## **Experiment 1**

**Aim – Introduction to Jupyter IDE and its libraries Pandas and NumPy.**

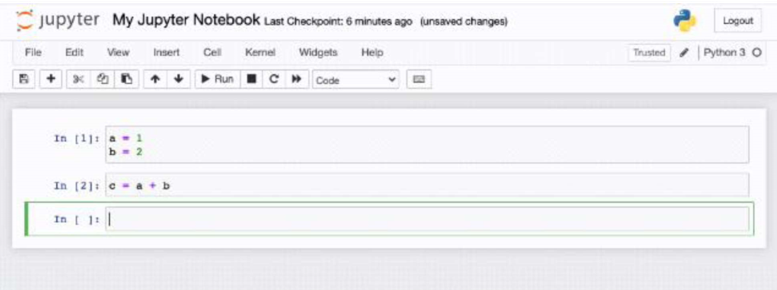
**Theory –**

Jupyter Notebook (sometimes called IPython Notebook) is a popular way to write and run Python code, especially for data analysis, data science and machine learning. Jupyter Notebooks are easy-to-use because they let you execute code and review the output quickly. This iterative process is central to data analytics and makes it easy to test hypotheses and record the results (just like a notebook).

For example. Let’s say you are visualizing a dataset about life expectancy by country. You only

want to show some countries, but you are not sure which ones to select. With a Jupyter Notebook, you can try multiple versions and easily compare. Even better, you have a written

record of what you’ve already tried that you can show a teammate (or your future self). This is just one example of the many benefits of working within a notebook-like environment.



Jupyter Notebook uses a back-end kernel called IPython. The ‘I’ stands for ‘Interactive’, which means that a program or script can be broken up into smaller pieces, and those pieces can be run independently from the rest of the program. You do not need to worry about the difference between Python and IPython. The important thing to know is that you can run small pieces of code, which can be helpful when working with data.

Integrated Development Environments (IDEs) - Jupyter Notebook is a type of Integrated Development Environment (IDE). IDEs are places to write code that offer some supportive features. Almost all IDEs provide syntax highlighting, debugging, and code completion. Jupyter Notebook also offers embedded help documentation and introspection (i.e., you can check each -line display of charts and images.

Pandas - Pandas is a very popular library for working with data (its goal is to be the most powerful and flexible open-source tool, and in our opinion, it has reached that go

DataFrames are at the center of pandas. A DataFrame is structured like a table or spreadsheet. The rows and the columns both have indexes, and you can perform operations on rows or columns separately. A pandas DataFrame can be easily changed and manipulated. Pandas has helpful functions for handling missing data, performing operations on columns and rows, and transforming data. If that wasn’t enough, a lot of SQL functions have counterparts in pandas, such as join, merge, filter by, and group by. With all of these powerful tools, it should come as no surprise that pandas is very popular among data scientists.

NumPy - NumPy is an open-source Python library that facilitates efficient numerical operations on large quantities of data. There are a few functions that exist in NumPy that we use on pandas DataFrames. For us, the most important part about NumPy is that pandas is built on top of it. So, NumPy is a dependency of Pandas.

**Installation**

pip install numpy

pip install pandas

import numpy as np

import pandas as pd

NumPy Arrays - NumPy arrays are unique in that they are more flexible than normal Python lists. They are called ndarrays since they can have any number (n) of dimensions (d). They hold a collection of items of any one data type and can be either a vector (one-dimensional) or a matrix (multi-dimensional). NumPy arrays allow for fast element access and efficient data manipulation.

The code below initializes a Python list named list1:

list1 = [1,2,3,4]

To convert this to a one-dimensional ndarray with one row and four columns, we can use the np.array() function:

**Input –**

array1 = np.array(list1) print(array1)

**Output –**

[1 2 3 4]

Numerical operations (min, max, mean, etc) - Mathematical operations can be performed on all values in a ndarray at one time rather than having to loop through values, as is necessary with a Python list. This is very helpful in many scenarios. Say you own a toy store and decide to decrease the price you can easily facilitate this operation.

Another important type of object in the pandas library is the DataFrame. This object is similar in form to a matrix as it consists of rows and columns. Both rows and columns can be indexed with integers or String names. One DataFrame can contain many different types of data types, but within a column, everything has to be the same data type. A column of a DataFrame is essentially a Series. All columns must have the same number of elements (rows).

There are different ways to fill a DataFrame such as with a CSV file, a SQL query, a Python list, or a dictionary. Here we have created a DataFrame using a Python list of lists. Each nested list represents the data in one row of the DataFrame. We use the keyword columns to pass in the list of our custom column names.

**Input –**

dataf = pd.DataFrame([

['John Smith','123 Main St',34],

['Jane Doe', '456 Maple Ave',28],

['Joe Schmo', '789 Broadway',51]

],

columns=['name','address','age'])

**Output –**

name | address | age

John Smith | 123 Main St | 34

Jane Doe | 456 Maple Ave | 28

Joe Schmo | 789 Broadway | 51

**Conclusion –**

The introduction to Jupyter IDE, along with libraries like Pandas and NumPy, highlighted their importance in data manipulation and analysis. Jupyter provides an interactive environment for coding, while Pandas simplifies data handling through DataFrames, and NumPy enhances numerical computations. Together, they form a robust foundation for data science and machine learning projects.

**Viva – Voce**

**Q1.** What are Pandas?

**Ans.** Pandas is an open-source Python library that is built on top of the NumPy library. It is made for working with relational or labelled data. It provides various data structures for manipulating, cleaning and analyzing numerical data. It can easily handle missing data as well. Pandas are fast and have high performance and productivity.

**Q2.** What are the Different Types of Data Structures in Pandas?

**Ans.** The two data structures that are supported by Pandas are Series and DataFrames. i. Pandas Series is a one-dimensional labelled array that can hold data of any type. It is mostly used to represent a single column or row of data. ii. Pandas DataFrame is a two-dimensional heterogeneous data structure. It stores data in a tabular form. Its three main components are data, rows, and columns.

**Q3.** List Key Features of Pandas.

**Ans.** Pandas are used for efficient data analysis. The key features of Pandas are as follows:

1. Fast and efficient data manipulation and analysis.
2. Provides time-series functionality.
3. Easy missing data handling.
4. Faster data merging and joining.
5. Flexible reshaping and pivoting of data sets.
6. Powerful group by functionality.
7. Data from different file objects can be loaded.
8. Integrates with NumPy.

**Q4.** What is NumPy, and why is it popular in the field of scientific computing?

**Ans.** NumPy is a powerful Python library for numerical and matrix operations. It provides support for large, multi-dimensional arrays and matrices, along with mathematical functions to operate on these arrays efficiently.

**Q5.** Which is faster - NumPy or Pandas?

**Ans.** Pandas is more user-friendly, but NumPy is faster. Pandas has a lot more options for handling missing data, but NumPy has better performance on large datasets. Pandas uses Python objects internally, making it easier to work with than NumPy (which uses C arrays).

**Experiment – 2**

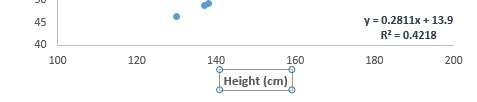
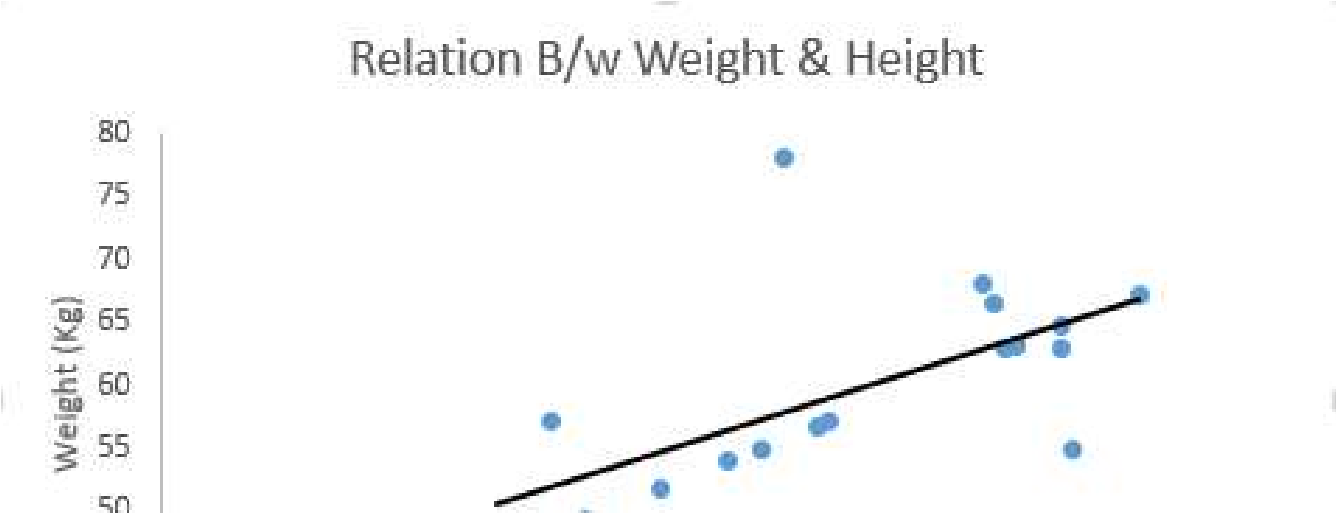
**Aim – Write program to demonstrate Simple Linear Regression.**

**Theory** –

Linear Regression is a machine learning algorithm based on supervised learning. It performs a regression task. Regression models a target prediction value based on independent variables. It is mostly used for finding out the relationship between variables and forecasting. Different regression models differ based on the kind of relationship between the dependent and independent variables, they are considering and the number of independent variables being used.

It is used to estimate real values (cost of houses, number of calls, total sales etc.) based on continuous variable(s). Here, we establish relationship between independent and dependent variables by fitting a best line. This best fit line is known as regression line and represented by a linear equation Y= a\*X + b.

Look at the below example. Here we have identified the best fit line having linear equation **y=0.2811x+13.9.** Now using this equation, we can find the weight, knowing the height of a person.



Linear Regression is of mainly two types: Simple Linear Regression and Multiple Linear Regression.

**Python Script**

# Importing all the required libraries

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

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

from sklearn.linear\_model import LinearRegression

data\_set= pd.read\_csv('Salary\_Data.csv')

df\_binary = data\_set[['Experience', 'Salary']]

# Taking only the selected two attributes from the dataset

df\_binary.columns = ['Salary', 'Experience']

# display the first 5 rows

print(df\_binary.head())

# Eliminating NaN or missing input numbers

df\_binary.fillna(method ='ffill', inplace = True)

# Converting each dataframe into a numpy array, since each dataframe contains only one column,

# Separating the data into independent and dependent variables

X = np.array(df\_binary['Salary']).reshape(-1, 1)

y = np.array(df\_binary['Experience']).reshape(-1, 1)

# Dropping any rows with Nan values

df\_binary.dropna(inplace=True)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.25)

# Splitting the data into training and testing data

regr = LinearRegression()

regr.fit(X\_train, y\_train)

print(regr.score(X\_test, y\_test))

y\_pred = regr.predict(X\_test)

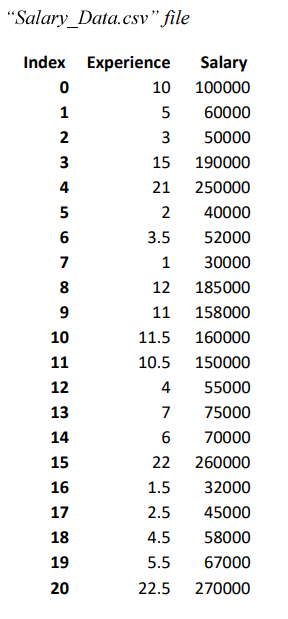
plt.scatter(X\_test, y\_test, color='b’)

plt.plot(X\_test, y\_pred, color='k')

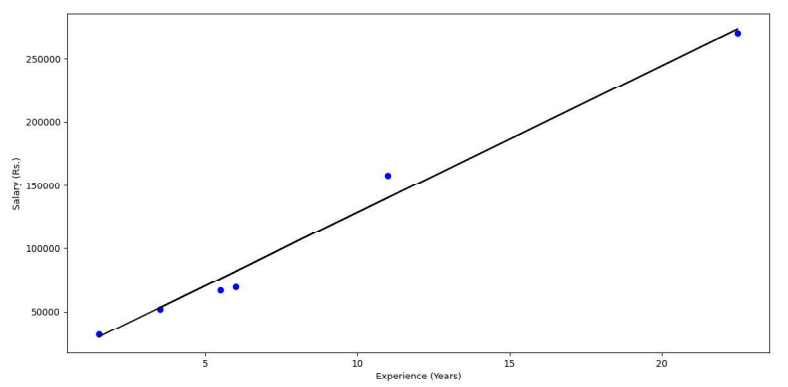
plt.xlabel('Experience (Years)')

plt.ylabel('Salary (Rs.)')

plt.show()

****

**Output –**

****

**Python Script Using Boston Dataset**

# Import necessary packages

import matplotlib.pyplot as plt

plt.style.use('ggplot')

from sklearn import datasets

from sklearn import linear\_model

# Load data

boston = datasets.load\_boston()

yb = boston.target.reshape(-1, 1)

Xb = boston['data'][:,5].reshape(-1, 1)

# Plot data

plt.scatter(Xb,yb)

plt.ylabel('value of house /1000 ($)')

plt.xlabel('number of rooms')

# Create linear regression object

regr = linear\_model.LinearRegression()

# Train the model using the training sets

regr.fit( Xb, yb)

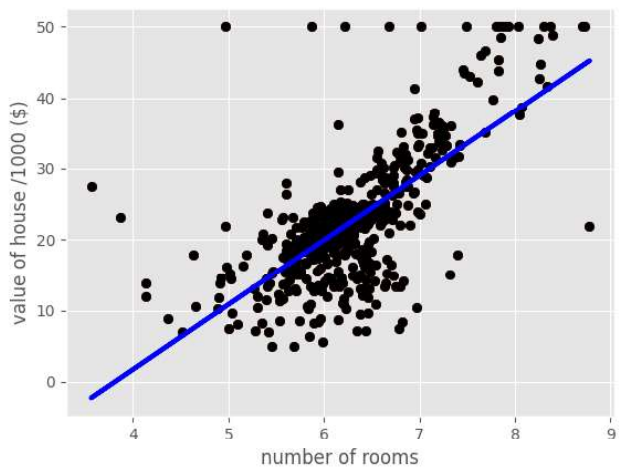
# Plot outputs

plt.scatter(Xb, yb, color='black')

plt.plot(Xb, regr.predict(Xb), color='blue', linewidth=3)

plt.show()

**Output –**

****

**Conclusion –**

The Simple Linear Regression experiment illustrated the relationship between two continuous variables. The model successfully predicted outputs based on a linear equation, showing the significance of the linear relationship in data analysis. This foundational regression technique serves as a basis for understanding more complex modelling.

**Viva – Voce**

**Q1.** What are the assumptions of a linear regression model?

**Ans.** The assumptions of a linear regression model are: The relationship between the independent and dependent variables is linear. The residuals, or errors, are normally distributed with a mean of zero and a constant variance. The independent variables are not correlated with each other (i.e. they are not collinear). The residuals are independent of each other (i.e. they are not autocorrelated). The model includes all the relevant independent variables needed to accurately predict the dependent variable.

**Q2.** What is the difference between simple and multiple linear regression?

**Ans.** Simple linear regression models the relationship between one independent variable and one dependent variable, while multiple linear regression models the relationship between multiple independent variables and one dependent variable. The goal of both methods is to find a linear model that best fits the data and can be used to make predictions about the dependent variable based on the independent variables.

**Q3.** What is the difference between linear regression and logistic regression?

**Ans.** Linear regression is a statistical method used for predicting a numerical outcome, such as the price of a house or the likelihood of a person developing a disease. Logistic regression, on the other hand, is used for predicting a binary outcome, such as whether a person will pass or fail a test, or whether a customer will churn or not.

**Q4.** What are the common techniques used to improve the accuracy of a linear regression model?

**Ans.**

1. Feature selection: selecting the most relevant features for the model to improve its predictive power.
2. Feature scaling: scaling the features to a similar range to prevent bias towards certain features.
3. Regularization: adding a penalty term to the model to prevent overfitting and improve generalization.
4. Cross-validation: dividing the data into multiple partitions and using a different partition for validation in each iteration to avoid overfitting.
5. Ensemble methods: combining multiple models to improve the overall accuracy and reduce variance.

**Q5.** What is the concept of overfitting in linear regression?

**Ans.** Overfitting in linear regression occurs when a model is trained on a limited amount of data and becomes too complex, resulting in poor performance when making predictions on unseen data. This happens because the model has learned to fit the noise or random fluctuations in the training data, rather than the underlying patterns and trends. As a result, the model is not able to generalize well to new data and may produce inaccurate or unreliable predictions. Overfitting can be avoided by using regularization techniques, such as introducing penalty terms to the objective function or using cross-validation to assess the model's performance.

**Experiment – 3**

**Aim – Write a program to demonstrate Logistic Regression**

**Theory –**

Classification techniques are an essential part of machine learning and data mining applications. Approximately 70% of data science problems are classification problems. There are lots of classification problems available, but logistic regression is common and is a useful regression method for solving the binary classification problem. Another category of classification is Multinomial classification, which handles the issues where multiple classes are present in the target variable. For example, the IRIS dataset is a very famous example of multi-class classification. Other examples are classifying article/blog/document categories.

Logistic regression can be used for various classification problems, such as spam detection. Some other examples include: diabetes prediction, whether a given customer will purchase a particular product; whether or not a customer will churn, whether the user will click on a given advertisement link or not, and many more examples.

Logistic Regression is one of the most simple and commonly used Machine Learning algorithms for two-class classification. It is easy to implement and can be used as the baseline for any binary classification problem. Its basic fundamental concepts are also constructive in deep learning. Logistic regression describes and estimates the relationship between one dependent binary variable and independent variables.

**What is Logistic Regression?**

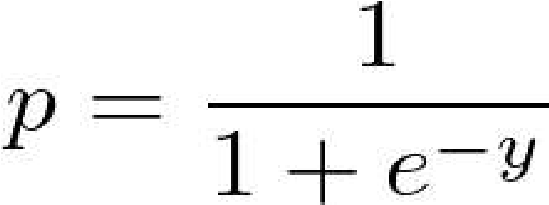
Logistic regression is a statistical method for predicting binary classes. The outcome or target variable is dichotomous in nature. Dichotomous means there are only two possible classes. For example, it can be used for cancer detection problems. It computes the probability of an event occurrence. It is used to estimate discrete values (Binary values like 0/1, yes/no, true/false) based on given set of independent variable(s). In simple words, it predicts the probability of occurrence of an event by fitting data to a logit function. Hence, it is also known as logit regression. Since, it predicts the probability, its output values lies between 0 and 1 (as expected). It is a special case of linear regression where the target variable is categorical in nature. It uses a log of odds as the dependent variable. Logistic Regression predicts the probability of occurrence of a binary event utilizing a logit function.

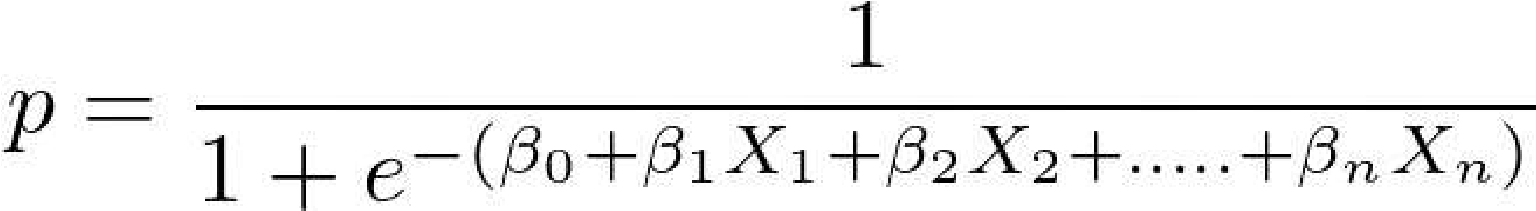
**Linear Regression Equation:**



Where y is a dependent variable and x1, x2 ... and Xn are explanatory variables.

**Sigmoid Function:**



Apply Sigmoid function on linear regression: 

Properties of Logistic Regression:

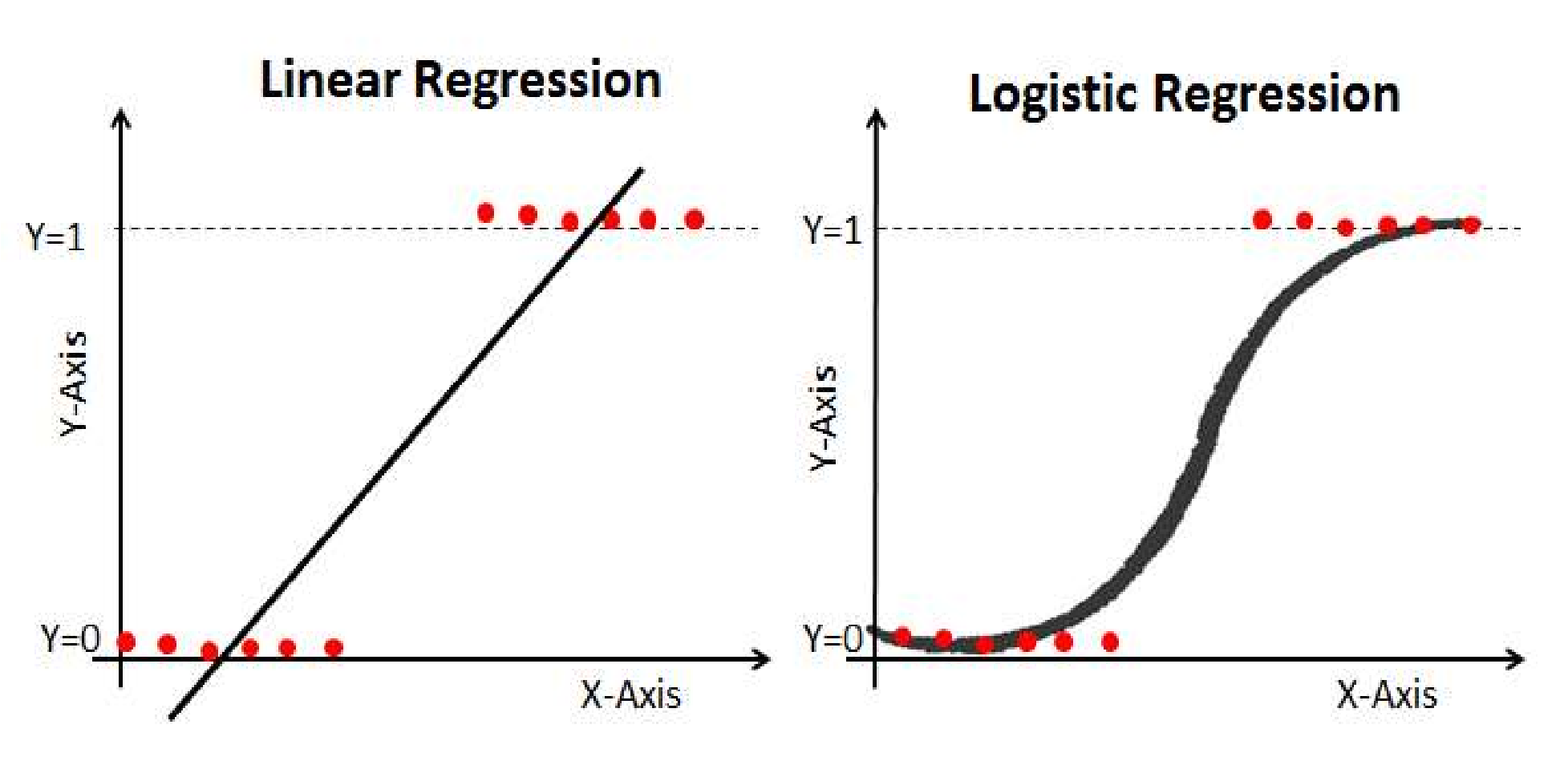
The dependent variable in logistic regression follows Bernoulli Distribution.

Estimation is done through maximum likelihood.

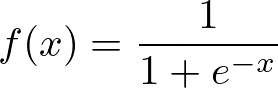
No R Square, Model fitness is calculated through Concordance, KS-Statistics.

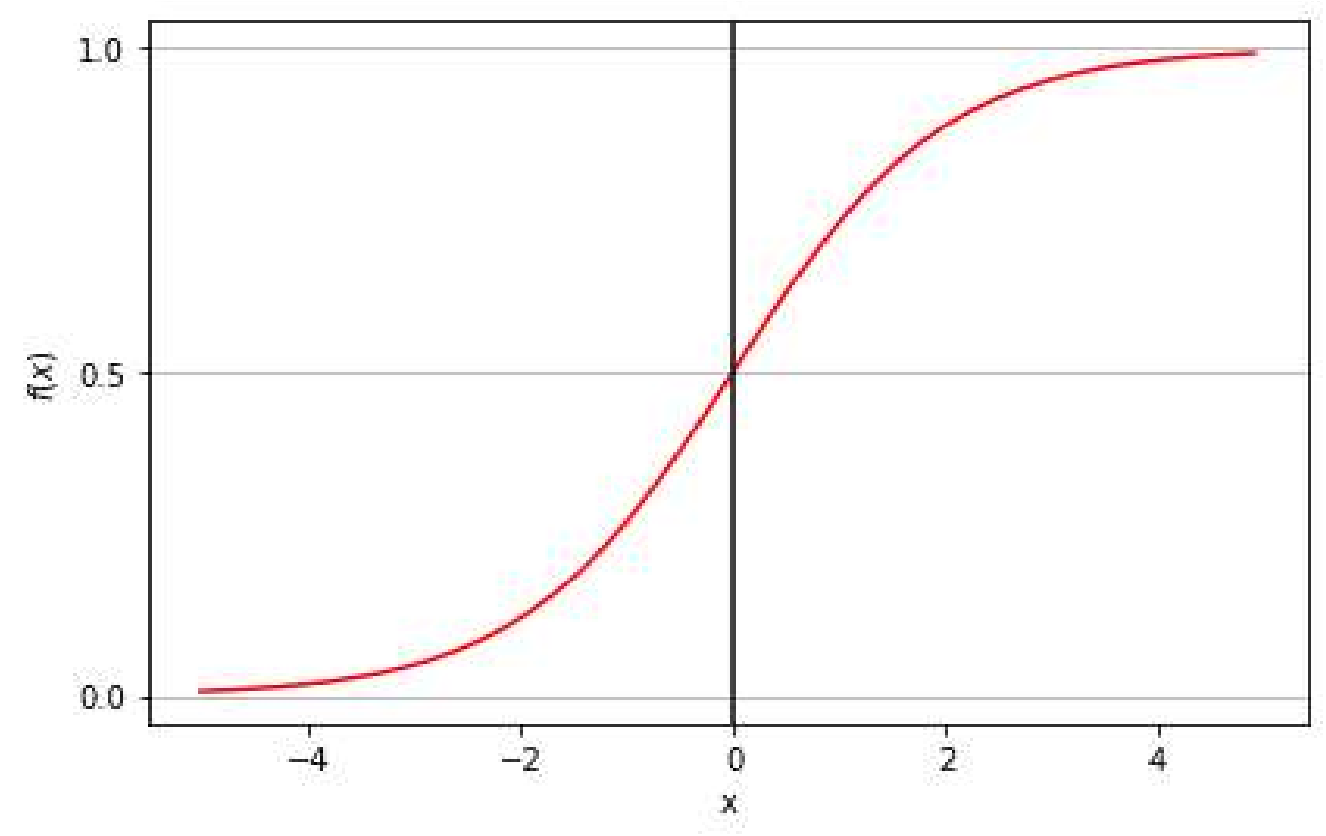
**Linear Regression Vs. Logistic Regression**

Linear regression gives you a continuous output, but logistic regression provides a constant output. An example of the continuous output is house price and stock price. Examples of the discrete output are predicting whether a patient has cancer or not and predicting whether a customer will churn. Logistic regression is estimated using the maximum likelihood estimation (MLE) approach, while linear regression is typically estimated using ordinary least squares (OLS), which can also be considered a special case of MLE when the errors in the model are normally distributed.



**Sigmoid function -** The Sigmoid Function, also called as logistic function, gives an ‘S’ shaped curve that can take any real-valued number and map it into a value between 0 and 1. If the curve goes to positive infinity, y predicted will become 1, and if the curve goes to negative infinity, y predicted will become 0. If the output of the sigmoid function is more than 0.5, we can classify the outcome as 1 or YES, and if it is less than 0.5, we can classify it as 0 or NO. For example, if the output is 0.75, we can say in terms of the probability that there is a 75 percent chance that a patient will suffer from cancer.





Types of Logistic Regression

Types of Logistic Regression:

Binary Logistic Regression: The target variable has only two possible outcomes such as Spam or Not Spam, Cancer or No Cancer.

Multinomial Logistic Regression: The target variable has three or more nominal categories, such as predicting the type of Wine.

Ordinal Logistic Regression: the target variable has three or more ordinal categories, such as restaurant or product rating from 1 to 5.

**Python Implementation**

import numpy as np from sklearn.linear\_model import LogisticRegression from sklearn.model\_selection import train\_test\_split from sklearn.metrics import accuracy\_score

# Sample data

X = np.array([[1, 2], [2, 3], [3, 1], [4, 3], [5, 3]]) y = np.array([0, 0, 0, 1, 1])

# Split the data into training and testing sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Create and fit the model model = LogisticRegression() model.fit(X\_train, y\_train) # Predict the output y\_pred = model.predict(X\_test) # Calculate accuracy accuracy = accuracy\_score(y\_test, y\_pred) print("Accuracy:", accuracy)

**Output –**

Accuracy: 1.0

**Python Implementation (Using Iris Dataset)**

# Import necessary libraries import pandas as pd from sklearn.model\_selection import train\_test\_split from sklearn.linear\_model import LogisticRegression from sklearn.metrics import accuracy\_score

# Load the dataset

#col\_names = ['sepal\_length', 'sepal\_width', 'petal\_length',

'petal\_width', 'species'] dataset = pd.read\_csv('IRIS.csv')

# The output class is in the categorical form in this data set,

# and we need to convert it into the numeric format. So We will use Label Encoder.

from sklearn.preprocessing import LabelEncoder le = LabelEncoder()

dataset['species'] = le.fit\_transform(dataset['species'])

print (dataset.head(100)) # Select only first 100 rows. Split dataset into features and target variable

feature\_cols = ['sepal\_length', 'sepal\_width', 'petal\_length', 'petal\_width']

#X = dataset.iloc[:, :-1] #y = dataset.iloc[:, -1]

X = dataset[feature\_cols]

Y = dataset.species

# Split dataset into training set and test set

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, Y, test\_size=0.3, random\_state=42)

# Initialize the model model = LogisticRegression()

# Train the model

model.fit(X\_train, y\_train)

# Predict the response for test dataset y\_pred = model.predict(X\_test)

# Evaluate accuracy print("Accuracy:", accuracy\_score(y\_test, y\_pred)\*100)

#Model Evaluation using Confusion Matrix

#A confusion matrix is a table that is used to evaluate the performance of a classification model.

# You can also visualize the performance of an algorithm.

# The fundamental part of a confusion matrix is the number of correct and incorrect predictions summed up class-wise.

# import the metrics class

from sklearn import metrics

cnf\_matrix = metrics.confusion\_matrix(y\_test, y\_pred)

print(cnf\_matrix)

#Diagonal values represent accurate predictions, while non-diagonal elements are inaccurate predictions.

#Visualizing confusion matrix using a heatmap

#Let's visualize the results of the model in the form of a confusion matrix using matplotlib and seaborn.

#Here, you will visualize the confusion matrix using Heatmap.

# import required modules

import numpy as np

import seaborn as sns

class\_names=[0,1] # name of classes

fig, ax = plt.subplots()

tick\_marks = np.arange(len(class\_names))

plt.xticks(tick\_marks, class\_names)

plt.yticks(tick\_marks, class\_names)

# create heatmap

sns.heatmap(pd.DataFrame(cnf\_matrix), annot=True, cmap="YlGnBu" ,fmt='g')

ax.xaxis.set\_label\_position("top")

plt.tight\_layout()

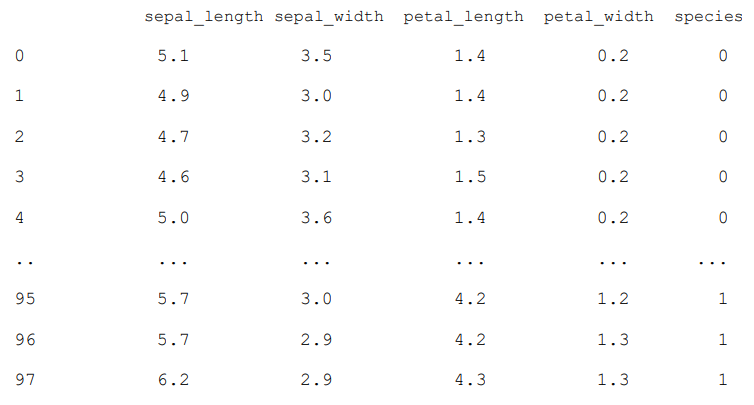
plt.title('Confusion matrix', y=1.1)

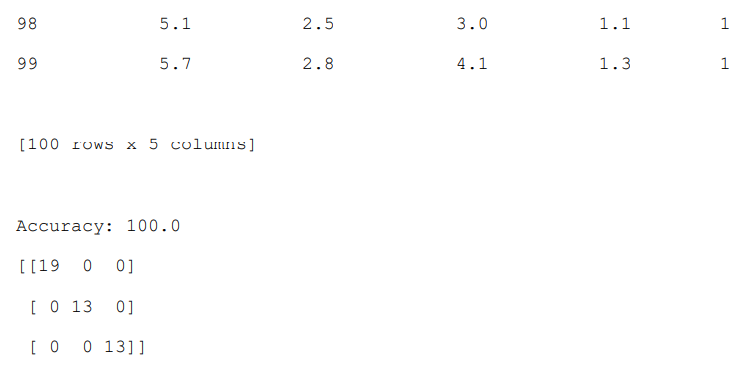
plt.ylabel('Actual label')

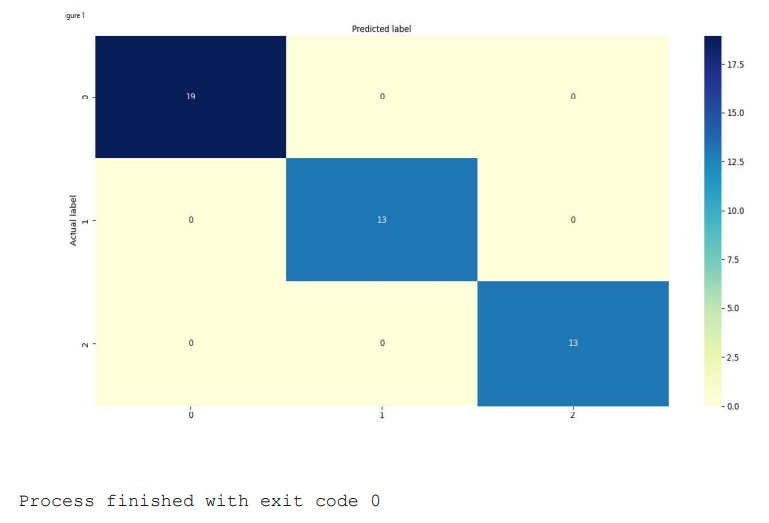
plt.xlabel('Predicted label')

plt.show()

**Output –**

****

****

****

**Conclusion –**

Logistic Regression demonstrated the ability to model binary outcomes based on one or more predictor variables. The model's output, expressed as probabilities, allowed for effective classification. This technique is essential in scenarios like medical diagnosis and credit scoring, where outcomes are categorical.

**Viva – Voce**

**Q1.** Can you use logistic regression for classification between more than two classes?

**Ans.** Yes, it is possible to use logistic regression for classification between more than two classes, and it is called multinomial logistic regression (e.g. SoftMax). However, this is not possible to implement without modifications to the conventional logistic regression model.

**Q2.** Why can’t we use the mean square error cost function used in linear regression for logistic regression?

**Ans.** If we use mean square error in logistic regression, the resultant cost function will be nonconvex, i.e., a function with many local minima, owing to the presence of the sigmoid function in h(x). As a result, an attempt to find the parameters using gradient descent may fail to optimize cost function properly. It may end up choosing a local minima instead of the actual global minima.

**Q3.** If you observe that the cost function decreases rapidly before increasing or stagnating at a specific high value, what could you infer?

**Ans.** A trend pattern of the cost curve exhibiting a rapid decrease before then increasing or stagnating at a specific high value indicates that the learning rate is too high. The gradient descent is bouncing around the global minimum but missing it owing to the larger than necessary step size.

**Q4.** How do you decide the cut-off for the output of logistic regression?

**Ans.** The cut-off is decided such that the accuracy is maximum. Confusion matrix is used here; true negative (actual = 0 and predicted = 0), false negative (actual = 1 and predicted = 0), false positive (actual = 0 and predicted = 1), true positive (actual = 1 and predicted = 1).

**Q5.** What is the importance of regularisation?

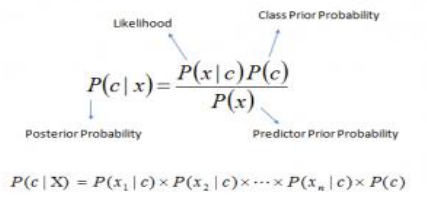
**Ans.** Regularisation is a technique that can help alleviate the problem of overfitting a model. It is beneficial when a large number of parameters are present, which help predict the target function. In these circumstances, it is difficult to select which features to keep manually. Regularisation essentially involves adding coefficient terms to the cost function so that the terms are penalized and are small in magnitude. This helps, in turn, to preserve the overall trends in the data while not letting the model become too complex. These penalties, in effect, restrict the influence a predictor variable can have over the target by compressing the coefficients, thereby preventing overfitting.

**Experiment – 4**

**Aim – Write a program to demonstrate Naive-Bayes Classifier.**

**Theory –**

Naive Bayes algorithm - It is a classification technique based on Bayes’ Theorem with an assumption of independence among predictors. In simple terms, a Naive Bayes classifier assumes that the presence of a particular feature in a class is unrelated to the presence of any other feature. For example, a fruit may be considered to be an apple if it is red, round, and about 3 inches in diameter. Even if these features depend on each other or upon the existence of the other features, all of these properties independently contribute to the probability that this fruit is an apple and that is why it is known as ‘Naive’. Naive Bayes model is easy to build and particularly useful for very large data sets. Along with simplicity, Naive Bayes is known to outperform even highly sophisticated classification methods. Bayes theorem provides a way of calculating posterior probability P(c|x) from P(c), P(x) and P(x|c). Look at the equation below:



Above,

* P(c|x) is the posterior probability of class (c, target) given predictor (x, attributes).
* P(c) is the prior probability of class.
* P(x|c) is the likelihood which is the probability of predictor given class. •
* P(x) is the prior probability of predictor.

**Pros and Cons of Naive Bayes**

Pros:

* It is easy and fast to predict the class of the test data set. It also performs well in multiclass prediction
* When the assumption of independence holds, a Naive Bayes classifier performs better compared to other models like logistic regression and you need less training data.
* It performs well in case of categorical input variables compared to a numerical variable(s). For a numerical variable, the normal distribution is assumed (bell curve, which is a strong assumption).

Cons:

* If the categorical variable has a category (in the test data set), which was not observed in training data set, then the model will assign a 0 (zero) probability and will be unable to make a prediction. This is often known as “Zero Frequency”. To solve this, we can use the smoothing technique. One of the simplest smoothing techniques is called Laplace estimation.
* On the other side, naive Bayes is also known as a bad estimator, so the probability outputs from predict\_proba are not to be taken too seriously.
* Another limitation of Naive Bayes is the assumption of independent predictors. In real life, it is almost impossible that we get a set of predictors which are completely independent.

**Applications of Naive Bayes Algorithms**

* **Real-time Prediction:** Naive Bayes is an eager learning classifier and it is sure fast. Thus, it could be used for making predictions in real time.
* **Multi-class Prediction:** This algorithm is also well known for multi-class prediction feature. Here we can predict the probability of multiple classes of the target variable.
* **Text classification/ Spam Filtering/ Sentiment Analysis:** Naive Bayes classifiers mostly used in text classification (due to a better result in multi-class problems and independence rule) have a higher success rate as compared to other algorithms. As a result, it is widely used in Spam filtering (identify spam e-mail) and Sentiment Analysis (in social media analysis, to identify positive and negative customer sentiments)
* **Recommendation System:** Naive Bayes Classifier and Collaborative Filtering techniques, together build a Recommendation System that uses machine learning and datamining techniques to filter unseen information and predict whether a user would like a given resource or not.

**Python Implementation for Naive Bayes Classifier**

import numpy as np

from sklearn.naive\_bayes

import GaussianNB from sklearn.model\_selection

import train\_test\_split from sklearn.metrics

import accuracy\_score

# Sample data

X = np.array([[1, 2], [2, 3], [3, 1], [4, 3], [5, 3], [6, 2]])

y = np.array([0, 0, 1, 1, 1, 0])

# Split the data into training and testing sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=0)

# Create and fit the Naive Bayes classifier

model = GaussianNB()

model.fit(X\_train, y\_train)

# Predict the output

y\_pred = model.predict(X\_test)

# Calculate accuracy

accuracy = accuracy\_score(y\_test, y\_pred)

print("Accuracy:", accuracy)

**Output** –

Accuracy: 0.5

**Python Implementation for Naive Bayes Classifier using Iris dataset**

from sklearn.datasets import load\_iris

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

from sklearn.naive\_bayes import GaussianNB

X, y = load\_iris(return\_X\_y=True)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.5, random\_state=0)

gnb = GaussianNB()

y\_pred = gnb.fit(X\_train, y\_train).predict(X\_test)

print("Number of mislabeled points out of a total %d points : %d" % (X\_test.shape[0], (y\_test != y\_pred).sum()))

**Output** **–**

Number of mislabelled points out of a total 75 points : 4

Process finished with exit code 0

**Conclusion –**

The Naïve-Bayes Classifier experiment showcased its efficiency in text classification tasks. Despite its simplicity, it performed well by leveraging the Bayes theorem and the assumption of feature independence. This method is particularly effective for applications like spam detection and sentiment analysis.

**Viva – Voce**

**Q1.** What is the Naïve-Bayes classifier based on?

**Ans.** The Naïve-Bayes classifier is based on Bayes' theorem and assumes that the features are independent given the class label.

**Q2.** What types of problems is the Naïve-Bayes classifier commonly used for?

**Ans.** It is commonly used for text classification tasks, such as spam detection and sentiment analysis.

**Q3.** How does the Naïve-Bayes classifier handle continuous features?

**Ans.** For continuous features, it typically assumes a Gaussian distribution and calculates probabilities accordingly.

**Q4.** What is the main advantage of using the Naïve-Bayes classifier?

**Ans.** Its main advantage is its simplicity and speed, making it very efficient for large datasets.

**Q5.** Can Naïve-Bayes perform well with a small amount of training data?

**Ans.** Yes, Naïve-Bayes often performs well even with small datasets due to its probabilistic approach.

**Experiment – 5**

**Aim – Write a program to demonstrate PCA and LDA on IRIS dataset.**

**Theory –**

Principal Component Analysis (PCA) - As the number of features or dimensions in a dataset increases, the amount of data required to obtain a statistically significant result increases exponentially. This can lead to issues such as overfitting, increased computation time, and reduced accuracy of machine learning models this is known as the curse of dimensionality problems that arise while working with high-dimensional data. Moreover, as the number of dimensions increases, the number of possible combinations of features increases exponentially, which makes it computationally difficult to obtain a representative sample of the data. It becomes expensive to perform tasks such as clustering or classification because the algorithms need to process a much larger feature space, which increases computation time and complexity. Additionally, some machine learning algorithms can be sensitive to the number of dimensions, requiring more data to achieve the same level of accuracy as lower-dimensional data. To address the curse of dimensionality, feature engineering techniques are used which include feature selection and feature extraction. Dimensionality reduction is a type of feature extraction technique that aims to reduce the number of input features while retaining as much of the original information as possible. PCA is one such technique that was introduced by the mathematician Karl Pearson in 1901. It works on the 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 is a statistical procedure that uses an orthogonal transformation that converts a set of correlated variables to a set of uncorrelated variables. PCA is the most widely used tool in exploratory data analysis and in machine learning for predictive models.
* It is an unsupervised learning algorithm technique used to examine the interrelations among a set of variables. It is also known as a general factor analysis where regression determines a line of best fit. It reduces the dimensionality of a data set by finding a new set of variables, smaller than the original set of variables, retaining most of the sample’s information, and useful for the regression and classification of data. It identifies a set of orthogonal axes, called principal components, that capture the maximum variance in the data. The principal components are linear combinations of the original variables in the dataset and are ordered in decreasing order of importance. The total variance captured by all the principal components is equal to the total variance in the original dataset. The first principal component captures the most variation in the data, but the second principal component captures the maximum variance that is orthogonal to the first principal component, and so on.

In scikit-learn, PCA is implemented as a transformer object that learns n components in its fit method, and can be used on new data to project it on these components. PCA centres but does not scale the input data for each feature before applying the singular-value-decomposition (SVD)

Linear Discriminant Analysis (LDA) - LDA and Quadratic Discriminant Analysis (QDA) are two classic classifiers, with, as their names suggest, a linear and a quadratic decision surface, respectively. These classifiers are attractive because they have closed-form solutions that can be easily computed, are inherently multiclass, have proven to work well in practice, and have no hyperparameters to tune.

Dimensionality reduction using Linear Discriminant Analysis

LDA can be used to perform supervised dimensionality reduction, by projecting the input data to a linear subspace consisting of the directions which maximize the separation between classes. The dimension of the output is necessarily less than the number of classes, so this is in general a rather strong dimensionality reduction, and only makes sense in a multiclass setting. This is implemented in the transform method. The desired dimensionality can be set using the n\_components parameter. This parameter has no influence on the fit and predict methods.

Both LDA and QDA can be derived from simple probabilistic models which model the class conditional distribution of the data P(X|y=k) for each class k. Predictions can then be obtained by using Bayes’ rule, for each training sample x ∈ Rd:

P(y=k|x)=P(x|y=k)P(y=k)/P(x) =P(x|y=k)P(y=k)/(∑l P(x|y=l)⋅P(y=l)),

and we select the class k which maximizes this posterior probability. More specifically, for linear and quadratic discriminant analysis, P(x|y) is modeled as a multivariate Gaussian distribution with density:

P(x|y=k)=1/((2π)d/2|Σk| 1/2). exp(−1/2.(x−μk)t Σk −1(x−μk)),

where d is the number of features and Σk is the covariance matrix.

LDA is a special case of QDA, where the Gaussians for each class are assumed to share the same covariance matrix: Σk = Σ for all k. Moreover, if in the QDA model we assume that the covariance matrices are diagonal, then the inputs are assumed to be conditionally independent in each class, and the resulting classifier is equivalent to the Gaussian Naive Bayes classifier.

**Python Script**

# PCA on Iris dataset

import matplotlib.pyplot as plt

# unused but required import for doing 3d projections with matplotlib # < 3.2

import mpl\_toolkits.mplot3d

import numpy as np

from sklearn import datasets, decomposition

np.random.seed(5)

iris = datasets.load\_iris()

X = iris.data

y = iris.target

fig = plt.figure(1, figsize=(4, 3))

plt.clf()

ax = fig.add\_subplot(111, projection="3d", elev=48, azim=134)

ax.set\_position([0, 0, 0.95, 1])

plt.cla()

pca = decomposition.PCA(n\_components=3)

pca.fit(X)

X = pca.transform(X)

for name, label in [("Setosa", 0), ("Versicolour", 1), ("Virginica", 2)]:

* ax.text3D(
  + X[y == label, 0].mean(),
  + X[y == label, 1].mean() + 1.5,
  + X[y == label, 2].mean(),
  + name,
  + horizontalalignment="center",
  + bbox=dict(alpha=0.5, edgecolor="w", facecolor="w"),

)

# Reorder the labels to have colors matching the cluster results

y = np.choose(y, [1, 2, 0]).astype(float)

ax.scatter(X[:, 0], X[:, 1], X[:, 2], c=y, cmap=plt.cm.nipy\_spectral, edgecolor="k")

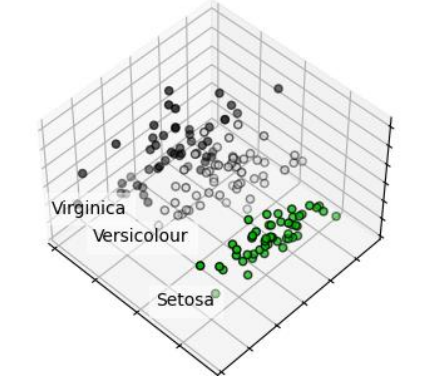
ax.xaxis.set\_ticklabels([])

ax.yaxis.set\_ticklabels([])

ax.zaxis.set\_ticklabels([])

plt.show()

**Output –**

****

**# Comparison of PCA and LDA 2D projection of Iris dataset**

# As we know, the Iris dataset represents 3 kind of Iris flowers (Setosa, Versicolour and Virginica) with 4 attributes: sepal length, sepal width, petal length and petal width.

# Principal Component Analysis (PCA) applied to this data identifies the combination of attributes (principal components, or directions in the feature space) that account for the most variance in the data. Here we plot the different samples on the 2 first principal components.

# Linear Discriminant Analysis (LDA) tries to identify attributes that account for the most variance between classes. In particular, LDA, in contrast to PCA, is a supervised method, using known class labels.

**Python Script**

import matplotlib.pyplot as plt

from sklearn import datasets

from sklearn.decomposition import PCA

from sklearn.discriminant\_analysis import LinearDiscriminantAnalysis

iris = datasets.load\_iris()

X = iris.data

y = iris.target

target\_names = iris.target\_names

pca = PCA(n\_components=2)

X\_r = pca.fit(X).transform(X)

lda = LinearDiscriminantAnalysis(n\_components=2)

X\_r2 = lda.fit(X, y).transform(X)

# Percentage of variance explained for each components

print(

* "Explained variance ratio (first two components): %s"
* % str(pca.explained\_variance\_ratio\_)

)

plt.figure()

colors = ["navy", "turquoise", "darkorange"]

lw = 2

for color, i, target\_name in zip(colors, [0, 1, 2], target\_names):

* plt.scatter(
  + X\_r[y == i, 0], X\_r[y == i, 1], color=color, alpha=0.8, lw=lw, label=target\_name

)

plt.legend(loc="best", shadow=False, scatterpoints=1)

plt.title("PCA of IRIS dataset")

plt.figure()

for color, i, target\_name in zip(colors, [0, 1, 2], target\_names):

* plt.scatter(
  + X\_r2[y == i, 0], X\_r2[y == i, 1], alpha=0.8, color=color, label=target\_name

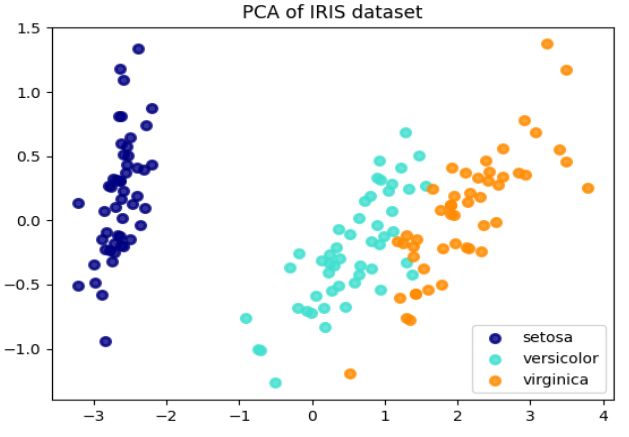
)

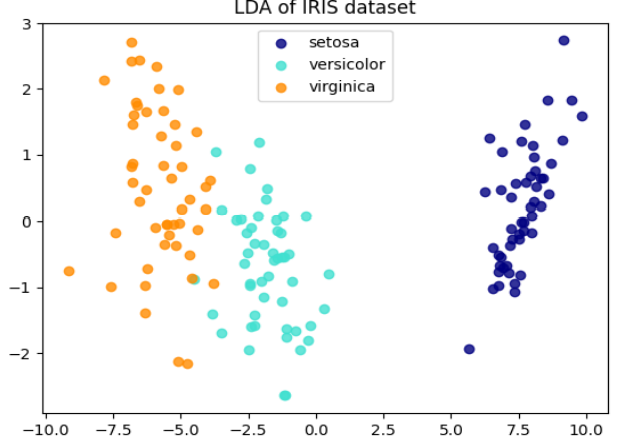
plt.legend(loc="best", shadow=False, scatterpoints=1)

plt.title("LDA of IRIS dataset")

plt.show()

**Output –**

****

****

Explained variance ratio (first two components): [0.92461872 0.05306648]

Process finished with exit code 0

**Conclusion –**

The demonstration of Principal Component Analysis (PCA) and Linear Discriminant Analysis (LDA) on the Iris dataset illustrated dimensionality reduction and class separation techniques. PCA efficiently reduced feature dimensions while preserving variance, while LDA enhanced classification accuracy by maximizing class separability, aiding in better data visualization and interpretation.

**Viva – Voce**

**Q1.** What is the primary purpose of PCA?

**Ans.** The primary purpose of PCA (Principal Component Analysis) is to reduce the dimensionality of a dataset while retaining as much variance as possible.

**Q2.** How does LDA differ from PCA?

**Ans.** LDA (Linear Discriminant Analysis) focuses on maximizing class separability, while PCA aims to maximize variance without considering class labels.

**Q3.** What is the Iris dataset used for?

**Ans.** The Iris dataset is a well-known dataset used for classification and clustering tasks, containing measurements of different Iris flower species.

**Q4.** Can PCA be used for supervised learning?

**Ans.** PCA is primarily an unsupervised technique; however, it can be used in a preprocessing step for supervised learning tasks to reduce dimensionality.

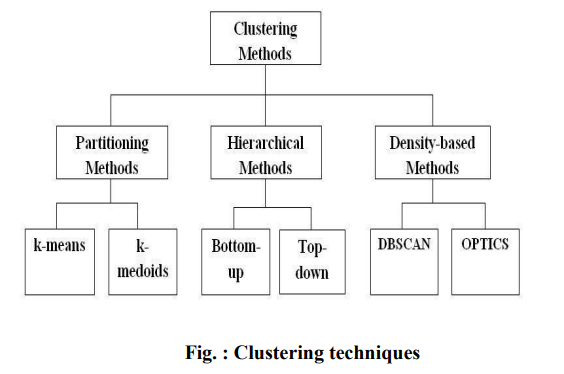
**Q5.** What are the key outputs of PCA?

**Ans.** The key outputs of PCA are the principal components and the explained variance ratio for each component.

**Experiment – 6**

**Aim – Write a program to demonstrate DBSCAN clustering algorithm.**

**Theory –**

****

DBSCAN (Density-Based Spatial Clustering of Applications with Noise) finds core samples in regions of high density and expands clusters from them. This algorithm is good for data which contains clusters of similar density.

Clustering algorithms are fundamentally unsupervised learning methods. However, since make\_blobs gives access to the true labels of the synthetic clusters, it is possible to use evaluation metrics that leverage this “supervised” ground truth information to quantify the quality of the resulting clusters. Examples of such metrics are the homogeneity, completeness, V-measure, Rand-Index, Adjusted Rand-Index and Adjusted Mutual Information (AMI). If the ground truth labels are not known, evaluation can only be performed using the model results itself. In that case, the Silhouette Coefficient comes in handy.

**Python Script**

# Data generation

# We use make\_blobs to create 3 synthetic clusters.

from sklearn.datasets import make\_blobs

from sklearn.preprocessing import StandardScaler

centers = [[1, 1], [-1, -1], [1, -1]]

X, labels\_true = make\_blobs(n\_samples=750, centers=centers,

cluster\_std=0.4, random\_state=0)

X = StandardScaler().fit\_transform(X)

# We can visualize the resulting data:

import matplotlib.pyplot as plt

plt.scatter(X[:, 0], X[:, 1])

plt.show()

# Compute DBSCAN

# One can access the labels assigned by DBSCAN using the labels\_ attribute. Noisy samples are given the label math:-1.

from sklearn import metrics

from sklearn.cluster import DBSCAN

db = DBSCAN(eps=0.3, min\_samples=10).fit(X)

labels = db.labels\_

# Number of clusters in labels, ignoring noise if present.

n\_clusters\_ = len(set(labels)) - (1 if -1 in labels else 0)

n\_noise\_ = list(labels).count(-1)

print("Estimated number of clusters: %d" % n\_clusters\_)

print("Estimated number of noise points: %d" % n\_noise\_)

print(f"Homogeneity: {metrics.homogeneity\_score(labels\_true, labels):.3f}")

print(f"Completeness: {metrics.completeness\_score(labels\_true, labels):.3f}")

print(f"V-measure: {metrics.v\_measure\_score(labels\_true, labels):.3f}")

print(f"Adjusted Rand Index: {metrics.adjusted\_rand\_score(labels\_true, labels):.3f}")

print(

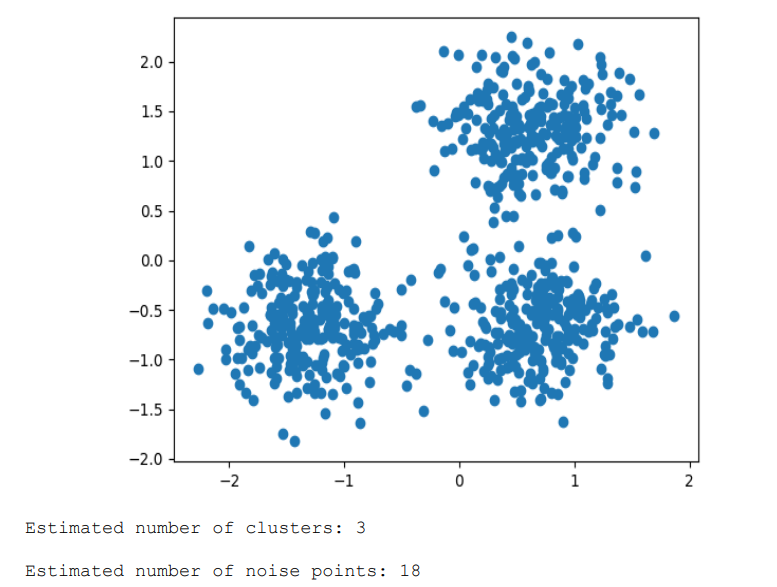
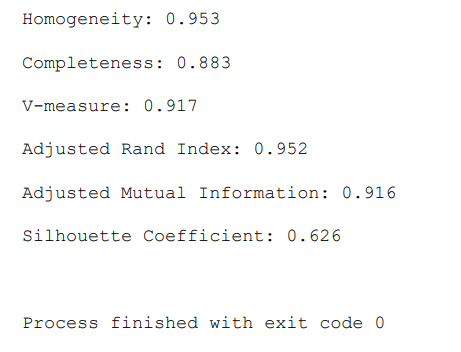
"Adjusted Mutual Information:"

f" {metrics.adjusted\_mutual\_info\_score(labels\_true, labels):.3f}"

)

print(f"Silhouette Coefficient: {metrics.silhouette\_score(X, labels):.3f}")

**Output –**

**** ****

**Conclusion –**

The DBSCAN clustering algorithm experiment highlighted its capability to identify clusters of varying shapes and sizes, effectively handling noise in data. Its density-based approach proved advantageous in scenarios where traditional clustering methods struggled, showcasing its utility in real-world applications like anomaly detection and geographic data analysis.

**Viva – Voce**

**Q1.** What does DBSCAN stand for?

**Ans.** DBSCAN stands for Density-Based Spatial Clustering of Applications with Noise.

**Q2.** How does DBSCAN define a cluster?

**Ans.** DBSCAN defines a cluster as a dense region of points separated by regions of lower density, identified through parameters like epsilon (ε) and minimum samples.

**Q3.** What is the main advantage of using DBSCAN over K-Means?

**Ans.** DBSCAN can identify clusters of arbitrary shapes and handle noise effectively, unlike K-Means, which assumes spherical clusters.

**Q4.** What are the two main parameters of DBSCAN?

**Ans.** The two main parameters are epsilon (ε), which defines the neighborhood radius, and MinPts, which specifies the minimum number of points required to form a dense region.

**Q5.** How does DBSCAN handle outliers?

**Ans.** DBSCAN labels points that do not belong to any cluster as noise, effectively identifying them as outliers.

**Experiment – 7**

**Aim – Write a program to demonstrate K-Medoids clustering algorithm**

**Theory –**

K-Medoids is an unsupervised clustering algorithm in which data points called “medoids" act as the cluster's center. A medoid is a point in the cluster whose sum of distances (also called dissimilarity) to all the objects in the cluster is minimal. The distance can be the Euclidean distance, Manhattan distance, or any other suitable distance function. Therefore, the K -medoids algorithm divides the data into K clusters by selecting K medoids from the data sample.

K-Medoids Clustering Algorithm - The K-medoids clustering algorithm can be summarized as follows –

* **Initialize k medoids −** Select k random data points from the dataset as the initial medoids.
* **Assign data points to medoids −** Assign each data point to the nearest medoid.
* **Update medoids −** For each cluster, select the data point that minimizes the sum of distances to all the other data points in the cluster, and set it as the new medoid.
* Repeat steps 2 and 3 until convergence or a maximum number of iterations is reached.

K-Medoids Clustering – Advantages

Here are the advantages of using K-medoids clustering –

1. Robust to outliers and noise − K-medoids clustering is more robust to outliers and noise than K-means clustering because it uses a representative data point, called a medoid, to represent the center of the cluster.
2. Can handle non-Euclidean distance metrics − K-medoids clustering can be used with any distance metric, including non-Euclidean distance metrics, such as Manhattan distance and cosine similarity.
3. Computationally efficient − K-medoids clustering has a computational complexity of O(k\*n^2), which is lower than the computational complexity of K-means clustering.

K-Medoids Clustering - Disadvantages The disadvantages of using K-medoids clustering are as follows –

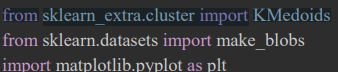
1. Sensitive to the choice of k − The performance of K-medoids clustering can be sensitive to the choice of k, the number of clusters.
2. Not suitable for high-dimensional data − K-medoids clustering may not perform well on highdimensional data because the medoid selection process becomes computationally expensive.

*Implementation in Python*

To implement K-medoids clustering in Python, we can use the scikit-learn library. The scikitlearn library provides the KMedoids class, which can be used to perform K-medoids clustering on a dataset.

Firstly, we need to install scikit-learn-extra using pip install scikit-learn-extra.

Then, we need to import the required libraries –



Next, we generate a sample dataset using the make\_blobs() function from scikit-learn –



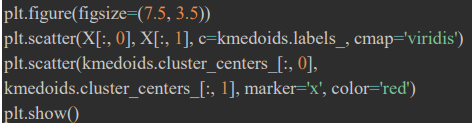
Here, we generate a dataset with 500 data points and 3 clusters.

Next, we initialize the KMedoids class and fit the data –



Here, we set the number of clusters to 3 and use the random\_state parameter to ensure reproducibility.

Finally, we can visualize the clustering results using a scatter plot –



**Complete Python Script**

from sklearn\_extra.cluster import KMedoids

from sklearn.datasets import make\_blobs

import matplotlib.pyplot as plt

# Generate sample data

X, y = make\_blobs(n\_samples=500, centers=3, random\_state=42)

# Cluster the data using KMedoids

kmedoids = KMedoids(n\_clusters=3, random\_state=42)

kmedoids.fit(X)

# Plot the results

plt.figure(figsize=(7.5, 3.5))

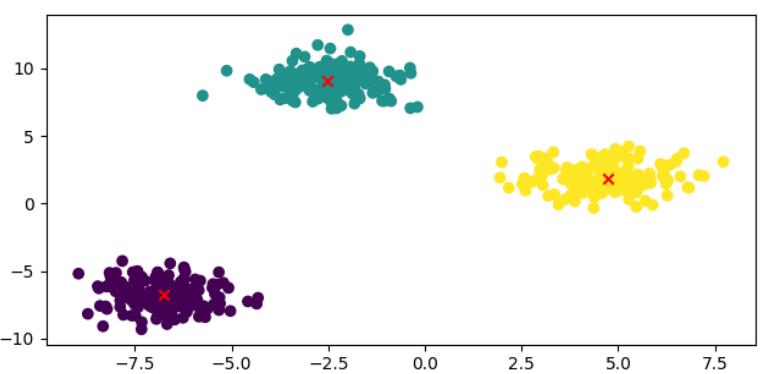
plt.scatter(X[:, 0], X[:, 1], c=kmedoids.labels\_, cmap='viridis')

plt.scatter(kmedoids.cluster\_centers\_[:, 0],

kmedoids.cluster\_centers\_[:, 1], marker='x', color='red')

plt.show()

**Output –**

****

# Here, the data points are plotted as a scatter plot, and colored based on their cluster labels. Also, the medoids are plotted as red crosses.

**Conclusion –**

The K-Medoid clustering algorithm demonstrated its effectiveness in partitioning datasets into clusters based on medoids, or central points. Unlike K-Means, K-Medoids is less sensitive to outliers, making it suitable for robust clustering. This method is valuable for tasks like customer segmentation and pattern recognition.

**Viva – Voce**

**Q1.** What is the main idea behind the K-Medoid clustering algorithm?

**Ans.** The K-Medoid algorithm partitions data into K clusters, using actual data points (medoids) as the center of each cluster instead of centroids.

**Q2.** How does K-Medoid differ from K-Means?

**Ans.** K-Medoid uses medoids, which are the most centrally located points in a cluster, making it more robust to noise and outliers than K-Means.

**Q3.** What distance metric is commonly used in K-Medoid?

**Ans.** K-Medoid commonly uses Manhattan distance or Euclidean distance to calculate the distance between points and medoids.

**Q4.** What is the typical process for updating medoids in K-Medoid?

**Ans.** The process involves iteratively selecting the most centrally located point in each cluster as the new medoid, minimizing the total distance within the cluster.

**Q5.** Can K-Medoid be used for categorical data?

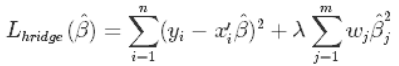
**Ans.** Yes, K-Medoid can handle categorical data using appropriate distance metrics, making it versatile for various data types.

**Experiment – 8**

**Aim – Write a program to demonstrate Lasso and Ridge regression.**

**Theory =**

Ridge Regression, also known as L2 regularization, is an extension to Linear Regression that introduces a regularization term to reduce model complexity and help prevent overfitting. In simple terms, Ridge Regression helps minimize the sum of the squared residuals and the parameters’ squared values scaled by a factor (lambda or α). This regularization term, λ, controls the strength of the constraint on the coefficients and acts as a tuning parameter. The Ridge Regression can help shrink the coefficients of less significant features close to zero but not exactly zero. By doing so, it reduces the model’s complexity while still preserving its interpretability



Lasso (Least Absolute Shrinkage and Selection Operator) Regression is another regularization technique that prevents overfitting in linear Regression models. Like Ridge Regression, Lasso Regression adds a regularization term to the linear Regression objective function. The difference lies in the loss function used — Lasso Regression uses L1 regularization, which aims to minimize the sum of the absolute values of coefficients multiplied by penalty factor λ. Unlike Ridge Regression, Lasso Regression can force coefficients of less significant features to be exactly zero. As a result, Lasso Regression performs both regularization and feature selection simultaneously.



**Python Script**

import pandas as pd

import numpy as np

from sklearn.model\_selection

import train\_test\_split

# Load the dataset

#col\_names = ['sepal\_length', 'sepal\_width', 'petal\_length', 'petal\_width', 'species']

dataset = pd.read\_csv('IRIS.csv')

# dataset = dataset.drop(columns = ['s.no.'])

# drop s.no. column which is not required

# Let’s have the information about the data type of the data set.

print(dataset.info())

#SepalLength, SepalWidth, PetalLength, and PetalWidth have float data types. 'Species' has an object data type.

# Let's check the number of samples of each class in Species.

print(dataset['species'].value\_counts())

# While training the model, we must remove all the null values.

# To check whether the data set contains the null values, we write

print(dataset.isnull().sum())

#It will display the number of null values in each column. There are no null or nan values in the datasets, as all entries in last column are displayed as 0.

#Now, We will visualize the data in the form of graphs. First, let's display some basic charts. For each column, let us create a histogram.

#dataset['SepalLengthCm'].hist()

#dataset['SepalWidthCm'].hist()

# The output class is in the categorical form in this data set,

# and we need to convert it into the numeric format. So We will use Label Encoder.

from sklearn.preprocessing import LabelEncoder

le = LabelEncoder()

dataset['species'] = le.fit\_transform(dataset['species'])

print (dataset.head(100))

# Select only first 100 rows

# Split dataset into features and target variable

feature\_cols = ['sepal\_length', 'sepal\_width', 'petal\_length', 'petal\_width']

X = dataset[feature\_cols]

Y = dataset.species

# Split dataset into training set and test set

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, Y, test\_size=0.3, random\_state=42)

# Lasso Regression model

from sklearn.linear\_model import Lasso

# Initialize the model

model = Lasso()

# Train the model

model.fit(X\_train, y\_train)

lasso = Lasso().fit(X\_train, y\_train)

print("Training set score (Lasso model):

{:.2f}".format(lasso.score(X\_train, y\_train)))

print("Test set score (Lasso model):

{:.2f}".format(lasso.score(X\_test, y\_test)))

print("Number of features used (Lasso model):

{}".format(np.sum(lasso.coef\_ != 0)))

# Ridge Regression model

from sklearn.linear\_model import Ridge

# Initialize the model

model = Ridge()

# Train the model

model.fit(X\_train, y\_train)

ridge = Ridge().fit(X\_train, y\_train)

print("Training set score (Ridge model):

{:.2f}".format(ridge.score(X\_train, y\_train)))

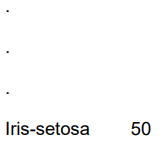
print("Test set score (Ridge model):

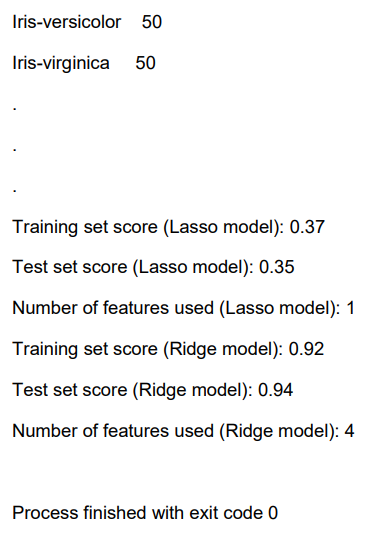
{:.2f}".format(ridge.score(X\_test, y\_test)))

print("Number of features used (Ridge model):

{}".format(np.sum(ridge.coef\_ != 0)))

**Output –**

****

****

**Conclusion –**

The Lasso and Ridge regression experiment illustrated the importance of regularization in linear models. Lasso performs variable selection by adding an L1 penalty, while Ridge reduces model complexity through L2 regularization. Both techniques effectively prevent overfitting, enhancing model generalization and performance.

**Viva – Voce**

**Q1.** What is the primary purpose of Lasso regression?

**Ans.** The primary purpose of Lasso regression is to perform variable selection and regularization to prevent overfitting in linear regression models.

**Q2.** How does Ridge regression differ from Lasso regression?

**Ans.** Ridge regression applies an L2 penalty to the coefficients, which shrinks them but does not eliminate any, while Lasso regression applies an L1 penalty, potentially driving some coefficients to zero.

**Q3.** What is the main advantage of using regularization in regression?

**Ans.** Regularization helps to improve model generalization by preventing overfitting, especially in datasets with many features.

**Q4.** In which scenarios would you prefer Lasso over Ridge regression?

**Ans.** Lasso is preferred when feature selection is essential, as it can eliminate irrelevant features by setting their coefficients to zero.

**Q5.** How do you choose the regularization parameter in Lasso and Ridge regression?

**Ans.** The regularization parameter is typically chosen using techniques like cross-validation to find the value that minimizes the error on validation data.

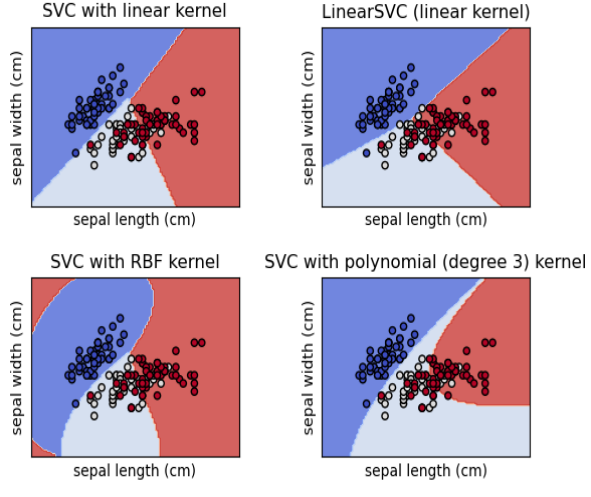
**Experiment – 9**

**Aim – Write a program to demonstrate SVM Classification method.**

**Theory –**

In machine learning, support vector machines (SVMs, also known as support vector networks) are supervised learning models with associated learning algorithms that analyze data used for classification and regression analysis and outliers detection. An SVM is a discriminative classifier formally defined by a separating hyperplane. In other words, given labeled training data (supervised learning), the algorithm outputs an optimal hyperplane which categorizes new examples.

An SVM model is a representation of the examples as points in space, mapped so that the examples of the separate categories are divided by a clear gap that is as wide as possible. In addition to performing linear classification, SVMs can efficiently perform a non-linear classification, implicitly mapping their inputs into high-dimensional feature spaces. Given a set of training examples, each marked as belonging to one or the other of two categories, an SVM training algorithm builds a model that assigns new examples to one category or the other, making it a non-probabilistic binary linear classifier.



The advantages of support vector machines are:

* Effective in high dimensional spaces.
* Still effective in cases where number of dimensions is greater than the number of samples.
* Uses a subset of training points in the decision function (called support vectors), so it is also memory efficient.
* Versatile: different Kernel functions can be specified for the decision function. Common kernels are provided, but it is also possible to specify custom kernels.

The disadvantages of support vector machines include:

* If the number of features is much greater than the number of samples, avoid over-fitting in choosing Kernel functions and regularization term is crucial.
* SVMs do not directly provide probability estimates, these are calculated using an expensive five-fold cross-validation

**Python Script (SVM classification using an array)**

# Based on array X and class label Y

from sklearn import svm

X = [[0, 0], [1, 1]]

y = [0, 1]

model = svm.SVC()

model.fit(X, y)

print(model.predict([[2, 2]]))

**Output –**

[1]

Process finished with exit code 0

**Python Script (SVM classification using IRIS dataset)**

# importing required libraries

import numpy as np

import pandas as pd

# reading csv file and extracting class column to y.

x = pd.read\_csv('IRIS.csv')

from sklearn.preprocessing import LabelEncoder

le = LabelEncoder()

x['species'] = le.fit\_transform(x['species'])

a = np.array(x)

print(a) y = a[:, 4]

# classes having encoded values 0, 1 and 2

# extracting two features

#x = np.column\_stack((x.sepal\_length, x.sepal\_width))

x = np.column\_stack((x.petal\_length, x.petal\_width))

print(x)

print(y)

# import support vector classifier

# "Support Vector Classifier"

from sklearn.svm import SVC

clf = SVC(kernel='linear')

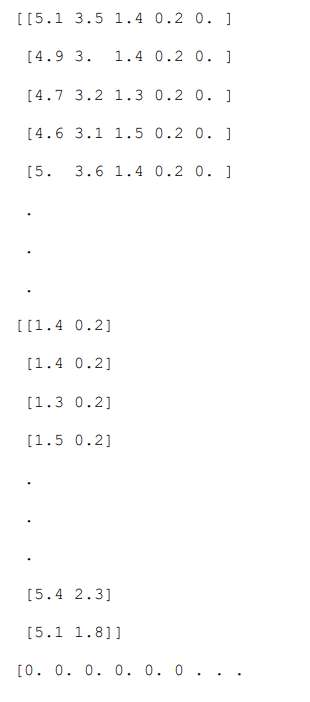
# fitting x samples and y classes

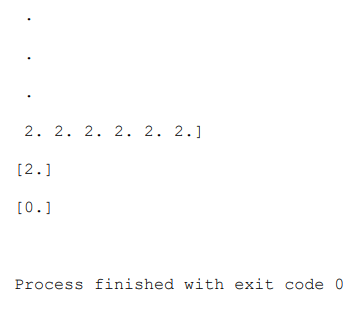
clf.fit(x, y)

print(clf.predict([[4, 5]]))

print(clf.predict([[0.2, 0.1]]))

**Output –**

****

****

**Conclusion –**

The SVM classification method experiment showcased its ability to classify data by finding the optimal hyperplane that separates different classes. SVM's effectiveness in high-dimensional spaces and flexibility with different kernels made it a powerful tool for complex classification tasks, such as image recognition and bioinformatics.

**Viva – Voce**

**Q1.** What is the main concept behind Support Vector Machines (SVM)?

**Ans.** The main concept behind SVM is to find the optimal hyperplane that maximizes the margin between different classes in the feature space.

**Q2.** How does SVM handle non-linearly separable data?

**Ans.** SVM can handle non-linearly separable data by using kernel functions to transform the data into a higher-dimensional space where it can be linearly separated.

**Q3.** What are the common types of kernels used in SVM?

**Ans.** Common kernels include linear, polynomial, and radial basis function (RBF) kernels.

**Q4.** What role do support vectors play in SVM?

**Ans.** Support vectors are the data points closest to the hyperplane, and they are crucial in defining the position and orientation of the hyperplane.

**Q5.** When would you prefer SVM over other classification methods?

**Ans.** SVM is preferred in high-dimensional spaces or when the data has clear margin separability, especially when the number of features exceeds the number of samples.

**Experiment – 10**

**Aim – Program to study various model evaluation metrics**

**Theory –**

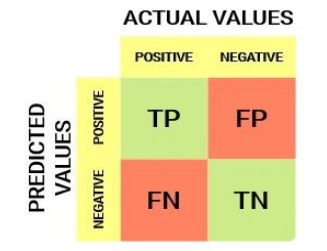
When we build a machine learning model, the next task is to evaluate and validate how good (or bad) the model is, so that we can decide whether to implement it or not. That’s where the “AreaUnder-the-Curve” (AUC) of the “Receiver-Operating-Characteristic” (ROC) comes into picture, where we calculate the Area-Under-the-Curve of the ROC. In other words, the AUC ROC curve helps us to visualize how well our machine learning classifier performs. Although it works only for binary classification problems, we can extend it to evaluate multi-class classification problems.

*Definitions*

An ROC curve, or receiver operating characteristic curve, is like a graph that shows how well a classification model performs. It helps us to see how the model makes decisions at different levels of certainty. The curve has two lines: one for how often the model correctly identifies positive cases (true positives) and another for how often it mistakenly identifies negative cases as positive (false positives). By looking at this graph, we can understand how good the model is and choose the threshold that gives us the right balance between correct and incorrect predictions. As mentioned earlier, it is an evaluation metric for binary classification problems. It is a probability curve that plots the True Positive Rate (TPR) against False Positive Rate (FPR) at various threshold values and essentially separates the ‘signal’ from the ‘noise.’ In other words, it shows the performance of a classification model at all classification thresholds. In other words, it shows how good a model is at telling things apart. It helps us see how often the model correctly identifies positive things and how often it correctly avoids labelling negative things as positive. So, it basically shows how well the model is working for binary classification tasks.

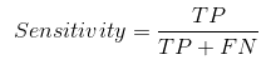
The AUC is the measure of the ability of a binary classifier to distinguish between classes and is used as a summary of the ROC curve. When AUC = 1, the classifier can correctly distinguish between all the Positive and the Negative class points. If, however, the AUC is 0, then the classifier would predict all Negatives as Positives and all Positives as Negatives. When 0.5 < AUC < 1, there is a high chance that the classifier will be able to distinguish the positive class values from the negative ones. This is so because the classifier is able to detect more numbers of True positives and True negatives than False negatives and False positives. When AUC = 0.5, then the classifier is not able to distinguish between Positive and Negative class points, i.e. the classifier either predicts a random class or a constant class for all the data points. So, the higher the AUC value for a classifier, the better it is.

*Confusion matrix [ROC curve]*

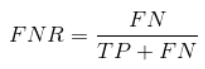


Defining the terms used in AUC and ROC Curve and summarizing

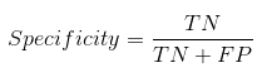
* AUC (Area Under the Curve): A single metric representing the overall performance of a binary classification model based on the area under its ROC curve.
* ROC Curve (Receiver Operating Characteristic Curve): A graphical plot illustrating the trade-off between TPR and FPR at various classification thresholds.
* True Positive Rate (also called Sensitivity/ Recall): Proportion of actual positives correctly identified by the model. A simple example would be determining what proportion of the actual sick people were correctly detected by the model.



* False Negative Rate (FNR): FNR tells us what proportion of the positive class got incorrectly classified by the classifier. A higher TPR and a lower FNR are desirable since we want to classify the positive class correctly.



* Specificity or True Negative Rate: Proportion of actual negatives correctly identified by the model. Taking the same example as in Sensitivity, Specificity would mean determining the proportion of healthy people who were correctly identified by the model.



* False Positive Rate: The model incorrectly classifies the proportion of actual negatives as positives.



A higher TNR and a lower FPR are desirable since we want to classify the negative class correctly.

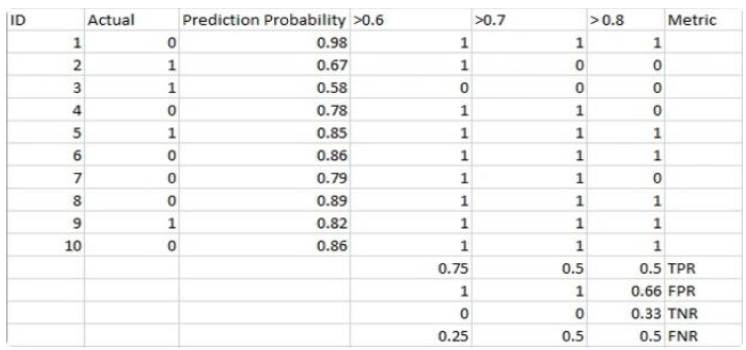
Out of these metrics, Sensitivity and Specificity are perhaps the most important, and we will see later on how these are used to build an evaluation metric. But before that, let’s understand why the probability of prediction is better than predicting the target class directly.

*Probability of Predictions*

A machine learning classification model can be used to naturally predict the data point’s actual class or predict its probability of belonging to different classes, employing an AUC-ROC curve for evaluation. The latter gives us more control over the result. We can determine our own threshold to interpret the result of the classifier, a valuable aspect when considering the nuances of the ROC Curve. This approach is sometimes more prudent than just building a completely new model.

Setting different thresholds for classifying positive classes for data points will inadvertently change the Sensitivity and Specificity of the model. And one of these thresholds will probably give a better result than the others, depending on whether we are aiming to lower the number of False Negatives or False Positives. Have a look at the table below:

*AUC-ROC curve example*

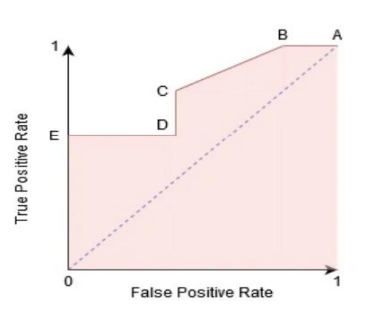
**

The metrics change with the changing threshold values. We can generate different confusion matrices and compare the various metrics that we discussed in the previous section. But that would not be a prudent thing to do. Instead, we can plot roc curves between some of these metrics to quickly visualize which threshold is giving us a better result.

How Does the AUC-ROC Curve Work?

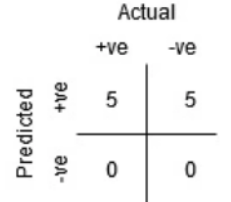
In an AUC-ROC curve, a higher X-axis value indicates a higher number of False positives than True negatives. While a higher Y-axis value indicates a higher number of True positives than False negatives. So, the choice of the threshold depends on the ability to balance False positives and False negatives naturally.

Let’s dig a bit deeper and understand what our ROC curve would look like for different threshold values and how the specificity and sensitivity would vary



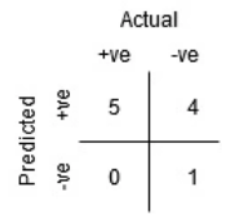
We can try and understand this graph by generating a confusion matrix for each point corresponding to a threshold and talk about the performance of our classifier:

*Sample Confusion matrix [ROC curve]*

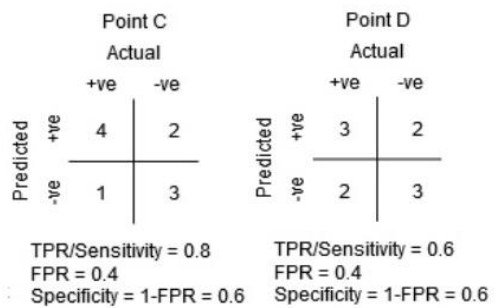
**

Point A is where the Sensitivity is the highest and Specificity the lowest. This means all the Positive class points are classified correctly, and all the Negative class points are classified incorrectly.

In fact, any point on the blue line corresponds to a situation where the True Positive Rate is equal to False Positive Rate. All points above this line correspond to the situation where the proportion of correctly classified points belonging to the Positive class is greater than the proportion of incorrectly classified points belonging to the Negative class.

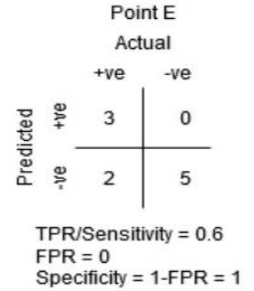


Although Point B has the same Sensitivity as Point A, it has a higher Specificity. Meaning the number of incorrectly Negative class points is lower than the previous threshold. This indicates that this threshold is better than the previous one.



Between points C and D, the Sensitivity at point C is higher than point D for the same Specificity. This means, for the same number of incorrectly classified Negative class points, the classifier predicted a higher number of Positive class points. Therefore, the threshold at point C is better than point D.

Now, depending on how many incorrectly classified points we want to tolerate for our classifier, we would choose between point B or C to predict whether you can defeat someone in PUBG or not.



Point E is where the Specificity becomes highest. Meaning the model classifies no False Positives. The model can correctly classify all the Negative class points! We would choose this point if our problem was to give perfect song recommendations to our users.

Going by this logic, we can guess where the point corresponding to a perfect classifier would lie on the graph. In the present case, it would be on the top-left corner of the ROC Curve graph corresponding to the coordinate (0, 1) in the cartesian plane. Here, both the Sensitivity and Specificity would be the highest, and the classifier would correctly classify all the Positive and Negative class points.

**Python Script**

# Let’s create an arbitrary data using the sklearn make\_classification method:

import pandas as pd

from sklearn.datasets import make\_classification

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

# generate two class dataset

X, y = make\_classification(n\_samples=1000, n\_classes=2, n\_features=20, random\_state=27)

# split into train-test sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=27)

print(pd.DataFrame(X))

print(pd.Series(y))

# We will test the performance of two classifiers on this dataset:

# train models

from sklearn.linear\_model import LogisticRegression

from sklearn.neighbors import KNeighborsClassifier

# logistic regression

model1 = LogisticRegression()

# knn

model2 = KNeighborsClassifier(n\_neighbors=4)

# fit model

model1.fit(X\_train, y\_train)

model2.fit(X\_train, y\_train)

# predict probabilities

pred\_prob1 = model1.predict\_proba(X\_test)

pred\_prob2 = model2.predict\_proba(X\_test)

#Sklearn has a very potent method, roc\_curve(), which computes the ROC for your classifier in a matter of seconds!

# It returns the FPR, TPR, and threshold values:

# roc curve for models

fpr1, tpr1, thresh1 = roc\_curve(y\_test, pred\_prob1[:,1], pos\_label=1)

print('Specificity = 1 - FPR (for Logistic Regression):', 1-fpr1)

fpr2, tpr2, thresh2 = roc\_curve(y\_test, pred\_prob2[:,1], pos\_label=1)

print('Specificity = 1 - FPR (for KNN):', 1-fpr2)

from sklearn.metrics import roc\_curve

# roc curve for tpr = fpr

random\_probs = [0 for i in range(len(y\_test))]

p\_fpr, p\_tpr, \_ = roc\_curve(y\_test, random\_probs, pos\_label=1)

# The AUC score can be computed using the roc\_auc\_score() method of sklearn:

from sklearn.metrics

import roc\_auc\_score

# auc scores

auc\_score1 = roc\_auc\_score(y\_test, pred\_prob1[:,1])

auc\_score2 = roc\_auc\_score(y\_test, pred\_prob2[:,1])

print('AUC Scores:', auc\_score1, auc\_score2)

# We can also plot the receiver operating characteristic curves for the two algorithms using matplotlib:

# matplotlib

import matplotlib.pyplot as plt

plt.style.use('seaborn')

# plot roc curves

plt.plot(fpr1, tpr1, linestyle='--',color='orange', label='Logistic Regression')

plt.plot(fpr2, tpr2, linestyle='-',color='green', label= 'KNN')

plt.plot(p\_fpr, p\_tpr, linestyle=':', color='blue', label='Probability(TPR) vs Probability(FPR)')

# title

plt.title('ROC curve')

# x label plt.xlabel('False Positive Rate')

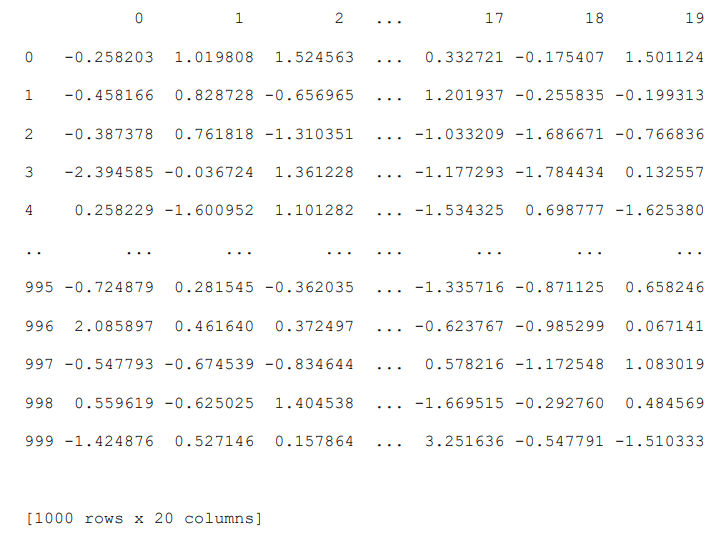
# y label plt.ylabel('True Positive rate/ Sensitivity')

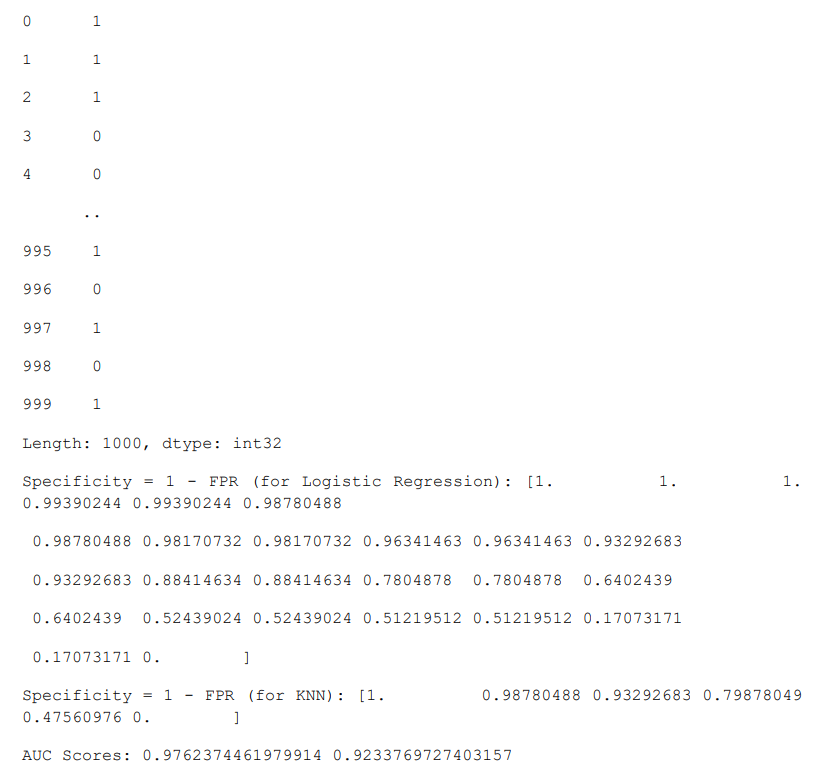
plt.legend(loc='best')

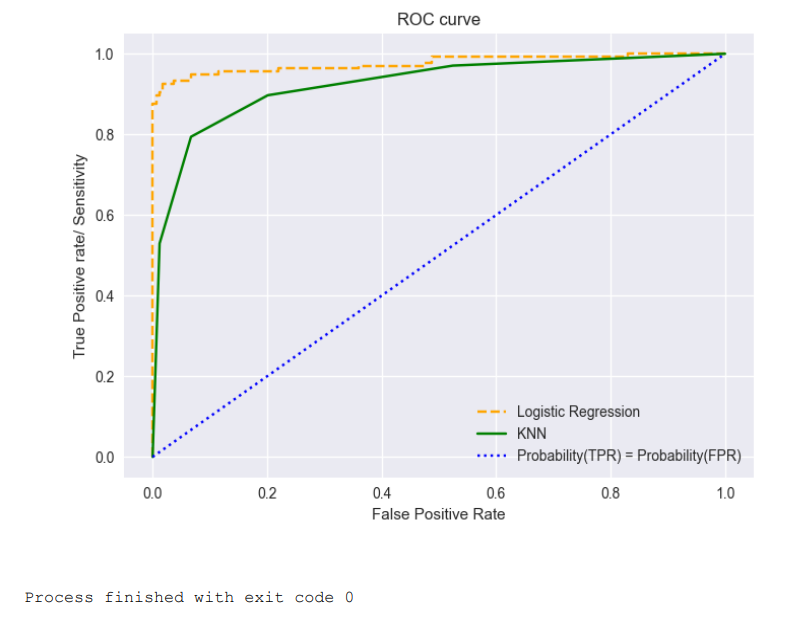
# Best Location

plt.show()

**Output –**

****

****

****

**Conclusion –**

The study of various model evaluation metrics provided insights into assessing model performance. Metrics like accuracy, precision, recall, and F1-score helped quantify the effectiveness of classification models. Understanding these metrics is crucial for model selection and refinement in machine learning projects, ensuring reliable predictions in practical applications.

**Viva – Voce**

**Q1.** What is the purpose of model evaluation metrics?

**Ans.** Model evaluation metrics are used to assess the performance of machine learning models, providing insights into their accuracy, precision, recall, and other characteristics.

**Q2.** What is the difference between precision and recall?

**Ans.** Precision measures the proportion of true positive predictions among all positive predictions, while recall measures the proportion of true positives among all actual positive instances.

**Q3.** What is the F1-score?

**Ans.** The F1-score is the harmonic mean of precision and recall, providing a single metric that balances both concerns, especially in imbalanced datasets.

**Q4.** Why is accuracy not always a reliable metric?

**Ans.** Accuracy can be misleading in imbalanced datasets where one class significantly outnumbers another, leading to high accuracy despite poor model performance on the minority class.

**Q5.** What are confusion matrices used for?

**Ans.** Confusion matrices are used to visualize the performance of a classification model, showing the counts of true positive, true negative, false positive, and false negative predictions.