (Max. + min. - |Max. - min.|)$$

$$D_{C_k,C_{i+j}} = 0.5 \cdot (Max. + min. - Max. + min.)$$

$$D_{C_k,C_{i+j}} = 0.5 \cdot (2 \cdot min.)$$

$$D_{C_k,C_{i+j}} = min.$$

This verifies the Lance William coefficients for singleton linkage method.

- After the Lance William algorithm updates the distance between the cluster C_{i+j} and the remaining clusters C_k, then merges the closest cluster C_k with C_{i+j} to produce a new cluster C_{i+j+k}. Then the Lance William algorithm will repeat step 3 and 4 until we end up with one cluster that contains all the points of the data set.

Chapter 4

Density Based clustering

In this chapter we will move on to another technique in unsupervised learning. We will explore the density based clustering method, analyze how it can work with the hierarchical clustering method and understand what distinguishes the density based method from the other clustering methods we have looked at so far.

4.1 Definitions

- **Density:** Density represent the number of points within a specific radius which we will call ϵ where $\epsilon \in R$
- **Core point:** A point is considered to be a core point if and only if it has more than a specific number of points, which we will call c within a predefined radius ϵ.
- **Border point:** A point is considered to be a border point if and only if it has fewer points than a specified number of points c, within a predefined radius ϵ and at the same time located in the neighborhood of a core point.
- **Noise point:** A point is considered to be a noise point if it is not a core point or a border point.
- **Density Reachability:** We call a point x density reachable from another point y if point x lies within ϵ distance from the point y, at

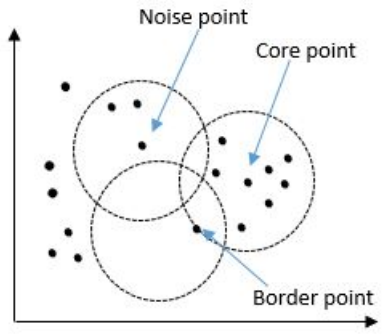

Figure 4.1: Illustrative diagram for density based clustering definitions

the same time y must have sufficient number of points that lie within the distance ϵ.

4.2 Density Based Clustering

The **density based clustering** technique is unsupervised learning method for data mining. The density based method divides data points into clusters based on their concentration around specific points. Those points are called core points. Furthermore, a scientist or a researcher who is using the density based method will need to specify a radius ϵ that will determine the core points in the data set, and determine the clusters. The density based clustering method can also be implemented in conjunction with the hierarchical method. The advantage of implementing these two techniques together is that the scientist working with the data doesn't need to specify a radius to implement the density based method. Rather, they define a starting point which is going to be used as the first radius, an incremental factor which refers to how much the radius will be incremented in each iteration, and finally an end point which will be the last radius used by the density method. The advantage of this approach is that sometimes it is not very clear what radius will allow the best classifications of the data, or what clusters would make better sense. Using the density based method with the hierarchical method algorithms will give an output of multiple radii and the resulted clusters, which can then be evaluated by humans to understand which data patterns give useful results.

The density based clustering method has played a vital role in finding non-linear structure based on the density. The Density Based Spatial Clustering of Applications with Noise (DBSCAN) is the most widely used density based algorithm. Here is how the Density Based Spatial Clustering of Applications with Noise works:

- Consider the data set $X = \{x_1, x_2, x_3, ..., x_n\}$. A preliminary step to cluster this data set with the density based method is to specify two parameters. The first parameter is the radius ϵ, and the second parameter is the minimum number of points (Minpts) required to form a cluster.

- The first step in the DBSCAN algorithm is to choose one point randomly. Then it will extract the neighborhood of this point using the radius ϵ which is predefined by the user.

- Next, the DBSCAN will determine if the initial point chosen has enough points i.e. Minpts around it. If this is the case then the clustering process will start. Otherwise, this initial point will be marked as a noise and DBSCAN will choose another point randomly.

- When DBSCAN finds a point that can be part of the cluster, then its ϵ neighborhood is also considered to be part of the cluster. Then, DBSCAN repeats the process, on all the other points that haven't been visited yet until all the points in the data have been classified.

4.3 Why Density Based Clustering

The density based clustering has few advantages over the partitioning method or the hierarchical method which we have discussed so far. Among the advantages of the density based clustering are:

- The Density based clustering algorithms do not require prior specifications for the number of clusters.

- The Density based clustering method is the only method we have seen so far that is able to identify noise data while clustering.

- The Density based clustering algorithm DBSCAN is known for its ability to find arbitrarily size and arbitrarily shaped clusters.

4.4 Examples of Density Based Clustering

Below is an example showing the original data points and the core, border and outliers identified by DBSCAN clustering.

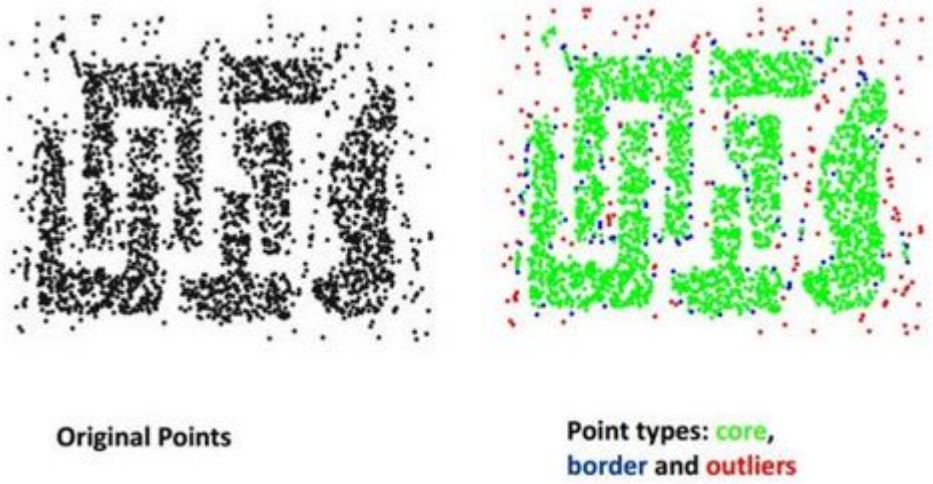

Figure 4.2: Illustrative diagram for Density Based Clustering[13]

In Figure 4.2 we have used $\epsilon = 10$ and $Minpts = 4$. The following example show a case when density based clustering did not work well. We will show the original data points and the resulted clusters and explain whey density based clustering did not work very well in this scenario.
The reason density based clustering did not work well with this set of data point is that density based clustering cannot handle varying densities which result in difficulties in determining the correct set of parameters.

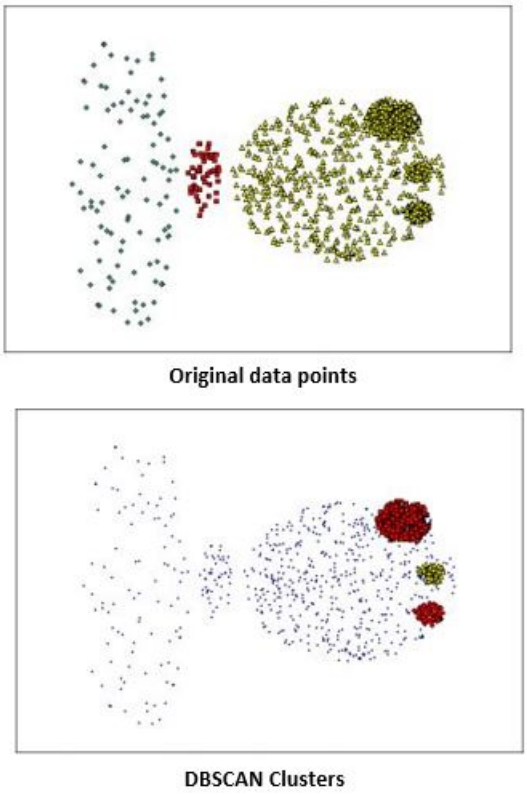

Figure 4.3: Illustrative diagram for Density Based Clustering problems[13]

Chapter 5

Neural Networks

In this chapter we will focus on giving a formal definition of neural networks and explain how neural networks are used in data mining. This chapter will help us develop critical concepts that will be used in the coming two chapters when we discuss competitive learning, cooperative learning and Kohonen maps.

5.1 Definitions

Neural Networks: neural networks (artificial neural networks) are an information processing paradigm that is inspired by the way biological nervous systems process information. An artificial neural network handles large number of highly interconnected simple processing elements (which are analogous to neurons in Biology) working together to solve specific problems. Figure 5.1 illustrates how a simple neural network works:

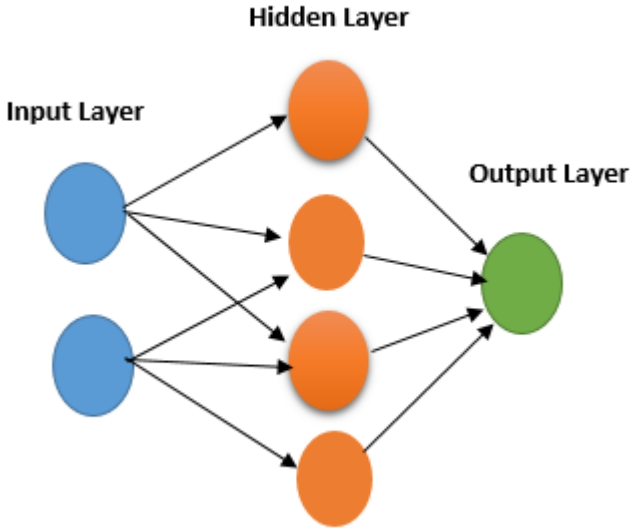

Figure 5.1: Simple Neural Network

5.2 How a Simple Neural Network Works

- In a simple artificial neural network the basic computational element is most often called a node. The node function is to receive input from some other node or from an external source like a file or a database.
- Each single input going to a node has a defined weight w, which can be modified in the algorithms to model synaptic learning.
- The next step of the process is that the node computes some function $f(x)$ of the weighted sum of its inputs where $x = \sum_j w_{ij} y_j$. The diagram below illustrates this process:

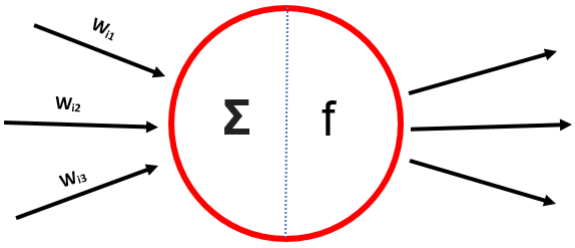

Figure 5.2: Neural network function example $y_i = f(\sum_j w_{ij} y_j)$

- A neural network is usually structured into an input layer of neurons, one or multiple hidden layers, and finally one output layer. This structure illustreated in Figure 5.1. Note that neurons belonging to adjacent layers are most often fully connected and the different types are usually identified by the different kind of topologies modified for the connections and the choice of the activation function. The values of the functions which are associated with the different connections are what we call the weights.

The whole point of using neural networks is the fact that in order for the network to yield appropriate outputs for the given inputs the user entered in the system, the weight must be set to suitable values. Choosing a cost function or a weight is very important process in neural networks. A specific cost function or a weight will be used either because of its desirable properties such as convexity, or because it arises naturally from a particular formulation of the problem.
Neural networks have several types, among the most poplar types are:

- **Self organized**: These type of neural networks are also known as Kohonen Maps, and they have applications in unsupervised learning which we will discuss in the last chapter.

- **Recurrent**: These refer to simple recurrent neural networks or what are also known as Hopfield networks.

- **Feedforward**: These kind of neural networks are also known as single layer perceptrons.

Chapter 6

Competitive Learning

In the coming two chapters we will analyze and implement Kohonen maps, also known as Self Organizing Maps (SOM). Kohonen maps are form of unsupervised learning in artificial neural networks. The advantage of Kohonen maps over any other unsupervised learning technique is that they combine two mechanisms of unsupervised learning known as competitive learning and cooperative learning to produce more accurate data clusters. In this chapter we will discuss the competitive learning mechanism before we dive in Kohonen maps.

6.1 Competitive Learning

Competitive learning is a form of unsupervised learning in artificial neural networks, in which the output neurons compete among themselves to be activated. At the end, only one neuron will be activated at any one time. The activated neuron is refereed to as the **winning neuron**. There are three essential elements to competitive learning:

- The initial set of neurons have randomly distributed synaptic weights, which will allow for different responses to a given set of data input. In other words, this principle is saying that our initial set of neurons will hold random values of all the attributes in the data set we are trying to cluster, and will respond differently to different data input points. For example, suppose the data set we are tying to cluster is a set of colors, and each color in the set has red, green and blue attributes (See Figure 6.1). Therefore, when using the competitive learning form of

unsupervised learning to cluster this set of colors, each neuron in our initial set of neurons will be assigned random attributes of red, green and blue colors (See Figure 6.2).

Figure 6.1 show an example of a data set of colors, where each color has red, green and blue attributes.

$$\text{Colors data set} = \begin{matrix} R & G & B \end{matrix} \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 0 & 0 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \\ 1 & 1 & 1 \end{bmatrix} = \begin{matrix} Colors \end{matrix} \begin{bmatrix} Black \\ Blue \\ Green \\ Cyan \\ Red \\ Pink \\ Yellow \\ White \end{bmatrix}$$

Figure 6.1: Example of a data set that can be handled using competitive learning

Figure 6.2 shows an initial 4×4 set of neurons that has random attributes of red, green and blue (R G B), where R, G and B are integers.

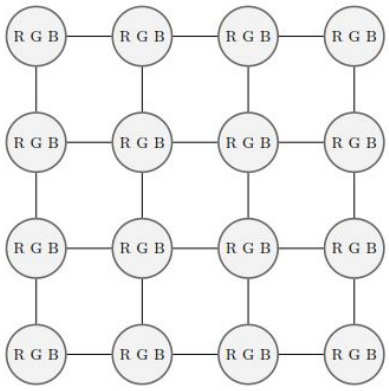

Figure 6.2: Example of a 4×4 neural network using the data set in Figure 6.1

- Each neuron has a limit imposed on its strength. This limit is imposed by the random values assigned to the neuron and the input point being compared to the neuron. In other words, the weights of each neuron and the input data points determine the competitivity of each neuron.

- We need a mechanism that allows the neurons to compete to find a winner neuron and activate the right response to a given input. The mechanism will be used to compare each input data point with all the neurons in the artificial neuron network, to activate the right neuron in the network and declare it as a winner. Figure 6.3 illustrates how this process works.

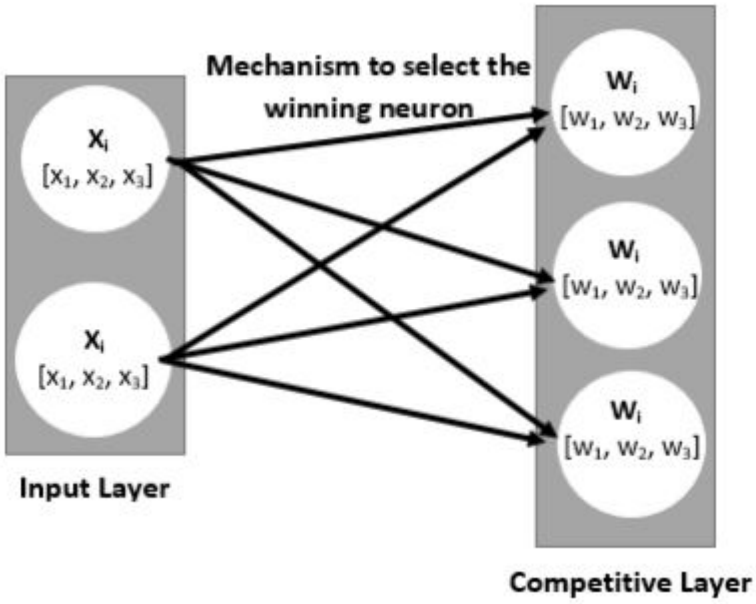

Figure 6.3: Illustrative diagram to show how the mechanism of selecting a winning neuron works

The input layer in Figure 6.3 show two input data points, and the competitive layer show all the neurons we have in a given network. Each input point and neuron is represented by a vector that has three entries. Note that each vector has three entries because each data point in our data set has three attributes: red, green and blue. The comparison mechanism compares each input data point with each neuron in the

network to find the winning neuron. Note that the mechanism implemented can vary depending on the data set we are handling and the desired output. We will implement the Euclidean metric $\sqrt{(\mathbf{x}-\mathbf{w})^{\mathbf{2}}}$ where $\mathbf{x}$ is a data vector and $\mathbf{w}$ is a neuron weights vector, in the algorithm we will develop for the competitive learning process.

6.2 Competitive Learning Algorithm

We will now turn our attention to developing an algorithm for competitive learning, and use it in an example.

- First, the algorithm will run on the data set to find out how many attributes each data point has and the same number of attributes will be assigned to each neuron. For instance, if our data set is the colors, then each data point (color) will have 3 attributes: red, green and blue. Therefore, each neuron in the network will have red, green and blue attributes.

- The algorithm must be provided with the number of neurons in the network that it should initialize. The more neurons you initialize, the more accurate the clusters will be but also, the more computational power you will need. In addition to initializing the number of neurons in the network, the algorithm must also be provided with lower and upper bounds for each attribute of each neuron. Finally, the learning rate α must also be defined. The **learning rate** refers to how fast the neurons attributes change value.

- Random values will be assigned to all the attributes/weights of each neuron, taking in consideration the lower and upper bounds defined in the previous step.

- Using the Euclidean distance formula $\sqrt{(\mathbf{x}-\mathbf{w})^{\mathbf{2}}}$ where $\mathbf{x}$ is a data point vector and $\mathbf{w}$ is a neuron weight vector, the distance between each data point and each neuron will be measured. For each data point, the closet neuron in distance will be announced as the winning neuron for this data point.

- In the next iteration, the winning neuron's i weights will be updated by the following formula:

$$w_i = w_i + \alpha \cdot (x_j - w_i)$$

  where $\mathbf{x}$ is the data point vector.

- The last two steps of the algorithm will be repeated for all the data point vectors.

Now, we will use this algorithm in an example to further illustrate how it works. Let's say that we have 2 data points and that each of them has two attributes (a length and thickness). Our data points are:

$$\mathbf{x_1} = \begin{bmatrix} 10 \\ 50 \end{bmatrix}, \mathbf{x_2} = \begin{bmatrix} 63 \\ 72 \end{bmatrix}$$

Suppose that we initiated our artificial neural network to have 3 neurons. We will set the lower bound of both the length and the thickness to be 1 and the upper bound to be 100. Now assigning random weights to our three neurons we get:

$$\mathbf{w_1} = \begin{bmatrix} 3 \\ 80 \end{bmatrix}, \mathbf{w_2} = \begin{bmatrix} 83 \\ 99 \end{bmatrix}, \mathbf{w_3} = \begin{bmatrix} 16 \\ 12 \end{bmatrix}$$

Now, we will calculate the distance between our first point x_1 and all the neurons in the network using the Euclidean distance formula:

$$d_1 = \sqrt{(x_1 - w_1)^2} = \sqrt{(10-3)^2 + (50-80)^2} = 30.8$$
$$d_2 = \sqrt{(x_1 - w_2)^2} = \sqrt{(10-83)^2 + (50-99)^2} = 87.9$$
$$d_3 = \sqrt{(x_1 - w_3)^2} = \sqrt{(10-16)^2 + (50-12)^2} = 38.4$$

Since w_1 is the most similar to our data point x_1, w_1 is the winner neuron. In the next iteration we will update the weights of this neuron. For simplicity we will let the learning rate $\alpha = 0.25$

$$w_{11} = w_{11} + \alpha \cdot (x_{11} - w_{11}) = 3 + (0.25 \cdot (10-3)) = 3 + 1.75 = 4.75$$
$$w_{12} = w_{12} + \alpha \cdot (x_{12} - w_{12}) = 80 + (0.25 \cdot (50-80)) = 80 - 7.5 = 72.5$$

Therefore, we have:

$$\mathbf{w_1} = \begin{bmatrix} 4.75 \\ 72.5 \end{bmatrix}$$

Now we will turn our attention to the second data point:

$$\mathbf{x_2} = \begin{bmatrix} 63 \\ 72 \end{bmatrix}$$

We then calculate the distance between our second point x_2 and all the neurons in the network using the Euclidean distance formula:

$$d_1 = \sqrt{(x_2 - w_1)^2} = \sqrt{(63-3)^2 + (72-80)^2} = 60.5$$
$$d_2 = \sqrt{(x_2 - w_2)^2} = \sqrt{(63-83)^2 + (72-99)^2} = 33.6$$
$$d_3 = \sqrt{(x_2 - w_3)^2} = \sqrt{(63-16)^2 + (72-12)^2} = 76.2$$

Based on the results we can conclude that the winning neuron is w_2. In the next iteration we will update the weights of w_2.

$$w_{21} = w_{21} + \alpha \cdot (x_{21} - w_{21}) = 83 + (0.25 \cdot (63 - 83)) = 83 - 5 = 78$$
$$w_{22} = w_{22} + \alpha \cdot (x_{22} - w_{22}) = 99 + (0.25 \cdot (72 - 99)) = 99 - 6.75 = 92.25$$

Therefore we have:

$$\mathbf{w_2} = \begin{bmatrix} 78 \\ 94.75 \end{bmatrix}$$

Finally our updated set of neurons is as the following:

$$\mathbf{w_1} = \begin{bmatrix} 4.75 \\ 72.5 \end{bmatrix}, \mathbf{w_2} = \begin{bmatrix} 78 \\ 94.75 \end{bmatrix}, \mathbf{w_3} = \begin{bmatrix} 16 \\ 12 \end{bmatrix}$$

In conclusion, we can see that as more data is received by the competitive learning algorithm each neuron converges to the center of the cluster that it has come to represent and reacts more strongly for inputs in this cluster and more weakly for inputs in other clusters.

Chapter 7

Kohonen Maps

In this chapter we will focus on developing an algorithm for Kohonen maps and analyze why the Kohonen maps algorithm represents a better alternative for algorithms that only use competitive learning.

7.1 Cooperative Learning

In competitive learning, we saw how the winning neuron adapts its weight to become more similar or closer to the data point being examined. The learning process in competitive learning only affects the winning neuron and no other neurons. Cooperative learning expands on competitive learning in the sense that it does not only adapt the weights of the winning neuron, but it also adapts the weights of the neurons within a predefined neighborhood of the winning neuron. There are various mechanisms that can be implemented to define the neighborhood around a winning neuron and to adapt the weights of the winning neuron and its neighborhood. In this chapter, we will be examining how Kohonen maps implement the cooperative learning concepts in addition to competitive learning.

7.2 Kohonen Maps (Self-Organizing Maps)

Kohonen maps use both competitive learning and cooperative learning to cluster a given data set. Kohonen maps have supervised and unsupervised learning applications, but we will only be focusing on the unsupervised learning algorithms and applications. We will develop our Kohonen maps algo-

rithm in three processes: the competitive process, the cooperative process and the adaptive process. The Kohonen maps algorithm works as follows:

- **Competitive Process:** We have covered competitive learning in the last chapter but we will go over the process briefly to understand how competitive learning fits in Kohonen maps. If our input space has D input units, then we can write our input pattern as $\mathbf{x} = \{\mathbf{x_i} : \mathbf{i} = \mathbf{1}, ..., \mathbf{D}\}$, and the weights of the neurons in the competitive layer can be expressed as $\mathbf{w} = \{\mathbf{w_j} : \mathbf{j} = \mathbf{1}, ..., \mathbf{N}\}$ where N is an integer that represents the total number of neurons in the network. As we have explained in the last chapter each weight must hold the same number of attributes as the input data points. We define a discriminant function as the Euclidean distance between input vector $\mathbf{x_i}$ and the weight vector $\mathbf{w_j}$ for each neuron:

  $$\sqrt{(\mathbf{x_i} - \mathbf{w_j})^\mathbf{2}}$$

  Basically, the neuron whose weight vector is most similar to (i.e. the closest) the input vector will be chosen as the winning neuron. Continuing in this fashion the input space will be mapped to the discrete output space of neurons by through competition between the neurons.

- **Cooperative Process:** The idea of cooperative learning started in Biology, where several studies have shown that there is a lateral interaction within a set of excited neurons[13]. When a neuron fires, neurons in the neighborhood closest to it tend to get excited more than those further away.

  Our goal is to define a similar neighborhood for the neurons in our network. If d_{ij} is the lateral distance between neurons i and j on a grid of neurons, we can define our neighborhood as:

  $$T_{i,j} = exp\left(\frac{-d_{ij}^2}{2\beta^2}\right)$$

  This function has critical properties that makes it will suited for the cooperative learning process:

  - The function is maximal at the winning neuron.
  - The function is symmetrical about the winning neuron.

- The function decreases monotonically to zero as the distance goes to infinity.
- The function is independent of the location of the winning neuron.

One of the important features of Kohonen maps is that the size β of the neighborhood needs to decrease with time. One way to achieve that is using a time dependence in an exponential decay as a scaling factor:

$$\beta(t) = \beta_0 \cdot exp\left(\frac{-t}{\tau_\beta}\right)$$

- **Adaptive Process:** Now we will turn our attention to the adaptive process for Kohonen maps. The adaptive process, or what is known as the learning process, is the process by which the outputs become self-organized and the feature map between the input data and the outputs is formed. The advantage that the cooperative learning introduces in the adaptive process is that not only the winning neuron gets its weights updated, but all the neurons in the neighborhood of the winning neuron will get their weights updated as well. Updating the weights of the winning neuron and the neurons in the neighborhood can be achieved using the formula below:

$$\Delta w_j = \sigma(t).T_{ij}(t) \cdot (x_i - w_j)$$

where σ is a time dependent learning rate that we will define as

$$\sigma(t) = \sigma_0 \cdot exp\left(\frac{-t}{\beta_\sigma}\right)$$

The effect of each weight update is to move the weight vectors w_j of the winning neuron and its neighborhood towards the input vector x. Repeating this process will lead to training data and producing a topological ordering.

7.3 Kohonen Maps Algorithms

In this section we aim to develop an algorithm for Kohonen maps using our analysis for the competitive, cooperative and adaptive processes. The algorithm runs as follows:

- Initialize the number of neurons in the network, and define the lower and upper bounds for each attribute the neurons have.
- Choose a data point randomly x_i from $\mathbf{x}$
- Use the Euclidean distance formula

  $$\sqrt{(x_i - w_j)^2}$$

  to measure the distance between data point x_i and all the neurons in the network.
- Choose the closest neuron as the winning neuron and update the weights of the winning neuron and its neighborhood by

  $$\Delta w_j = \sigma(t).T_{ij}(t) \cdot (x_i - w_j)$$

- Repeat all the steps except step 1 until the feature map stops changing.

7.4 Applications of Self Organizing Maps

One of the very interesting projects that has been carried out using self organizing maps (Kohonen maps) is the world poverty map. As we have explained earlier self organizing maps can be used to portray complex correlations in statistical data. Using data from the World Bank statistics of countries in 1992 the world poverty map used 39 indicators describing various quality of life factors such as state of health, nutrition, income, pollution, and educational services to draw the world poverty map. Even though there were 39 different indicators the complex joint effect of these factors can by visualized be organizing the countries using Kohonen maps.

The project found that countries that had similar values of the indicators are in places near each other on the map. Furthermore, the project encoded different clusters on the map with different bright colors so that colors change smoothly on the map display. This has resulted in assigning a color to each country that describes its poverty type in relation to other countries. Figure 7.1 shows a table of the countries that participated in the project. In addition, Figures 7.2 and 7.3 show the results of the project.

The Country Names

AFG	Afghanistan	GTM	Guatemala	NZL	New Zealand
AGO	Angola	HKG	Hong Kong	OAN	Taiwan, China
ALB	Albania	HND	Honduras	OMN	Oman
ARE	United Arab Emirates	HTI	Haiti	PAK	Pakistan
ARG	Argentina	HUN	Hungary	PAN	Panama
AUS	Australia	HVO	Burkina Faso	PER	Peru
AUT	Austria	IDN	Indonesia	PHL	Philippines
BDI	Burundi	IND	India	PNG	Papua New Guinea
BEL	Belgium	IRL	Ireland	POL	Poland
BEN	Benin	IRN	Iran, Islamic Rep.	PRT	Portugal
BGD	Bangladesh	IRQ	Iraq	PRY	Paraguay
BGR	Bulgaria	ISR	Israel	ROM	Romania
BOL	Bolivia	ITA	Italy	RWA	Rwanda
BRA	Brasil	JAM	Jamaica	SAU	Saudi Arabia
BTN	Bhutan	JOR	Jordan	SDN	Sudan
BUR	Myanmar	JPN	Japan	SEN	Senegal
BWA	Botswana	KEN	Kenya	SGP	Singapore
CAF	Central African Rep.	KHM	Cambodia	SLE	Sierra Leone
CAN	Canada	KOR	Korea, Rep.	SLV	El Salvador
CHE	Switzerland	KWT	Kuwait	SOM	Somalia
CHL	Chile	LAO	Lao PDR	SWE	Sweden
CHN	China	LBN	Lebanon	SYR	Syrian Arab Rep.
CIV	Cote d'Ivoire	LBR	Liberia	TCD	Chad
CMR	Cameroon	LBY	Libya	TGO	Togo
COG	Congo	LKA	Sri Lanka	THA	Thailand
COL	Colombia	LSO	Lesotho	TTO	Trinidad and Tobago
CRI	Costa Rica	MAR	Morocco	TUN	Tunisia
CSK	Czechoslovakia	MDG	Madagascar	TUR	Turkey
DEU	Germany	MEX	Mexico	TZA	Tanzania
DNK	Denmark	MLI	Mali	UGA	Uganda
DOM	Dominican Rep.	MNG	Mongolia	URY	Uruguay
DZA	Algeria	MOZ	Mozambique	USA	United States
ECU	Ecuador	MRT	Mauritania	VEN	Venezuela
EGY	Egypt, Arab Rep.	MUS	Mauritius	VNM	Viet Nam
ESP	Spain	MWI	Malawi	YEM	Yemen, Rep.
ETH	Ethiopia	MYS	Malaysia	YUG	Yugoslavia
FIN	Finland	NAM	Namibia	ZAF	South Africa
FRA	France	NER	Niger	ZAR	Zaire
GAB	Gabon	NGA	Nigeria	ZMB	Zambia
GBR	United Kingdom	NIC	Nicaragua	ZWE	Zimbabwe
GHA	Ghana	NLD	Netherlands		
GIN	Guinea	NOR	Norway		
GRC	Greece	NPL	Nepal		

Figure 7.1: Countries that participated in the World Poverty Map project[14]

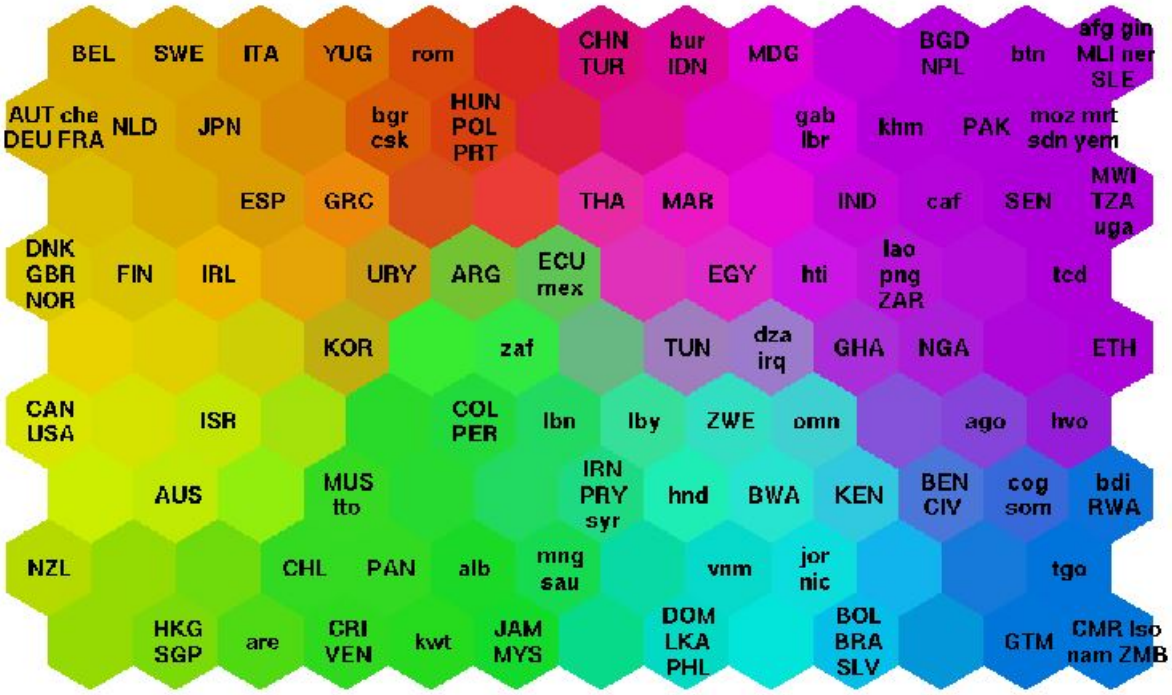

Figure 7.2: Countries organized on a self-organizing map based on indicators related to poverty[14]

As we can see in Figure 7.1 the countries with similar poverty level has been encoded with similar colors to reflect how they compare to each other.

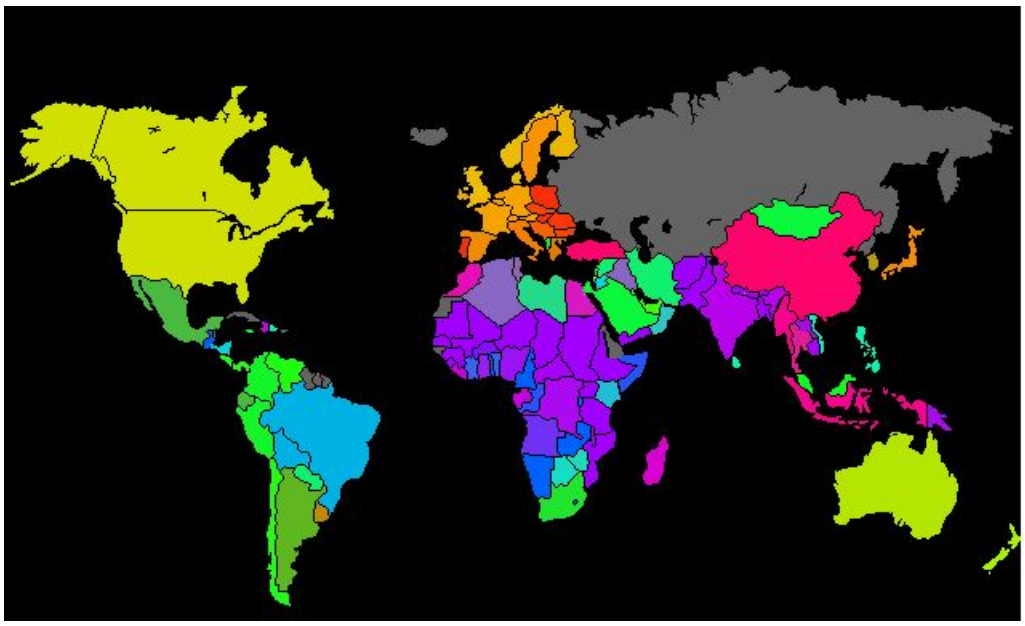

Figure 7.3: A map of the world where countries have been colored with the color describing their poverty type (the color was obtained with the SOM in figure 7.2)[14]

Conclusion

In this paper we have seen different unsupervised learning techniques to manipulate numerical data and classify them into clusters which allow us to analyze how the data points or data vectors relate to each other. In other words the data clusters reveals new properties about the data that were unknown to us before. Our analysis mainly focused on numerical data sets, however the techniques discussed in this paper could be further developed to process other data types such as alphanumeric strings for example. In addition more research could be done on supervised learning techniques to develop mathematical functions that could map the output result to a pre-registered target value given a set of data. Data mining is an endless ocean that always has more discover.

Bibliography

[1] Clustering Lecture 2: Partitional Methods, Jing Gao, SUNY Buffalo

[2] Course notes, Introduction to Mathematical Modeling, D. Hundley, 2008.

[3] Andrew Moore: K-means and Hierarchical Clustering - Tutorial Slides" http://www-2.cs.cmu.edu/ awm/tutorials/kmeans.html

[4] Drake, Jonathan (2012). "Accelerated k-means with adaptive distance bounds".

[5] Clustering Lecture 4: Density-based Methods by Jing Gao and SUNY Buffalob

[6] SCAN: A structural Clustering Algorithm for Networks by Xiaowei Xu, Nurcan Yuruk, Zhidan Fend and Thomas Schweiger

[7] https://onlinecourses.science.psu.edu/stat505/node/138

[8] G. N. Lance and W. T. Williams. A general theory of classificatory sorting strategies. 1. hierarchical systems. Computer Journal, 9:373-380, 1967.

[9] Lecture notes, School of Informatics, Victor Lavrenko and Nigel Goddard, 2014.

[10] http://www.solver.com/xlminer/help/hierarchical-clustering-intro

[11] Supervised and Unsupervised Learning by Ciro Donalek published on April, 2011.

[12] Introduction to Neural Networks : Lecture 16 by John A. Bullinaria, 2004.

[13] Rumelhart, David E., and David Zipser. "Feature discovery by competitive learning." Cognitive science 9.1 (1985): 75-112.

[14] http://www.cis.hut.fi/research/som-research/worldmap.html