KURUNJI VENKATRAMANA GOWDA POLYTECHNIC SULLIA-574327

5TH SEMESTER

AI/ML WEEK-9

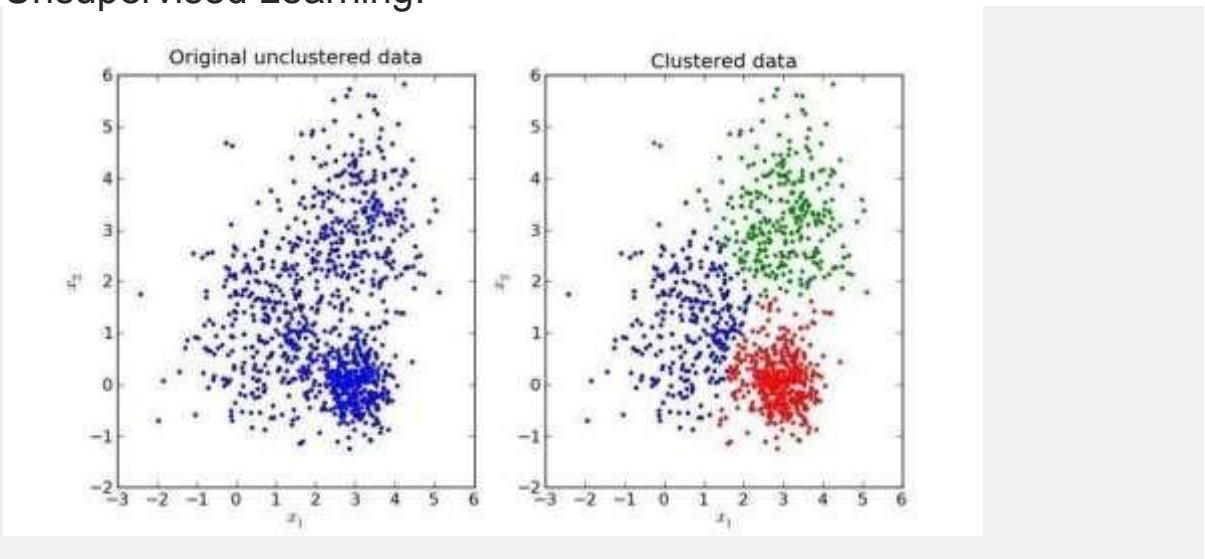
WEEK-9

Unsupervised Learning

Unsupervised Learning is the type of Machine Learning where no human intervention is required to make the data machine-readable and train the algorithm. Also, contrary to supervised learning, unlabeled data is used in the case of unsupervised learning.

Since there is no human intervention and unlabeled data is used, the algorithm can work on a larger data set. Unlike supervised learning, unsupervised learning does not require labels to establish relationships between two data points.

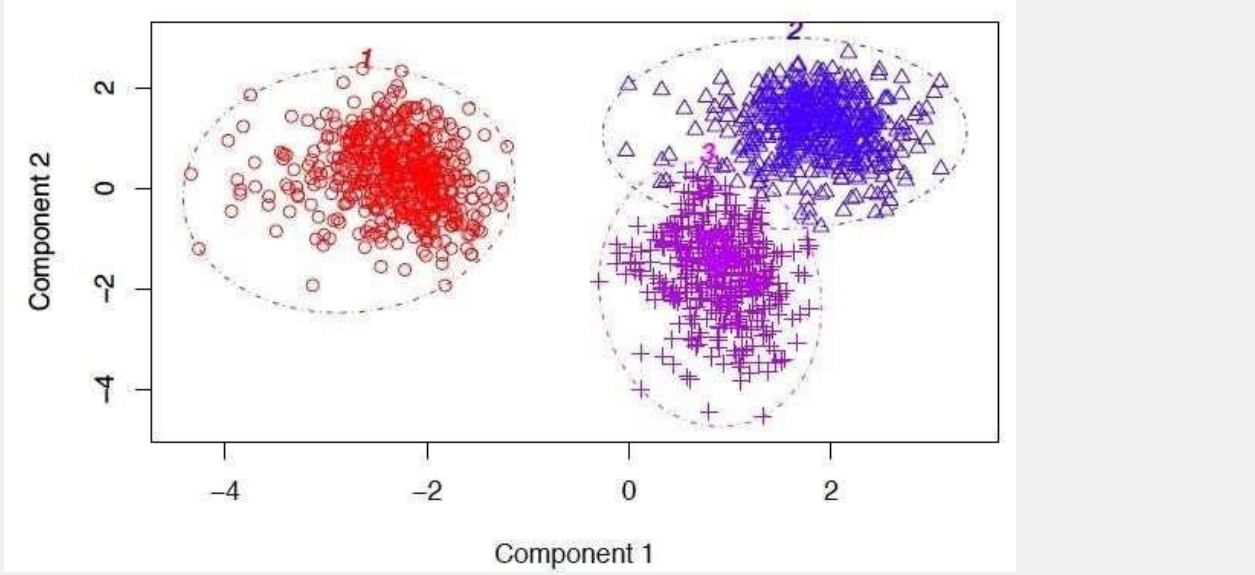
Unsupervised Learning:



In unsupervised learning, an AI system is presented with unlabeled, un-categorized data and the system's algorithms act on the data without prior training. The output is dependent upon the coded algorithms. Subjecting a system to unsupervised learning is one way of testing AI.
The unsupervised learning is categorized into 2 other categories which are "**Clustering**" and "**Association**".

Clustering:

A set of inputs is to be divided into groups. Unlike in classification, the groups are not known beforehand, making this typically an unsupervised task.

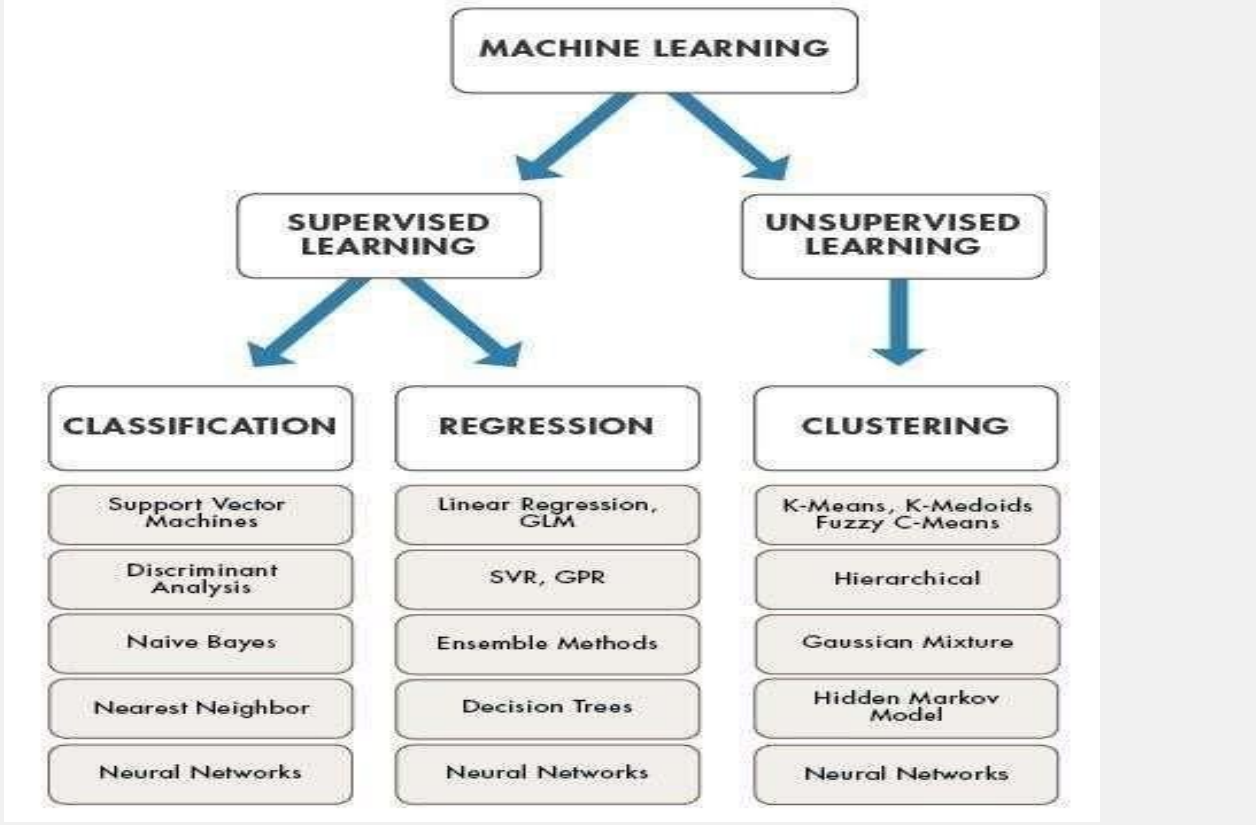


Clustering

Methods used for clustering are:

- **Gaussian mixtures**
- **K-Means Clustering**
- **Boosting**
- **Hierarchical Clustering**
- **K-Means Clustering**
- **Spectral Clustering**

Overview of models under categories:



Overview of models

Unsupervised learning is often focused on clustering.

Clustering is the grouping of objects or data points that are similar to each other and dissimilar to objects in other clusters.

Machine learning engineers and data scientists can use different algorithms for clustering, with the algorithms themselves falling into different categories based on how they work. The categories include the following:

- exclusive clustering
- overlapping clustering
- hierarchical clustering
- probabilistic clustering

Some of the more widely used algorithms include the k-means clustering algorithm and the fuzzy k-means algorithm, as well as the hierarchical clustering and the density-based clustering algorithms.

The Latent Dirichlet Allocation (LDA) model and Gaussian mixture models are also commonly used in clustering.

In addition to clustering, unsupervised learning may be used to determine how data is distributed in space (density estimation).

Application of unsupervised learning

Machine learning techniques have become a common method to improve a product user experience and to test systems for quality assurance. Unsupervised learning provides an exploratory path to view data, allowing businesses to identify patterns in large volumes of data more quickly when compared to manual observation. Some of the most common real-world applications of unsupervised learning are:

- **News Sections:** Google News uses unsupervised learning to categorize articles on the same story from various online news outlets. For example, the results of a presidential election could be categorized under their label for "US" news.
- **Computer vision:** Unsupervised learning algorithms are used for visual perception tasks, such as object recognition.
- **Medical imaging:** Unsupervised machine learning provides essential features to medical imaging devices, such as image detection, classification and segmentation, used in radiology and pathology to diagnose patients quickly and accurately.
- **Anomaly detection:** Unsupervised learning models can comb through large amounts of data and discover atypical data points within a dataset. These anomalies can raise awareness around faulty equipment, human error, or breaches in security.
- **Customer personas:** Defining customer personas makes it easier to understand common traits and business clients' purchasing habits. Unsupervised learning allows businesses to build better buyer persona profiles, enabling organizations to align their product messaging more appropriately.
- **Recommendation Engines:** Using past purchase behavior data, unsupervised learning can help to discover data trends that can be used to develop more effective cross-selling strategies. This is used to make relevant add-on recommendations to customers during the checkout process for online retailers.

K-Means Clustering Algorithm

K-Means Clustering is an unsupervised learning algorithm that is used to solve the clustering problems in machine learning or data science. In this topic, we will learn what is K-means clustering algorithm, how the algorithm works, along with the Python implementation of k-means clustering.

What is K-Means Algorithm?

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

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

It allows us to cluster the data into different groups and a convenient way to discover the categories of groups in the unlabeled dataset on its own without the need for any training.

It is a centroid-based algorithm, where each cluster is associated with a centroid.

The main aim of this algorithm is to minimize the sum of distances between the data point and their corresponding clusters.

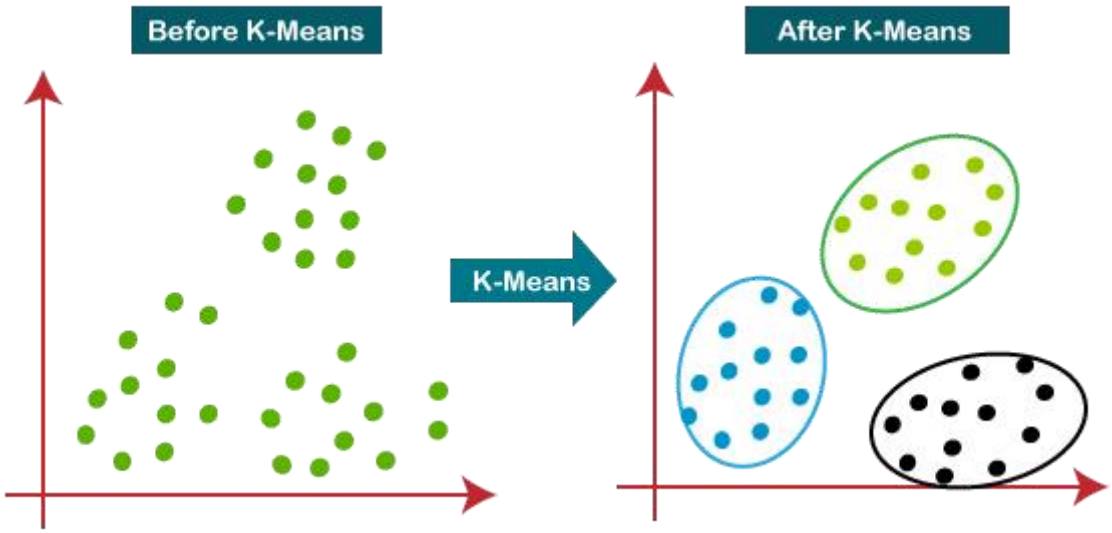
The algorithm takes the unlabeled dataset as input, divides the dataset into k-number of clusters, and repeats the process until it does not find the best clusters. The value of k should be predetermined in this algorithm.

The k-means clustering algorithm mainly performs two tasks:

- Determines the best value for K center points or centroids by an iterative process.
- Assigns each data point to its closest k-center. Those data points which are near to the particular k-center, create a cluster.

Hence each cluster has datapoints with some commonalities, and it is away from other clusters.

The below diagram explains the working of the K-means Clustering Algorithm:



How does the K-Means Algorithm Work?

The working of the K-Means algorithm is explained in the below steps:

Step-1: Select the number K to decide the number of clusters.

Step-2: Select random K points or centroids. (It can be other from the input dataset).

Step-3: Assign each data point to their closest centroid, which will form the predefined K clusters.

Step-4: Calculate the variance and place a new centroid of each cluster.

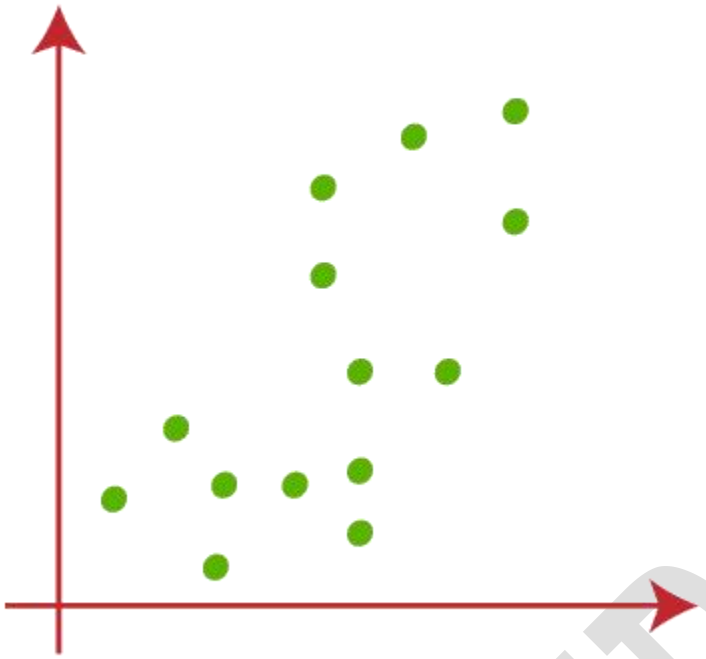
Step-5: Repeat the third steps, which means reassign each datapoint to the new closest centroid of each cluster.

Step-6: If any reassignment occurs, then go to step-4 else go to FINISH.

Step-7: The model is ready.

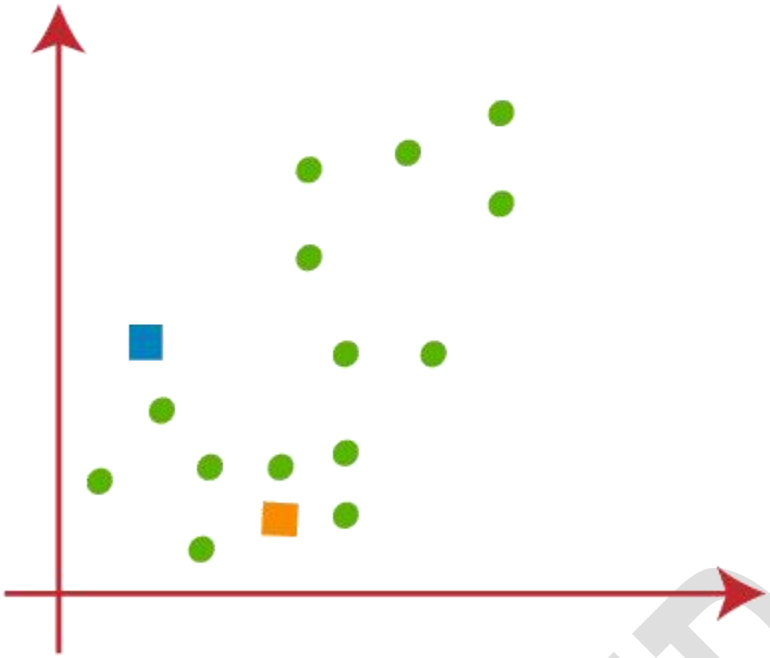
Let's understand the above steps by considering the visual plots:

Suppose we have two variables M1 and M2. The x-y axis scatter plot of these two variables is given below:

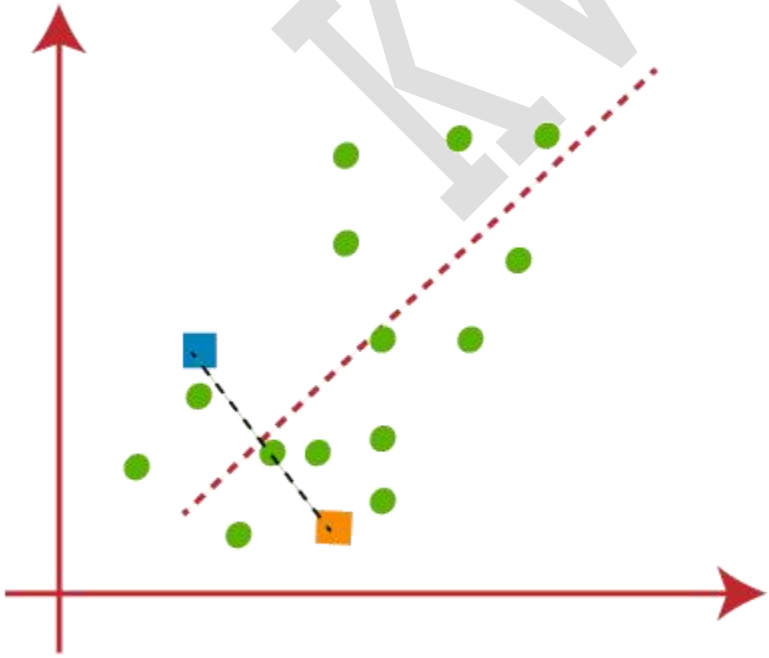


- Let's take number k of clusters, i.e., K=2, to identify the dataset and to put them into different clusters. It means here we will try to group these datasets into two different clusters.
- We need to choose some random k points or centroid to form the cluster. These points can be either the points from the dataset or any other point. So, here we are selecting the below two points as k points, which are not the part of our dataset.

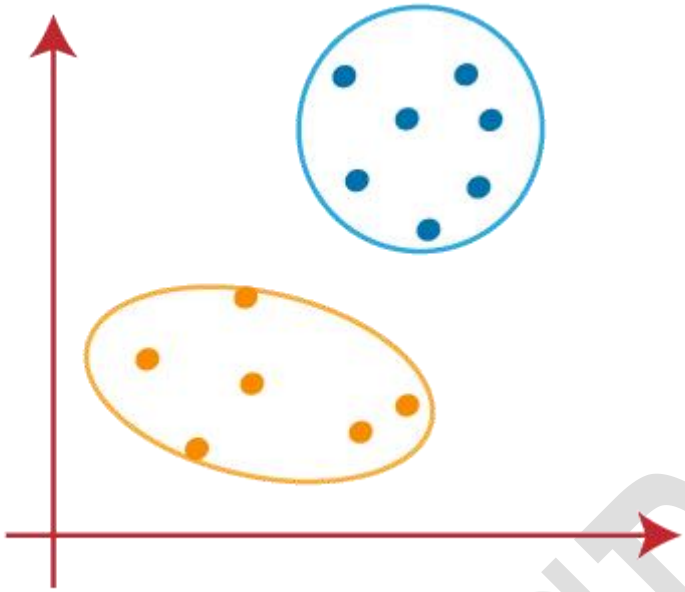
Consider the below image:



- Now we will assign each data point of the scatter plot to its closest K-point or centroid. We will compute it by applying some mathematics that we have studied to calculate the distance between two points. So, we will draw a median between both the centroids. Consider the below image:



As our model is ready, so we can now remove the assumed centroids, and the two final clusters will be as shown in the below image:



How to choose the value of "K number of clusters" in Kmeans Clustering?

The performance of the K-means clustering algorithm depends upon highly efficient clusters that it forms. But choosing the optimal number of clusters is a big task. There are some different ways to find the optimal number of clusters, but here we are discussing the most appropriate method to find the number of clusters or value of K. The method is given below:

Elbow Method

The Elbow method is one of the most popular ways to find the optimal number of clusters.

This method uses the concept of WCSS value. **WCSS** stands for **Within Cluster Sum of Squares**, which defines the total variations within a cluster. The formula to calculate the value of WCSS (for 3 clusters) is given below:

In the above formula of WCSS,

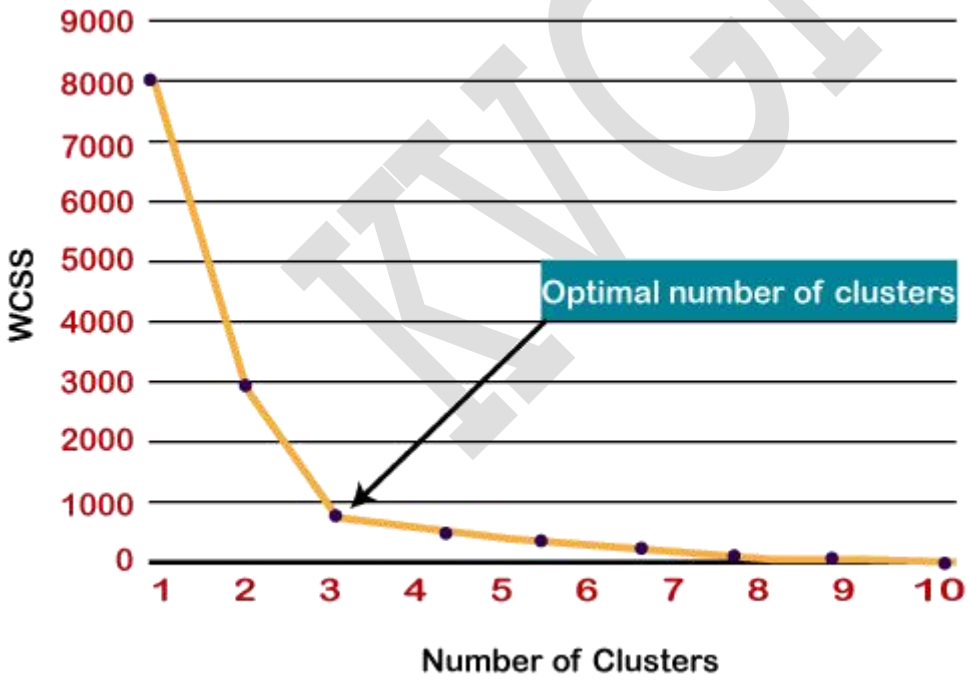
$\sum_{Pi \text{ in Cluster1}}$ distance$(P_i C_1)^2$: It is the sum of the square of the distances between each data point and its centroid within a cluster1 and the same for the other two terms.

To measure the distance between data points and centroid, we can use any method such as Euclidean distance or Manhattan distance.

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

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

Since the graph shows the sharp bend, which looks like an elbow, hence it is known as the elbow method. The graph for the elbow method looks like the below image:



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

Python Implementation of K-means Clustering Algorithm

In the above section, we have discussed the K-means algorithm, now let's see how it can be implemented using Python.

Before implementation, let's understand what type of problem we will solve here. So, we have a dataset of **Mall_Customers**, which is the data of customers who visit the mall and spend there.

In the given dataset, we have **Customer_Id, Gender, Age, Annual Income ($), and Spending Score** (which is the calculated value of how much a customer has spent in the mall, the more the value, the more he has spent). From this dataset, we need to calculate some patterns, as it is an unsupervised method, so we don't know what to calculate exactly.

The steps to be followed for the implementation are given below:

- **Data Pre-processing**
- **Finding the optimal number of clusters using the elbow method** o **Training the K-means algorithm on the training dataset** o **Visualizing the clusters**

Step-1: Data pre-processing Step

The first step will be the data pre-processing, as we did in our earlier topics of Regression and Classification. But for the clustering problem, it will be different from other models. Let's discuss it:

o **Importing Libraries**

As we did in previous topics, firstly, we will import the libraries for our model, which is part of data pre-processing. The code is given below:

```
1. # importing libraries
2. import numpy as nm
3. import matplotlib.pyplot as mtp
4. import pandas as pd
```

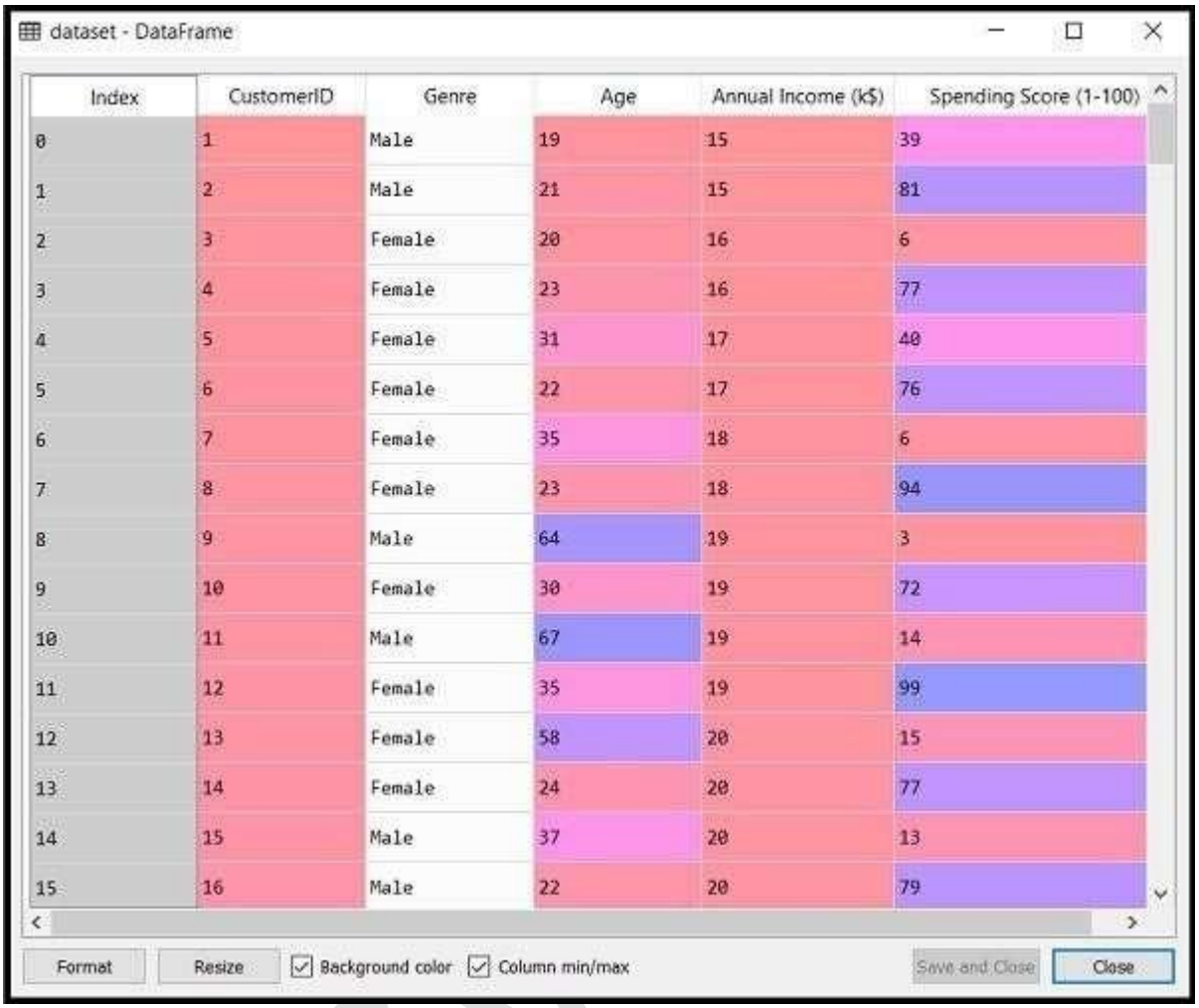
In the above code, the **numpy** we have imported for the performing mathematics calculation, **matplotlib** is for plotting the graph, and **pandas** are for managing the dataset.

o **Importing the Dataset:**

Next, we will import the dataset that we need to use. So here, we are using the Mall_Customer_data.csv dataset. It can be imported using the below code:

```
1. # Importing the dataset
2. dataset = pd.read_csv('Mall_Customers_data.csv')
```

By executing the above lines of code, we will get our dataset in the Spyder IDE. The dataset looks like the below image:



Index	CustomerID	Genre	Age	Annual Income (k$)	Spending Score (1-100)
0	1	Male	19	15	39
1	2	Male	21	15	81
2	3	Female	20	16	6
3	4	Female	23	16	77
4	5	Female	31	17	40
5	6	Female	22	17	76
6	7	Female	35	18	6
7	8	Female	23	18	94
8	9	Male	64	19	3
9	10	Female	30	19	72
10	11	Male	67	19	14
11	12	Female	35	19	99
12	13	Female	58	20	15
13	14	Female	24	20	77
14	15	Male	37	20	13
15	16	Male	22	20	79

From the above dataset, we need to find some patterns in it.

- **Extracting Independent Variables**

Here we don't need any dependent variable for data pre-processing step as it is a clustering problem, and we have no idea about what to determine. So we will just add a line of code for the matrix of features.

```
1. x = dataset.iloc[:, [3, 4]].values
```

As we can see, we are extracting only 3rd and 4th feature. It is because we need a 2d plot to visualize the model, and some features are not required, such as customer_id.

Step-2: Finding the optimal number of clusters using the elbow method

In the second step, we will try to find the optimal number of clusters for our clustering problem. So, as discussed above, here we are going to use the elbow method for this purpose.

As we know, the elbow method uses the WCSS concept to draw the plot by plotting WCSS values on the Y-axis and the number of clusters on the X-axis. So we are going to calculate the value for WCSS for different k values ranging from 1 to 10. Below is the code for it:

```
#finding optimal number of clusters using the elbow method
from sklearn.cluster import KMeans
wcss_list= [] #Initializing the list for the values of WCSS

#Using for loop for iterations from 1 to 10.
for i in range(1, 11):
kmeans = KMeans(n_clusters=i, init='k-means++', random_state= 42)
kmeans.fit(x)
wcss_list.append(kmeans.inertia_)
mtp.plot(range(1, 11), wcss_list)
mtp.title('The Elobw Method Graph')
mtp.xlabel('Number of clusters(k)')
mtp.ylabel('wcss_list')
mtp.show()
```

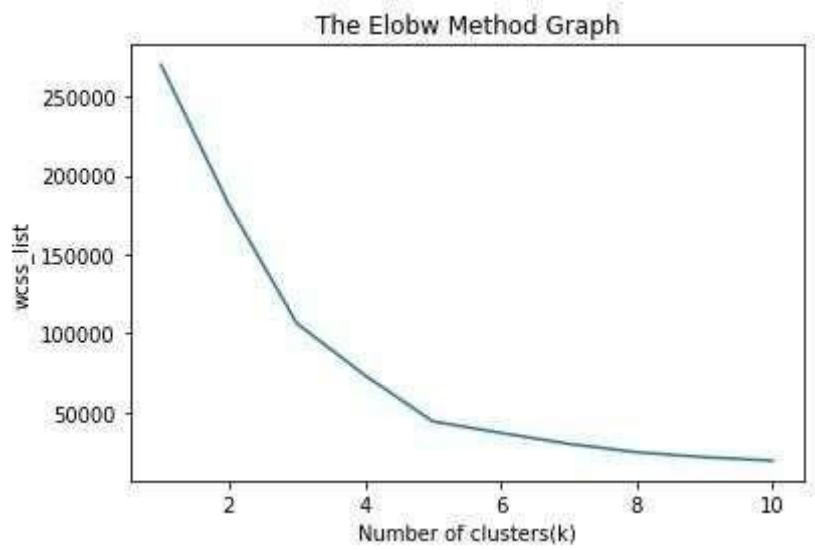
As we can see in the above code, we have used **the KMeans** class of sklearn. cluster library to form the clusters.

Next, we have created the **wcss_list** variable to initialize an empty list, which is used to contain the value of wcss computed for different values of k ranging from 1 to 10.

After that, we have initialized the for loop for the iteration on a different value of k ranging from 1 to 10; since for loop in Python, exclude the outbound limit, so it is taken as 11 to include 10^{th} value.

The rest part of the code is similar as we did in earlier topics, as we have fitted the model on a matrix of features and then plotted the graph between the number of clusters and WCSS.

Output: After executing the above code, we will get the below output:



From the above plot, we can see the elbow point is at **5. So the number of clusters here will be 5.**

wcss_list - List (10 elements)

Index	Type	Size	Value
0	float64	1	269981.28
1	float64	1	181363.59595959596
2	float64	1	106348.37306211118
3	float64	1	73679.78903948834
4	float64	1	44448.45544793371
5	float64	1	37233.81451071001
6	float64	1	30259.65720728547
7	float64	1	25011.83934915659
8	float64	1	21850.165282585633
9	float64	1	19672.07284901432

Save and Close | Close

Step- 3: Training the K-means algorithm on the training dataset

As we have got the number of clusters, so we can now train the model on the dataset.

To train the model, we will use the same two lines of code as we have used in the above section, but here instead of using i, we will use 5, as we know there are 5 clusters that need to be formed. The code is given below:

```
1. #training the K-means model on a dataset
2. kmeans = KMeans(n_clusters=5, init='k-means++', random_state= 42)
3. y_predict= kmeans.fit_predict(x)
```

Index	CustomerID	Genre	Age	ınnua
0	1	Male	19	15
1	2	Male	21	15
2	3	Female	20	16
3	4	Female	23	16
4	5	Female	31	17
5	6	Female	22	17
6	7	Female	35	18
7	8	Female	23	18
8	9	Male	64	19
9	10	Female	30	19

The first line is the same as above for creating the object of KMeans class.

In the second line of code, we have created the dependent variable **y_predict** to train the model.

By executing the above lines of code, we will get the y_predict variable. We can check it under **the variable explorer** option in the Spyder IDE. We can now compare the values of y_predict with our original dataset. Consider the below image:

From the above image, we can now relate that the CustomerID 1 belongs to a cluster

3(as index starts from 0, hence 2 will be considered as 3), and 2 belongs to cluster 4, and so on.

Step-4: Visualizing the Clusters

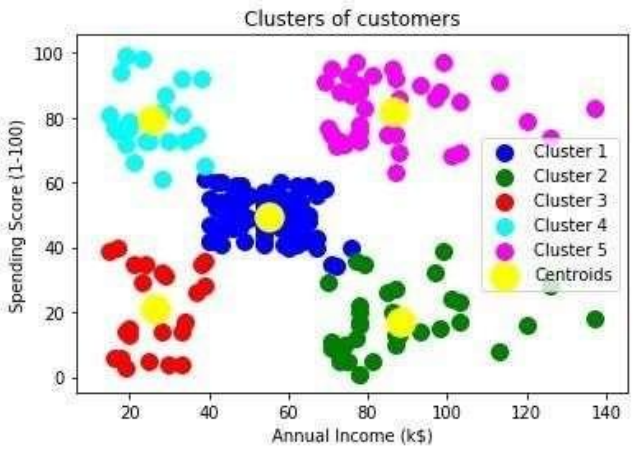
The last step is to visualize the clusters. As we have 5 clusters for our model, so we will visualize each cluster one by one.

To visualize the clusters will use scatter plot using mtp.scatter() function of matplotlib.

```
1. #visulaizing the clusters
2. mtp.scatter(x[y_predict == 0, 0], x[y_predict == 0, 1], s = 100, c = 'blue', label = 'C luster 1') #for first cluster
3. mtp.scatter(x[y_predict == 1, 0], x[y_predict == 1, 1], s = 100, c = 'green', label = ' Cluster 2') #for second cluster
4. mtp.scatter(x[y_predict== 2, 0], x[y_predict == 2, 1], s = 100, c = 'red', label = 'Clu ster 3') #for third cluster
5. mtp.scatter(x[y_predict == 3, 0], x[y_predict == 3, 1], s = 100, c = 'cyan', label = 'C luster 4') #for fourth cluster
6. mtp.scatter(x[y_predict == 4, 0], x[y_predict == 4, 1], s = 100, c = 'magenta', label = 'Cluster 5') #for fifth cluster
7. mtp.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1], s = 300, c = 'yellow', label = 'Centroid')
8. mtp.title('Clusters of customers')
9. mtp.xlabel('Annual Income (k$)')
10. mtp.ylabel('Spending Score (1-100)')
11. mtp.legend()
12. mtp.show()
```


In above lines of code, we have written code for each clusters, ranging from 1 to 5. The first coordinate of the mtp.scatter, i.e., x[y_predict == 0, 0] containing the x value for the showing the matrix of features values, and the y_predict is ranging from 0 to 1.

Output:



The output image is clearly showing the five different clusters with different colors. The clusters are formed between two parameters of the dataset; Annual income of customer and Spending. We can change the colors and labels as per the requirement or choice. We can also observe some points from the above patterns, which are given below:

Evalution matrics for Classification:

Confusion Matrix

Confusion Matrix is **a useful machine learning method which allows you to measure Recall, Precision, Accuracy, and AUCROC curve**. Below given is an example to know the terms True Positive, True Negative, False Negative, and True Negative. True Positive: You projected positive and its turn out to be true.

For better visualization of the performance of a model, these four outcomes are plotted on a confusion matrix.

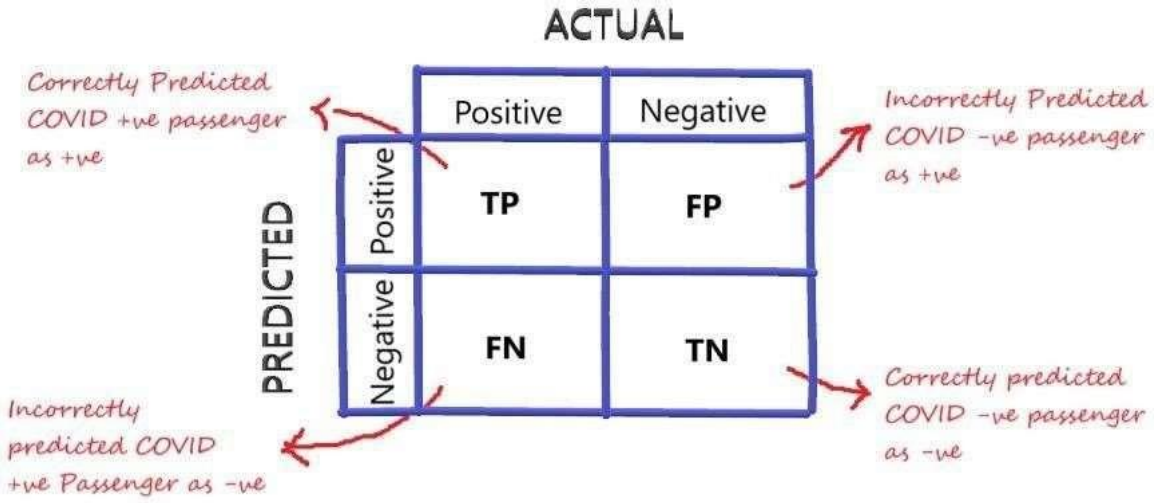


Fig.5 Confusion Matrix

Accuracy

Yes! You got that right, we want our model to focus on True positive and True Negative. Accuracy is one metric which gives the fraction of predictions our model got right. Formally, accuracy has the following definition:

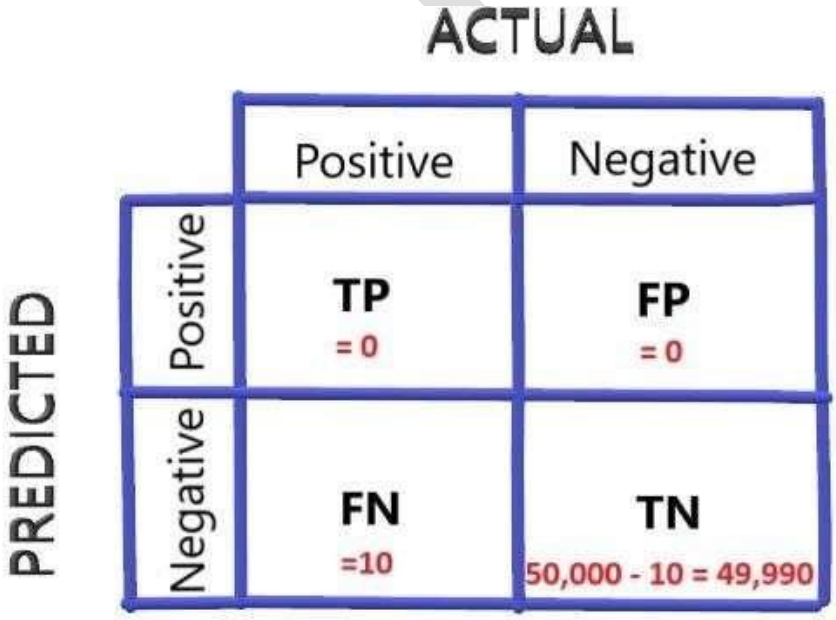
Accuracy = Number of correct predictions / Total number of predictions.

$$\text{Accuracy} = \frac{TP + TN}{TP + FP + FN + TP}$$

Fig.-6

Now, let's consider 50,000 passengers travel per day on an average. Out of which, 10 are actually COVID positive.

One of the easy ways to increase accuracy is to **classify every passenger as COVID negative**. So our confusion matrix looks like:



Accuracy for this case will be:

Accuracy = 49,990/50,000 = 0.9998 or **99.98**%

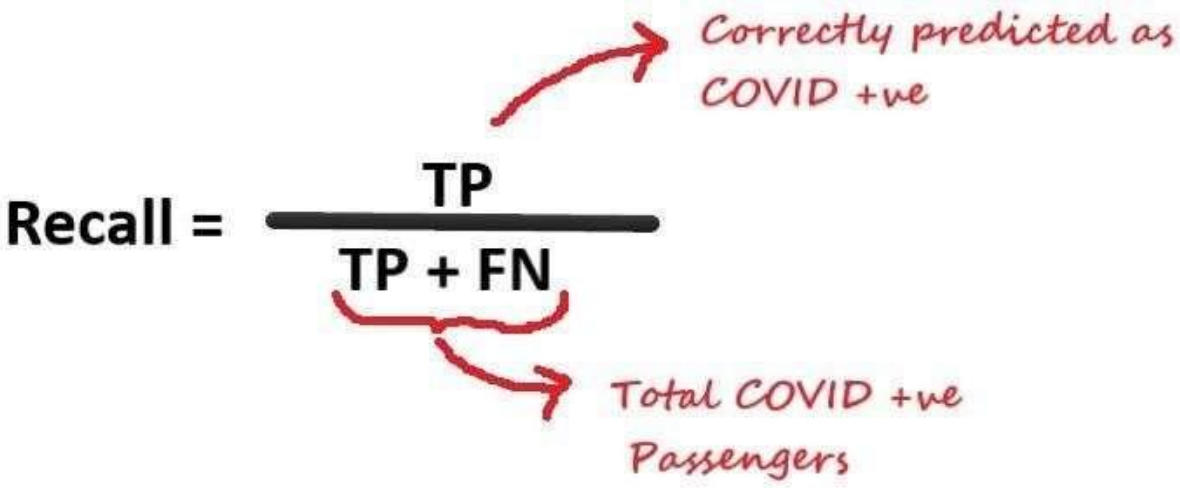
Impressive!! Right? Well, does that really solve our purpose of classifying COVID positive passengers correctly?

For this particular example where we are trying to label passengers as COVID positive and negative with the hope of identifying the right ones, I can get 99.98% accuracy by simply labeling everyone as COVID negative. Obviously, this is a way more accuracy than we have ever seen in any model. But it doesn't solve the purpose. The purpose here is to identify COVID positive passengers. Not labeling 10 of actually positive is a lot more expensive in this scenario. Accuracy in this context is a terrible measure because its easy to get extremely good accuracy but that's not what we are interested in.

So in this context accuracy is not a good measure to evaluate a model. Let's look at a very popular measure called Recall.

Recall (Sensitivity or True positive rate)

Recall gives the fraction you correctly identified as positive out of all positives.



Now, this is an important measure. Out of all positive passengers what fraction you identified correctly. Going back to our previous strategy of labeling every passenger as negative that will give recall of Zero.

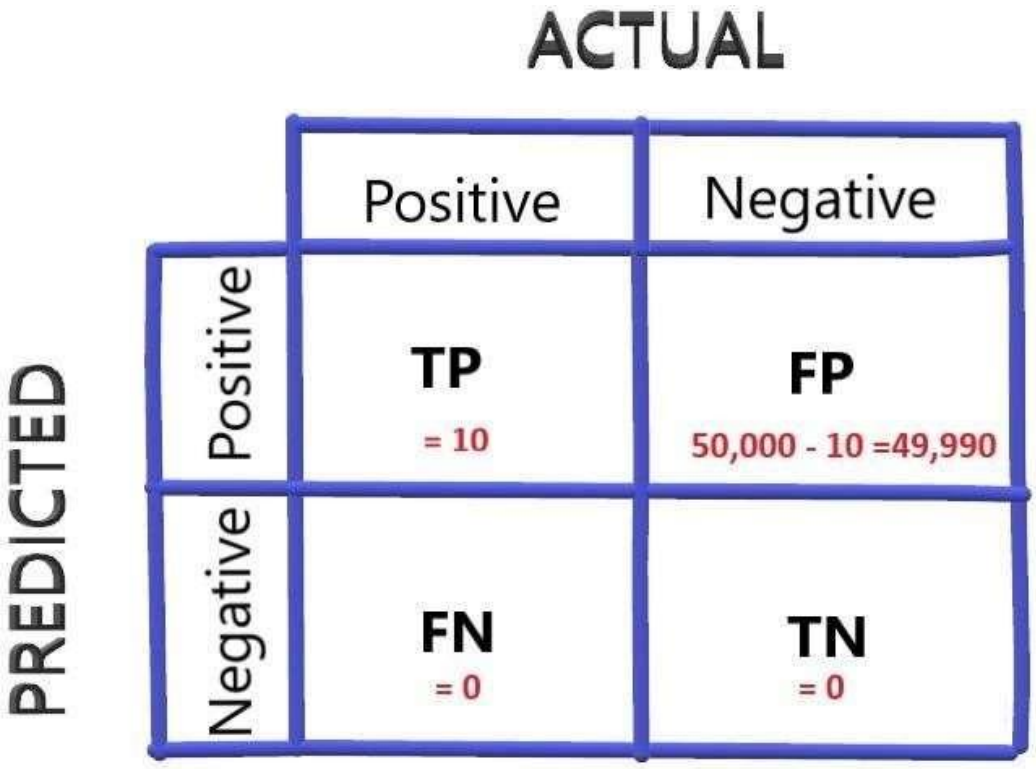
Recall = 0/10 = 0

So, in this context, Recall is a good measure. It says that the terrible strategy of identifying every passenger as COVID negative leads to zero recall. And we want to maximize the recall.

Wait, Wait!! Before considering recall as a good measure for evaluation. Just think: Is recall alone good enough to evaluate the performance of a classification model?

To answer the above question, consider another scenario of labeling every passenger as COVID positive. Everybody walks into the airport and the model just labels them as positive. Labeling every passenger as positive is bad in terms of the amount of cost that needs to be spent in actually investigating each one before they board the flight.

The confusion matrix will look like:



Recall for this case would be:

Recall = 10/(10+0) = 1

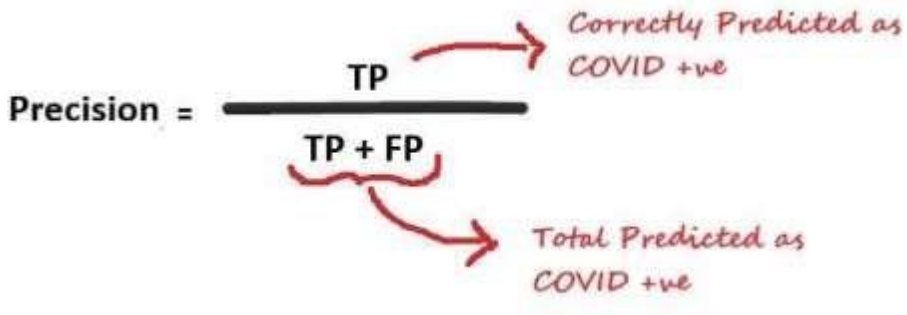
That's a huge problem. So concluding, it turns out that accuracy was a bad idea because labeling everyone as negative can increase the accuracy but hoping Recall will be a good measure in this context but then realized that labeling everyone as positive will increase recall as well.

So recall independently is not a good measure.

There is another measure called Precision

Precision

Precision gives the fraction of correctly identified as positive out of all predicted as positives.



Considering our second bad strategy of labeling every passenger as positive, the precision would be :

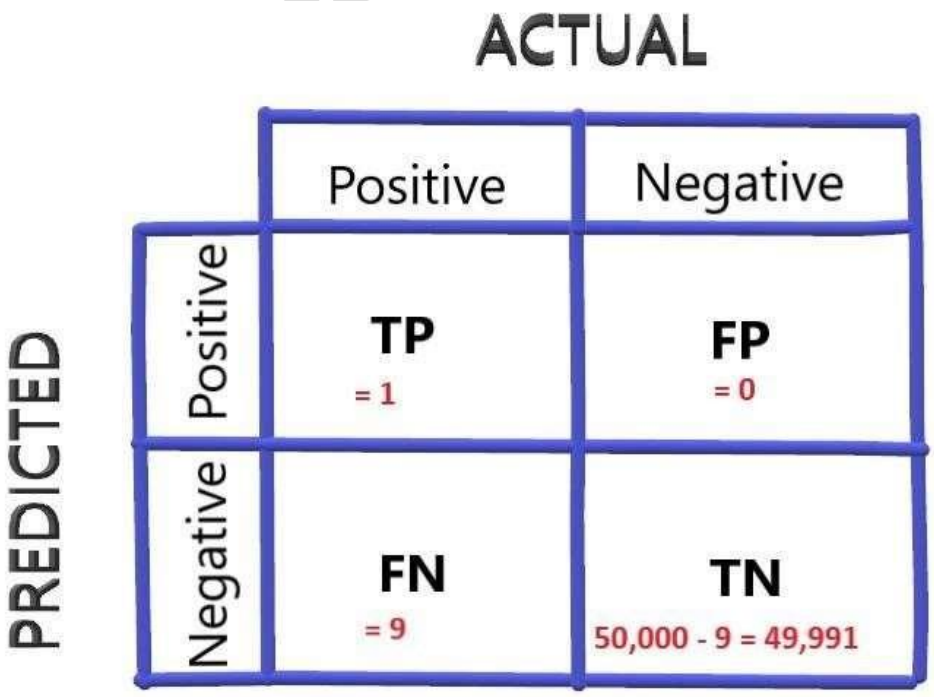
Precision = 10 / (10 + 49990) = 0.0002

While this bad strategy has a good recall value of **1** but it has a terrible precision value of **0.0002.**

This clarifies that recall alone is not a good measure, we need to consider precision value.

Considering another Case (This will be the last one, I promise :P) of labeling the top passengers as COVID positive that is labeling passengers with the highest probability of having

COVID. Let's say we got only one such passenger. The confusion matrix in this case will be:



Precision comes out to be: 1/ (1 + 0) = **1**

Precision value is good in this case but let's check for recall value once:

Recall = 1 / (1 + 9) = 0.1

Precision value is good in this context but recall value is low.

Scenario	Accuracy	Recall	Precision
Classifying all the passengers as –ve	High	Low	Low
Classifying all the passengers as positive	Low	High	Low
Classifying passengers with max. probability as positive	High	Low	Low

In some cases, we are pretty sure that we want to maximize either recall or precision at the cost of others. As in this case of labeling passengers, we really want to get the predictions right for COVID positive passengers because it is really expensive to not predict the passenger right as allowing COVID positive person to proceed will result in increasing the spread. So we are more interested in recall here.

Unfortunately, you can't have it both ways: increasing precision reduces recall and vice versa. This is called precision/recall tradeoff.

SPECIFICITY:

Specificity is the metric that evaluates a model's ability to predict true negatives of each available category. These metrics apply to any categorical model.

F1 Score

It is defined as the harmonic mean of the model's precision and recall.

$$\text{F1 Score} = 2 * \frac{\text{Precision} * \text{Recall}}{\text{Precision} + \text{Recall}}$$

You must be wondering why Harmonic mean and not simple average?
You're heading in the right direction.

We use Harmonic mean because it is not sensitive to extremely large values, unlike simple averages. Say, we have a model with a precision of 1, and recall of 0 gives a simple average as 0.5 and an F1 score of 0. If one of the parameters is low, the second one no longer matters in the F1 score. The F1 score favors classifiers that have similar precision and recall. Thus, the F1 score is a better measure to use if you are seeking a balance between Precision and Recall.

ROC/AUC Curve

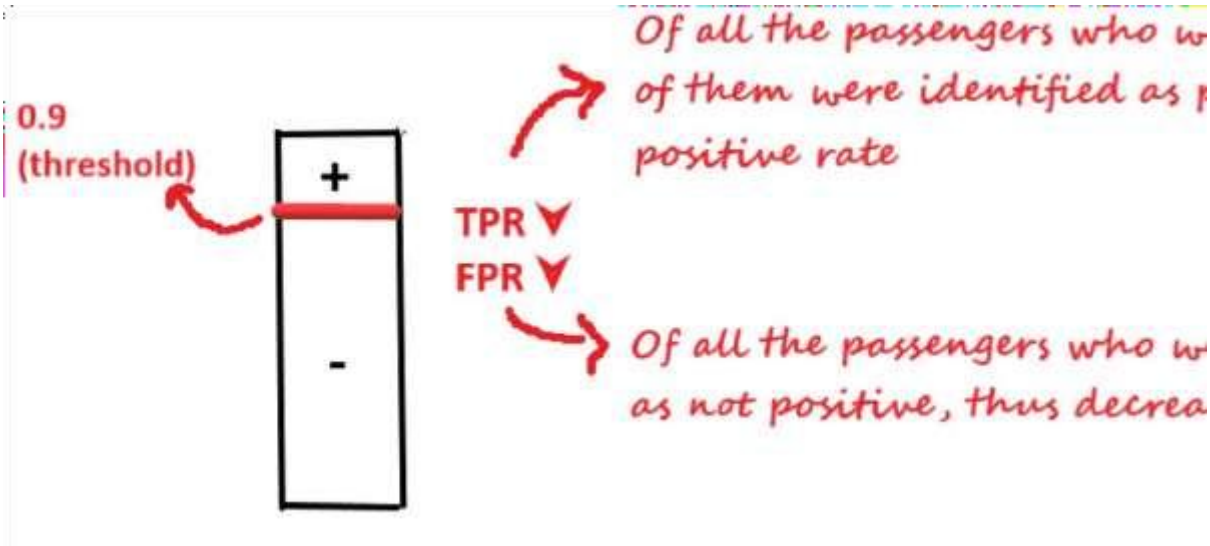
The receiver operator characteristic is another common tool used for evaluation. It plots out the sensitivity and specificity for every possible decision rule cutoff between 0 and 1 for a model. For classification problems with probability outputs, a threshold can convert probability outputs to classifications. we get the ability to control the confusion matrix a little bit. So by changing the threshold, some of the numbers can be changed in the confusion matrix. But the most important question here is, how to find the right threshold? Of course, we don't want to look at the confusion matrix every time the threshold is changed, therefore here comes the use of the ROC curve.

For each possible threshold, the ROC curve plots the False positive rate versus the true positive rate.

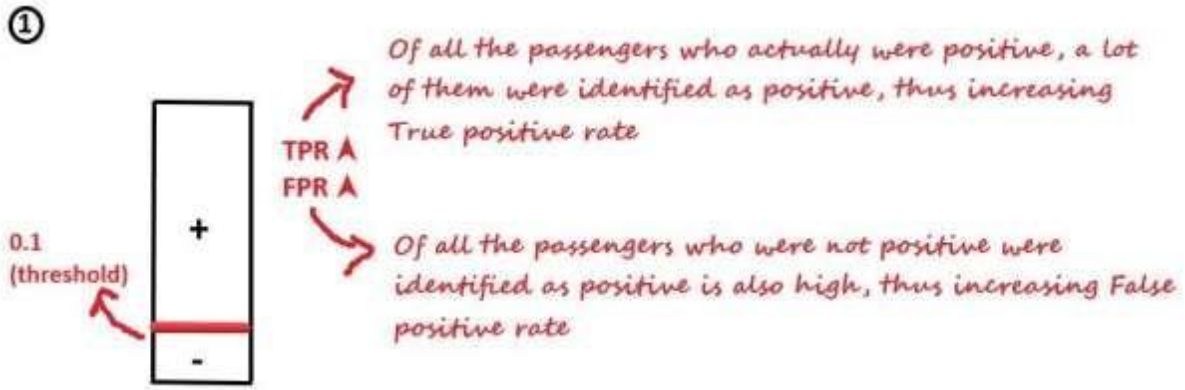
False Positive Rate: Fraction of negative instances that are incorrectly classified as positive.

True Positive Rate: Fraction of positive instances that are correctly predicted as positive.

Now, think about having a **low** threshold. So amongst all the probabilities arranged in ascending order, everything below bad is

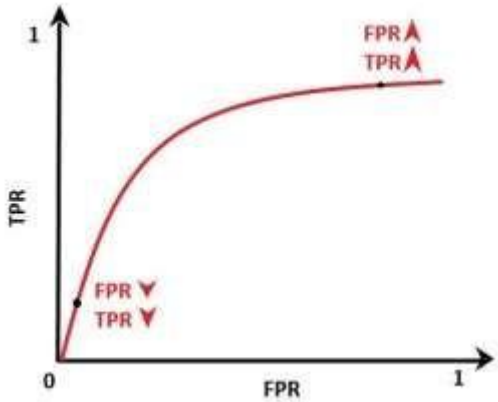


considered as negative and everything above 0.1 is considered as positive. By choosing this, you're being very liberal.



But if you set your threshold as **high**, say 0.9.

Below is the ROC curve for the same model at different threshold values.



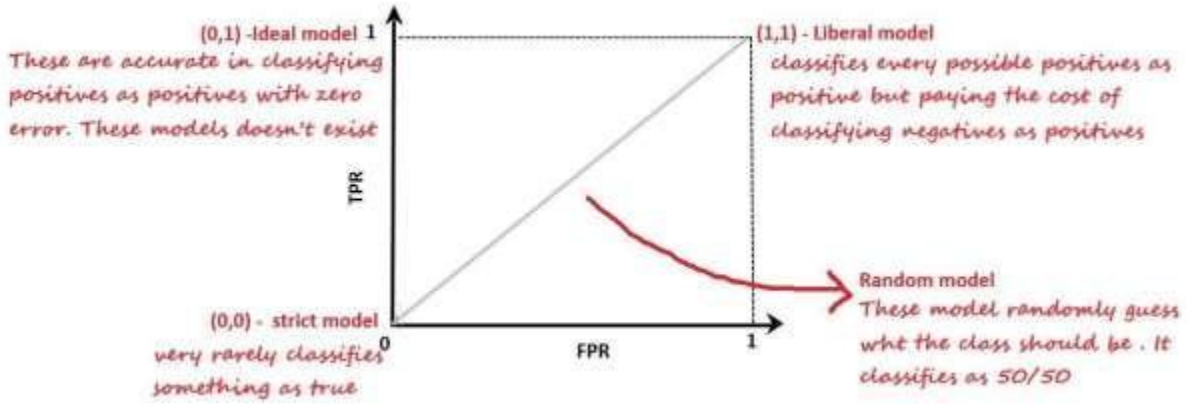
From the above graph, it can be seen that the true positive rate increases at a higher rate but suddenly at a certain threshold, the TPR starts to taper off.

For every increase in TPR, we have to pay a cost, the cost of an increase in FPR.

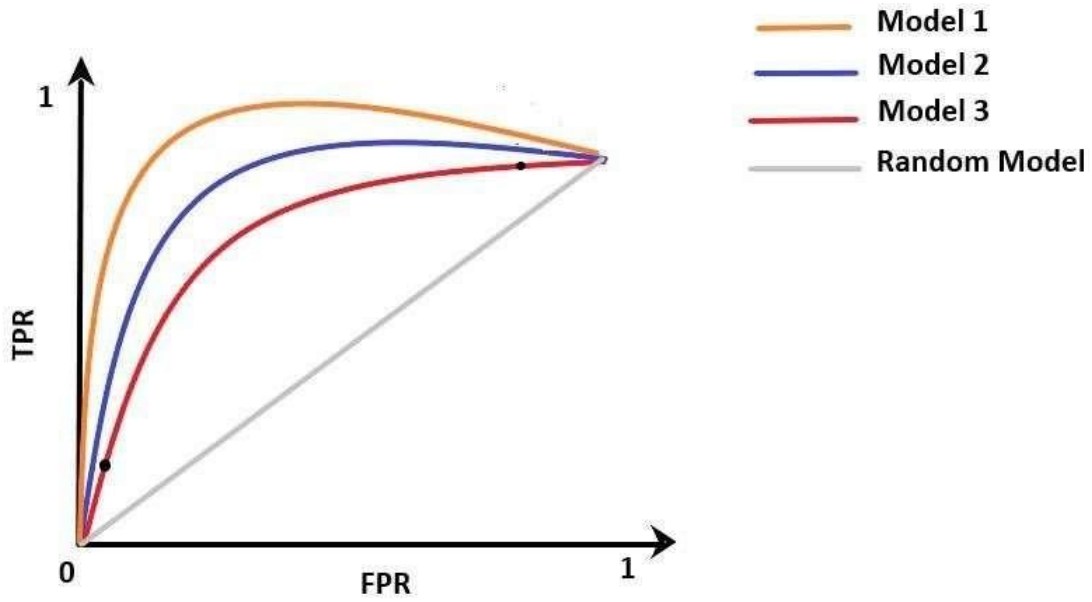
At the initial stage, the TPR increase is higher than FPR

So, we can select the threshold for which the TPR is high and FPR is low.

Now, let's see what different values about TPR and FPR tell us about the model.



For different models, we will have a different ROC curve. Now, how to compare different models? From the above plot, it is clear that the curve is in the upper triangle, the good the model is. One way to compare classifiers is to measure the area under the curve for ROC.



AUC(Model 1) > AUC(Model 2) > AUC(Model 2)

Thus Model 1 is the best of all.

What is Dimensionality Reduction?

In machine learning classification problems, there are often too many factors on the basis of which the final classification is done. These factors are basically variables called features. The higher the number of features, the harder it gets to visualize the training set and then work on it. Sometimes, most of these features are correlated, and hence redundant. This is where dimensionality reduction algorithms come into play. Dimensionality reduction is the process of reducing the number of random variables under consideration, by obtaining a set of principal variables. It can be divided into feature selection and feature extraction.

Why is Dimensionality Reduction important in Machine Learning and Predictive Modeling?

An intuitive example of dimensionality reduction can be discussed through a simple e-mail classification problem, where we need to classify whether the e-mail is spam or not. This can involve a large number of features, such as whether or not the e-mail has a generic title, the content of the e-mail, whether the e-mail uses a template, etc.

However, some of these features may overlap. In another condition, a classification problem that relies on both humidity and rainfall can be collapsed into just one underlying feature, since both of the aforementioned are correlated to a high degree. Hence, we can reduce the number of features in such problems. A 3-D classification problem can be hard to visualize, whereas a 2-D one can be mapped to a simple 2 dimensional space, and a 1-D problem to a simple line. The below figure illustrates this concept, where a 3-D feature space is split into two 2-D feature spaces, and later, if found to be correlated, the number of features can be reduced even further.

Components of Dimensionality Reduction

There are two components of dimensionality reduction:

Feature selection: In this, we try to find a subset of the original set of variables, or features, to get a smaller subset which can be used to model the problem. It usually involves three ways: Filter

Wrapper

Embedded

Feature extraction: This reduces the data in a high dimensional space to a lower dimension space, i.e. a space with lesser no. of dimensions.

Machine-Learning-Course

Methods of Dimensionality Reduction

The various methods used for dimensionality reduction include:

Principal Component Analysis (PCA)

Linear Discriminant Analysis (LDA)

Generalized Discriminant Analysis (GDA)

Dimensionality reduction may be both linear or non-linear, depending upon the method used. The prime linear method, called Principal Component Analysis, or PCA, is discussed below.

Principal Component Analysis

This method was introduced by Karl Pearson. It works on a condition that while the data in a higher dimensional space is mapped to data in a lower dimension space, the variance of the data in the lower dimensional space should be maximum. It involves the following steps:

Construct the covariance matrix of the data.

Compute the eigenvectors of this matrix.

Eigenvectors corresponding to the largest eigenvalues are used to reconstruct a large fraction of variance of the original data.

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

Advantages of Dimensionality Reduction

It helps in data compression, and hence reduced storage space.

It reduces computation time.

It also helps remove redundant features, if any.

Disadvantages of Dimensionality Reduction

It may lead to some amount of data loss.

PCA tends to find linear correlations between variables, which is sometimes undesirable.

PCA fails in cases where mean and covariance are not enough to define datasets.

We may not know how many principal components to keep- in practice, some thumb rules are applied.

This article is contributed by Anannya Uberoi. If you like GeeksforGeeks and would like to contribute, you can also write an article using write.geeksforgeeks.org or mail your article to reviewteam@geeksforgeeks.org. See your article appearing on the GeeksforGeeks main page and help other Geeks.

Common Dimensionality Reduction Techniques

Dimensionality reduction can be done in two different ways:

- By only keeping the most relevant variables from the original dataset (this technique is called feature selection)
- By finding a smaller set of new variables, each being a combination of the input variables, containing basically the same information as the input variables (this technique is called dimensionality reduction)

We will now look at various dimensionality reduction techniques and how to implement each of them in Python.

3.1 Missing Value Ratio

Suppose you're given a dataset. What would be your first step? You would naturally want to explore the data first before building model. While exploring the data, you find that your

dataset has some missing values. Now what? You will try to find out the reason for these missing values and then impute them or drop the variables entirely which have missing values (using appropriate methods).

What if we have too many missing values (say more than 50%)? Should we impute the missing values or drop the variable? I would prefer to drop the variable since it will not have much information. However, this isn't set in stone. We can set a threshold value and if the percentage of missing values in any variable is more than that threshold, we will drop the variable.

First, let's load the data:

```
# read the data
train=pd.read_csv("Train_UWu5bXk.csv")
```

Note: The path of the file should be added while reading the data.

Now, we will check the percentage of missing values in each variable. We can use *.isnull().sum()* to calculate this.

```
# checking the percentage of missing values in each variable
train.isnull().sum()/len(train)*100
```

```
Item_Identifier              0.000000
Item_Weight                 17.165317
Item_Fat_Content             0.000000
Item_Visibility              0.000000
Item_Type                    0.000000
Item_MRP                     0.000000
Outlet_Identifier            0.000000
Outlet_Establishment_Year    0.000000
Outlet_Size                 28.276428
Outlet_Location_Type         0.000000
Outlet_Type                  0.000000
Item_Outlet_Sales            0.000000
dtype: float64
```

As you can see in the above table, there aren't too many missing values (just 2 variables have them actually). We can impute the values using appropriate methods, or we can set a threshold of, say 20%, and remove the variable having more than 20% missing values. Let's look at how this can be done in Python:

```
# saving missing values in a variable a =
train.isnull().sum()/len(train)*100 # saving column
names in a variable variables = train.columns
variable = [ ] for i in range(0,12):
    if a[i]<=20:   #setting the threshold as 20%          variable.append(variables[i])
```

So the variables to be used are stored in "variable", which contains only those features where the missing values are less than 20%.

3.2 Low Variance Filter

Consider a variable in our dataset where all the observations have the same value, say 1. If we use this variable, do you think it can improve the model we will build? The answer is no, because this variable will have zero variance.

So, we need to calculate the variance of each variable we are given. Then drop the variables having low variance as compared to other variables in our dataset. The reason for doing this, as I mentioned above, is that variables with a low variance will not affect the target variable.

Let's first impute the missing values in the *Item_Weight* column using the median value of the known *Item_Weight* observations. For the *Outlet_Size* column, we will use the mode of the known *Outlet_Size* values to impute the missing values:

```
train['Item_Weight'].fillna(train['Item_Weight'].median(), inplace=True)
train['Outlet_Size'].fillna(train['Outlet_Size'].mode()[0], inplace=True)
```

Let's check whether all the missing values have been filled:

```
train.isnull().sum()/len(train)*100
```

```
Item_Identifier              0.0
Item_Weight                  0.0
Item_Fat_Content             0.0
Item_Visibility              0.0
Item_Type                    0.0
Item_MRP                     0.0
Outlet_Identifier            0.0
Outlet_Establishment_Year    0.0
Outlet_Size                  0.0
Outlet_Location_Type         0.0
Outlet_Type                  0.0
Item_Outlet_Sales            0.0
dtype: float64
```

Voila! We are all set. Now let's calculate the variance of all the numerical variables.

```
train.var()
```

```
Item_Weight                  1.786956e+01
Item_Visibility              2.662335e-03
Item_MRP                     3.878184e+03
Outlet_Establishment_Year    7.008637e+01
Item_Outlet_Sales            2.912141e+06
dtype: float64
```

As the above output shows, the variance of *Item_Visibility* is very less as compared to the other variables. We can safely drop this column. This is how we apply low variance filter. Let's implement this in Python:

```
numeric = train[['Item_Weight', 'Item_Visibility', 'Item_MRP',
'Outlet_Establishment_Year']] var =
numeric.var() numeric =
numeric.columns variable = [ ] for i in
range(0,len(var)):
    if var[i]>=10:   #setting the threshold as 10%        variable.append(numeric[i+1])
```

The above code gives us the list of variables that have a variance greater than 10.

3.3 High Correlation filter

High correlation between two variables means they have similar trends and are likely to carry similar information. This can bring down the performance of some models drastically (linear

and logistic regression models, for instance). We can calculate the correlation between independent numerical variables that are numerical in nature. If the correlation coefficient crosses a certain threshold value, we can drop one of the variables (dropping a variable is highly subjective and should always be done keeping the domain in mind).

As a general guideline, we should keep those variables which show a decent or high correlation with the target variable.

Let's perform the correlation calculation in Python. We will drop the dependent variable (*Item_Outlet_Sales*) first and save the remaining variables in a new dataframe (*df*).

```
df=train.drop('Item_Outlet_Sales', 1) df.corr()
```

	Item_Weight	Item_Visibility	Item_MRP	Outlet_Establishment_Year
Item_Weight	1.000000	-0.014168	0.024951	0.007739
Item_Visibility	-0.014168	1.000000	-0.001315	-0.074834
Item_MRP	0.024951	-0.001315	1.000000	0.005020
Outlet_Establishment_Year	0.007739	-0.074834	0.005020	1.000000

Wonderful, we don't have any variables with a high correlation in our dataset. Generally, if the correlation between a pair of variables is greater than 0.5-0.6, we should seriously consider dropping one of those variables.

3.4 Random Forest

Random Forest is one of the most widely used algorithms for feature selection. It comes packaged with in-built feature importance so you don't need to program that separately. This helps us select a smaller subset of features.

We need to convert the data into numeric form by applying one hot encoding, as Random Forest (Scikit-Learn Implementation) takes only numeric inputs. Let's also drop the ID variables
(*Item_Identifier* and *Outlet_Identifier*) as these are just unique numbers and hold no significant importance for us currently.

```
from sklearn.ensemble import RandomForestRegressor df=df.drop(['Item_Identifier',
'Outlet_Identifier'], axis=1) model = RandomForestRegressor(random_state=1,
max_depth=10) df=pd.get_dummies(df)
model.fit(df,train.Item_Outlet_Sales)
```

After fitting the model, plot the feature importance graph:

```
features = df.columns
importances = model.feature_importances_
indices = np.argsort(importances)[-9:]  # top 10 features plt.title('Feature
Importances')
plt.barh(range(len(indices)), importances[indices], color='b', align='center')
plt.yticks(range(len(indices)), [features[i] for i in indices]) plt.xlabel('Relative
Importance') plt.show()
```

Reduce Data Dimensionality using PCA – Python

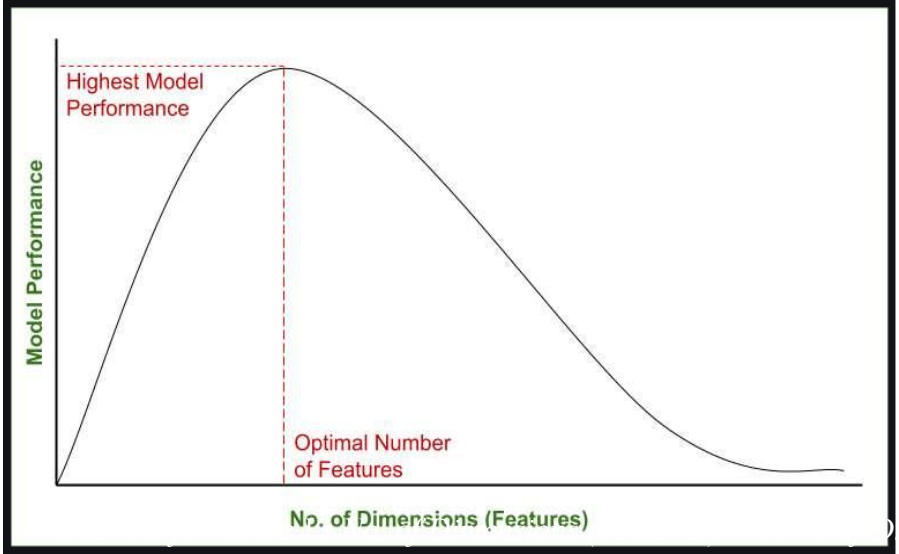
- Difficulty Level : Expert
- Last Updated : 18 Jul, 2022

- Read
- Discuss
- Practice
- Video
- Courses

Introduction

overfitting

ber of dimensions of the dataset. It can be observed that the model
at an option dimension, beyond which it starts decreasing.

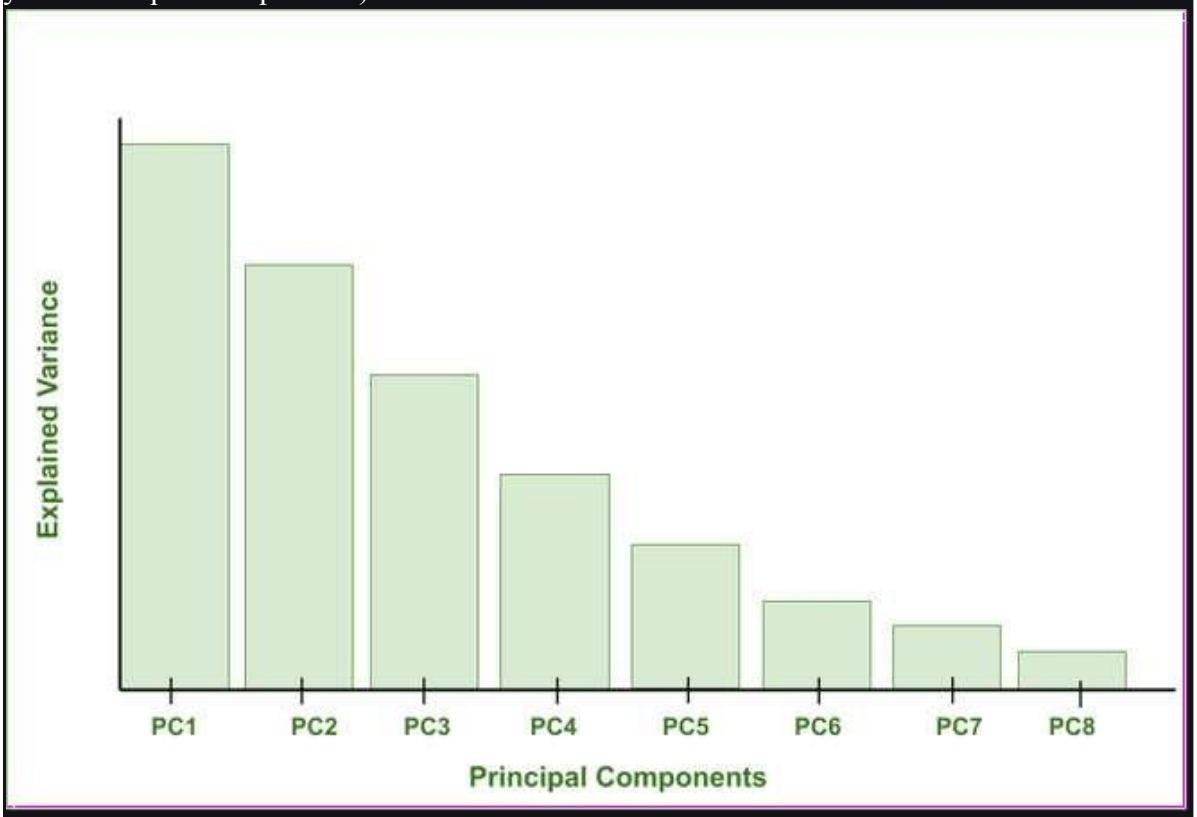


ensionality Reduction is a statistical/ML-based technique wherein
ber of features in our dataset and obtain a dataset with an optimal num

of the most common ways to accomplish Dimensionality Reduction is
rein we reduce the number of dimensions by mapping a higher dimen
r-dimensional feature space. The most popular technique of F

As stated earlier, Principal Component Analysis is a technique of feature extraction that maps a higher dimensional feature space to a lowerdimensional feature space. While reducing the number of dimensions, PCA ensures that maximum information of the original dataset is retained in the dataset with the reduced no. of dimensions and the co-relation between the newly obtained Principal Components is minimum. The new features obtained after applying PCA are called Principal Components and are denoted as ***PCi (i=1,2,3…n)***. Here, (Principal Component-1) PC1 captures the maximum information of the original dataset, followed by PC2, then PC3 and so on.

The following bar graph depicts the amount of Explained Variance captured by various Principal Components. (The Explained Variance defines the amount of information captured by the Principal Components).



Explained Variance Vs Principal Components

In order to understand the mathematical aspects involved in Principal Component Analysis do check out Mathematical Approach to PCA. In this article, we will focus on how to use PCA in Python for Dimensionality Reduction.

Steps to Apply PCA in Python for Dimensionality Reduction

We will understand the step by step approach of applying Principal Component Analysis in Python with an example. In this example, we will use the iris dataset, which is already present in the sklearn library of Python.

Step-1: Import necessary libraries

All the necessary libraries required to load the dataset, pre-process it and then apply PCA on it are mentioned below:

Python3

```
# Import necessary libraries

from sklearn import datasets  # to retrieve the iris Dataset

import pandas as pd  # to load the dataframe

from sklearn.preprocessing import StandardScaler # to standardize the features from sklearn.decomposition import PCA # to apply PCA import seaborn as sns # to plot the heat maps
```

Step-2: Load the dataset

After importing all the necessary libraries, we need to load the dataset. Now, the iris dataset is already present in sklearn. First, we will load it and then convert it into a pandas data frame for ease of use.

Python3

```
#Load the Dataset iris =

datasets.load_iris()

#convert the dataset into a pandas data frame

df = pd.DataFrame(iris['data'], columns =
iris['feature_names'])

#display the head (first 5 rows) of the dataset df.head()
```

Output:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
0	5.1	3.5	1.4	0.2
1	4.9	3.0	1.4	0.2
2	4.7	3.2	1.3	0.2
3	4.6	3.1	1.5	0.2
4	5.0	3.6	1.4	0.2

iris dataset

Step-3: Standardize the features

Before applying PCA or any other Machine Learning technique it is always considered good practice to standardize the data. For this, Standard Scalar is the most commonly used scalar. Standard Scalar is already present in sklearn. So, now we will standardize the feature set using Standard Scalar and store the scaled feature set as a pandas data frame.

Python3

```
#Standardize the features
#Create an object of StandardScaler which is present insklearn.preprocessing scalar =
scaled_data = pd.DataFrame(scalar.fit_transform(df))
#scaling the data scaled_data
```

Output

tal width (cm)
-1.315444
-1.315444

Scaled iris

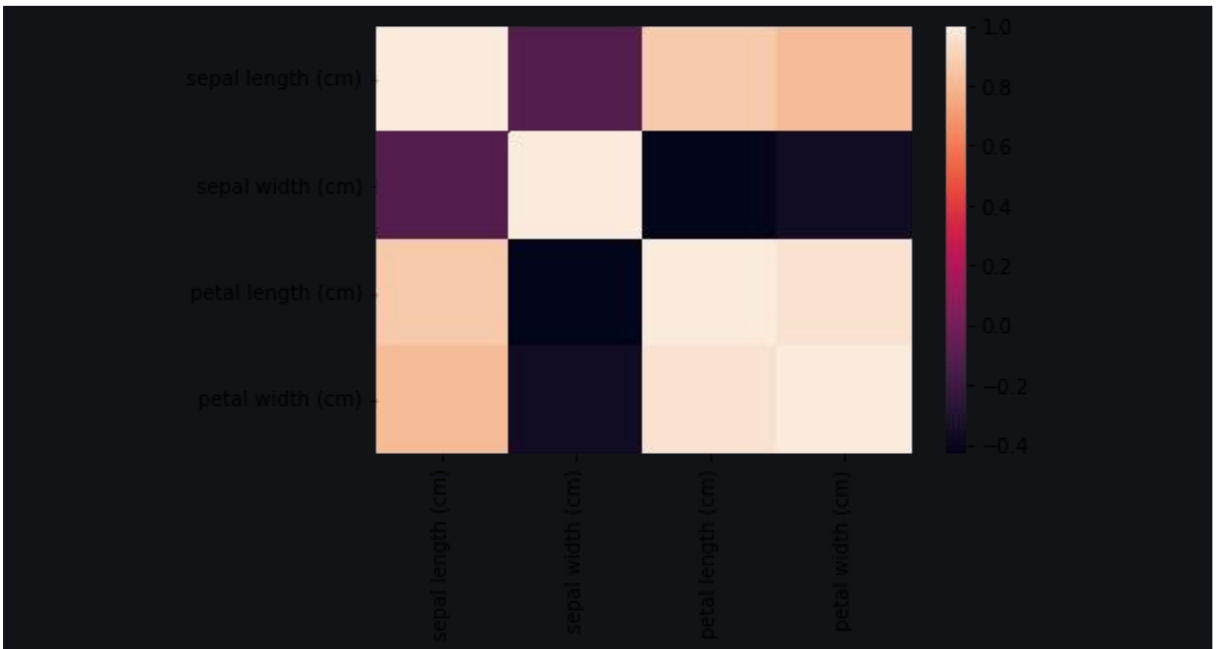
Step-3: Check the Co-relation between features without PCA

Now, we will check the co-relation between our scaled dataset using a heat map. For this, we have already imported the seaborn library in Step-1. The correlation between various features is given by the *corr()* function and then the heat map is

Python3

```
#Check the Co-relation between features without PCA sns.heatmap(scaled_data.corr())
```

Output:



Co-relation Heatmap of Iris dataset without PCA

We can observe from the above heatmap that sepal length & petal length and petal length & petal width have high co-relation. Thus, we evidently need to apply dimensionality reduction. If you are already aware that your dataset needs dimensionality reduction – you can skip this step.

Step-4: Applying Principal Component Analysis

We will apply PCA on the scaled dataset. For this Python offers yet another in-built class called PCA which is present in*sklearn.decomposition*, which we have already imported in step -1. We need to create an object of PCA and while doing so we also need to initialize n_components – which is the number of principal components we want in our final dataset. Here, we have taken n_components = 3, which means our final feature set will have 3 columns. We fit our scaled data to the PCA object which gives us our reduced dataset.

Python

```
#Applying PCA
#Taking no. of Principal Components as 3 pca = PCA(n_components = 3)
```

```
pca.fit(scaled_data) data_pca =
pca.transform(scaled_data)
data_pca =
pd.DataFrame(data_pca,columns=['PC1','PC2','PC3']) data_pca.head()
```

Output:

	PC1	PC2	PC3
0	-2.264703	0.480027	-0.127706
1	-2.080961	-0.674134	-0.234609
2	-2.364229	-0.341908	0.044201
3	-2.299384	-0.597395	0.091290
4	-2.389842	0.646835	0.015738

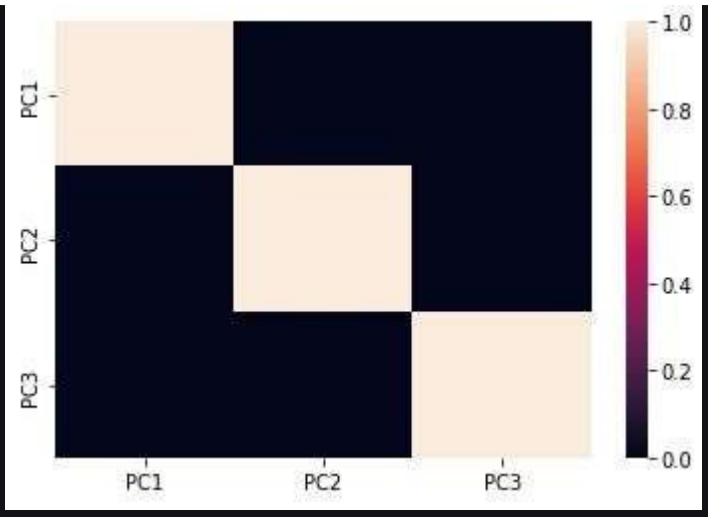
PCA Dataset

Step-5: Checking Co-relation between features after PCA

Now that we have applied PCA and obtained the reduced feature set, we will check the co-relation between various Principal Components, again by using a heatmap.

```
#Checking Co-relation between features after PCA sns.heatmap(data_pca.corr())
```

Output:



Heatmap after PCA

The above heatmap clearly depicts that there is no correlation between various obtained principal components (PC1, PC2, and PC3). Thus, we have moved from higher dimensional feature space to a lower -dimensional feature space while ensuring that there is no correlation between the so obtained PCs is minimum. Hence, we have accomplished the objectives of PCA.

Advantages of Principal Component Analysis (PCA):

1. For efficient working of ML models, our feature set needs to have features with no co-relation. After implementing the PCA on our dataset, all the Principal Components are independent – there is no correlation among them.
2. A Large number of feature sets lead to the issue of overfitting in models. PCA reduces the dimensions of the feature set – thereby reducing the chances of overfitting.
3. PCA helps us reduce the dimensions of our feature set; thus, the newly formed dataset comprising Principal Components need less disk/cloud space for storage while retaining maximum information.

Deployment process

Imagine building a supervised machine learning(ML) model to decide whether a loan application should be approved. With the model confidence level (probability) in successful applications, we can calculate the risk-free loanable amount. The deployment of such ML-model is the goal of this project.

Join me as I walk you through my internship project at Data Science Nigeria where I have been opportune to work alongside crème de la crème of Data Scientists, Software Engineers

and Researchers in Artificial intelligence. The goal is to raise 1 million Ai talent in 10 years which am proud to be part of.

Whereas data scientists build Machine learning models in jupyter lab, google colab and the likes, Machine learning engineers take the built model into production.

Deployment of an ML-model simply means the integration of the model into an existing production environment which can take in an input and return an output that can be used in making practical business decisions.

To demystify the deployment process, I created this post which is split into four chapters. In chapter 1, we cover the process of building a baseline model and in chapter 2, we adopt this model intro production-ready code. Chapters 3 and 4 are covered in Part 2 of this series.

Chapter 1: Building a base-line model

A base-line model in this context refers to an ML-model having the minimum possible number of features but with good evaluation measures. When building an ML-model for deployment purposes, you must always have your end-users in mind. A model with thousands of features to attain an accuracy 90+% on evaluation might not be good enough for deployment for several reasons:

- **Portability**: Is the ability of software to be transferred from one machine or system to another. A portable model decreases the response time of your code and the amount of code you have to rewrite when your goals change will be minimal.
- **Scalability**: Is the ability of a program to scale. A model is considered scalable when it doesn't need to be redesigned to maintain effective performance.
- **Operationalization**: Refers to the deployment of models to be consumed by business applications to predict the target value of a classification/regression problem.
- **Test**: Refers to the validation of output to processes and input. A robust model will be unnecessarily difficult to test.

Click to download the dataset that will be used for this project, the notebook and the exclusive summary that contain Exploratory Data
Analysis, Data Preprocessing, Feature Engineering, Feature Selection, Training the ML model and Model Evaluation.

As a first step, we use pandas to import our test and training data.

```
train = pd.read_csv("train.csv") test =
pd.read_csv("test.csv")
```

Data Preprocessing: This involves the transformation of raw data into an understandable format suitable to train a model. It entails taking care of missing values, outliers, encoding categorical variables.

For the purpose of deployment, you want to always keep a record of the values you are using to fill the missing values and the approach has to be replicable.