**Practical No 1**

**Aim: Design an Expert System using AIML**

**Library:**

**AIML**

AIML was developed by Richard Wallace. He made a bot called A.L.I.C.E. (Artificial Linguistics Internet Computer Entity) which won several artificial intelligence awards. Interestingly, one of the Turing tests to look for artificial intelligence is to have a human chat with a bot through a text interface for several minutes and see if they thought it, was a human. AIML is a form of XML that defines rules for matching patterns and determining responses.

**What is Expert System?**

In artificial intelligence, an expert system is a computer system that emulates the decision-making ability of a human expert. Expert systems are designed to solve complex problems by reasoning through bodies of knowledge, represented mainly as if-then rules rather than through conventional procedural code.

**Code:**

**AAIPractical\_1.py**

**import** aiml

**import** time

time**.**clock **=** time**.**time

kernel **=** aiml**.**Kernel**()**

kernel**.**learn**(**"std-startup.xml"**)**

kernel**.**respond**(**"LOAD AIML B"**)**

**while** **True:**

input\_text **=** **input(**">Human: "**)**

response **=** kernel**.**respond**(**input\_text**)**

**print(**">Bot: " **+** response**)**

**basic\_chat.aiml**

<aiml version="1.0.1" encoding="UTF-8">

<category>

<pattern>HELLO</pattern>

<template>

Hello my new friend Osama!

</template>

</category>

<category>

<pattern>What are you</pattern>

<template>

I am bot,and i am Silly

</template>

</category>

<category>

<pattern>WHAT DO YOU DO</pattern>

<template>

I am here to annoy you

</template>

</category>

</aiml>

**std-startup.xml**

<aiml version=**"1.0.1"** encoding=**"UTF-8"** >

<category>

<pattern>**LOAD AIML B** </pattern>

<template>

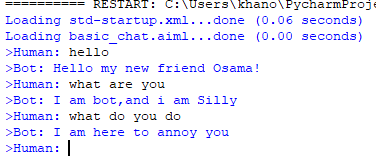
<learn>**basic\_chat.aiml**</learn>

</template>

</category>

</aiml>

**Output:**

****

**Practical No 2**

**Aim: Design a BOT using AIML**

AIML stands for Artificial Intelligence Modelling Language. AIML is an XML based markup language meant to create artificial intelligent applications. AIML makes it possible to create human interfaces while keeping the implementation simple to program, easy to understand and highly maintainable.

AIML Tags/Description:

● <aiml> defines the beginning and end of a AIML document.

● <category> defines the unit of knowledge in bot's knowledge base.

● <pattern> defines the pattern to match what is user may input to an bot.

● <template> defines the response of a bot to user's input.

**Code:**

**Python file:**

**import** aiml

**import** time

kernel **=** aiml**.**Kernel**()**

time**.**clock **=** time**.**time

kernel**.**learn**(**"std2\_startup.xml"**)**

kernel**.**respond**(**"LOAD PRAC 2"**)**

**while** 1**:**

input\_text **=** **input(**"-->Osama-->"**)**

response **=** kernel**.**respond**(**input\_text**)**

**print(**"-->Bot-->"**+**response**)**

**P2\_chat.aiml**

<aiml version='1.0.1' encoding = 'UTF-8'>

<category>

<pattern>HELLO</pattern>

<template> Hello user </template>

</category>

<category>

<pattern>SUNDAY</pattern>

<template>the day before Monday following Saturday</template>

</category>

<category>

<pattern>MONDAY</pattern>

<template>the day before Tuesday following Sunday</template>

</category>

<category>

<pattern>TUESDAY</pattern>

<template>the day before Wednesday following Monday</template>

</category>

<category>

<pattern>WEDNESDAY</pattern>

<template>the day before Thursday following Tuesday</template>

</category>

<category>

<pattern>THURSDAY</pattern>

<template>the day before Monday following Wednesday</template>

</category>

<category>

<pattern>FRIDAY</pattern>

<template>the day before Saturday following Thursday</template>

</category>

<category>

<pattern>SATURDAY</pattern>

<template>the day before Sunday following Friday</template>

</category>

</aiml>

**std2\_startup.xml**

<aiml version=**"1.0.1"** encoding=**"UTF-8"**>

<category>

<pattern>**LOAD PRAC 2**</pattern>

<template>

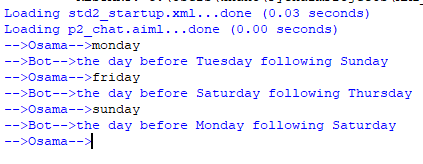
<learn>**p2\_chat.aiml**</learn>

</template>

</category>

</aiml>

**Output:**



**Practical No 3**

**Aim: Implement Bayes Theorem using Python.**

**Library:**

*sklearn*

Scikit-learn is a free machine learning library for Python. It features various algorithms like support vector machine, random forests, and k-neighbours, and it also supports Python numerical and scientific libraries like NumPy and SciPy.

**Description:**

**What Is Bayes' Theorem?**

Bayes' Theorem, named after 18th-century British mathematician Thomas Bayes, is a mathematical formula for determining [conditional probability](https://www.investopedia.com/terms/c/conditional_probability.asp). Conditional probability is the likelihood of an outcome occurring, based on a previous outcome having occurred in similar circumstances. Bayes' theorem provides a way to revise existing predictions or theories (update probabilities) given new or additional evidence.

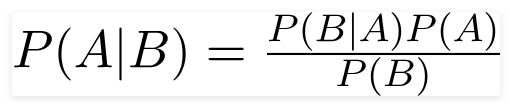
**What is a classifier?**

A classifier is a machine learning model that is used to discriminate different objects based on certain features.

**Principle of Naive Bayes Classifier:**

A Naive Bayes classifier is a probabilistic machine learning model that’s used for classification task. The crux of the classifier is based on the Bayes theorem.

**Bayes Theorem:**



Using Bayes theorem, we can find the probability of **A** happening, given that **B** has occurred. Here, **B** is the evidence and **A** is the hypothesis. The assumption made here is that the predictors/features are independent. That is presence of one particular feature does not affect the other. Hence it is called naive.

**Types of Naive Bayes Classifier:**

**Multinomial Naive Bayes:**

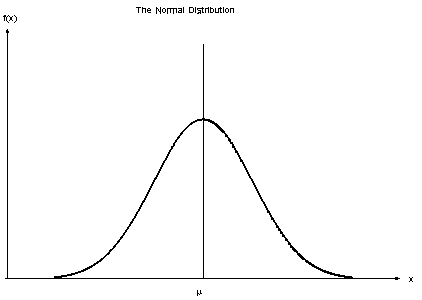
This is mostly used for document classification problem, i.e whether a document belongs to the category of sports, politics, technology etc. The features/predictors used by the classifier are the frequency of the words present in the document.

**Bernoulli Naive Bayes:**

This is similar to the multinomial naive bayes but the predictors are boolean variables. The parameters that we use to predict the class variable take up only values yes or no, for example if a word occurs in the text or not.

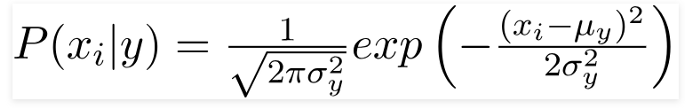
**Gaussian Naive Bayes:**

When the predictors take up a continuous value and are not discrete, we assume that these values are sampled from a gaussian distribution.



Gaussian Distribution(Normal Distribution)

Since the way the values are present in the dataset changes, the formula for conditional probability changes to,



**Code:**

**from** sklearn**.**datasets **import** load\_breast\_cancer

**from** sklearn**.**model\_selection **import** train\_test\_split

**from** sklearn**.**naive\_bayes **import** GaussianNB

**from** sklearn**.**metrics **import** accuracy\_score

**print(**"Osama Khan "**)**

data **=** load\_breast\_cancer**()**

**print(**data**.**keys**())**

targets\_name **=** data**[**'target\_names'**]**

target\_value **=** data**[**'target'**]**

features\_name **=** data**[**'feature\_names'**]**

features **=** data**[**'data'**]**

**print(**targets\_name**[:**10**])**

**print(**target\_value**[:**10**])**

**print(**features\_name**[:**10**])**

**print(**features**[:**10**])**

X\_train**,** X\_test**,** y\_train**,** y\_test **=** train\_test\_split**(**features**,** target\_value**,** test\_size**=**0.30**,**random\_state**=**42**)**

gnb **=** GaussianNB**()**

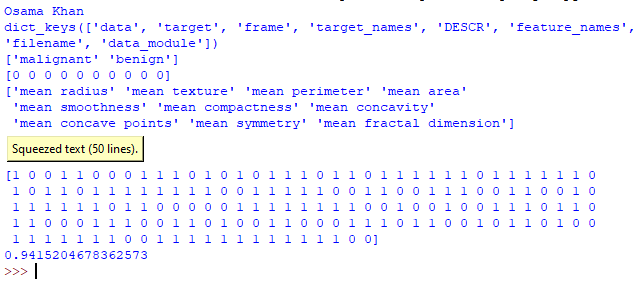
model **=** gnb**.**fit**(**X\_train**,**y\_train**)**

preds **=** gnb**.**predict**(**X\_test**)**

**print(**preds**)**

**print(**accuracy\_score**(**y\_test**,** preds**))**

**Output:**



**Practical No 4:**

**Aim: Implement Conditional Probability and Joint Probability using python.**

**Library:**

*enum:*

Enum is a class in python for creating enumerations, which are a set of symbolic names (members) bound to unique, constant values. The members of an enumeration can be compared by these symbolic anmes, and the enumeration itself can be iterated over. An enum has the following characteristics.

*random:*

Python **Random module** is an in-built module of Python which is used to generate random numbers. These are pseudo-random numbers means these are not truly random. This module can be used to perform random actions such as generating random numbers, print random a value for a list or string, etc.

**Description:**

What is conditional Probability?

**What Is Conditional Probability?**

Conditional probability is defined as the likelihood of an event or outcome occurring, based on the occurrence of a previous event or outcome. Conditional probability is calculated by multiplying the [probability](https://www.investopedia.com/terms/c/compound-probability.asp) of the preceding event by the updated probability of the succeeding, or conditional, event.

Conditional probability can be contrasted with [unconditional probability](https://www.investopedia.com/terms/u/unconditional_probability.asp). Unconditional probability refers to the likelihood that an event will take place irrespective of whether any other events have taken place or any other conditions are present.

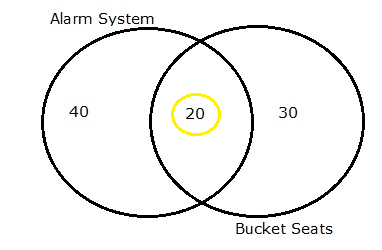
In a group of 100 sports car buyers, 40 bought alarm systems, 30 purchased bucket seats, and 20 purchased an alarm system and bucket seats. If a car buyer chosen at random bought an alarm system, what is the probability they also bought bucket seats?

Step 1: Figure out P(A). It’s given in the question as 40%, or 0.4.

Step 2: Figure out P(A∩B). This is the [intersection](https://www.calculushowto.com/intersection-of-two-lines/) of A and B: both happening together. It’s given in the question 20 out of 100 buyers, or 0.2.

Step 3: Insert your answers into the formula:  
P(B|A) = P(A∩B) / P(A) = 0.2 / 0.4 = 0.5.

The probability that a buyer bought bucket seats, given that they purchased an alarm system, is 50%.

[](https://www.statisticshowto.com/wp-content/uploads/2012/10/venn-diagram-of-conditional-probability.png)

Venn diagram for 90 buyers, showing that 20 alarm buyers also purchased bucket seats.

**Conditional Probability vs. Joint Probability and Marginal Probability**

* **Conditional probability**: p(A|B) is the probability of event A occurring, **given that** event B occurs. For example, given that you drew a red card, what’s the probability that it’s a four (p(four|red))=2/26=1/13. So out of the 26 red cards (given a red card), there are two fours so 2/26=1/13.
* **Marginal probability**: the probability of an event occurring (p(A)) in isolation. It may be thought of as an unconditional probability. It is not conditioned on another event. Example: the probability that a card drawn is red (p(red) = 0.5). Another example: the probability that a card drawn is a 4 (p(four)=1/13).
* **Joint probability**: p(A ∩B). [Joint probability](https://www.investopedia.com/terms/j/jointprobability.asp) is that of event A **and** event B occurring. It is the probability of the intersection of two or more events. The probability of the intersection of A and B may be written p(A ∩ B). Example: the probability that a card is a four and red =p(four and red) = 2/52=1/26. (There are two red fours in a deck of 52, the 4 of hearts and the 4 of diamonds).

## Bayes' Theorem and Conditional Probability

[Bayes' theorem](https://www.investopedia.com/terms/b/bayes-theorem.asp), named after 18th-century British mathematician Thomas Bayes, is a mathematical formula for determining conditional probability. The theorem provides a way to revise existing predictions or theories (update probabilities) given new or additional evidence. In finance, Bayes' theorem can be used to rate the [risk](https://www.investopedia.com/terms/r/risk.asp) of lending money to potential borrowers.

Bayes' theorem is also called Bayes' Rule or Bayes' Law and is the foundation of the field of Bayesian statistics. This set of rules of probability allows one to update their predictions of events occurring based on new information that has been received, making for better and more dynamic estimates.

**Code:**

**def** conditional**():**

pas\_stats **=** 0.15

pass\_codingWstats **=** 0.60

pass\_codingWOstats **=** 0.40

prob\_both **=** pas\_stats **\*** pass\_codingWstats

**print(**"the probability that applicant passes both is:"**,round(**prob\_both**,**3**))**

prob\_coding **=** **(**prob\_both**)+((**1**-**pas\_stats**)\***pass\_codingWOstats**)**

**print(**"The probability that he/she passes only coding is:"**,round(**prob\_coding**,** 3**))**

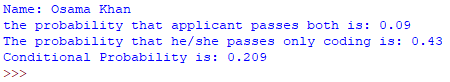
stats\_given\_coding **=** prob\_both**/**prob\_coding

**print(**"Conditional Probability is:"**,round(**stats\_given\_coding**,**3**))**

**print(**"Name: Osama Khan"**)**

conditional**()**

**Output:**



**What is Joint Probability?**

Joint probability is a statistical measure that calculates the likelihood of two events occurring together and at the same point in time. Joint probability is the probability of event Y occurring at the same time that event X occurs.

**The Formula for Joint Probability Is**

Notation for joint probability can take a few different forms. The following formula represents the probability of events intersection:

​P (X⋂Y)

where:X,Y=Two different events that intersect

P(X and Y),P(XY)=The joint probability of X and Y​

**What Does Joint Probability Tell You?**

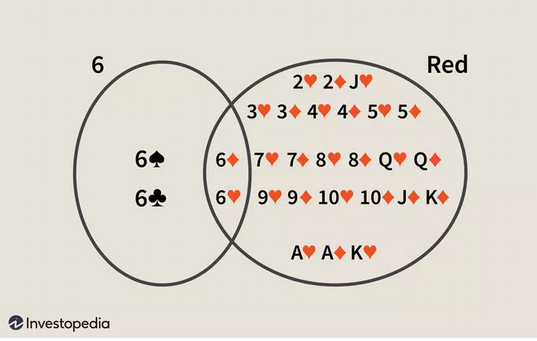
Probability is a field closely related to [statistics](https://www.investopedia.com/terms/s/statistics.asp) that deals with the likelihood of an event or phenomena occurring. It is quantified as a number between 0 and 1 inclusive, where 0 indicates an impossible chance of occurrence and 1 denotes the certain outcome of an event.

For example, the probability of drawing a red card from a deck of cards is 1/2 = 0.5. This means that there is an equal chance of drawing a red and drawing a black; since there are 52 cards in a deck, of which 26 are red and 26 are black, there is a 50-50 probability of drawing a red card versus a black card.

Joint probability is a measure of two events happening at the same time, and can only be applied to situations where more than one observation can occur at the same time. For example, from a deck of 52 cards, the joint probability of picking up a card that is both red and 6 is P(6 ∩ red) = 2/52 = 1/26, since a deck of cards has two red sixes—the six of hearts and the six of diamonds. Because the events "6" and "red" are independent in this example, you can also use the following formula to calculate the joint probability:

P(6∩red)=P(6)×P(red)=4/52×26/52=1/26

The symbol “∩” in a joint probability is referred to as an intersection. The probability of event X and event Y happening is the same thing as the point where X and Y intersect. Therefore, joint probability is also called the intersection of two or more events. A [Venn diagram](https://www.investopedia.com/terms/v/venn-diagram.asp) is perhaps the best visual tool to explain an intersection:



From the Venn above, the point where both circles overlap is the intersection, which has two observations: the six of hearts and the six of diamonds.

**Practical No 5**

**Aim: Design a Fuzzy based application using Python.**

**Library:**

*fuzzywuzzy:*

• FuzzyWuzzy is a library of Python which is used for string matching.

• Fuzzy string matching is the process of finding strings that match a given pattern.

• Basically, it uses Levenshtein Distance to calculate the differences between sequences

*Lenvenshtein:*

The Levenshtein Distance ("Edit Distance" on LeetCode)

• Given two strings str1 and str2 and below operations that can performed on str1. Find minimum number of

edits (operations) required to convert ‘str1’ into ‘str2’.

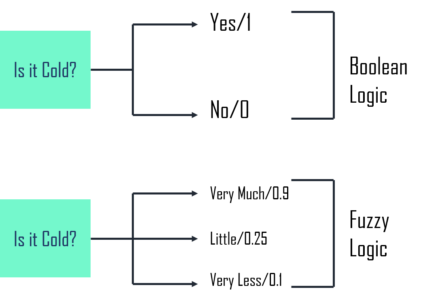
• Insert

• Remove

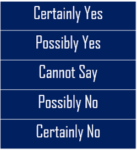
• Replace

## ****What is Fuzzy Logic?****

## **Fuzzy Logic** (FL) is a method of reasoning that resembles **human reasoning**. This approach is similar to how humans perform decision making. And it involves all intermediate possibilities between **YES** and **NO**.



The **conventional logic block** that a computer understands takes precise input and produces a definite output as TRUE or FALSE, which is equivalent to a human being’s YES or NO. The Fuzzy logic was invented by **Lotfi Zadeh** who observed that unlike computers, humans have a different range of possibilities between YES and NO, such as:



The Fuzzy logic works on the levels of possibilities of input to achieve a definite output. Now, talking about the implementation of this logic:

* It can be implemented in systems with different sizes and capabilities such as **micro-controllers, large networked** or **workstation-based systems.**
* Also, it can be implemented in **hardware, software** or a combination of **both**.

**Code:**

**from** fuzzywuzzy **import** fuzz

**from** fuzzywuzzy **import** process

#Ratio

**print(**"Ratio"**)**

Str\_A **=** 'FuzzyWuzzy is a lifesaver!'

Str\_B **=** 'fuzzy wuzzy is = LIFE SAVER.'

ratio **=** fuzz**.**ratio **(**Str\_A**.**lower **(),** Str\_B**.**lower**())**

**print(**'Similarity score: {}'**.format** **(**ratio**))**

**print(**"================================================================="**)**

#Partial Ratio

**print(**"Partial Ratio"**)**

Str\_A **=** 'Chicago, Illinois'

Str\_B **=** 'Chicago'

ratio **=** fuzz**.**partial\_ratio **(**Str\_A**.**lower**(),** Str\_B**.**lower**())**

**print(**'partial\_ratio: {}'**.format** **(**ratio**))**

**print(**"================================================================="**)**

#Token Sort Ratio

**print(**"Token Sort Ratio"**)**

Str\_A **=** 'Gunner William Kline'

Str\_B **=** 'Kline,Gunner William'

ratio **=** fuzz**.**token\_sort\_ratio **(**Str\_A**,** Str\_B**)**

**print** **(**'token\_sort\_ratio: {}' **.format** **(**ratio**))**

**print(**"================================================================="**)**#Token Set Ratio

**print(**"Token Set Ratio"**)**

Str\_A **=** 'The 3000 meter steeplechase winner, Soufiane and El Bakkali'

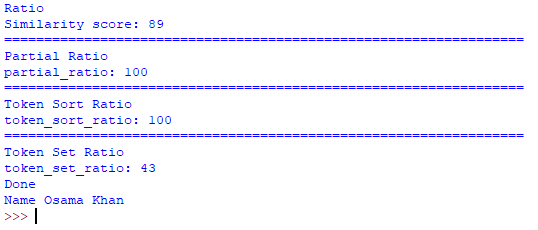
Str\_B **=** 'Soufian E1 Bakkli'

ratio **=** fuzz**.**token\_set\_ratio**(**Str\_A**,** Str\_B**)**

**print(**'token\_set\_ratio: {}'**.format(**ratio**))**

**print(**"Done"**)**

**print(**"Name Osama Khan"**)**

**Output:**

**Practical No 6**

**Aim: Supervised Learning model: Linear Regression (multiple linear regression).**

**Library :**

*matplotlib:*

Matplotlib is a comprehensive library for creating static, animated, and interactive visualizations in Python. Matplotlib makes easy things easy and hard things possible

*pandas:*

pandas is a [Python](https://www.python.org) package providing fast, flexible, and expressive data structures designed to make working with “relational” or “labeled” data both easy and intuitive. It aims to be the fundamental high-level building block for doing practical, **real-world** data analysis in Python. Additionally, it has the broader goal of becoming **the most powerful and flexible open-source data**

*numpy:*

NumPy is the fundamental package for scientific computing in Python. It is a Python library that provides a multidimensional array object, various derived objects (such as masked arrays and matrices), and an assortment of routines for fast operations on arrays, including mathematical, logical, shape manipulation, sorting, selecting, I/O, discrete Fourier transforms, basic linear algebra, basic statistical operations, random simulation and much more.

**What is supervised learning?**

Supervised learning, also known as supervised machine learning, is a subcategory of [machine learning](https://www.ibm.com/cloud/learn/machine-learning) and [artificial intelligence](https://www.ibm.com/cloud/learn/what-is-artificial-intelligence). It is defined by its use of labeled datasets to train algorithms that to classify data or predict outcomes accurately. As input data is fed into the model, it adjusts its weights until the model has been fitted appropriately, which occurs as part of the cross validation process. Supervised learning helps organizations solve for a variety of real-world problems at scale, such as classifying spam in a separate folder from your inbox.

**How supervised learning works**

Supervised learning uses a training set to teach models to yield the desired output. This training dataset includes inputs and correct outputs, which allow the model to learn over time. The algorithm measures its accuracy through the loss function, adjusting until the error has been sufficiently minimized.

Supervised learning can be separated into two types of problems when data mining—classification and regression:

* **Classification** uses an algorithm to accurately assign test data into specific categories. It recognizes specific entities within the dataset and attempts to draw some conclusions on how those entities should be labeled or defined. Common classification algorithms are linear classifiers, support vector machines (SVM), decision trees, k-nearest neighbor, and random forest, which are described in more detail below.
* **Regression** is used to understand the relationship between dependent and independent variables. It is commonly used to make projections, such as for sales revenue for a given business. [Linear regression](https://www.ibm.com/analytics/learn/linear-regression), [logistical regression](https://www.ibm.com/analytics/learn/logistic-regression), and polynomial regression are popular regression algorithms.

**Supervised learning algorithms**

**Various algorithms and computation techniques are used in supervised machine learning processes. Below are brief explanations of some of the most commonly used learning methods, typically calculated through use of programs like R or Python:**

**Neural networks**

Primarily leveraged for deep learning algorithms, [neural networks](https://www.ibm.com/cloud/learn/neural-networks) process training data by mimicking the interconnectivity of the human brain through layers of nodes. Each node is made up of inputs, weights, a bias (or threshold), and an output. If that output value exceeds a given threshold, it “fires” or activates the node, passing data to the next layer in the network. Neural networks learn this mapping function through supervised learning, adjusting based on the loss function through the process of gradient descent. When the cost function is at or near zero, we can be confident in the model’s accuracy to yield the correct answer.

**Naive Bayes**

Naive Bayes is classification approach that adopts the principle of class conditional independence from the Bayes Theorem. This means that the presence of one feature does not impact the presence of another in the probability of a given outcome, and each predictor has an equal effect on that result. There are three types of Naïve Bayes classifiers: Multinomial Naïve Bayes, Bernoulli Naïve Bayes, and Gaussian Naïve Bayes. This technique is primarily used in text classification, spam identification, and recommendation systems.

**Linear regression**

Linear regression is used to identify the relationship between a dependent variable and one or more independent variables and is typically leveraged to make predictions about future outcomes. When there is only one independent variable and one dependent variable, it is known as simple linear regression. As the number of independent variables increases, it is referred to as multiple linear regression. For each type of linear regression, it seeks to plot a line of best fit, which is calculated through the method of least squares. However, unlike other regression models, this line is straight when plotted on a graph.

**Logistic regression**

While linear regression is leveraged when dependent variables are continuous, logistical regression is selected when the dependent variable is categorical, meaning they have binary outputs, such as "true" and "false" or "yes" and "no." While both regression models seek to understand relationships between data inputs, logistic regression is mainly used to solve binary classification problems, such as spam identification.

**Support vector machine (SVM)**

A support vector machine is a popular supervised learning model developed by Vladimir Vapnik, used for both data classification and regression. That said, it is typically leveraged for classification problems, constructing a hyperplane where the distance between two classes of data points is at its maximum. This hyperplane is known as the decision boundary, separating the classes of data points (e.g., oranges vs. apples) on either side of the plane.

**K-nearest neighbor**

K-nearest neighbor, also known as the KNN algorithm, is a non-parametric algorithm that classifies data points based on their proximity and association to other available data. This algorithm assumes that similar data points can be found near each other. As a result, it seeks to calculate the distance between data points, usually through Euclidean distance, and then it assigns a category based on the most frequent category or average.

Its ease of use and low calculation time make it a preferred algorithm by data scientists, but as the test dataset grows, the processing time lengthens, making it less appealing for classification tasks. KNN is typically used for recommendation engines and image recognition.

**Random forest**

Random forest is another flexible supervised machine learning algorithm used for both classification and regression purposes. The "forest" references a collection of uncorrelated decision trees, which are then merged together to reduce variance and create more accurate data predictions.

**Code:**

**print(**"Osama Khan"**)**

**import** matplotlib**.** pyplot **as** plt

**import** pandas **as** pd

**import** numpy **as** np

**from** sklearn **import** datasets**,** linear\_model**,**metrics

**from** sklearn**.**model\_selection **import** train\_test\_split

bostondata **=** datasets**.**load\_boston**(**return\_X\_y**=False)**

a **=** bostondata**.**data

b **=** bostondata**.**target

a\_train**,**a\_test**,**b\_train**,**b\_test **=** train\_test\_split**(**a**,**b**,**

test\_size**=**0.4**,** random\_state**=**42**)**

regression **=** linear\_model**.**LinearRegression**()**

regression**.**fit**(**a\_train**,**b\_train**)**

**print(**"Ceffient of Model:"**,** regression**.**coef\_**)**

**print(**"Variance of the Model:"**,**regression**.**score**(**a\_test**,**b\_test**))**

plt**.**style**.**use**(**'fivethirtyeight'**)**

plt**.**scatter**(**regression**.**predict**(**a\_train**),**regression**.**predict**(**a\_train**)-**b\_train**,** color**=**'green'**,**s**=**10**,**label**=**'Train\_data'**)**

plt**.**scatter**(**regression**.**predict**(**a\_test**),**regression**.**predict**(**a\_test**)-**b\_test**,** color**=**'blue'**,**s**=**10**,**label**=**'Test\_data'**)**

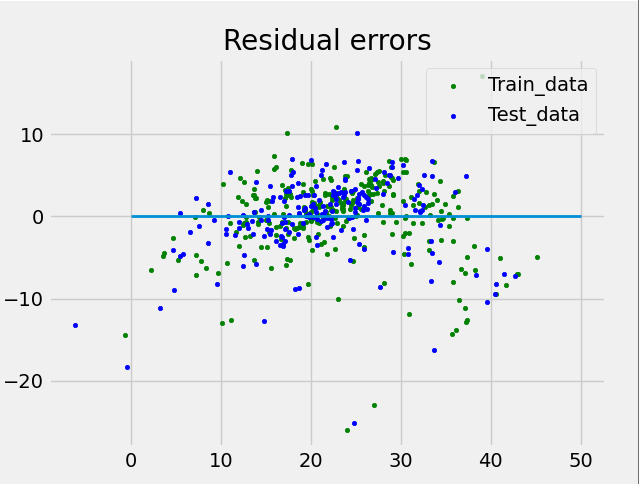
plt**.**hlines**(**y**=**0**,**xmin**=**0**,**xmax**=**50**,**linewidth**=**2**)**

plt**.**legend**(**loc**=**'upper right'**)**

plt**.**title**(**'Residual errors'**)**

plt**.**show**()**

**Output:**

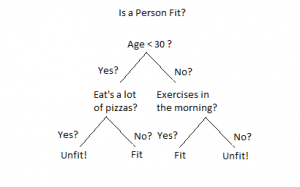


**Practical No 7**

**Aim:** **Supervised learning model: decision tree.**

**Description:**

Decision Trees are a type of Supervised Machine Learning (that is you explain what the input is and what the corresponding output is in the training data) where the data is continuously split according to a certain parameter. The tree can be explained by two entities, namely decision nodes and leaves. The leaves are the decisions or the final outcomes. And the decision nodes are where the data is split.

[](https://www.xoriant.com/sites/default/files/uploads/2017/08/Decision-Trees-modified-1.png)

An example of a decision tree can be explained using above binary tree. Let’s say you want to predict whether a person is fit given their information like age, eating habit, and physical activity, etc. The decision nodes here are questions like ‘What’s the age?’, ‘Does he exercise?’, ‘Does he eat a lot of pizzas’? And the leaves, which are outcomes like either ‘fit’, or ‘unfit’. In this case this was a binary classification problem (a yes no type problem). There are two main types of Decision Trees:

1. **Classification trees** (Yes/No types)

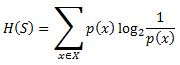
What we’ve seen above is an example of classification tree, where the outcome was a variable like ‘fit’ or ‘unfit’. Here the decision variable is **Categorical**.

1. **Regression trees** (Continuous data types)

Here the decision or the outcome variable is **Continuous**, e.g. a number like 123. **Working** Now that we know what a Decision Tree is, we’ll see how it works internally. There are many algorithms out there which construct Decision Trees, but one of the best is called as **ID3 Algorithm**. ID3 Stands for **Iterative Dichotomiser 3**. Before discussing the ID3 algorithm, we’ll go through few definitions.

* **Entropy:**

Entropy, also called as Shannon Entropy is denoted by H(S) for a finite set S, is the measure of the amount of uncertainty or randomness in data.

[](https://www.xoriant.com/sites/default/files/uploads/2017/08/Decision-Trees-modified-2.jpg)

Intuitively, it tells us about the predictability of a certain event. Example, consider a coin toss whose probability of heads is 0.5 and probability of tails is 0.5. Here the entropy is the highest possible, since there’s no way of determining what the outcome might be. Alternatively, consider a coin which has heads on both the sides, the entropy of such an event can be predicted perfectly since we know beforehand that it’ll always be heads. In other words, this event has no randomness hence it’s entropy is zero. In particular, lower values imply less uncertainty while higher values imply high uncertainty.

* **Information Gain:**

nformation gain is also called as Kullback-Leibler divergence denoted by IG(S,A) for a set S is the effective change in entropy after deciding on a particular attribute A. It measures the relative change in entropy with respect to the independent variables.

[Decision Trees modified](https://www.xoriant.com/sites/default/files/uploads/2017/08/Decision-Trees-modified-3.jpg)

Alternatively,

[Decision Trees modified](https://www.xoriant.com/sites/default/files/uploads/2017/08/Decision-Trees-modified-4.jpg)

where IG(S, A) is the information gain by applying feature A. H(S) is the Entropy of the entire set, while the second term calculates the Entropy after applying the feature A, where P(x) is the probability of event x.

**Code:**

**print(**"Name: Osama Khan"**)**

**import** pandas **as** pd

**from** sklearn**.**tree **import** DecisionTreeClassifier

**from** sklearn**.**model\_selection **import** train\_test\_split

**from** sklearn **import** metrics

#load dataset

pima **=** pd**.**read\_csv**(**"diabetes.csv"**)**

col\_names **=** pima**.**columns

pima**.**head**()**

#split dataset in features and target variable

feature\_cols **=** **[**'insulin'**,** 'bmi'**,** 'age'**,**'glucose'**,**'bp'**,**'pedigree'**]**

X **=** pima**[**feature\_cols**]** # Features

y **=** pima**.**label # Target variable

X\_train**,** X\_test**,** y\_train**,** y\_test **=** train\_test\_split**(**X**,** y**,** test\_size**=**0.3**,** random\_state**=**1**)** # 70% training and 30% test

clf **=** DecisionTreeClassifier**()**

# Train Decision Tree Classifer

clf **=** clf**.**fit**(**X\_train**,**y\_train**)**

#Predict the response for test dataset

y\_pred **=** clf**.**predict**(**X\_test**)**

**print(**"Accuracy:"**,**metrics**.**accuracy\_score**(**y\_test**,** y\_pred**))**

lf **=** DecisionTreeClassifier**(**criterion**=**"entropy"**,** max\_depth**=**3 **)**

# Train Decision Tree Classifer

clf **=** clf**.**fit**(**X\_train**,**y\_train**)**

#Predict the response for test dataset

y\_pred **=** clf**.**predict**(**X\_test**)**

# Model Accuracy, how often is the classifier correct?

**print(**"Accuracy:"**,**metrics**.**accuracy\_score**(**y\_test**,** y\_pred**))**

**from** six **import** StringIO

**from** IPython**.**display **import** Image

**from** sklearn**.**tree **import** export\_graphviz

**import** pydotplus

**import** graphviz

dot\_data **=** StringIO**()**

export\_graphviz**(**clf**,** out\_file**=**dot\_data**,**

filled**=True,** rounded**=True,**

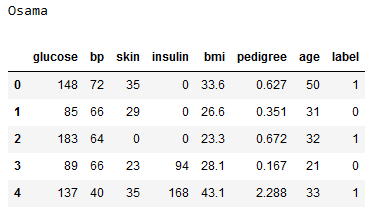
special\_characters**=True,** feature\_names **=** feature\_cols**,**class\_names**=[**'0'**,**'1'**])**

graph **=** pydotplus**.**graph\_from\_dot\_data**(**dot\_data**.**getvalue**())**

graph**.**write\_png**(**'diabetes2.png'**)**

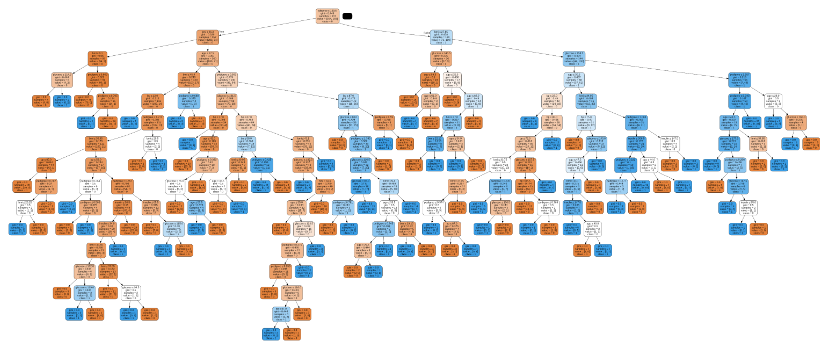
Image**(**graph**.**create\_png**())**

**Output:**







****

**Practical No 8**

**Aim: Unsupervised learning model: (Principal Component Analysis).**

**Library:**

*seaborn:*

Seaborn is an open-source Python library built on top of [matplotlib](https://www.section.io/engineering-education/matplotlib-visualization-python/). It is used for data visualization and exploratory data analysis. Seaborn works easily with data frames and the Pandas library. The graphs created can also be customized easily. Below are a few benefits of Data Visualization.

Graphs can help us find data trends that are useful in any machine learning or forecasting project.

* Graphs make it easier to explain your data to non-technical people.
* Visually attractive graphs can make presentations and reports much more appealing to the reader.

**What is unsupervised learning?**

Unsupervised learning, also known as [unsupervised machine learning](https://www.ibm.com/cloud/learn/machine-learning), uses machine learning algorithms to analyze and cluster unlabeled datasets. These algorithms discover hidden patterns or data groupings without the need for human intervention. Its ability to discover similarities and differences in information make it the ideal solution for exploratory data analysis, cross-selling strategies, customer segmentation, and image recognition.

**Common unsupervised learning approaches**

Unsupervised learning models are utilized for three main tasks—clustering, association, and dimensionality reduction. Below we’ll define each learning method and highlight common algorithms and approaches to conduct them effectively.

**Clustering**

Clustering isa data mining technique which groups unlabeled data based on their similarities or differences. Clustering algorithms are used to process raw, unclassified data objects into groups represented by structures or patterns in the information. Clustering algorithms can be categorized into a few types, specifically exclusive, overlapping, hierarchical, and probabilistic.

***Exclusive and Overlapping Clustering***

Exclusive clustering is a form of grouping that stipulates a data point can exist only in one cluster. This can also be referred to as “hard” clustering. The K-means clustering algorithm is an example of exclusive clustering.

* **K-means clustering** is a common example of an exclusive clustering method where data points are assigned into K groups, where K represents the number of clusters based on the distance from each group’s centroid. The data points closest to a given centroid will be clustered under the same category. A larger K value will be indicative of smaller groupings with more granularity whereas a smaller K value will have larger groupings and less granularity. K-means clustering is commonly used in market segmentation, document clustering, image segmentation, and image compression.

Overlapping clusters differs from exclusive clustering in that it allows data points to belong to multiple clusters with separatedegrees of membership. “Soft” or fuzzy k-means clustering is an example of overlapping clustering.

***Hierarchical clustering***

Hierarchical clustering, also known as hierarchical cluster analysis (HCA), is an unsupervised clustering algorithm that can be categorized in two ways; they can be agglomerative or divisive. Agglomerative clustering is considered a “bottoms-up approach.” Its data points are isolated as separate groupings initially, and then they are merged together iteratively on the basis of similarity until one cluster has been achieved. Four different methods are commonly used to measure similarity:

1. **Ward’s linkage:** This method states that the distance between two clusters is defined by the increase in the sum of squared after the clusters are merged.
2. **Average linkage:** This method is defined by the mean distance between two points in each cluster
3. **Complete (or maximum) linkage:** This method is defined by the maximum distance between two points in each cluster
4. **Single (or minimum) linkage:** This method is defined by the minimum distance between two points in each cluster

Euclidean distance is the most common metric used to calculate these distances; however, other metrics, such as Manhattan distance, are also cited in clustering literature.

Divisive clustering can be defined as the opposite of agglomerative clustering; instead it takes a “top-down” approach. In this case, a single data cluster is divided based on the differences between data points. Divisive clustering is not commonly used, but it is still worth noting in the context of hierarchical clustering. These clustering processes are usually visualized using a dendrogram, a tree-like diagram that documents the merging or splitting of data points at each iteration.

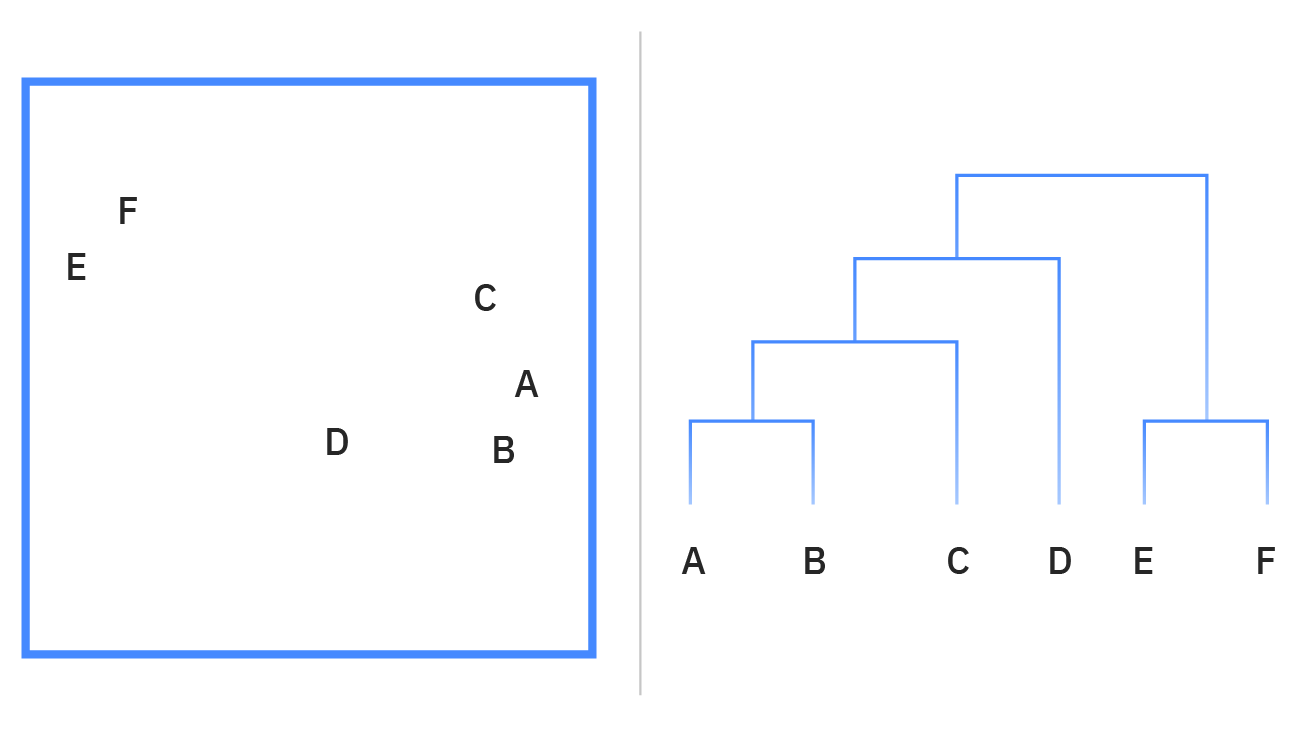
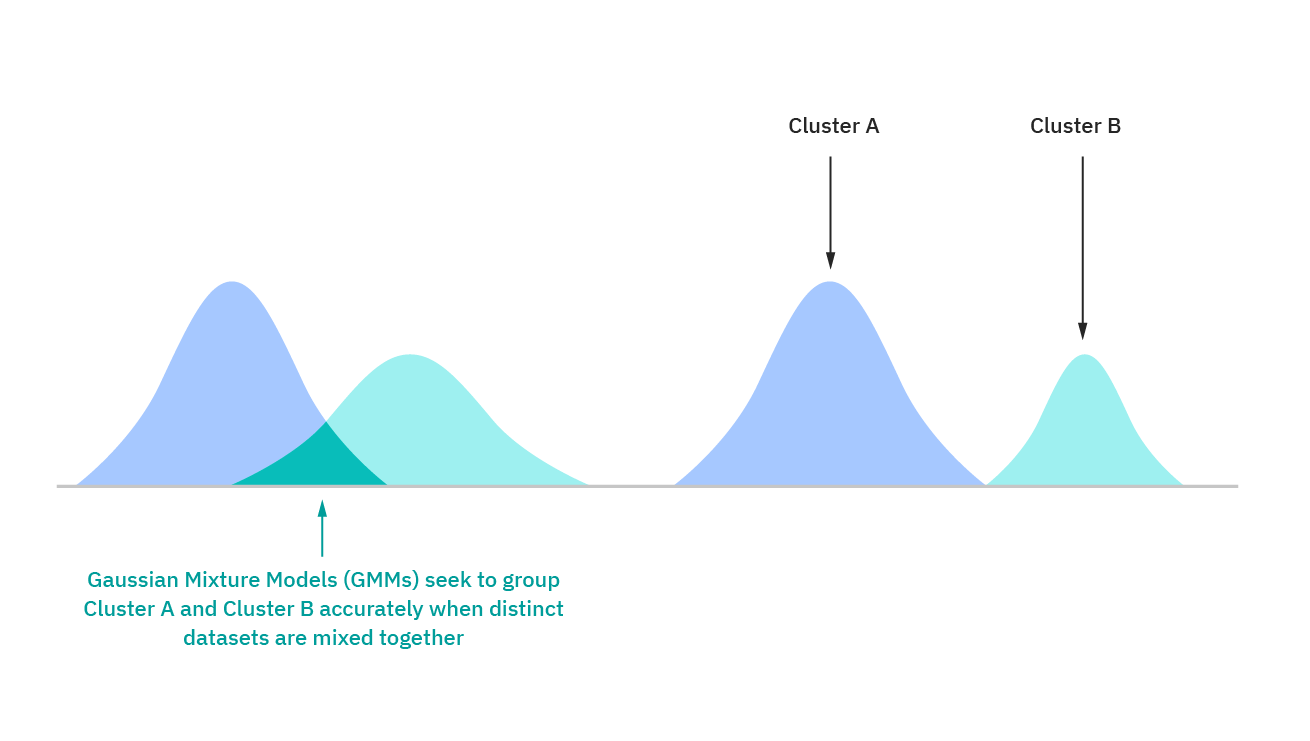


Diagram of a Dendrogram; reading the chart "bottom-up" demonstrates agglomerative clustering while "top-down" is indicative of divisive clustering

***Probabilistic clustering***

A probabilistic model is an unsupervised technique that helps us solve density estimation or “soft” clustering problems. In probabilistic clustering, data points are clustered based on the likelihood that they belong to a particular distribution. The Gaussian Mixture Model (GMM) is the one of the most commonly used probabilistic clustering methods.

* **Gaussian Mixture Models** are classified as mixture models, which means that they are made up of an unspecified number of probability distribution functions. GMMs are primarily leveraged to determine which Gaussian, or normal, probability distribution a given data point belongs to. If the mean or variance are known, then we can determine which distribution a given data point belongs to. However, in GMMs, these variables are not known, so we assume that a latent, or hidden, variable exists to cluster data points appropriately. While it is not required to use the Expectation-Maximization (EM) algorithm, it is a commonly used to estimate the assignment probabilities for a given data point to a particular data cluster.



**Association Rules**

An association rule is a rule-based method for finding relationships between variables in a given dataset. These methods are frequently used for market basket analysis, allowing companies to better understand relationships between different products. Understanding consumption habits of customers enables businesses to develop better cross-selling strategies and recommendation engines. Examples of this can be seen in Amazon’s “Customers Who Bought This Item Also Bought” or Spotify’s "Discover Weekly" playlist. While there are a few different algorithms used to generate association rules, such as Apriori, Eclat, and FP-Growth, the Apriori algorithm is most widely used.

***Apriori algorithms***

Apriori algorithms have been popularized through market basket analyses, leading to different recommendation engines for music platforms and online retailers. They are used within transactional datasets to identify frequent itemsets, or collections of items, to identify the likelihood of consuming a product given the consumption of another product. For example, if I play Black Sabbath’s radio on Spotify, starting with their song “Orchid”, one of the other songs on this channel will likely be a Led Zeppelin song, such as “Over the Hills and Far Away.” This is based on my prior listening habits as well as the ones of others. Apriori algorithms use a hash tree to count itemsets, navigating through the dataset in a breadth-first manner.

**Dimensionality reduction**

While more data generally yields more accurate results, it can also impact the performance of machine learning algorithms (e.g. overfitting) and it can also make it difficult to visualize datasets. Dimensionality reduction is a technique used when the number of features, or dimensions, in a given dataset is too high. It reduces the number of data inputs to a manageable size while also preserving the integrity of the dataset as much as possible. It is commonly used in the preprocessing data stage, and there are a few different dimensionality reduction methods that can be used, such as:

***Principal component analysis***

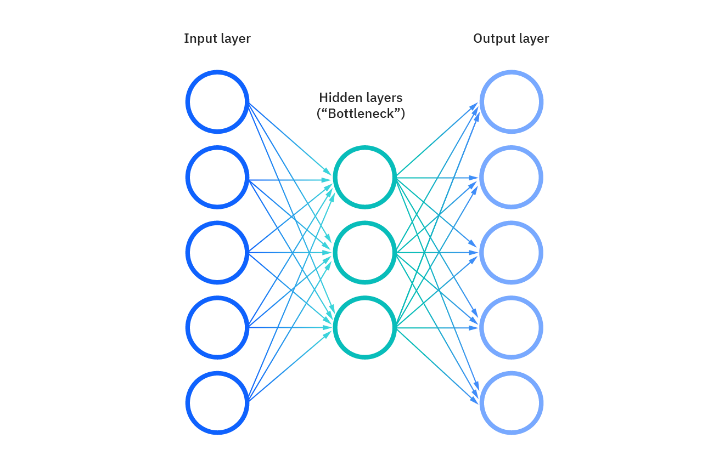
Principal component analysis (PCA) is a type of dimensionality reduction algorithm which is used to reduce redundancies and to compress datasets through feature extraction. This method uses a linear transformation to create a new data representation, yielding a set of "principal components." The first principal component is the direction which maximizes the variance of the dataset. While the second principal component also finds the maximum variance in the data, it is completely uncorrelated to the first principal component, yielding a direction that is perpendicular, or orthogonal, to the first component. This process repeats based on the number of dimensions, where a next principal component is the direction orthogonal to the prior components with the most variance.

***Singular value decomposition***

Singular value decomposition (SVD) is another dimensionality reduction approach which factorizes a matrix, A, into three, low-rank matrices. SVD is denoted by the formula, A = USVT, where U and V are orthogonal matrices. S is a diagonal matrix, and S values are considered singular values of matrix A. Similar to PCA, it is commonly used to reduce noise and compress data, such as image files.

***Autoencoders***

Autoencoders leverage [neural networks](https://www.ibm.com/cloud/learn/neural-networks) to compress data and then recreate a new representation of the original data’s input. Looking at the image below, you can see that the hidden layer specifically acts as a bottleneck to compress the input layer prior to reconstructing within the output layer. The stage from the input layer to the hidden layer is referred to as “encoding” while the stage from the hidden layer to the output layer is known as “decoding.”



**Code:**

**from** sklearn**.**datasets **import** load\_breast\_cancer

**from** sklearn**.**preprocessing **import** StandardScaler

**from** sklearn**.**decomposition **import** PCA

**import** matplotlib**.**pyplot **as** plt

**import** seaborn **as** sns

cancer **=** load\_breast\_cancer**()**

cancer**.**target

sc **=** StandardScaler**()**

x\_sc **=** sc**.**fit\_transform**(**cancer**.**data**)**

pca **=** PCA**(**n\_components**=**2**)**

pca**.**fit**(**x\_sc**)**

x\_pca **=** pca**.**transform**(**x\_sc**)**

**print(**"Original shape: {}"**.format(str(**x\_sc**.**shape**)))**

**print(**"Reduced shape: {}"**.format(str(**x\_pca**.**shape**)))**

ax **=** plt**.**figure**(**figsize**=(**12**,**8**))**

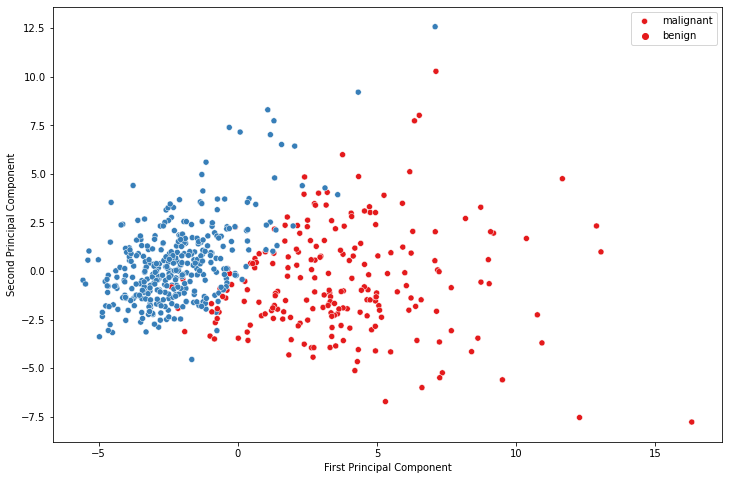
sns**.**scatterplot**(**x\_pca**[:,**0**],** x\_pca**[:,**1**],**hue**=**cancer**.**target**,** palette **=**'Set1' **)**

plt**.**legend**(**cancer**.**target\_names**,** loc**=**'best'**)**

plt**.**xlabel**(**'First Principal Component'**)**

plt**.**ylabel**(**'Second Principal Component'**)**

**Output:**

****

**Practical No 9**

**Aim: Write an application to implement k-means clustering algorithm.**

## What is Meant by the K-Means Clustering Algorithm?

K-Means clustering is an unsupervised learning algorithm. There is no labeled data for this clustering, unlike in supervised learning. K-Means performs the division of objects into clusters that share similarities and are dissimilar to the objects belonging to another cluster.

The term ‘K’ is a number. You need to tell the system how many clusters you need to create. For example, K = 2 refers to two clusters. There is a way of finding out what is the best or optimum value of K for a given data.

**Code:**

**print(**"Osama Khan"**)**

**import** numpy **as** np

**import** pandas **as** pd

**from** matplotlib **import** pyplot **as** plt

**from** sklearn**.**datasets **import** make\_blobs

**from** sklearn**.**cluster **import** KMeans

a**,**b **=** make\_blobs**(**n\_samples**=**300**,** centers**=**4**,**cluster\_std**=**0.60**,**random\_state**=**0**)**

plt**.**scatter**(**a**[:,**0**],**a**[:,**1**])**

plt**.**title**(**'osama'**)**

plt**.**show**()**

wcss **=** **[]**

**for** i **in** **range(**1**,** 11**):**

kmean **=** KMeans**(**n\_clusters**=**i**,** init**=**'k-means++'**,**max\_iter**=**300**,** n\_init**=**10**,**random\_state**=**0**)**

kmean**.**fit**(**a**)**

wcss**.**append**(**kmean**.**inertia\_**)**

plt**.**plot**(range(**1**,**11**),**wcss**)**

plt**.**title**(**'Elbow method'**)**

plt**.**xlabel**(**"Number of cluster"**)**

plt**.**ylabel**(**"wcss"**)**

plt**.**show**()**

kmean **=** KMeans**(**n\_clusters**=**4**,** init**=**'k-means++'**,**max\_iter**=**300**,** n\_init**=**10**,**random\_state**=**0**)**

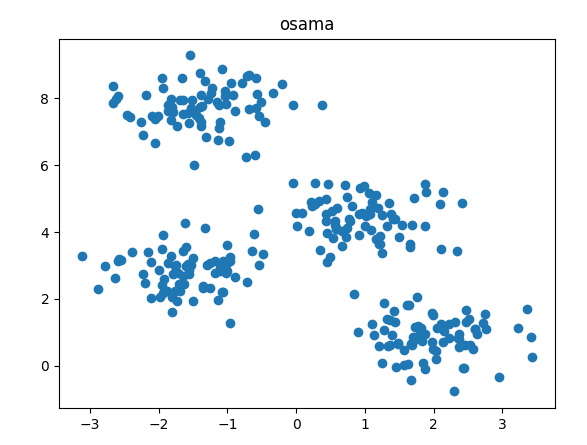
pred\_b **=** kmean**.**fit\_predict**(**a**)**

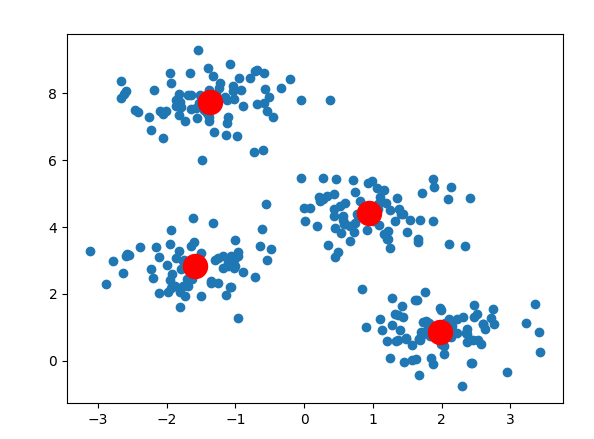
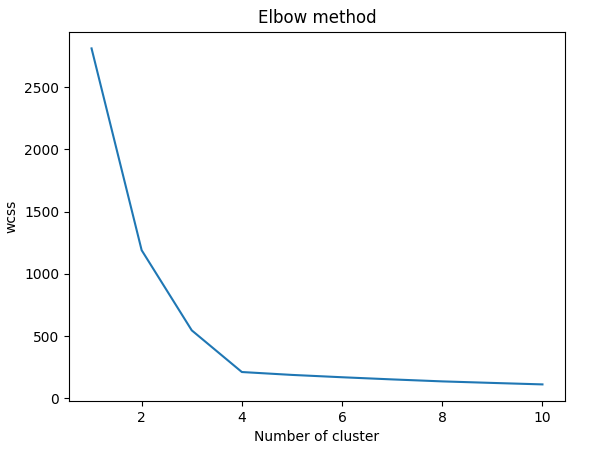
plt**.**scatter**(**a**[:,**0**],**a**[:,**1**])**

plt**.**scatter**(**kmean**.**cluster\_centers\_**[:,**0**],** kmean**.**cluster\_centers\_**[:,**1**],** s**=**300**,** c**=**'red'**)**

plt**.**show**()**

**Output:**





**OR**

**Code:**

**from** numpy **import** where

**from** sklearn**.**datasets **import** make\_classification

**from** matplotlib **import** pyplot

X**,**y **=** make\_classification**(**n\_samples**=**1000**,** n\_features**=**2**,**n\_informative**=**2**,**n\_redundant**=**0**,**n\_clusters\_per\_class**=**1**,** random\_state**=**42**)**

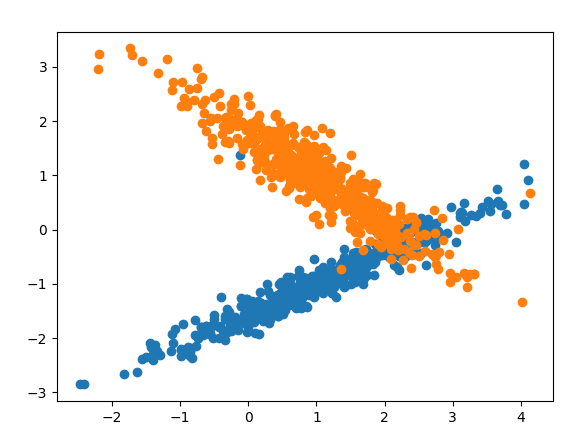
**for** class\_val **in** **range(**2**):**

row\_ix **=** where**(**y**==**class\_val**)**

pyplot**.**scatter**(**X**[**row\_ix**,**0**],**X**[**row\_ix**,**1**])**

pyplot**.**show**()**

**Output:**



**Practical No 10**

**Aim: Write an application to implement support vector machine algorithm.**

**Support Vector Machine Algorithm**

Support Vector Machine or SVM is one of the most popular Supervised Learning algorithms, which is used for Classification as well as Regression problems. However, primarily, it is used for Classification problems in Machine Learning.

The goal of the SVM algorithm is to create the best line or decision boundary that can segregate n-dimensional space into classes so that we can easily put the new data point in the correct category in the future. This best decision boundary is called a hyperplane.

SVM chooses the extreme points/vectors that help in creating the hyperplane. These extreme cases are called as support vectors, and hence algorithm is termed as Support Vector Machine. Consider the below diagram in which there are two different categories that are classified using a decision boundary or hyperplane:



**Example:** SVM can be understood with the example that we have used in the KNN classifier. Suppose we see a strange cat that also has some features of dogs, so if we want a model that can accurately identify whether it is a cat or dog, so such a model can be created by using the SVM algorithm. We will first train our model with lots of images of cats and dogs so that it can learn about different features of cats and dogs, and then we test it with this strange creature. So as support vector creates a decision boundary between these two data (cat and dog) and choose extreme cases (support vectors), it will see the extreme case of cat and dog. On the basis of the support vectors, it will classify it as a cat. Consider the below diagram:



SVM algorithm can be used for **Face detection, image classification, text categorization,** etc.

**Types of SVM**

**SVM can be of two types:**

* **Linear SVM:** Linear SVM is used for linearly separable data, which means if a dataset can be classified into two classes by using a single straight line, then such data is termed as linearly separable data, and classifier is used called as Linear SVM classifier.
* **Non-linear SVM:** Non-Linear SVM is used for non-linearly separated data, which means if a dataset cannot be classified by using a straight line, then such data is termed as non-linear data and classifier used is called as Non-linear SVM classifier.

## Hyperplane and Support Vectors in the SVM algorithm:

**Hyperplane:** There can be multiple lines/decision boundaries to segregate the classes in n-dimensional space, but we need to find out the best decision boundary that helps to classify the data points. This best boundary is known as the hyperplane of SVM.

The dimensions of the hyperplane depend on the features present in the dataset, which means if there are 2 features (as shown in image), then hyperplane will be a straight line. And if there are 3 features, then hyperplane will be a 2-dimension plane.

We always create a hyperplane that has a maximum margin, which means the maximum distance between the data points.

**Support Vectors:**

The data points or vectors that are the closest to the hyperplane and which affect the position of the hyperplane are termed as Support Vector. Since these vectors support the hyperplane, hence called a Support vector.

**Code:**

**print(**"Osama Khan "**)**

**import** numpy **as** np

**import** matplotlib**.**pyplot **as** plt

**import** pandas **as** pd

**from** sklearn**.**svm **import** SVC

**from** sklearn**.**model\_selection **import** train\_test\_split

**from** sklearn**.**metrics **import** classification\_report**,** confusion\_matrix

datasrc **=** "https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data"

colnames **=** **[**"sepal-length"**,** "sepal-width"**,** "petal-length"**,**"petal-width"**,** "Class"**]**

DS **=** pd**.**read\_csv**(**datasrc**,**names**=**colnames**)**

P **=** DS**.**drop**(**"Class"**,**axis**=**1**)**

Q **=** DS**[**"Class"**]**

p\_tr**,**p\_ts**,**q\_tr**,**q\_ts **=** train\_test\_split**(**P**,**Q**,**test\_size**=**0.2**)**

#polynomial

cls **=** SVC**(**kernel**=**"poly"**,** degree**=**8**)**

cls**.**fit**(**p\_tr**,**q\_tr**)**

q\_pred **=** cls**.**predict**(**p\_ts**)**

**print(**confusion\_matrix**(**q\_ts**,**q\_pred**))**

**print(**classification\_report**(**q\_ts**,**q\_pred**))**

#gaussion

cls **=** SVC**(**kernel**=**"rbf"**,** degree**=**8**)**

cls**.**fit**(**p\_tr**,**q\_tr**)**

q\_pred **=** cls**.**predict**(**p\_ts**)**

**print(**confusion\_matrix**(**q\_ts**,**q\_pred**))**

**print(**classification\_report**(**q\_ts**,**q\_pred**))**

#polynomial

cls **=** SVC**(**kernel**=**"poly"**,** degree**=**8**)**

cls**.**fit**(**p\_tr**,**q\_tr**)**

q\_pred **=** cls**.**predict**(**p\_ts**)**

**print(**confusion\_matrix**(**q\_ts**,**q\_pred**))**

**print(**classification\_report**(**q\_ts**,**q\_pred**))**

**Output:**

