Unsupervised Learning

ScholarHat

Last lecture reminder

We learned about:

- Unsupervised Learning - Introduction
- Supervised vs Unsupervised Learning
- K-Mean Clustering - Introduction
- K-Mean Clustering - Simple Python Example
- K-Mean Clustering - Optimal K Value
- Optimal K Value - Python Example

K-Mean Clustering - Image Quantization

Image quantization → Image quantization is a digital image processing technique often used in computer graphics and image analysis. Its goal is to reduce the number of distinct colors used in an image while attempting to maintain its visual appearance as closely as possible. This technique greatly simplifies and reduces the computational and memory requirements of digital imaging applications.

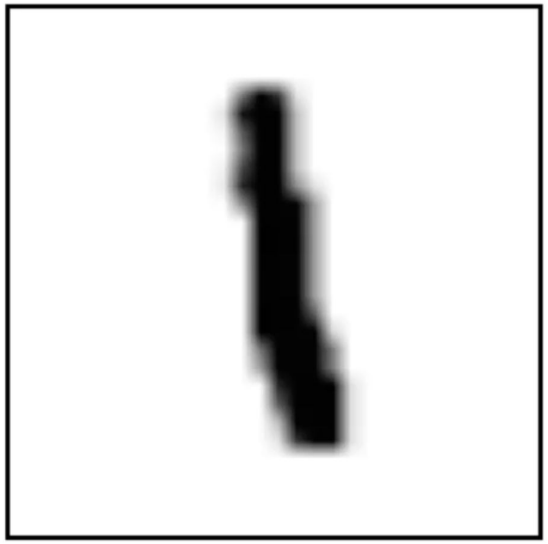
There are two types of image quantization:

- **Color Quantization** → In this process of image quantization, the number of distinct colors in an image is reduced. This is typically used in image storage and processing that require less computing strength.
- **Spatial Quantization** → This process involves reducing the spatial resolution of an image. The image is divided into non-overlapping regions and each region is substituted by a representative intensity value, mainly the mean intensity.

K-Mean Clustering - Image Quantization

The process of color quantization can be done by K-Mean Clustering. Meaning we can divide the picture colors into K different colors that will represent the K clusters in the algorithm result.
In order to understand how K-Mean Clustering perform this color division task we first need to understand how a picture been represented digitally.

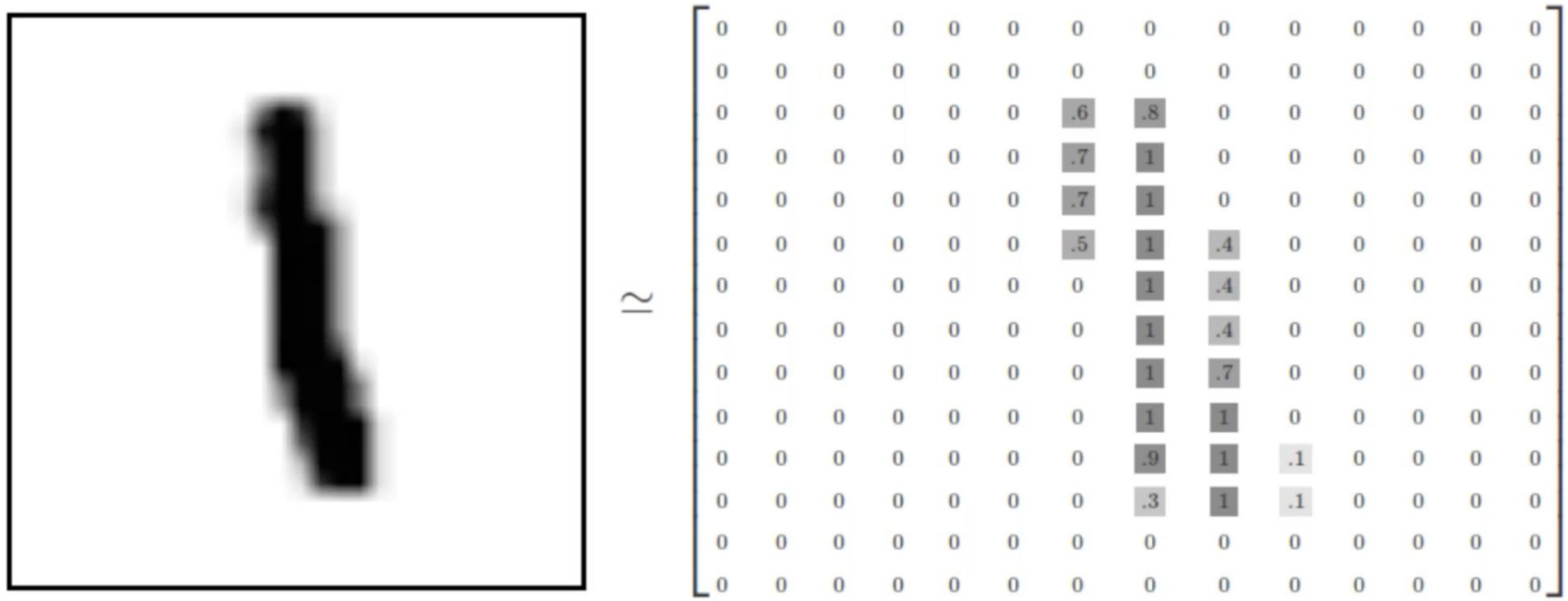
First, let's take a look at the image below. This image is in **grayscale**, meaning the image color range goes from black to white.



K-Mean Clustering - Image Quantization

The computer will store the image information as an 2D array of numbers.

Each cell in the array represent a pixel (according to the row and column of the pixel) and each number represent the amount of black in the image. Number 0 meaning white (no black) and number 1 meaning black. Any number between 0 and 1 represent grey color (combination of white and black).



K-Mean Clustering - Image Quantization

In case of images that are not grayscale, the computer can represent each pixel color by the combinations of 3 main colors - Red, Green, Blue, this called **RGB**.

RGB (Red Green Blue) → RGB stands for Red, Green and Blue, and it is a color model used in digital imaging and computer graphics. Essentially, it is a system for representing colors by combining these three primary colors of light.

In RGB, Red, Green, Blue are primary colors of light. Each of these colors is assigned a range of integer values, usually from 0 to 255. When combined, these three colors can produce over 16 million colors (256x256x256).

K-Mean Clustering - Image Quantization

So in RGB, in order to represent a specific color we need to provide 3 numbers between 0 - 255.

First number for red, Second number for green and third number for blue.

For example →

Now, when a computer store a color image it store a 3D array:

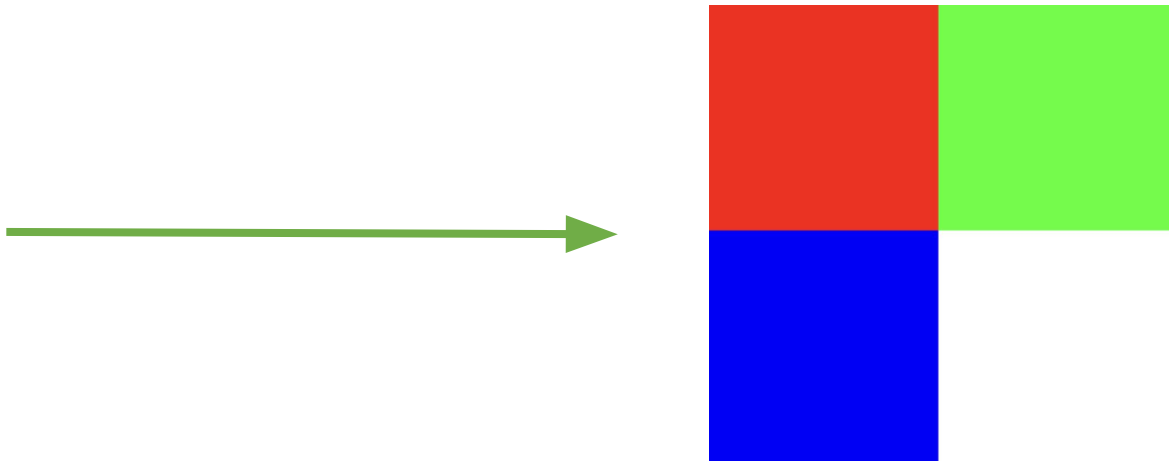
- **First dimension** → Represent the Y coordinates of each pixel (In total its represent the height of the picture).
- **Second dimension** → Represent the X coordinates of each pixel (In total its represent the width of the picture).
- **Third dimension** → Represent the RGB color of the pixel, The value of the third dimension is an array of itself with 3 numbers for the red, green and blue in the pixel color.

K-Mean Clustering - Image Quantization

For example → Let's consider that we have an image of 2x2 pixels with red, green, blue and white colors.

The 3D array for this image could look like this:

```
image = [
        [
            [255, 0, 0], [0, 255, 0]
        ],
        [
            [0, 0, 255], [255, 255, 255]
        ]
    ]
```

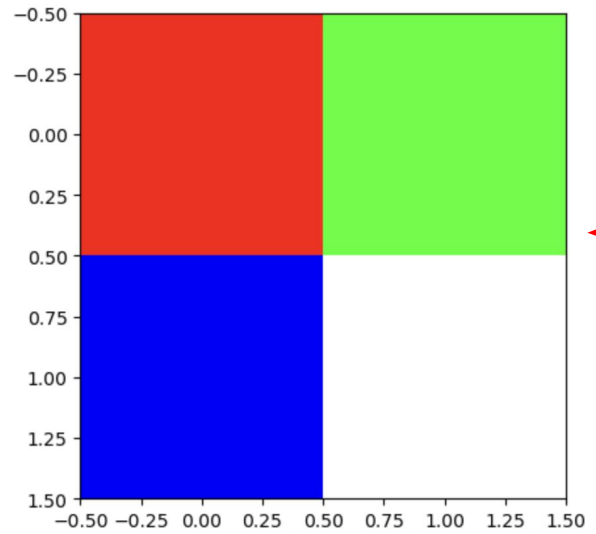


K-Mean Clustering - Image Quantization

In order to see the actual picture of this 3D array we can use the imshow() method from matplotlib.

plt.imshow() → A method in matplotlib that can display data as an image and interprets 2D grids in which the values are represented as colors.

```
In [2]: image = np.array([
            [
              [255, 0, 0], [0, 255, 0]
            ],
            [
              [0, 0, 255], [255, 255, 255]
            ]
          ], dtype=np.uint8)

plt.imshow(image)
plt.show()
```



The 3D array that represent the image we want to generate.

The image itself that been generated from that 3D array. Each cell in the grid is a different pixel and the color been determine from the RGB value.

K-Mean Clustering - Image Quantization

Now that we understand how computers can represent colorful images, it's time to see how K-Mean Clustering algorithm can help us in the task of dividing the colors in a picture into K groups of colors.

First, we need to create the dataset. As we learned in unsupervised learning the dataset contain only features and no labels. The algorithm goal is to decide how to divide the data points into the different groups. In our case the data points will be all the pixels in the picture and the features will be the Red, Green and Blue values of each data point.

So basically what we are doing is flatten the 3D picture array into 2D array:

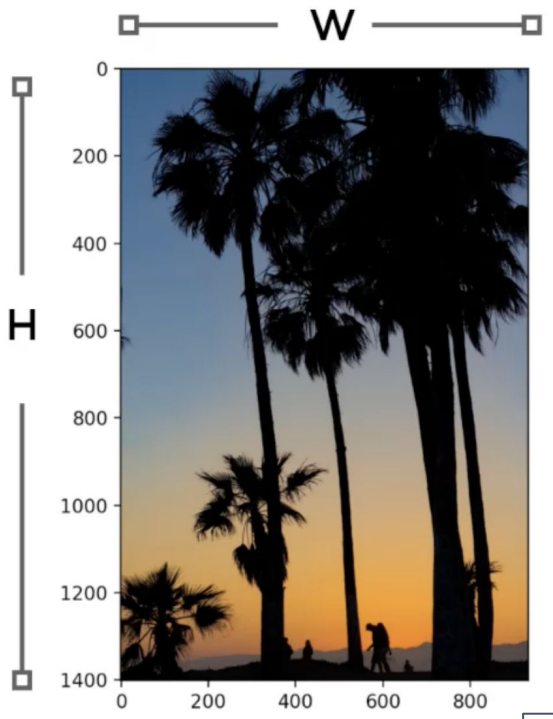
```
array([[[255,   0,   0],
        [  0, 255,   0]],

       [[  0,   0, 255],
        [255, 255, 255]]])
```

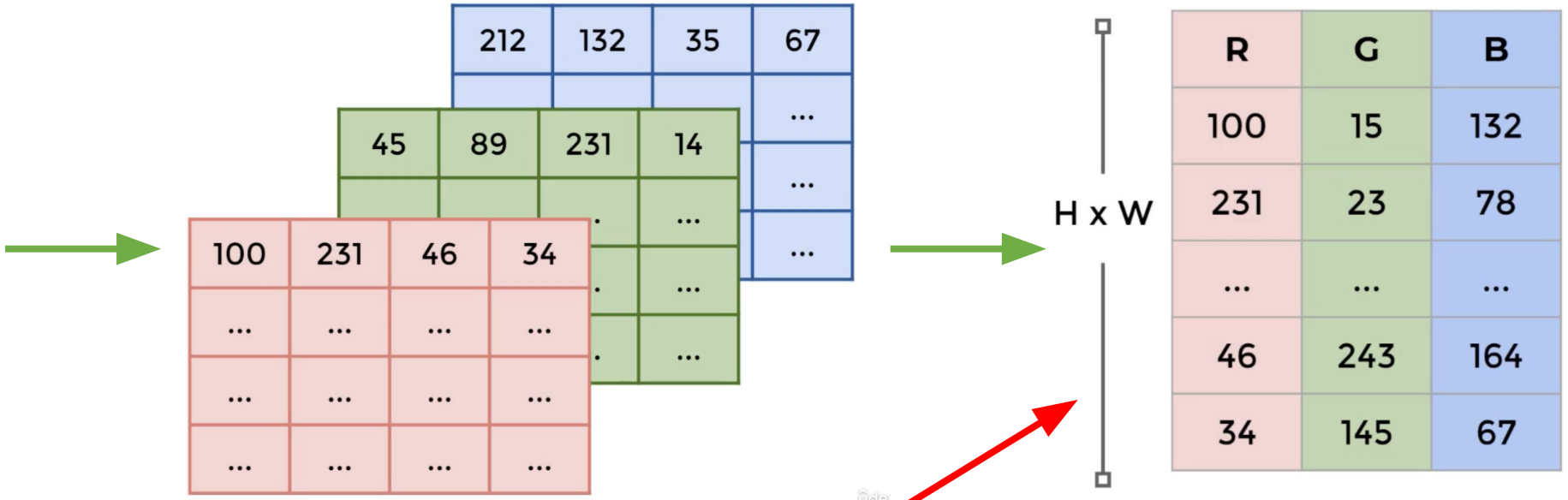
	Red	Green	Blue
0	255	0	0
1	0	255	0
2	0	0	255
3	255	255	255

Image Quantization - Python Example

Now that we understand the basics of our task we can take a real colorful picture and use K-Mean Clustering model to divide the colors in the picture into K different color groups.

For this example we will use the **‘palm_trees.jpg’** image file:





Original 3D array representation of the picture

Flatten 2D array, each row is a pixel in the picture and the columns are the RGB values of the color of that pixel

Image Quantization - Python Example

Now let's import the image as an array using 'mpimg' method from Matplotlib:

```
In [14]: import matplotlib.image as mpimg

         image_as_array = mpimg.imread('/Users/ben.meir/Downloads/palm_trees.jpg')
         image_as_array

Out[14]: array([[[ 25,  89, 127],
                 [ 25,  89, 127],
                 [ 25,  89, 127],
                 ...,
                 [ 23,  63,  99],
                 [ 51,  91, 127],
                 [ 50,  90, 126]],
```

The 'palm_trees' picture represented as a 3D array.

We can also take a look at the array shape and see the picture height and width size:

```
In [15]: image_as_array.shape

Out[15]: (1401, 934, 3)
```

The picture has 1401X934 pixels and each pixel has its own color represent by the RGB values

Now, let's reshape the array from 3D to 2D using the **np.reshape()** method:

```
In [16]: (h,w,c) = image_as_array.shape
         reshaped_image_array = image_as_array.reshape(h*w, c)
         reshaped_image_array

Out[16]: array([[ 25,  89, 127],
                [ 25,  89, 127],
                [ 25,  89, 127],
                ...,
                [  9,   9,  11],
                [ 10,  10,  12],
                [ 10,  10,  12]], dtype=uint8)
```

We reshaped the original 3D array into 2D array. The first dimension is the entire pixels and the second dimension is the RGB value of each pixel.

Image Quantization - Python Example

Now that we have the dataset ready, we can train our K-Mean Clustering model and request from the model to divide the picture pixels into 6 different colors. We will determine that the K value will be 6.

```
In [19]: from sklearn.cluster import KMeans

         model = KMeans(n_clusters=6)
         labels = model.fit_predict(reshaped_image_array)

         labels

Out[19]: array([3, 3, 3, ..., 1, 1, 1], dtype=int32)
```

In order to understand what color each group is represent we need to check the centroid value of each group. Those values are RGB values that represent the color of each group.

```
In [23]: rgb_codes = model.cluster_centers_
         rgb_codes

Out[23]: array([[136.73533325, 143.74401484, 143.9844705 ],
                [  2.75659154,   2.57529968,   3.70012508],
                [219.02710886, 135.42650506,  47.00342746],
                [ 71.12526864, 109.27679682, 137.69417556],
                [ 67.25337203,  61.54496264,  62.01242432],
                [191.45660313, 154.52547084, 109.63137723]])
```

The model provided 6 RGB values. A RGB value for each group. This allowing us the get the actual RGB color of each group.

Image Quantization - Python Example

Next, in order to see the final image we first need to round those centroid numbers (because RGB only works with round numbers) and then according to the model clustering add the corresponding RGB value as the pixel RGB value.

```
In [25]: rgb_codes = model.cluster_centers_.round(0).astype(int)
         rgb_codes

Out[25]: array([[137, 144, 144],
                [  3,   3,   4],
                [219, 135,  47],
                [ 71, 109, 138],
                [ 67,  62,  62],
                [191, 155, 110]])
```

Rounding the centroid values to be valid RGB values

```
In [30]: rgb_codes[labels]

Out[30]: array([[ 71, 109, 138],
                [ 71, 109, 138],
                [ 71, 109, 138],
                ...,
                [  3,   3,   4],
                [  3,   3,   4],
                [  3,   3,   4]])
```

Takes the label predictions from the model and according to the indexing match each index with the corresponding RGB value. So for example pixel with label 0 will get (137,144,144) RGB color.

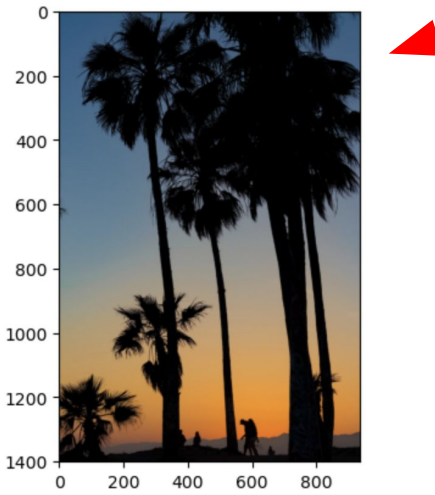
Image Quantization - Python Example

Finally, we will take the 2D array with the RGB values and reshape it again to 3D array so Matplotlib will be able to display the image result.

```
In [32]: quantized_image = np.reshape(rgb_codes[labels], (h,w,c))
         quantized_image

Out[32]: array([[[ 71, 109, 138],
                 [ 71, 109, 138],
                 [ 71, 109, 138],
                 ...,
```

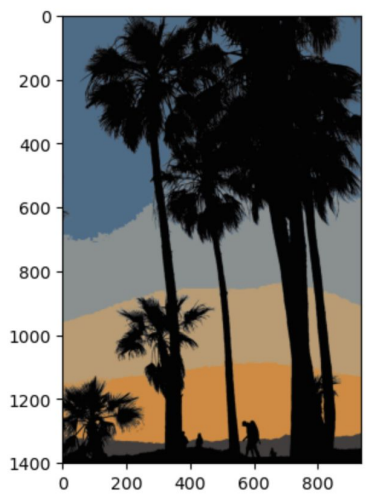
Reshaping the array back to 3D array, but now the RGB value of each pixel is just from those 6 RGB values the model found as the group centroids.

```
In [34]: plt.imshow(image_as_array)
Out[34]: <matplotlib.image.AxesImage at 0x14f195ff0>
```



Original image

```
In [33]: plt.imshow(quantized_image)
Out[33]: <matplotlib.image.AxesImage at 0x14f0faad0>
```



Quantized image

Class Exercise - Image Quantization

Instructions:

Use the **‘dog_image.jpeg’** file and implement the following:

- Represent the image as a 3D array and find the image height and width values
- Perform image quantization using K-Mean Clustering model
- Check the image quantization result for → K = 4, K = 10, K = 25 and K = 50

Class Exercise Solution - Image Quantization

Hierarchical Clustering - Introduction

Hierarchical Clustering → Hierarchical clustering is an algorithm that builds a hierarchy of clusters. It is a type of unsupervised machine learning model which is meant to perform grouping of data without having any prior knowledge of the dataset.

The main advantage of Hierarchical Clustering on K-Mean Clustering is that the Hierarchical Clustering model not request the number of clusters as parameter. We can still pass a cluster number but its not mandatory as with K-Mean Clustering model.

How the algorithm works?

The algorithm starts with all the data points assigned to a cluster of their own. Then two closest clusters (according to distance metric) are joined into the same cluster. The algorithm will keep combining clusters together until there is only one single cluster left.

Hierarchical Clustering - Introduction

In Hierarchical Clustering there are two main approaches:

- **Agglomerative Hierarchical Clustering** → A bottom-up approach, where each observation starts in its own cluster and pairs of clusters are merged together as one moves up the hierarchy.
- **Divisive Hierarchical Clustering** → A top-down approach, where all observations start in one cluster and splits are performed recursively as one moves down the hierarchy.

The hierarchical clustering model is highly explainable, intuitive, and highly useful in understanding complex relationships between data points. However, its computation can be expensive and inefficient when dealing with large datasets. This is because it requires a distance matrix that analyses distances between each pair of samples in the dataset, thus, the higher the number of features, the more computationally intensive it becomes.

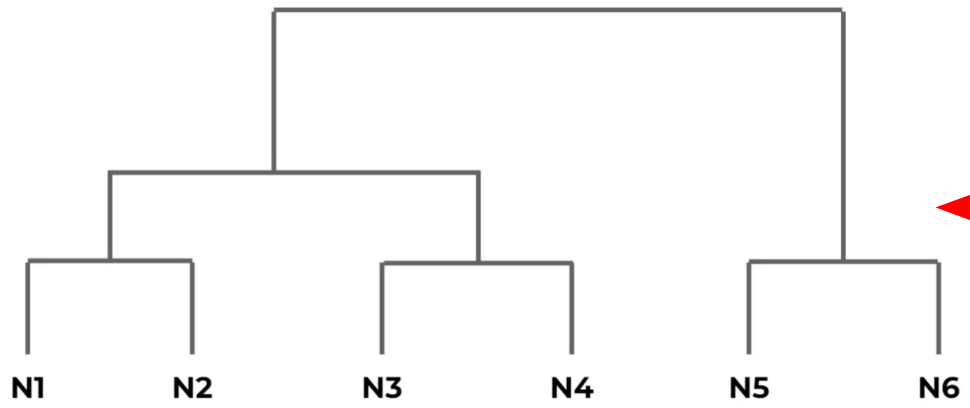
Hierarchical Clustering - Dendrogram

The most common way to visualize and represent a Hierarchical Clustering is by using a **Dendrogram**.

Dendrogram → A dendrogram is a tree-like diagram that displays the sequence of merges or splits in hierarchical clustering. It's a visual representation of the clustering process.

The individual data points are represented on the 'x' axis and then as you move up the 'y' axis, it shows which data points merge, creating a link between them. The height of that link on the 'y' axis shows the distance or dissimilarity between those points—the higher the link, the less similar the data points.

For example →

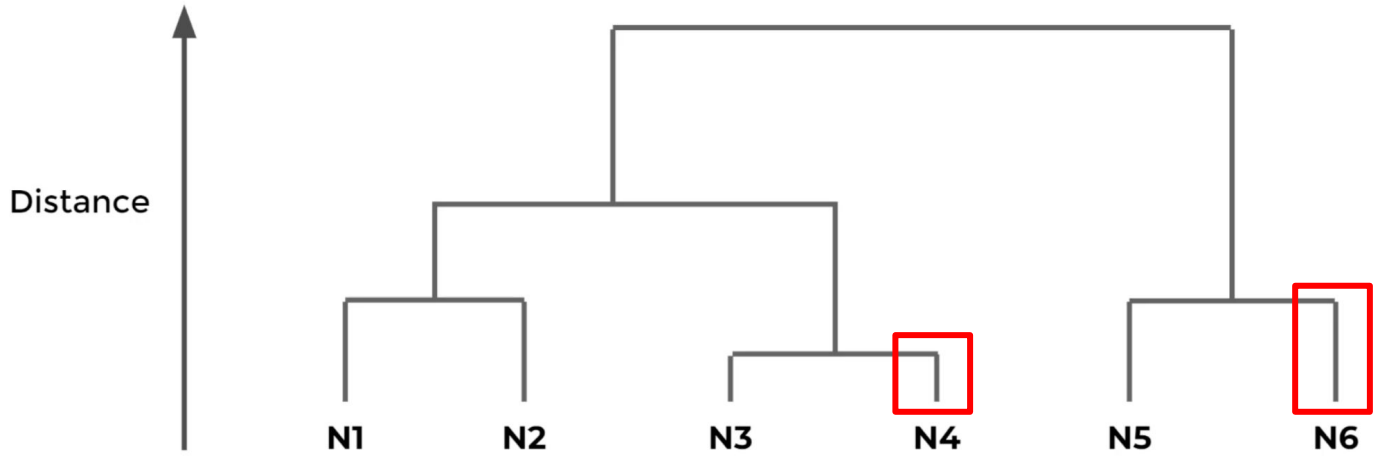


A agglomerative type Dendrogram represent Hierarchical Clustering of 6 data points.

Hierarchical Clustering - Dendrogram

In the Dendrogram, the height of the y-axis represent the distance between the two merged clusters. The higher the line meaning the higher the distance.

For example → The data points N3 and N4 are much more similar to each other then data points N5 and N6. This is because the line that merge N3 and N4 together into one cluster are lower than the lines that merge data points N5 and N6 into one cluster.

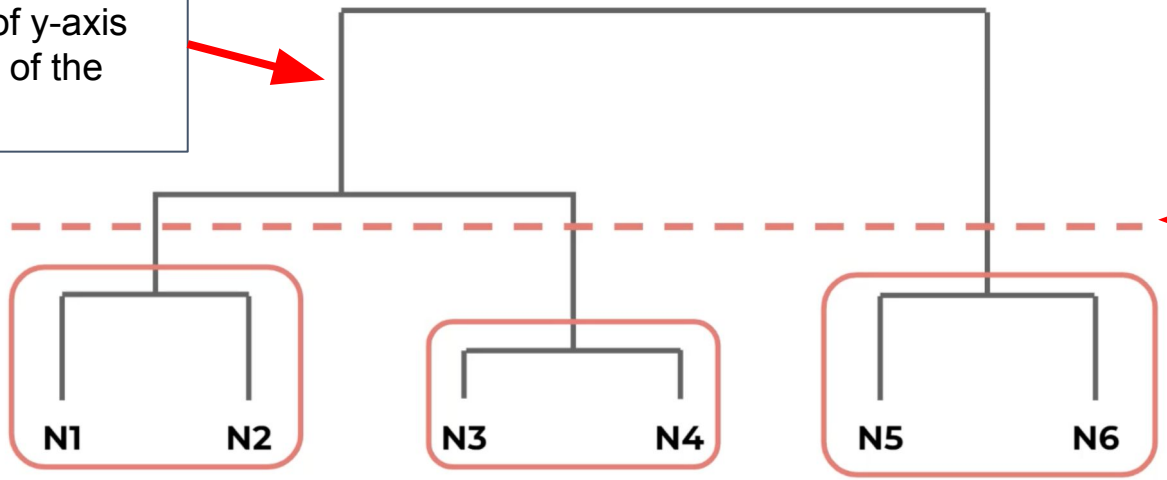


Hierarchical Clustering - Dendrogram

We can also utilize the fact that the Dendrogram y-axis represent the distance between the cluster to help us understand where we should slice the Dendrogram in order to determine the optimal number of clusters.

For example → If we will see that from a specific level in the Dendrogram the distance between the clusters becomes to large we can decide to slice the Dendrogram right before this cluster merge.

The number of clusters been used determine by the number of y-axis lines in the highest degree of the Dendrogram.



Slicing the Dendrogram at this point will determine the number of clusters to be 3 instead of 2 in the unsliced Dendrogram.

Hierarchical Clustering - Distance Calculation

As mentioned, the Hierarchical Clustering model is based on calculating the distance between each cluster. Because at the beginning of the algorithm it referred to each data point as an individual cluster it also need to first calculate the distance between data points.

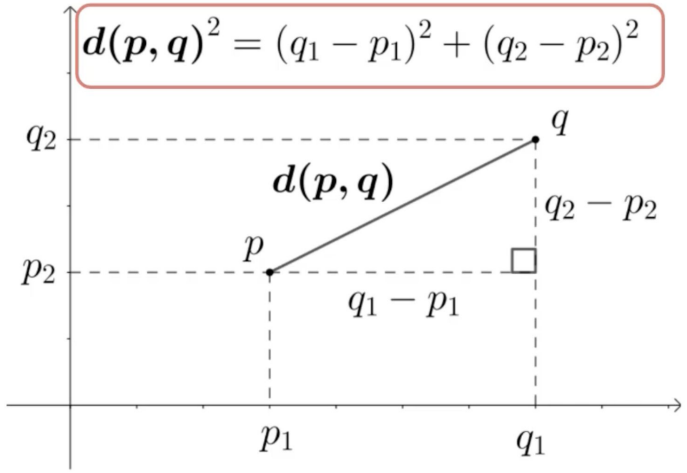
How a distance been two data points been calculated?

There are many ways to calculate this distance but the most common way (and the default way in Hierarchical Clustering) is by using **euclidean distance**.

Euclidean Distance → A method that allowing to calculate the distance between two points by It's calculate the square root of the sum of the squared differences between the individual dimensions (features) of the two points. It is suitable for data that has continuous numerical values.

Hierarchical Clustering - Distance Calculation

The euclidean distance been calculated according to the following formula (derived from the Pythagorean theorem):



For data points q & p with feature values (q1, q2), (p1, p2) the distance between those data points based on the feature values will be equal to:
D = sqrt((q1 - p1)^2 + (q2 - p2)^2)

So for a data points with N number of features we can say that the euclidean distance between 2 data points is equal to →

D = sqrt(X1 - Y1)^2 + (X2 - Y2)^2 + … + (Xn - Yn)^2)

Note: This is the same formula that is used to calculate also the distance between cluster centroids in K-Mean Clustering algorithms.

Hierarchical Clustering - Distance Calculation

Once the algorithm finished to calculate the distance between individual data points and start merging them into clusters it's now need to also start calculating the distance between clusters.
This is not the same as calculating distance between individual data points because remember that each cluster now contain more than 1 data point.

Here we can choose from different ways to calculate those distances:

- **Single Linkage** → The shortest distance between any two points each in a different cluster.
- **Complete Linkage** → The longest distance between any two points each in a different cluster.
- **Average Linkage** → The average distance between every pair of points in a different cluster.
- **Centroid Linkage** → The distance between centroids (center point) of the two clusters (This method is been used by K-Mean Clustering algorithm).