VARDHAMAN COLLEGE OF ENGINEERING, HYDERABAD

(AUTONOMOUS)

Affiliated to **JNTUH**, approved by **AICTE**, Accredited by **NAAC** with **A^{++}** Grade,
ISO 9001:2015 Certified, Kacharam, Shamshabad, Hyderabad – 501218, Telangana, India

MACHINE LEARNING(A8704)

Laboratory Manual

Department of Computer Science & Engineering (AI&ML)

Class & Semester	**II B.Tech & II Semester**
Course Type	**INTEGRATED**
Category	**CORE ENGINEERING**
Regulation	**VCE-R22**
Academic Year	**2023-2024**
Course Instructors	**Dr M. Ramachandro, Associate Professor, Department of CSE(AI&ML), Vardhaman College of Engineering, Kacharam Village, Shamshabad, Hyderabad.**

PROGRAM OUTCOMES (POS)

PO1: Engineering Knowledge: Apply knowledge of mathematics, science, engineering fundamentals and an engineering specialization to the solution complex engineering problems.

PO2: Problem Analysis: Identify, formulate, research literature and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences and engineering sciences.

PO3: Design/ Development of Solutions: Design solutions for complex engineering problems and design system components or processes that meet specified needs with appropriate consideration for public health and safety, cultural, societal and environmental considerations.

PO4: Conduct investigations of complex problems: Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data and synthesis of information to provide valid conclusions.

PO5: Modern Tool Usage: Create, select and apply appropriate techniques, resources and modern engineering and IT tools including prediction and modelling to complex engineering activities with an understanding of the limitations.

PO6: The Engineer and Society: Apply reasoning informed by contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to professional engineering practice.

PO7: Environment and Sustainability: Understand the impact of professional engineering solutions in societal and environmental contexts and demonstrate knowledge of and need for sustainable development.

PO8: Ethics: Apply ethical principles and commit to professional ethics and responsibilities and norms of engineering practice.

PO9: Individual and Team Work: Function effectively as an individual, and as a member or leader in diverse teams and in multi-disciplinary settings.

PO10: Communication: Communicate effectively on complex engineering activities with the engineering community and with society at large, such as being able to comprehend and write effective reports and design documentation, make effective presentations and give and receive clear instructions.

PO11: Project Management and Finance: Demonstrate knowledge and understanding of engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.

PO12: Life-long Learning: Recognize the need for and have the preparation and ability to Engage in independent and life- long learning in the broadest context of technological Change.

PROGRAM SPECIFIC OUTCOMES (PSOs)

PSO1: To collect requirements, analyze, design, implement and test software Systems.

PSO2: To analyze the errors and debug them within minimal time.

COURSE OVERVIEW

This course will introduce the field of Machine Learning, in particular focusing on the core concepts of supervised and unsupervised learning. In supervised learning we will discuss algorithms which are trained on input data labelled with a desired output, for instance an image of a face and the name of the person whose face it is, and learn a function mapping from the input to the output. Unsupervised learning aims to discover latent structure in an input signal where no output labels are available, an example of which is grouping web-pages based on the topics they discuss. Students will learn the algorithms which underpin many popular Machine Learning techniques, as well as developing an understanding of the theoretical relationships between these algorithms. The practical's will concern the application of machine learning to a range of real-world problems.

COURSE OBJECTIVE

This course covers a wide variety of topics in machine learning and statistical modelling. The primary goal of the class is to help participants gain a deep understanding of the concepts, techniques and mathematical frameworks used by experts in machine learning. It is designed to make valuable machine learning skills more accessible to individuals with a strong math background, including software developers, experimental scientists, engineers and financial professionals.

COURSE OUTCOMES (COs)

After the completion of the course, the student will be able to:

CO#	Course Outcomes	POs	PSOs
A8703.1	Identify the various concepts and challenges in machine Learning.	---	---
A8703.2	Select modelling and evaluation technique for handling real time data.	1,5	1,2
A8703.3	Use supervised learning algorithms for a given problem.	2,5	1,2
A8703.4	Examine supervised and unsupervised learning algorithms for analyzing data.	1,5	1,2
A8703.5	Identify the various concepts of neural network to develop AI based applications.	3,5	1,2

BLOOM'S LEVEL OF THE COURSE OUTCOMES

CO#	BLOOM'S LEVEL					
	Remember (L1)	Understand (L2)	Apply (L3)	Analyze (L4)	Evaluate (L5)	Create (L6)
A8703.1		✓				
A8703.2			✓			
A8703.3				✓		
A8703.4			✓			
A8703.5					✓	

COURSE ARTICULATION MATRIX

CO#/ POs	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12	PSO1	PSO2
A8703.1	3												2	2
A8703.2		3	3										3	3
A8703.3			3										3	2
A8703.4					3								3	2
A8703.5		3											3	3

Note: 1-Low, 2-Medium, 3-High

COURSE ASSESSMENT

S No	Component		Duration in Hours	Component Wise Marks	Total Marks	Weight age	Marks
1	Continuous Internal Evaluationn (CIE)	Theory: CAT-I	2	30	100	0.3	30
2		Theory: CAT-II	2	30			
3		Practical: CAT-I	2	10			
4		Practical: CAT-II	2	10			
5		Alternate Assessment*	-	20			
6		Practical Exam	2	20			
7	Semester End Exam (SEE)		3	100	70	0.7	70
Total Marks							**100**

LIST OF PROGRAMS FOR PRACTICE

S.NO	TITLE OF THE EXPERIMENT	TOOLS AND TECHNIQUES	EXPECTED SKILLS /ABILITY
1	Demonstrate the Data Pre-Processing techniques by taking real datasets.	Open source Tools: JDK 8 and above versions, Weka 3.8 above, Anaconda Python, Spider, Jupyter Notebook, pycharm IDE: Pycharm OS: Windows / Linux Web browser: Internet Explorer/ Google/ Firefox	Apply Machine Learning principles to Analyze the given data
2	Demonstrate Feature subset selection and implement dimensionality reduction (PCA) technique.		
3	Write a program to demonstrate the working of the decision tree-based ID3 algorithm. Use an appropriate data set for building the decision tree and apply this knowledge to classify a new sample.		
4	Write a program to implement the naïve Bayesian classifier for a sample training data set stored as a .CSV file. Compute the accuracy of the classifier, considering a few test data sets.		
5	Write a program to implement the SVM classifier for a sample training data set stored as a .CSV file.		
6	Write a program to implement the K-NN classifier for a sample training data set stored as a .CSV file. Compute the accuracy of the classifier.		
7	Write a program to construct a Bayesian network considering medical data. Use this model to demonstrate the diagnosis of heart patients using the standard Heart Disease Data Set.		
8	Implement a linear Regression algorithm in order to fit data points. Select the appropriate data set for your experiment and draw graphs.		
9	Implement multiple linear Regression algorithms in order to fit data points. Select the appropriate data set for your experiment and draw graphs		
10	Write a program to implement the K-Means Clustering algorithm and find the SSE of set of clusters.		
11	Write a program to implement an agglomerative hierarchal Clustering algorithm.		

Books and Materials

TEXT BOOKS

1. G Amit Kumar Das, Saikat Dutt, Subramanian Chandra mouli, Machine Learning, Pearson India Education Services, 2019.

REFERENCE BOOKS:

1. Tom M. Mitchell, Machine Learning, McGraw Hill Education, 2013.
2. Rudolph Russell, Machine Learning: Step-by-Step Guide to Implement Machine
3. Learning Algorithms with Python, Create Space Independent Publishing Platform, 2018
4. Machine Learning Methods in the Environmental Sciences, Neural Networks, William W Hsieh, Cambridge Univ. Press.
5. Richard o. Duda, Peter E. Hart and David G. Stork, pattern classification, John Wiley & Sons Inc., 2001

JOURNALS/MAGAZINES

1. Journal of Machine Learning Research, JMLR.org
2. https://ieeexplore.ieee.org/xpl/RecentIssue.jsp?punumber=34, IEEETA, PAMI,
3. https://ieeexplore.ieee.org/xpl/RecentIssue.jsp?punumber=10207, IEEE computational Intelligence Magazine
4. https://ieeexplore.ieee.org/xpl/RecentIssue.jsp?punumber=5962385

WEEK -01

1Q: Demonstrate the Data Pre-Processing Techniques by Taking Real Datasets.

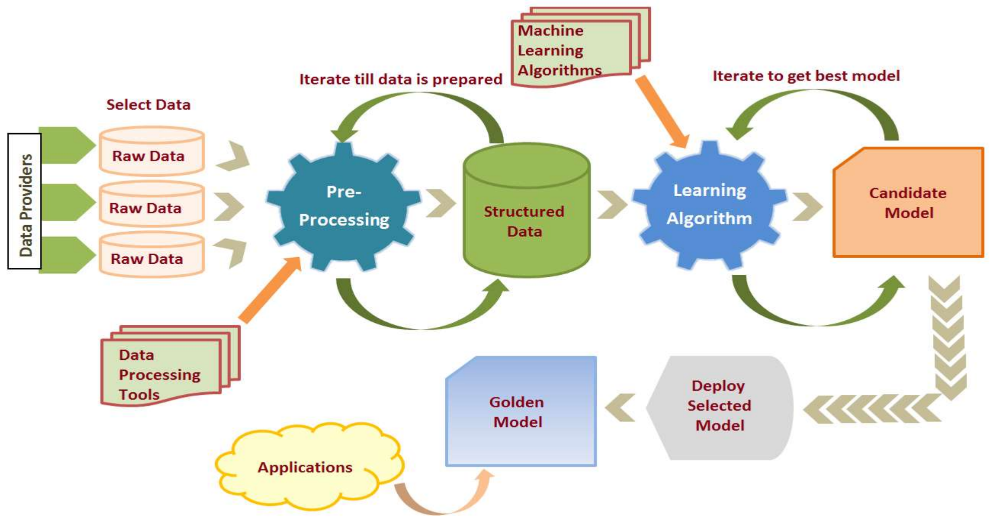
Solution:

- Data pre-processing is a process of preparing the raw data and making it suitable for a machine learning model. It is the first and crucial step while creating a machine learning model.
- When creating a machine learning project, it is not always a case that we come across the clean and formatted data. And while doing any operation with data, it is mandatory to clean it and put in a formatted way. So for this, we use data pre-processing task.

Why is Data Pre-processing Important?

Data Pre-processing is an important step in the Data Preparation stage of a Data Science development lifecycle that will ensure reliable, robust, and consistent results. The main objective of this step is to ensure and check the quality of data before applying any Machine Learning or Data Mining methods. Let's review some of its benefits -

1. **Accuracy -** Data Pre-processing will ensure that input data is accurate and reliable by ensuring there are no manual entry errors, no duplicates, etc.
2. **Completeness -** It ensures that missing values are handled, and data is complete for further analysis.
3. **Consistent -** Data Pre-processing ensures that input data is consistent, i.e., the same data kept in different places should match.
4. **Timeliness -** Whether data is updated regularly and on a timely basis or not.
5. **Trustable -** Whether data is coming from trustworthy sources or not.
6. **Interpretability -** Raw data is generally unusable, and Data Pre-processing converts raw data into an interpretable format.



It involves below steps:

1. Getting the dataset
2. Importing libraries
3. Importing datasets
4. Finding Missing Data
5. Encoding Categorical Data
6. Splitting dataset into training and test set
7. Feature scaling

Program Source Code

Step 1: Importing Libraries and the Dataset

```
# importing libraries
import pandas as pd
import scipy
import numpy as np
from sklearn.preprocessing import MinMaxScaler
import seaborn as sns
import matplotlib.pyplot as plt
```

Step 2: Load the dataset

Dataset link: [https://www.kaggle.com/datasets/uciml/pima-indians-diabetes-database]

# Load the dataset

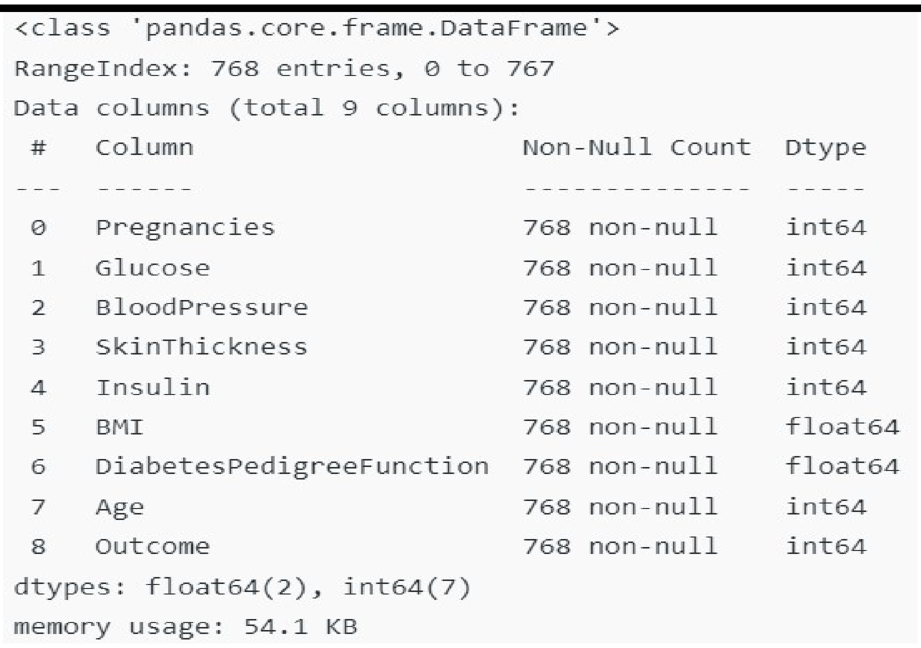
1. df = pd.read_csv('diabetes.csv')
2. print(df.head())

output:

```
   Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin   BMI
0            6      148             72             35        0  33.6
1            1       85             66             29        0  26.6
2            8      183             64              0        0  23.3
3            1       89             66             23       94  28.1
4            0      137             40             35      168  43.1

   DiabetesPedigreeFunction  Age  Outcome
0                     0.627   50        1
1                     0.351   31        0
2                     0.672   32        1
3                     0.167   21        0
4                     2.288   33        1
```

Check the data info

df.info( )

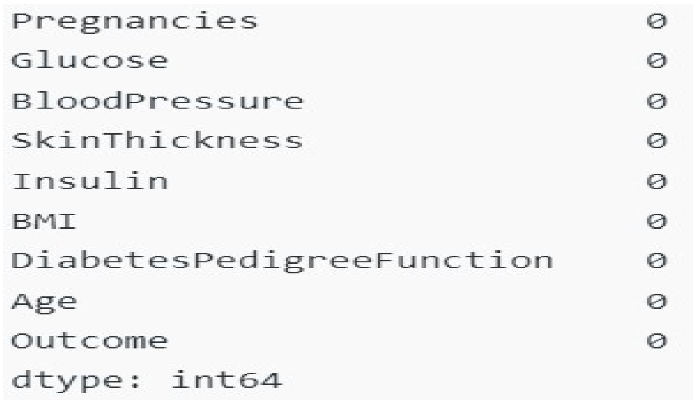


```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 768 entries, 0 to 767
Data columns (total 9 columns):
 #   Column                    Non-Null Count  Dtype  
---  ------                    --------------  -----  
 0   Pregnancies               768 non-null    int64  
 1   Glucose                   768 non-null    int64  
 2   BloodPressure             768 non-null    int64  
 3   SkinThickness             768 non-null    int64  
 4   Insulin                   768 non-null    int64  
 5   BMI                       768 non-null    float64
 6   DiabetesPedigreeFunction  768 non-null    float64
 7   Age                       768 non-null    int64  
 8   Outcome                   768 non-null    int64  
dtypes: float64(2), int64(7)
memory usage: 54.1 KB
```

As we can see from the above info that, the dataset has 9 columns and each columns has 768 values. There is no Null values in the dataset.

We can also check the null values using df.isnull( )

df.isnull( ).sum( )

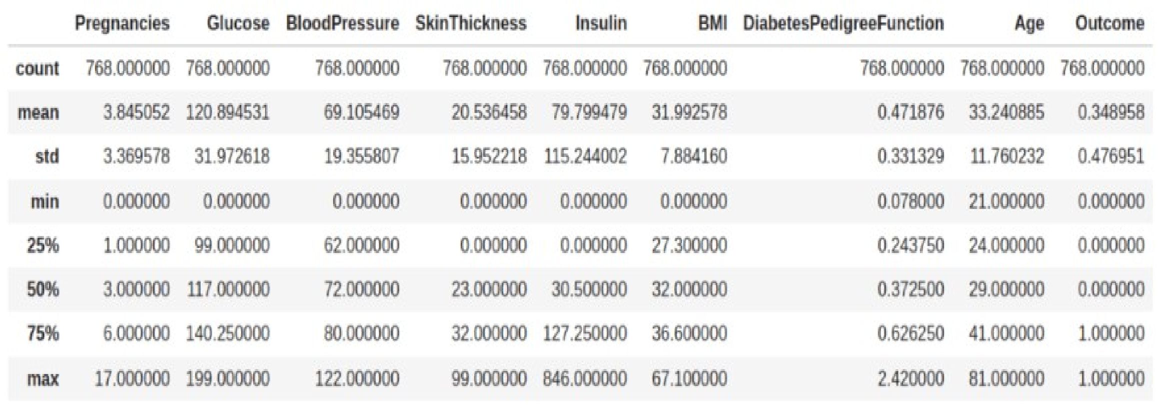


```
Pregnancies                 0
Glucose                     0
BloodPressure               0
SkinThickness               0
Insulin                     0
BMI                         0
DiabetesPedigreeFunction    0
Age                         0
Outcome                     0
dtype: int64
```

Step 3: Statistical Analysis

- In statistical analysis, first, we use the df.describe() which will give a descriptive overview of the dataset.

df.describe( )



	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
count	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000
mean	3.845052	120.894531	69.105469	20.536458	79.799479	31.992578	0.471876	33.240885	0.348958
std	3.369578	31.972618	19.355807	15.952218	115.244002	7.884160	0.331329	11.760232	0.476951
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.078000	21.000000	0.000000
25%	1.000000	99.000000	62.000000	0.000000	0.000000	27.300000	0.243750	24.000000	0.000000
50%	3.000000	117.000000	72.000000	23.000000	30.500000	32.000000	0.372500	29.000000	0.000000
75%	6.000000	140.250000	80.000000	32.000000	127.250000	36.600000	0.626250	41.000000	1.000000
max	17.000000	199.000000	122.000000	99.000000	846.000000	67.100000	2.420000	81.000000	1.000000

Data summary

Note: The above table shows the count, mean, standard deviation, min, 25%, 50%, 75%, and max values for each column. When we carefully observe the table we will find that. Insulin, Pregnancies, BMI, Blood Pressure columns has outliers.

- Let's plot the boxplot for each column for easy understanding.

Step 4: Normalization or Standardization

- Normalization
- MinMaxScaler scales the data so that each feature is in the range [0, 1].
- It works well when the features have different scales and the algorithm being used is sensitive to the scale of the features, such as k-nearest neighbors or neural networks.
- Rescale your data using scikit-learn using the MinMaxScaler.

# Initialising the MinMaxScaler

scaler = MinMaxScaler(feature_range=(0, 1))

# learning the statistical parameters for each of the data and transforming

rescaledX = scaler.fit_transform(X)

rescaledX[:5]

output

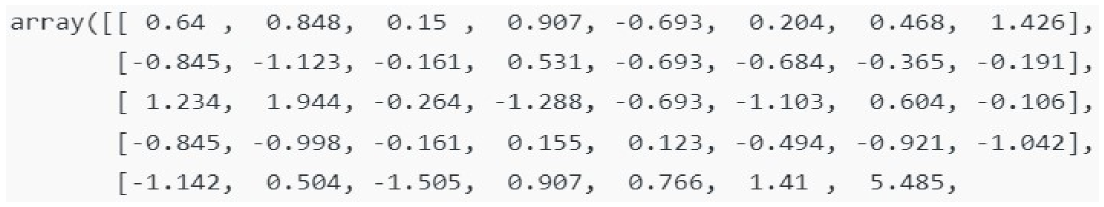
```
array([[0.353, 0.744, 0.59 , 0.354, 0.   , 0.501, 0.234, 0.483],
       [0.059, 0.427, 0.541, 0.293, 0.   , 0.396, 0.117, 0.167],
       [0.471, 0.92 , 0.525, 0.   , 0.   , 0.347, 0.254, 0.183],
       [0.059, 0.447, 0.541, 0.232, 0.111, 0.419, 0.038, 0.   ],
       [0.   , 0.688, 0.328, 0.354, 0.199, 0.642, 0.944, 0.2  ]])
```

Standardization

- Standardization is a useful technique to transform attributes with a Gaussian distribution and differing means and standard deviations to a standard Gaussian distribution with a mean of 0 and a standard deviation of 1.
- We can standardize data using scikit-learn with the StandardScaler class.

- It works well when the features have a normal distribution or when the algorithm being used is not sensitive to the scale of the features

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler().fit(X)
rescaledX = scaler.transform(X)
rescaledX[:5]
```



```
array([[ 0.64 ,  0.848,  0.15 ,  0.907, -0.693,  0.204,  0.468,  1.426],
       [-0.845, -1.123, -0.161,  0.531, -0.693, -0.684, -0.365, -0.191],
       [ 1.234,  1.944, -0.264, -1.288, -0.693, -1.103,  0.604, -0.106],
       [-0.845, -0.998, -0.161,  0.155,  0.123, -0.494, -0.921, -1.042],
       [-1.142,  0.504, -1.505,  0.907,  0.766,  1.41 ,  5.485,
```

WEEK -02

2Q: Demonstrate Feature subset selection and implement dimensionality reduction (PCA) technique.

SOLUTION:

Principal Component Analysis is an unsupervised learning algorithm that is used for the dimensionality reduction in machine learning. It is a statistical process that converts the observations of correlated features into a set of linearly uncorrelated features with the help of orthogonal transformation. These new transformed features are called the Principal Components. It is one of the popular tools that is used for exploratory data analysis and predictive modelling. It is a technique to draw strong patterns from the given dataset by reducing the variances.

The PCA algorithm is based on some mathematical concepts such as:

1. Variance and Covariance
2. Eigenvalues and Eigen factors

Some common terms used in PCA algorithm:

1. **Dimensionality:** It is the number of features or variables present in the given dataset. More easily, it is the number of columns present in the dataset.
2. **Correlation:** It signifies that how strongly two variables are related to each other. Such as if one changes, the other variable also gets changed. The correlation value ranges from -1 to +1. Here, -1 occurs if variables are inversely proportional to each other, and +1 indicates that variables are directly proportional to each other.
3. **Orthogonal:** It defines that variables are not correlated to each other, and hence the correlation between the pair of variables is zero.
4. **Eigenvectors:** If there is a square matrix M, and a non-zero vector v is given. Then v will be eigenvector if Av is the scalar multiple of v.
5. **Covariance Matrix:** A matrix containing the covariance between the pair of variables is called the Covariance Matrix.

Steps for PCA algorithm

1. **Getting the dataset**
    - Firstly, we need to take the input dataset and divide it into two subparts X and Y, where X is the training set, and Y is the validation set.
2. **Representing data into a structure**
    - Now we will represent our dataset into a structure. Such as we will represent the two-dimensional matrix of independent variable X. Here each row corresponds to the data items, and the column corresponds to the Features. The number of columns is the dimensions of the dataset.

3. **Standardizing the data**
    - In this step, we will standardize our dataset. Such as in a particular column, the features with high variance are more important compared to the features with lower variance.
    - If the importance of features is independent of the variance of the feature, then we will divide each data item in a column with the standard deviation of the column. Here we will name the matrix as Z.
4. **Calculating the Covariance of Z**
    - To calculate the covariance of Z, we will take the matrix Z, and will transpose it. After transpose, we will multiply it by Z. The output matrix will be the Covariance matrix of Z.
5. **Calculating the Eigen Values and Eigen Vectors**
    - Now we need to calculate the eigenvalues and eigenvectors for the resultant covariance matrix Z. Eigenvectors or the covariance matrix are the directions of the axes with high information. And the coefficients of these eigenvectors are defined as the eigenvalues.
6. **Sorting the Eigen Vectors**
    - In this step, we will take all the eigenvalues and will sort them in decreasing order, which means from largest to smallest. And simultaneously sort the eigenvectors accordingly in matrix P of eigenvalues. The resultant matrix will be named as P*.
7. **Calculating the new features Or Principal Components**
    - Here we will calculate the new features. To do this, we will multiply the P* matrix to the Z. In the resultant matrix Z*, each observation is the linear combination of original features. Each column of the Z* matrix is independent of each other.
8. **Remove less or unimportant features from the new dataset.**
    - The new feature set has occurred, so we will decide here what to keep and what to remove. It means, we will only keep the relevant or important features in the new dataset, and unimportant features will be removed out.

Applications of Principal Component Analysis

- PCA is mainly used as the dimensionality reduction technique in various AI applications such as computer vision, image compression, etc.
- It can also be used for finding hidden patterns if data has high dimensions. Some fields where PCA is used are Finance, data mining, Psychology, etc.

Step 1: Import Libraries and Load Data
Step 2: Standardize the Data
Step 3: Compute Covariance Matrix
Step 4: Compute Eigenvectors and Eigenvalues
Step 5: Sort Eigenvalues in Descending Order
Step 6: Choose Principal Components
Step 7: Project Data Onto Lower-Dimensional Linear Subspace

STEP 1: IMPORT LIBRARIES AND LOAD DATA

The first step is to import the necessary libraries and load the data that you want to perform PCA on. For this example, we will use the Iris dataset, which is a commonly used dataset in machine learning.

1. import numpy as np
2. from sklearn.datasets import load_iris
3. iris = load_iris()
4. X = iris.data
5. y = iris.target

NOTE: Here, we import NumPy for numerical computing and load_iris from Scikit-Learn's datasets module to get the Iris dataset. We then assign the feature values to X and target values to y.

STEP 2: STANDARDIZE THE DATA

PCA is sensitive to the scale of the input data, so it's important to standardize the data before performing PCA. Standardization involves scaling the data so that it has a mean of 0 and a standard deviation of 1.

6. from sklearn.preprocessing import StandardScaler
7. scaler = StandardScaler()
8. X_scaled = scaler.fit_transform(X)

NOTE: Here, we import StandardScaler from Scikit-Learn's preprocessing module and create an instance of it. We then fit and transform the feature values in X to get X_scaled.

STEP 3: COMPUTE COVARIANCE MATRIX

The next step is to compute the covariance matrix of the standardized data. The covariance matrix represents the relationships between the different features in the data.

9. cov_matrix = np.cov(X_scaled.T)

NOTE: Here, we use NumPy's cov function to compute the covariance matrix of X_scaled after transposing it.

STEP 4: COMPUTE EIGENVECTORS AND EIGENVALUES

The eigenvectors and eigenvalues of the covariance matrix are used to determine the principal components of the data. The eigenvectors represent the directions of maximum variance in the data, while the corresponding eigenvalues represent the amount of variance explained by each eigenvector.

10. eigen_values, eigen_vectors = np.linalg.eig(cov_matrix)

NOTE: Here, we use NumPy’s linalg.eig function to compute both the eigenvalues and eigenvectors of cov_matrix.

STEP 5: SORT EIGENVALUES IN DESCENDING ORDER

The next step is to sort the eigenvalues in descending order. This will allow us to choose the principal components that explain the most variance in the data.

```
sorted_index = np.argsort(eigen_values)[::-1]
sorted_eigenvalue = eigen_values[sorted_index]
sorted_eigenvectors = eigen_vectors[:,sorted_index]
```

NOTE: Here, we use NumPy’s argsort function to get the indices that would sort the eigenvalues in ascending order. We then reverse the order using [::-1] and use it to sort both the eigenvalues and eigenvectors.

STEP 6: CHOOSE PRINCIPAL COMPONENTS

The next step is to choose the principal components that we want to keep. We can do this by Selecting the top k eigenvectors that correspond to the k largest eigenvalues.

```
k = 2
principal_components = sorted_eigenvectors[:,:k]
```

NOTE: Here, we choose k=2 as an example and select the first two eigenvectors from sorted_eigenvectors to be our principal components.

STEP 7: PROJECT DATA ONTO LOWER-DIMENSIONAL LINEAR SUBSPACE

The final step is to project the data onto the lower-dimensional linear subspace defined by the principal components. This will transform the data from its original high-dimensional space into a lower-dimensional space while retaining as much of the original information as possible.

```
X_new = np.dot(X_scaled, principal_components)
```

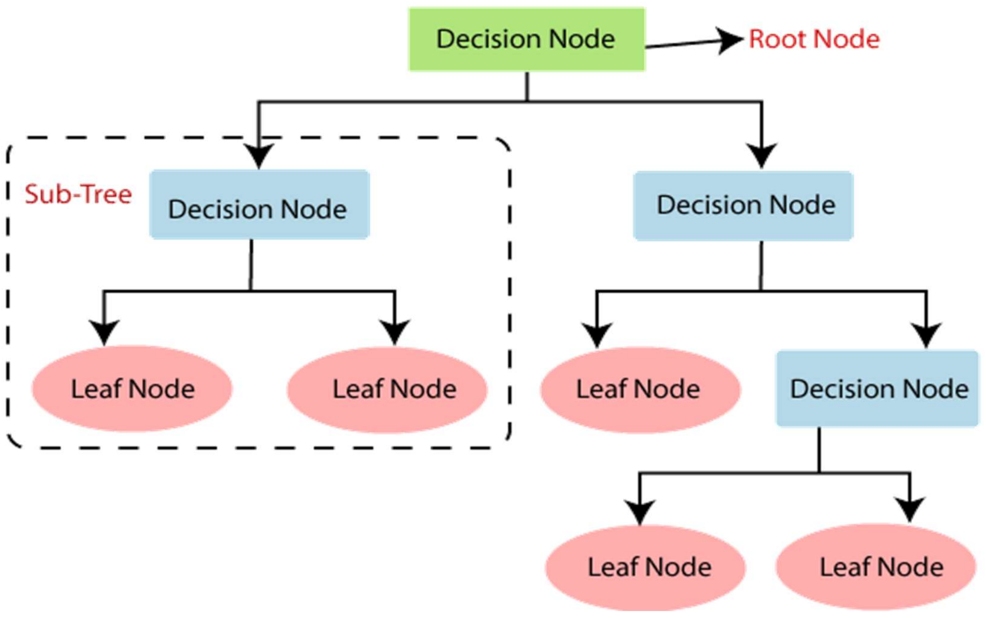
NOTE: Here, we use NumPy’s dot function to compute the dot product between X_scaled and principal_components to obtain X_new, which is our transformed data.

WEEK -03

3Q: Write a program to demonstrate the working of the Decision Tree-Based Id3 algorithm. Use an appropriate data set for building the decision tree and apply this knowledge to classify a new sample.

Solution:

1. Decision Tree is a supervised learning technique that can be used for both classification and Regression problems, but mostly it is preferred for solving Classification problems. It is a tree-structured classifier, where internal nodes represent the features of a dataset, branches represent the decision rules and each leaf node represents the outcome.
2. In a Decision tree, there are two nodes, which are the Decision Node and Leaf Node. Decision nodes are used to make any decision and have multiple branches, whereas Leaf nodes are the output of those decisions and do not contain any further branches.
3. The decisions or the test are performed on the basis of features of the given dataset.
4. It is a graphical representation for getting all the possible solutions to a problem/decision based on given conditions.
5. It is called a decision tree because, similar to a tree, it starts with the root node, which expands on further branches and constructs a tree-like structure.
6. In order to build a tree, we use the CART algorithm, which stands for Classification and Regression Tree algorithm.
7. A decision tree simply asks a question, and based on the answer (Yes/No), it further split the tree into subtrees.



How ID3(Iterative Dichotomiser 3) Works:

The ID3 algorithm is specifically designed for building decision trees from a given dataset. Its primary objective is to construct a tree that best explains the relationship between attributes in the data and their corresponding class labels.

1. Selecting the Best Attribute

- ID3 employs the concept of entropy and information gain to determine the attribute that best separates the data. Entropy measures the impurity or randomness in the dataset.
- The algorithm calculates the entropy of each attribute and selects the one that results in the most significant information gain when used for splitting the data.

2. Creating Tree Nodes

- The chosen attribute is used to split the dataset into subsets based on its distinct values.
- For each subset, ID3 recurses to find the next best attribute to further partition the data, forming branches and new nodes accordingly.

3. Stopping Criteria

- The recursion continues until one of the stopping criteria is met, such as when all instances in a branch belong to the same class or when all attributes have been used for splitting.

4. Handling Missing Values

- ID3 can handle missing attribute values by employing various strategies like attribute mean/mode substitution or using majority class values.

5. Tree Pruning

- Pruning is a technique to prevent overfitting. While not directly included in ID3, post-processing techniques or variations like C4.5 incorporate pruning to improve the tree's generalization.

Why use Decision Trees?

Decision trees are popular in machine learning for several reasons:

Advantages:

1. **Interpretability:** Decision trees mimic human decision-making processes and are easy to understand and interpret. The tree structure is intuitive, making it simple to visualize and explain to non-experts. Each decision rule can be easily traced back to the features of the data.
2. **Feature Importance:** Decision trees provide a natural ranking of features based on their importance in predicting the target variable. Features that appear higher in the tree (closer to the root) are more important in making decisions.

3. **Handling Non-linearity and Interactions:** Decision trees can capture non-linear relationships between features and the target variable. They can also model interactions between features, which can be challenging for linear models.
4. **Handling Mixed Data Types:** Decision trees can handle both numerical and categorical data without the need for pre-processing like normalization or one-hot encoding.
5. **Robustness to Outliers and Irrelevant Features:** Decision trees are robust to outliers and can handle noisy data well. They also automatically select relevant features and ignore irrelevant ones during the tree-building process.
6. **Scalability:** Decision trees can scale well to large datasets with many features and samples. Techniques like pruning and ensemble methods (e.g., Random Forests) can further improve performance and scalability.
7. **No Assumptions about Data Distribution:** Decision trees make no assumptions about the underlying distribution of the data. They can perform well even if the data distribution is unknown or complex.
8. **Easy to Handle Missing Values:** Decision trees can handle missing values in the data without requiring imputation. They simply make decisions based on the available information for each instance.
9. **Versatility:** Decision trees can be used for both classification and regression tasks. Various extensions and ensemble methods, such as Random Forests, Gradient Boosting Machines, and XG Boost, build upon decision trees to further enhance their performance.

Disadvantages and Limitations:

1. **Overfitting:** Decision trees are prone to overfitting, especially with deep trees. They can memorize noise in the training data, leading to poor generalization performance on unseen data.
2. **High Variance:** Small changes in the training data can lead to significantly different decision trees. This high variance makes decision trees unstable compared to other algorithms.
3. **Bias towards Features with Many Levels:** Decision trees tend to bias towards features with a large number of levels. Features with many levels may appear more informative than they actually are.
4. **Difficulty in Learning XOR, Parity, and Multiplexer Problems:** Decision trees struggle to learn certain logical functions like XOR, parity, and multiplexer problems, which require complex decision boundaries.

5. **Not Suitable for Regression on Continuous Variables:** While decision trees can handle regression tasks, they may not perform well when the target variable has a high degree of continuity.
6. **Inability to Extrapolate:** Decision trees cannot extrapolate beyond the range of values seen in the training data. They may perform poorly when applied to data outside the training range.
7. **Greedy Nature:** Decision trees use a greedy approach to select attributes at each node, which may not always result in the optimal tree structure.
8. **Difficulty with Imbalanced Classes:** Decision trees may produce biased trees when dealing with imbalanced class distributions. Techniques like class weighting or resampling may be needed to address this issue.

Let's examine the formulas linked to the main theoretical ideas in the ID3 algorithm:

1. Entropy

- A measure of disorder or uncertainty in a set of data is called entropy. Entropy is a tool used in ID3 to measure a dataset's disorder or impurity. By dividing the data into as homogenous subsets as feasible, the objective is to minimize entropy.
- For a set S with classes {c1, c2, …, cn}, the entropy is calculated as:

$$H(S) = \Sigma_{i=1}^{n} p_i log_2(p_i)$$

2. Information Gain

- A measure of how well a certain quality reduces uncertainty is called Information Gain. ID3 splits the data at each stage, choosing the property that maximizes Information Gain. It is computed using the distinction between entropy prior to and following the split.
- Information Gain measures the effectiveness of an attribute A in reducing uncertainty in set S.

$$IG(A,S) = H(S) - \Sigma_{v \epsilon values(A)} \frac{|S_v|}{|S|} \cdot H(S_v)$$

- Where, |Sv | is the size of the subset of S for which attribute A has value v.

3. Gain Ratio

- Gain Ratio is an improvement on Information Gain that considers the inherent worth of characteristics that have a wide range of possible values. It deals with the bias of Information Gain in favour of characteristics with more pronounced values.

$$GR(A,S) = \frac{IG(A,S)}{\Sigma_{v \epsilon values(A)} \frac{|S_v|}{S} \cdot log_2(\frac{|S_v|}{|S|})}$$

Program Source Code:

```
import numpy as np
from collections import Counter
class Node:
    def __init__(self, attribute=None):
        self.attribute = attribute
        self.children = {}
        self.value = None  # For leaf nodes

def entropy(y):
    class_counts = Counter(y)
    entropy_val = 0
    total_samples = len(y)
    for count in class_counts.values():
        p = count / total_samples
        entropy_val -= p * np.log2(p)
    return entropy_val

def information_gain(X, y, attribute):
    total_entropy = entropy(y)
    unique_values = np.unique(X[:, attribute])
    weighted_entropy = 0
    for value in unique_values:
        subset_indices = np.where(X[:, attribute] == value)[0]
        subset_entropy = entropy(y[subset_indices])
        weight = len(subset_indices) / len(y)
        weighted_entropy += weight * subset_entropy
    return total_entropy - weighted_entropy

def id3(X, y, attributes):
    if len(set(y)) == 1:
        leaf = Node()
        leaf.value = y[0]
        return leaf
    if len(attributes) == 0:
        leaf = Node()
        leaf.value = Counter(y).most_common(1)[0][0]
        return leaf
    gains = [information_gain(X, y, attribute) for attribute in attributes]
    best_attribute = attributes[np.argmax(gains)]
    node = Node(best_attribute)
    unique_values = np.unique(X[:, best_attribute])
    for value in unique_values:
        subset_indices = np.where(X[:, best_attribute] == value)[0]
        subset_X = X[subset_indices]
        subset_y = y[subset_indices]
        if len(subset_y) == 0:
            leaf = Node()
            leaf.value = Counter(y).most_common(1)[0][0]
            node.children[value] = leaf
        else:
            node.children[value] = id3(subset_X, subset_y, np.setdiff1d(attributes, [best_attribute]))
    return node

def print_tree(node, depth=0):
    if node.attribute is None:
        print('  ' * depth, 'Predict:', node.value)
    else:
        print('  ' * depth, 'Attribute', node.attribute)
        for value, child_node in node.children.items():
            print('  ' * (depth + 1), 'Value', value)
            print_tree(child_node, depth + 2)

# Example usage:
X = np.array([
    ['Sunny', 'Hot', 'High', 'Weak'],
    ['Sunny', 'Hot', 'High', 'Strong'],
    ['Overcast', 'Hot', 'High', 'Weak'],
    ['Rain', 'Mild', 'High', 'Weak'],
    ['Rain', 'Cool', 'Normal', 'Weak'],
    ['Rain', 'Cool', 'Normal', 'Strong'],
    ['Overcast', 'Cool', 'Normal', 'Strong'],
    ['Sunny', 'Mild', 'High', 'Weak'],
    ['Sunny', 'Cool', 'Normal', 'Weak'],
    ['Rain', 'Mild', 'Normal', 'Weak'],
    ['Sunny', 'Mild', 'Normal', 'Strong'],
    ['Overcast', 'Mild', 'High', 'Strong'],
    ['Overcast', 'Hot', 'Normal', 'Weak'],
    ['Rain', 'Mild', 'High', 'Strong']
])
y = np.array(['No', 'No', 'Yes', 'Yes', 'Yes', 'No', 'Yes', 'No', 'Yes', 'Yes', 'Yes', 'Yes', 'Yes', 'No'])
attributes = [0, 1, 2, 3]  # Indices of attributes
tree = id3(X, y, attributes)
print_tree(tree)
```

WEEK -04

4Q: Write a program to implement the naïve Bayesian classifier for a sample training data set stored as a .CSV file. Compute the accuracy of the classifier, considering a few test data sets.

Solution:

1. Naïve Bayes algorithm is a supervised learning algorithm, which is based on Bayes theorem and used for solving classification problems.
2. It is mainly used in text classification that includes a high-dimensional training dataset.
3. Naïve Bayes Classifier is one of the simple and most effective Classification algorithms which helps in building the fast machine learning models that can make quick predictions.
4. It is a probabilistic classifier, which means it predicts on the basis of the probability of an object.
5. Some popular examples of Naïve Bayes Algorithm are spam filtration, Sentimental analysis, and classifying articles.

Bayes' Theorem:

Bayes' theorem is also known as Bayes' Rule or Bayes' law, which is used to determine the probability of a hypothesis with prior knowledge. It depends on the conditional probability.

The formula for Bayes' theorem is given as:

$$P(A|B) = \frac{P(B|A)\,P(A)}{P(B)}$$

1. P(A|B) is Posterior probability: Probability of hypothesis A on the observed event B.
2. P(B|A) is Likelihood probability: Probability of the evidence given that the probability of a hypothesis is true.
3. P(A) is Prior Probability: Probability of hypothesis before observing the evidence.
4. P(B) is Marginal Probability: Probability of Evidence.

Naïve Bayesian classifier Advantages:

1. **Simplicity:** Naive Bayes is simple and easy to understand, making it suitable for beginners and as a baseline classifier.
2. **Efficiency:** It is computationally efficient and scales well to large datasets, making it suitable for real-time applications and big data scenarios.
3. **Fast Training:** Naive Bayes requires minimal training data compared to more complex algorithms, and the training process is fast.

4. **Handles High-Dimensional Data:** It performs well in high-dimensional feature spaces, such as text classification, where the number of features (words) can be large.
5. **Handles Irrelevant Features:** Naive Bayes can handle irrelevant features well due to its independence assumption. It effectively ignores irrelevant features during classification.
6. **Robust to Overfitting:** Despite its simplicity, Naive Bayes is less prone to overfitting, especially with large datasets, due to its strong regularization.
7. **Works Well with Imbalanced Data:** Naive Bayes can handle imbalanced class distributions effectively, as it computes probabilities based on class frequencies.
8. **Interpretability:** Predictions made by Naive Bayes are easy to interpret, as they are based on probabilities and conditional probabilities.

Disadvantages:

1. **Strong Independence Assumption:** The "naive" assumption of feature independence may not hold in real-world datasets. This can lead to suboptimal performance, especially if features are correlated.
2. **Inability to Capture Interactions:** Naive Bayes cannot capture interactions between features, as it assumes independence. This may limit its performance on tasks where feature interactions are important.
3. **Sensitivity to Feature Distribution:** Naive Bayes can be sensitive to the distribution of features. It assumes that features follow a specific distribution (e.g., Gaussian, multinomial), which may not always hold true.
4. **Poor Estimation of Probabilities:** In cases where a feature-value combination is not present in the training data, Naive Bayes assigns zero probability, leading to poor estimation. Laplace smoothing can help mitigate this issue.
5. **Limited Expressiveness:** Naive Bayes has limited expressiveness compared to more complex models like decision trees or neural networks. It may not capture complex patterns in the data effectively.
6. **Cannot Handle Continuous Features Well:** While Naive Bayes can handle categorical and discrete features effectively, it may not perform well with continuous features without appropriate discretization techniques.
7. **Requires Balanced Training Data:** While Naive Bayes can handle imbalanced class distributions, extreme class imbalances may still lead to biased predictions.

Program Source Code

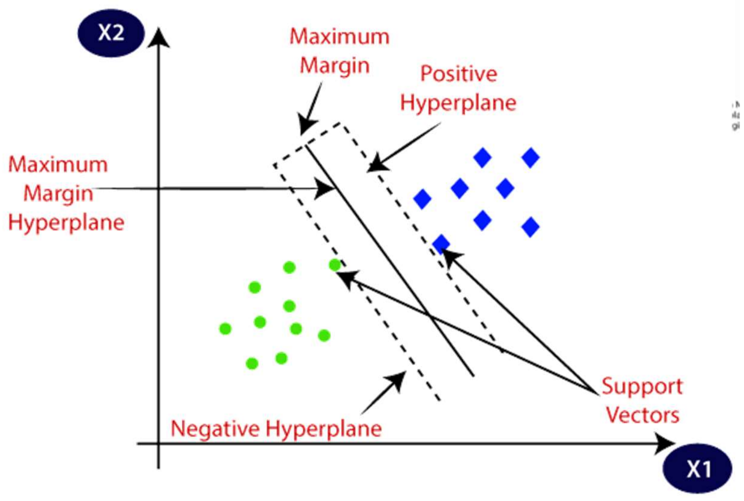
```
# load the iris dataset
from sklearn.datasets import load_iris
iris = load_iris()
# store the feature matrix (X) and response vector (y)
X = iris.data
y = iris.target
# splitting X and y into training and testing sets
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.4,
random_state=1)
# training the model on training set
from sklearn.naive_bayes import GaussianNB
gnb = GaussianNB()
gnb.fit(X_train, y_train)
# making predictions on the testing set
y_pred = gnb.predict(X_test)
# comparing actual response values (y_test) with predicted response values (y_pred)
from sklearn import metrics
print("Gaussian Naive Bayes model accuracy(in %):", metrics.accuracy_score(y_test,
y_pred)*100)
OUTPUT:
Gaussian Naive Bayes model accuracy(in %): 95.0
```

WEEK -05

5Q: Write a program to implement the SVM classifier for a sample training data set stored as a .CSV file.

Solution:

- 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:



Types of SVM

SVM can be of two types:

1. **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.
2. **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:

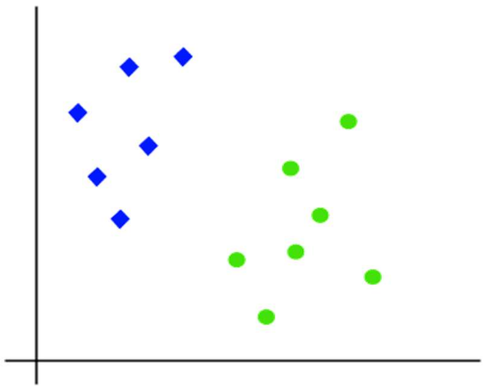
1. **Hyperplane:** There can be multiple lines/decision boundaries to segregate the classes in n-dimensional space, but we need to find out the best decision boundary that helps to classify the data points. This best boundary is known as the hyperplane of SVM.

2. 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.
3. We always create a hyperplane that has a maximum margin, which means the maximum distance between the data points.

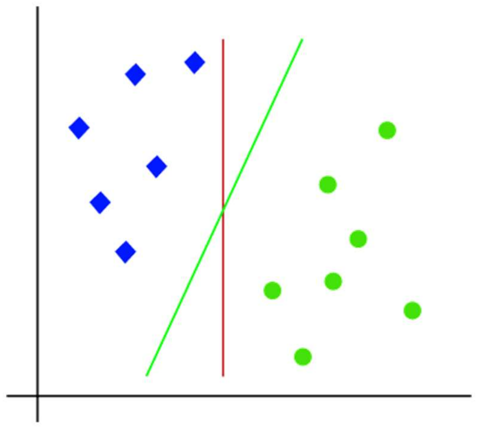
How does SVM works?

Linear SVM:

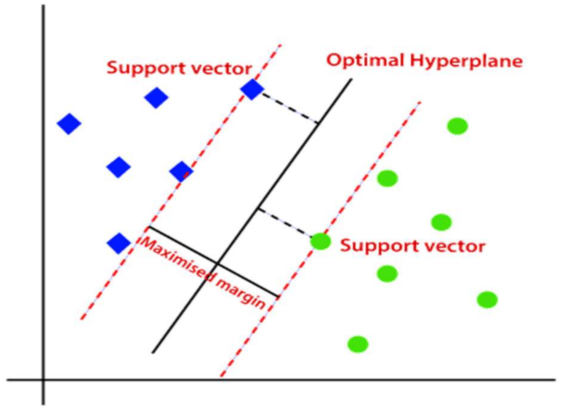
The working of the SVM algorithm can be understood by using an example. Suppose we have a dataset that has two tags (green and blue), and the dataset has two features x1 and x2. We want a classifier that can classify the pair(x1, x2) of coordinates in either green or blue. Consider the below image:



So as it is 2-d space so by just using a straight line, we can easily separate these two classes. But there can be multiple lines that can separate these classes. Consider the below image:

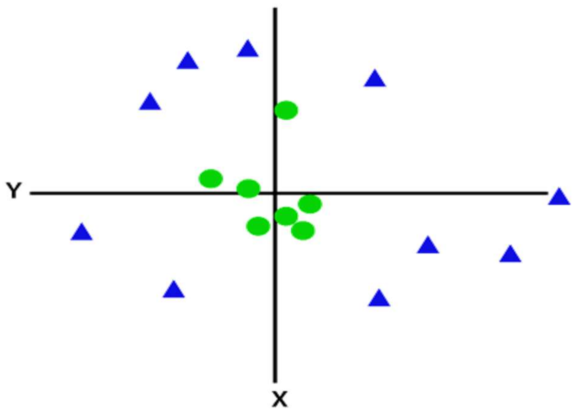


Hence, the SVM algorithm helps to find the best line or decision boundary; this best boundary or region is called as a hyperplane. SVM algorithm finds the closest point of the lines from both the classes. These points are called support vectors. The distance between the vectors and the hyperplane is called as margin. And the goal of SVM is to maximize this margin. The hyperplane with maximum margin is called the optimal hyperplane.



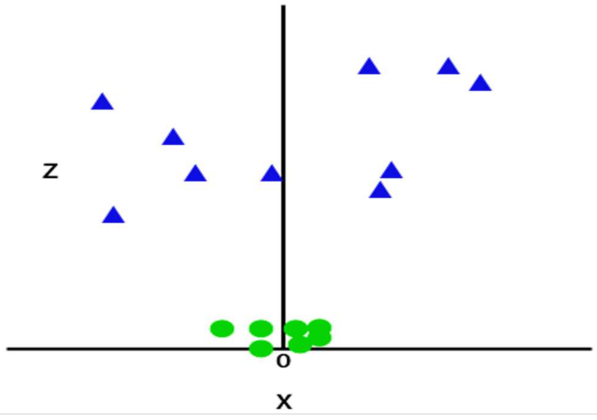
Non-Linear SVM:

If data is linearly arranged, then we can separate it by using a straight line, but for non-linear data, we cannot draw a single straight line. Consider the below image:

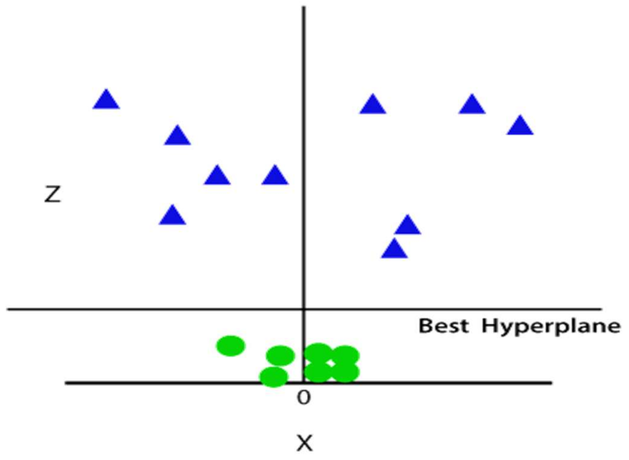


So to separate these data points, we need to add one more dimension. For linear data, we have used two dimensions x and y, so for non-linear data, we will add a third dimension z. It can be calculated as:

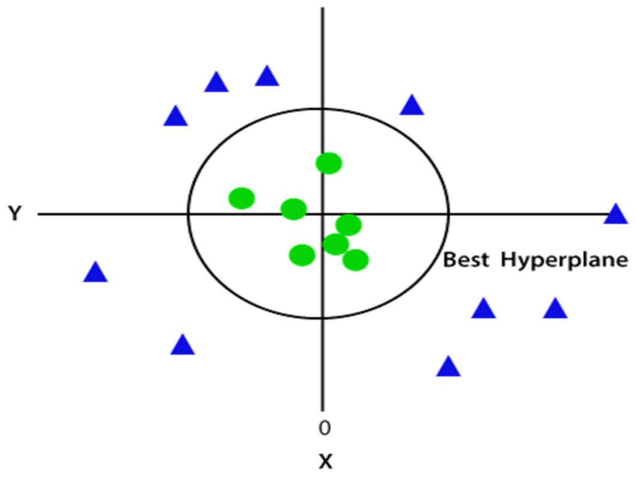
$$Z=x^2 + y^2$$



So now, SVM will divide the datasets into classes in the following way. Consider the below image



Since we are in 3-d Space, hence it is looking like a plane parallel to the x-axis. If we convert it in 2d space with z=1, then it will become as:



Program Source Code

import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score

# Load the dataset from the CSV file 	
def load_dataset(file_path):
   df = pd.read_csv(file_path)
   X = df.drop('label', axis=1) # Features
   y = df['label']   # Target variable
   return X, y

Split the dataset into training and testing sets

```
def split_dataset(X, y, test_size=0.2):
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=test_size, random_state=42)
    return X_train, X_test, y_train, y_test
```

```
# Train the SVM classifier
def train_SVM(X_train, y_train, kernel='linear'):
    svm_classifier = SVC(kernel=kernel)
    svm_classifier.fit(X_train, y_train)
    return svm_classifier
```

```
# Evaluate the trained classifier
def evaluate_classifier(classifier, X_test, y_test):
    y_pred = classifier.predict(X_test)
    accuracy = accuracy_score(y_test, y_pred)
    print("Accuracy:", accuracy)
```

```
# Main function
def main():
    # File path to the CSV dataset
    file_path = "dataset.csv"
    # Load dataset
    X, y = load_dataset(file_path)
    # Split dataset into training and testing sets
    X_train, X_test, y_train, y_test = split_dataset(X, y)
    # Train SVM classifier
    svm_classifier = train_SVM(X_train, y_train)
    # Evaluate the classifier
    evaluate_classifier(svm_classifier, X_test, y_test)
if __name__ == "__main__":
    main()
```

WEEK -06

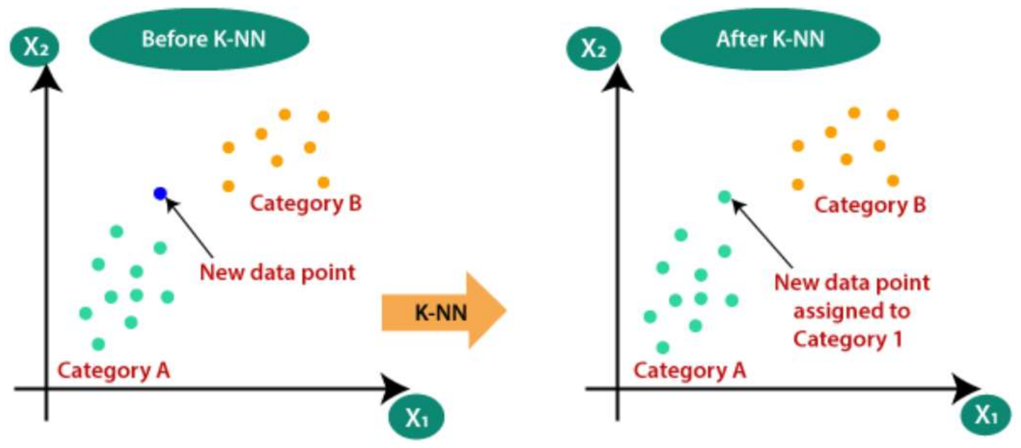
6Q: Write a program to implement the K-NN classifier for a sample training data set stored as a .CSV file. Compute the accuracy of the classifier.

SOLUTION:

1. K-Nearest Neighbour is one of the simplest Machine Learning algorithms based on Supervised Learning technique.
2. K-NN algorithm assumes the similarity between the new case/data and available cases and put the new case into the category that is most similar to the available categories.
3. K-NN algorithm stores all the available data and classifies a new data point based on the similarity. This means when new data appears then it can be easily classified into a well suite category by using K- NN algorithm.
4. K-NN algorithm can be used for Regression as well as for Classification but mostly it is used for the Classification problems.
5. K-NN is a non-parametric algorithm, which means it does not make any assumption on underlying data.
6. It is also called a lazy learner algorithm because it does not learn from the training set immediately instead it stores the dataset and at the time of classification, it performs an action on the dataset.
7. KNN algorithm at the training phase just stores the dataset and when it gets new data, then it classifies that data into a category that is much similar to the new data.

Why do we need a K-NN Algorithm?

Suppose there are two categories, i.e., Category A and Category B, and we have a new data point x1, so this data point will lie in which of these categories. To solve this type of problem, we need a K-NN algorithm. With the help of K-NN, we can easily identify the category or class of a particular dataset. Consider the below diagram:



How does K-NN work?

The K-NN working can be explained on the basis of the below algorithm:

Step-1: Select the number K of the neighbors

Step-2: Calculate the Euclidean distance of K number of neighbors

Step-3: Take the K nearest neighbors as per the calculated Euclidean distance.

Step-4: Among these k neighbors, count the number of the data points in each category.

Step-5: Assign the new data points to that category for which the number of the neighbor is maximum.

Step-6: Our model is ready.

Distance Metrics Used in KNN Algorithm

- As we know that the KNN algorithm helps us identify the nearest points or the groups for a query point. But to determine the closest groups or the nearest points for a query point we need some metric. For this purpose, we use below distance metrics:

Euclidean Distance

This is nothing but the Cartesian distance between the two points which are in the plane/hyperplane. Euclidean distance can also be visualized as the length of the straight line that joins the two points which are into consideration. This metric helps us calculate the net displacement done between the two states of an object.

$$\text{distance}(x, X_i) = \sqrt{\sum_{j=1}^{d}(x_j - X_{i_j})^2}]$$

Manhattan Distance

Manhattan Distance metric is generally used when we are interested in the total distance travelled by the object instead of the displacement. This metric is calculated by summing the absolute difference between the coordinates of the points in n-dimensions.

$$d\left(x, y\right) = \sum_{i=1}^{n} \left|x_i - y_i\right|$$

Minkowski Distance

We can say that the Euclidean, as well as the Manhattan distance, are special cases of the Minkowski distance.

Advantages of the KNN Algorithm

1. Easy to implement as the complexity of the algorithm is not that high.
2. **Adapts Easily –** As per the working of the KNN algorithm it stores all the data in memory storage and hence whenever a new example or data point is added then the algorithm adjusts itself as per that new example and has its contribution to the future predictions as well.

3. **Few Hyper parameters –** The only parameters which are required in the training of a KNN algorithm are the value of k and the choice of the distance metric which we would like to choose from our evaluation metric.

Disadvantages of the KNN Algorithm

1. **Does not scale** – As we have heard about this that the KNN algorithm is also considered a Lazy Algorithm. The main significance of this term is that this takes lots of computing power as well as data storage. This makes this algorithm both time-consuming and resource exhausting.
2. **Curse of Dimensionality –** There is a term known as the peaking phenomenon according to this the KNN algorithm is affected by the curse of dimensionality which implies the algorithm faces a hard time classifying the data points properly when the dimensionality is too high.
3. **Prone to Overfitting –** As the algorithm is affected due to the curse of dimensionality it is prone to the problem of overfitting as well. Hence generally feature selection as well as dimensionality reduction techniques are applied to deal with this problem.

Program Source Code: KNN

1.Import necessary libraries:

```
import pandas as pd
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
```

2.Load the dataset:

```
# Replace 'your_dataset.csv' with the actual path to your dataset
data = pd.read_csv('dataset.csv')
```

3.Separate features (X) and target variable (y):

```
X = data.drop('target_column', axis=1)  # Features
y = data['target_column']  # Target variable
```

4.Split 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)
```

5. Scale features (optional, but often recommended for KNN):

```
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

6.Create a KNN classifier instance:

```
knn = KNeighborsClassifier(n_neighbors=5)  # Adjust 'n_neighbors' as needed
```

7.Train the model:

```
knn.fit(X_train, y_train)
```

8.Make predictions on the test set:

```
y_pred = knn.predict(X_test)
```

9.Evaluate model performance:

```
from sklearn.metrics import accuracy_score
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
```

WEEK -07

7Q: Write a program to construct a Bayesian network considering medical data. Use this model to demonstrate the diagnosis of heart patients using the standard Heart Disease Data Set.

Solution:

A Bayesian network, also known as a Bayes network, is a probabilistic graphical model that represents a set of variables and their conditional dependencies via a directed acyclic graph (DAG). In the context of medical data, Bayesian networks are widely used for modelling complex relationships between symptoms, diseases, risk factors, and other relevant variables.

Architecture of Bayesian Network for Medical Data:

1. Nodes: Each node represents a variable, which could be a symptom, a medical condition, a risk factor, or any other relevant attribute in the medical domain. For example:
    a. Symptom nodes (e.g., fever, cough, chest pain)
    b. Disease nodes (e.g., flu, pneumonia, heart disease)
    c. Risk factor nodes (e.g., smoking, high cholesterol, family history)
2. Edges: Directed edges between nodes indicate the causal or probabilistic relationships between them. An edge from node A to node B signifies that node A has a direct influence on node B. For example:

- An edge from a risk factor node (e.g., smoking) to a disease node (e.g., lung cancer) indicates that smoking increases the risk of lung cancer.
- An edge from a symptom node (e.g., chest pain) to a disease node (e.g., heart disease) indicates that chest pain is a symptom associated with heart disease.

3. Conditional Probability Distributions (CPDs): Each node in the Bayesian network has a conditional probability distribution that quantifies the probability of that node given its parents (nodes that directly influence it). These distributions can be learned from data or specified by domain experts.

Example of Bayesian Network for Medical Diagnosis:

Let's consider a simplified example of a Bayesian network for diagnosing whether a patient has the flu based on symptoms:

1. **Nodes:**
   Symptom nodes: Fever, Cough, Fatigue, Sore throat
2. **Disease node:** Flu

3. **Edges:**
   Fever, Cough, Fatigue, and Sore throat are connected to the Flu node, indicating they are symptoms associated with the flu.
4. **Conditional Probability Distributions (CPDs):**
   The CPD for the Flu node represents the probability of having the flu given the presence or absence of symptoms.
   For example, P(Flu = True | Fever = True, Cough = True, Fatigue = True, Sore throat = False) might be higher than P(Flu = True | Fever = False, Cough = False, Fatigue = True, Sore throat = True), indicating that the presence of fever, cough, and fatigue increases the likelihood of having the flu.
5. **Bayesian Network Inference:**
   Given observed evidence (e.g., patient's symptoms), inference in a Bayesian network involves calculating the posterior probability distribution of the target variable (e.g., disease) given the evidence. This is typically done using probabilistic inference algorithms such as Variable Elimination or Belief Propagation.

While Bayesian networks offer several advantages for medical diagnosis, they also have some limitations and disadvantages:

1. **Complexity in Model Construction:** Constructing a Bayesian network for medical diagnosis requires expertise in both the medical domain and probabilistic modeling. Defining accurate relationships between variables and estimating conditional probability distributions can be challenging and time-consuming.
2. **Data Requirements:** Bayesian networks require a significant amount of data to accurately estimate the conditional probability distributions. In medical diagnosis, obtaining large and high-quality datasets may be difficult, especially for rare diseases or complex conditions.
3. **Assumption of Conditional Independence:** Bayesian networks assume conditional independence between variables given their parents in the graph. However, this assumption may not always hold true in real-world medical data, leading to model inaccuracies.
4. **Difficulty in Handling Continuous Variables:** Bayesian networks are typically designed to handle discrete variables, and handling continuous variables can be complex. Discretization of continuous variables may lead to loss of information and affect the accuracy of the model.
5. **Sensitivity to Model Structure:** The performance of a Bayesian network heavily depends on the correctness of its structure (i.e., the arrangement of nodes and edges). Small changes in the model structure can significantly impact its predictive capabilities.

6. **Limited Representation of Temporal Dynamics:** Bayesian networks are static models and may not fully capture the dynamic nature of diseases and medical conditions over time. They may struggle to represent temporal dependencies and changes in patient states accurately.
7. **Difficulty in Incorporating External Knowledge:** Integrating external domain knowledge or expert opinions into Bayesian networks can be challenging. Incorporating such knowledge requires careful consideration and may introduce biases or uncertainties into the model.
8. **Computational Complexity:** Performing inference in Bayesian networks, especially for large and complex networks, can be computationally intensive. Exact inference algorithms may become impractical for networks with many nodes and dependencies.

Program Source Code:

1.Import necessary libraries:

```
import pandas as pd
from pgmpy.models import BayesianModel
from pgmpy.estimators import MaximumLikelihoodEstimator
```

2.Load the dataset:

```
data = pd.read_csv("heart.csv")  # Replace with the actual path to your data
```

3. Define the Bayesian Network Structure:

```
model = BayesianModel([
    ('age', 'cholestrol'),
    ('age', 'lifestyle'),
    ('gender', 'lifestyle'),
    ('family', 'heartdisease'),
    ('cholestrol', 'heartdisease'),
    ('lifestyle', 'heartdisease')
])
```

4. Fit the Model with Data:

```
estimator = MaximumLikelihoodEstimator()
estimator.fit(model, data)
```

5. Inference and Diagnosis:

#Example: Diagnose a patient with age=30, gender=Male, family=No, cholestrol=High, and lifestyle=Sedentary.

```
query = {'heartdisease': True}
evidence = {'age': 30, 'gender': 'Male', 'family': 'No', 'cholestrol': 'High', 'lifestyle': 'Sedentary'}
inference = VariableElimination(model)
probability = inference.query(query, evidence)
print(f"Probability of heart disease: {probability['heartdisease'][1]:.2f}")
```

WEEK -08

8Q: Implement a linear Regression algorithm in order to fit data points. Select the appropriate data set for your experiment and draw graphs.

Solution:

Linear regression is a type of supervised machine learning algorithm that computes the linear relationship between a dependent variable and one or more independent features. When the number of the independent feature, is 1 then it is known as Univariate Linear regression, and in the case of more than one feature, it is known as multivariate linear regression.

Why Linear Regression is Important?

The interpretability of linear regression is a notable strength. The model's equation provides clear coefficients that elucidate the impact of each independent variable on the dependent variable, facilitating a deeper understanding of the underlying dynamics. Its simplicity is a virtue, as linear regression is transparent, easy to implement, and serves as a foundational concept for more complex algorithms.

Linear regression is not merely a predictive tool; it forms the basis for various advanced models. Techniques like regularization and support vector machines draw inspiration from linear regression, expanding its utility. Additionally, linear regression is a cornerstone in assumption testing, enabling researchers to validate key assumptions about the data.

Types of Linear Regression

Linear regression can be further divided into two types of the algorithm:

Simple Linear Regression:

If a single independent variable is used to predict the value of a numerical dependent variable, then such a Linear Regression algorithm is called Simple Linear Regression.

Multiple Linear regression:

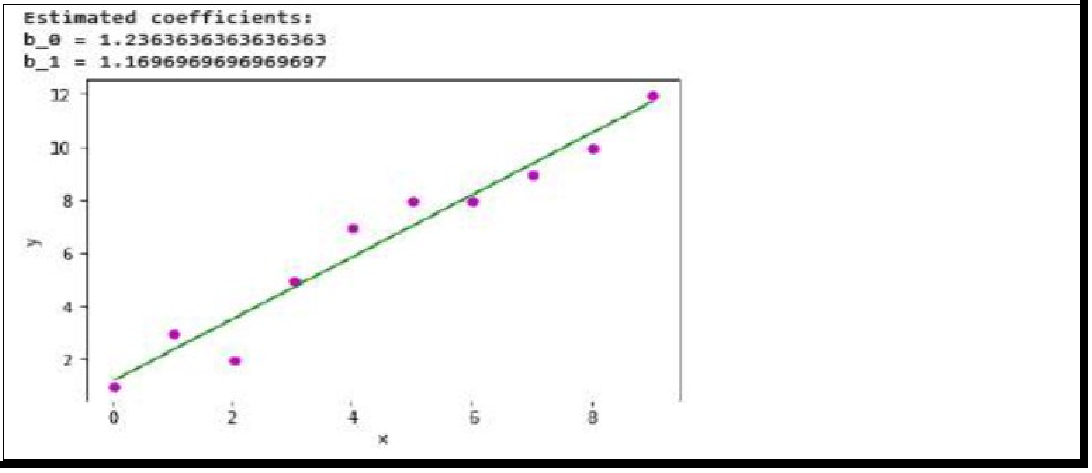
If more than one independent variable is used to predict the value of a numerical dependent variable, then such a Linear Regression algorithm is called Multiple Linear Regression.

Program Source Code

```
import numpy as np
import matplotlib.pyplot as plt
def estimate_coef(x, y):
# number of observations/points
n = np.size(x)
# mean of x and y vector
m_x = np.mean(x)
m_y = np.mean(y)
# calculating cross-deviation and deviation about x
```

```
SS_xy = np.sum(y*x) - n*m_y*m_x
SS_xx = np.sum(x*x) - n*m_x*m_x
# calculating regression coefficients
b_1 = SS_xy / SS_xx
b_0 = m_y - b_1*m_x
return (b_0, b_1)
def plot_regression_line(x, y, b):
# plotting the actual points as scatter plot
plt.scatter(x, y, color = "m",
marker = "o", s = 30)
# predicted response vector
y_pred = b[0] + b[1]*x
# plotting the regression line
plt.plot(x, y_pred, color = "g")
# putting labels
plt.xlabel('x')
plt.ylabel('y')
# function to show plot
plt.show()
def main():
# observations / data
x = np.array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
y = np.array([1, 3, 2, 5, 7, 8, 8, 9, 10, 12])
# estimating coefficients
b = estimate_coef(x, y)
print("Estimated coefficients:\nb_0 = {} \
\nb_1 = {}".format(b[0], b[1]))
# plotting regression line
plot_regression_line(x, y, b)
if __name__ == "__main__":
main()
```

Output



```
Estimated coefficients:
b_0 = 1.2363636363636363
b_1 = 1.1696969696969697
```

WEEK -09

9Q: Implement multiple linear Regression algorithms in order to fit data points. Select the appropriate data set for your experiment and draw graphs

Solutions:

Multiple linear regression is an extension of simple linear regression that involves predicting a dependent variable (target) based on multiple independent variables (features). It assumes a linear relationship between the independent variables and the dependent variable. The goal of multiple linear regression is to find the best-fitting linear equation that minimizes the difference between the observed and predicted values.

Advantages of Multiple Linear Regression:

1. Interpretability: The coefficients in the model provide insights into the relationship between independent and dependent variables.
2. Flexibility: Can handle multiple independent variables simultaneously, capturing more complex relationships in the data.
3. Simple and Easy to Implement: The model is relatively simple and easy to understand, making it accessible to users with basic statistical knowledge.

Disadvantages of Multiple Linear Regression:

1. Assumption of Linearity: Assumes a linear relationship between independent and dependent variables, which may not always hold true in real-world scenarios.
2. Assumption of Independence: Assumes that independent variables are not correlated with each other, violating this assumption can lead to biased estimates.
3. Sensitive to Outliers: Outliers in the data can significantly impact the model's performance and estimation of coefficients.

Program Source Code

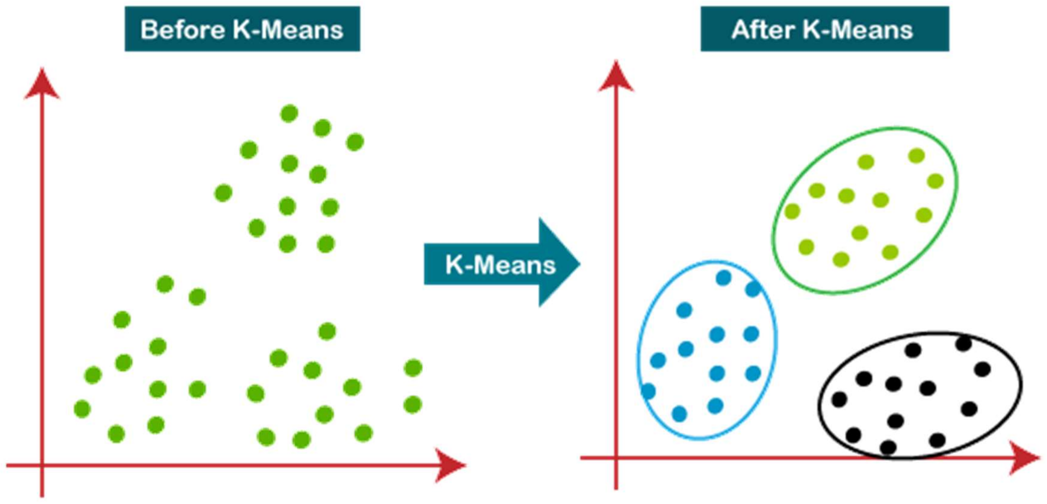
```
import numpy as np
class MultipleLinearRegression:
  def __init__(self):
    self.coefficients = None

  def fit(self, X, y):
    # Add a column of ones for the intercept term
    X = np.hstack((np.ones((X.shape[0], 1)), X))
        # Calculate coefficients using the normal equation
    self.coefficients = np.linalg.inv(X.T.dot(X)).dot(X.T).dot(y)
    def predict(self, X):
        # Add a column of ones for the intercept term
        X = np.hstack((np.ones((X.shape[0], 1)), X))

        # Make predictions
        return X.dot(self.coefficients)

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

# Create and fit the model
model = MultipleLinearRegression()
model.fit(X, y)

# Make predictions
X_test = np.array([[6, 7], [7, 8]])
predictions = model.predict(X_test)

print("Coefficients:", model.coefficients)
print("Predictions:", predictions)
```

WEEK -10

10Q: Write a program to implement the K-Means Clustering algorithm and find the SSE of set of clusters.

Solution:

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



How does the K-Means Algorithm Work?

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

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

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

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

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

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

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

Step-7: The model is ready.

Advantages of K-Means Algorithm:

1. Simple and Easy to Implement: K-Means is straightforward and easy to understand, making it accessible even to those with limited machine learning experience.
2. Efficient: It scales well to large datasets and is computationally efficient, making it suitable for clustering large amounts of data.
3. Versatile: K-Means can be applied to various types of data and is effective in many real-world scenarios.

4. Fast Convergence: It typically converges quickly, especially with a large number of dimensions.

Disadvantages of K-Means Algorithm:

1. Requires Specifying Number of Clusters: The user needs to specify the number of clusters (k) beforehand, which may not always be known or optimal.
2. Sensitive to Initial Centroid Selection: Results can vary based on the initial selection of centroids, and it may converge to suboptimal solutions.
3. Assumes Spherical Clusters: K-Means assumes that clusters are spherical and isotropic, which may not hold true for all datasets.
4. May Converge to Local Optima: It may converge to local optima, resulting in suboptimal clustering solutions.

Applications of K-Means Algorithm:

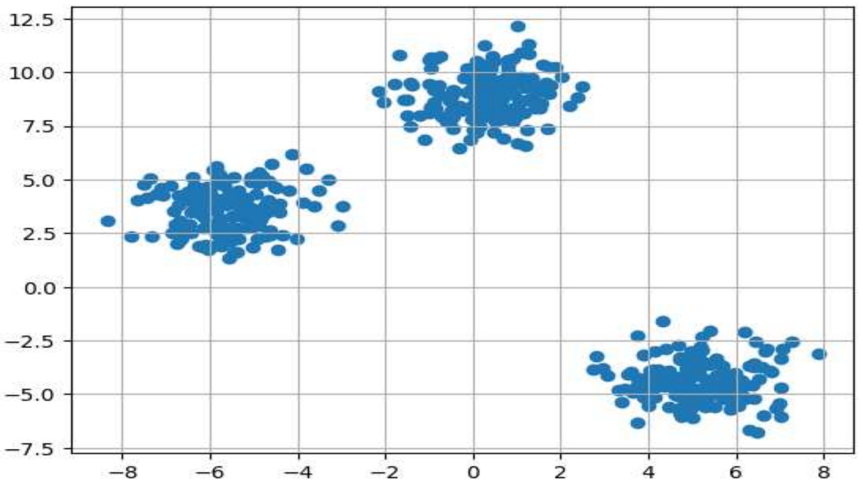
1. Customer Segmentation: Clustering customers based on purchase history, demographics, or behavior.
2. Image Compression: Reducing the size of images by clustering similar pixels together.
3. Anomaly Detection: Identifying unusual patterns or outliers in data.
4. Document Clustering: Grouping similar documents together in text data analysis.
5. Recommendation Systems: Grouping users or items with similar preferences in recommendation systems.

Program source Code

Import the necessary Libraries

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import make_blobs
```

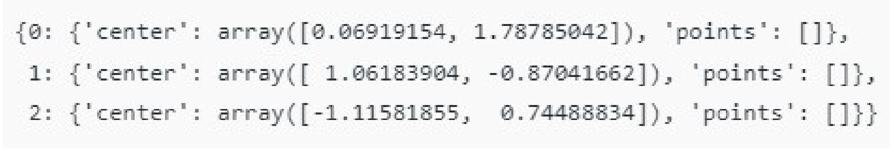
```
X,y = make_blobs(n_samples = 500,n_features = 2,centers = 3,random_state = 23)
fig = plt.figure(0)
plt.grid(True)
plt.scatter(X[:,0],X[:,1])
plt.show()
```



Initialize the random centroids

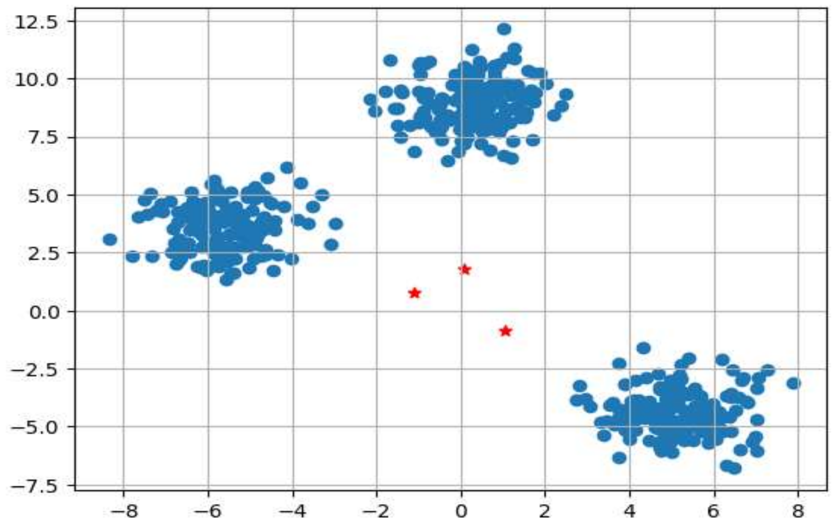
```
k = 3
clusters = {}
np.random.seed(23)
for idx in range(k):
    center = 2*(2*np.random.random((X.shape[1],))-1)
    points = []
    cluster = {
        'center' : center,
        'points' : []
    }
    clusters[idx] = cluster
```

clusters



```
{0: {'center': array([0.06919154, 1.78785042]), 'points': []},
 1: {'center': array([ 1.06183904, -0.87041662]), 'points': []},
 2: {'center': array([-1.11581855,  0.74488834]), 'points': []}}
```

```
plt.scatter(X[:,0],X[:,1])
plt.grid(True)
for i in clusters:
    center = clusters[i]['center']
    plt.scatter(center[0],center[1],marker = '*',c = 'red')
plt.show()
```



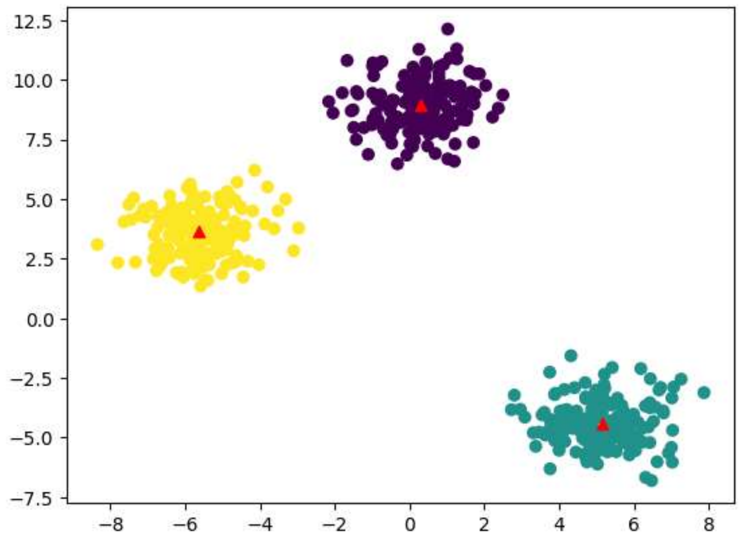
```
def distance(p1,p2):
    return np.sqrt(np.sum((p1-p2)**2))
```

```
#Implementing E step
def assign_clusters(X, clusters):
    for idx in range(X.shape[0]):
        dist = [ ]
        curr_x = X[idx]
        for i in range(k):
        dis = distance(curr_x,clusters[i]['center'])
        dist.append(dis)
        curr_cluster = np.argmin(dist)
        clusters[curr_cluster]['points'].append(curr_x)
    return clusters
#Implementing the M-Step
def update_clusters(X, clusters):
    for i in range(k):
        points = np.array(clusters[i]['points'])
        if points.shape[0] > 0:
            new_center = points.mean(axis =0)
            clusters[i]['center'] = new_center

            clusters[i]['points'] = []
    return clusters
```

```
def pred_cluster(X, clusters):
    pred = [ ]
    for i in range(X.shape[0]):
        dist = [ ]
        for j in range(k):
            dist.append(distance(X[i],clusters[j]['center']))
        pred.append(np.argmin(dist))
    return pred
```

```
clusters = assign_clusters(X,clusters)
clusters = update_clusters(X,clusters)
pred = pred_cluster(X,clusters)
```

```
plt.scatter(X[:,0],X[:,1],c = pred)
for i in clusters:
        center = clusters[i]['center']
        plt.scatter(center[0],center[1],marker = '^',c = 'red')
plt.show()
```



Program Source Code

```
import numpy as np
import matplotlib.pyplot as plt
# Sample data (replace with your actual data)
data = np.array([[1, 1], [1.5, 1.8], [5, 5], [8, 8], [1, 0.6], [9, 11]])
# Function to calculate Euclidean distance
def euclidean_distance(p1, p2):
  return np.sqrt(np.sum((p1 - p2)**2))
# Function to initialize centroids
def initialize_centroids(data, K):
  centroids = [ ]
  # Randomly select K data points as initial centroids
  for _ in range(K):
    random_index = np.random.randint(0, len(data))
    centroids.append(data[random_index])
  return np.array(centroids)
# Function to assign data points to closest centroid and calculate SSE
def assign_to_clusters_and_calculate_sse(data, centroids):
  clusters = []
  sse = np.zeros(len(centroids))  # Initialize SSE for each cluster
  for i, point in enumerate(data):
    min_distance = float('inf')
    min_index = -1
    for j, centroid in enumerate(centroids):
      distance = euclidean_distance(point, centroid)
      if distance < min_distance:
        min_distance = distance
        min_index = j
      sse[j] += distance**2  # Accumulate squared distance for each cluster
    clusters.append(min_index)
  return np.array(clusters), sse
# Function to update centroids based on assigned clusters
def update_centroids(data, clusters, K):
  new_centroids = np.zeros((K, data.shape[1]))
  for i in range(K):
    data_points_in_cluster = data[clusters == i]
    # If no points are assigned to a cluster, avoid division by zero
    if len(data_points_in_cluster) > 0:
      new_centroids[i] = np.mean(data_points_in_cluster, axis=0)
  return new_centroids
# Function to perform K-Means clustering
def kmeans(data, K):
  centroids = initialize_centroids(data, K)
  clusters = None
  # Iterate until centroids no longer change significantly
  while True:
    prev_centroids = centroids.copy()
    clusters, sse = assign_to_clusters_and_calculate_sse(data, centroids)
    centroids = update_centroids(data, clusters, K)
    # Check for convergence (stopping condition)
    if np.all(np.equal(prev_centroids, centroids)):
      break
  return clusters, centroids, sse
# Set the number of clusters (K)
K = 2
# Run K-Means clustering
clusters, centroids, sse = kmeans(data, K)
# Print the total SSE and individual SSE for each cluster
print("Total SSE:", np.sum(sse))
for i in range(K):
  print("Cluster", i, "SSE:", sse[i])
# Visualize the results
colors = ['red', 'blue']
for i, point in enumerate(data):
  plt.plot(point[0], point[1], 'o', color=colors[clusters[i]])
plt.scatter(centroids[:, 0], centroids[:, 1], s=100, marker='x', color='black', label='Centroids')
plt.title('K-Means Clustering (K=' + str(K) + ')')
plt.xlabel('X')
plt.ylabel('Y')
plt.legend()
plt.show()
```

WEEK -11

11Q: Write a program to implement an agglomerative hierarchal Clustering algorithm.
Solution:

Agglomerative Hierarchical Clustering is a popular method for clustering data points into a hierarchy of clusters. Unlike K-Means, which partitions data into a fixed number of clusters, agglomerative hierarchical clustering builds a tree-like hierarchy of clusters, where each data point starts in its own cluster and clusters are successively merged together. Here's how the agglomerative hierarchical clustering algorithm works:

Algorithm Steps:

1. **Initialization:**

   Start by assigning each data point to its own cluster. Therefore, initially, there are as many clusters as there are data points.

2. **Compute Pairwise Distances:**

   Compute the pairwise distance (or similarity) between each pair of clusters. The choice of distance metric (e.g., Euclidean distance, Manhattan distance, etc.) depends on the nature of the data.

3. **Merge Closest Clusters:**

   Find the closest pair of clusters based on the computed distances.

   Merge the closest pair of clusters into a single cluster. This reduces the total number of clusters by one.

4. **Update Distance Matrix:**

   Recalculate the pairwise distances between the newly formed cluster and all other clusters. This step is necessary as the merged cluster now needs to be compared to the remaining clusters.

5. **Repeat:**

6. Repeat steps 3 and 4 iteratively until only a single cluster remains, or until a stopping criterion is met. This stopping criterion could be a predefined number of clusters or a threshold distance beyond which clusters are not merged.

7. **Output:**

   At the end of the algorithm, you obtain a dendrogram, which is a tree-like structure that represents the hierarchy of clusters. The vertical lines in the dendrogram indicate clusters, and the height of each vertical line represents the distance at which clusters were merged.

Advantages of Agglomerative Hierarchical Clustering:

1. **Hierarchical Structure:** Provides a hierarchical representation of the data, allowing users to explore different levels of granularity.
2. **No Need to Specify Number of Clusters:** Unlike K-Means, there's no need to specify the number of clusters beforehand. The dendrogram can be cut at different levels to obtain different numbers of clusters.
3. **Can Handle Different Distance Metrics:** Allows for the use of various distance metrics, making it applicable to different types of data.

Disadvantages of Agglomerative Hierarchical Clustering:

1. **Computational Complexity:** Can be computationally expensive, especially for large datasets, as the pairwise distance matrix needs to be computed at each step.
2. **Memory Requirements:** The algorithm requires storing the entire distance matrix, which can be memory-intensive for large datasets.
3. **Difficulty in Interpreting Large Dendrograms:** Dendrograms can become complex and difficult to interpret, especially for large datasets with many data points.
4. **Applications of Agglomerative Hierarchical Clustering:**
    a. **Biology:** Clustering genes or proteins based on their expression levels.
    b. **Text Analysis:** Clustering documents or text snippets based on similarity of content.
    c. **Image Analysis:** Segmenting images into regions with similar characteristics.
    d. **Social Network Analysis:** Identifying communities or clusters of users in social networks based on interaction patterns.

Program Source Code: agglomerative hierarchal Clustering algorithm.

```
from sklearn.cluster import AgglomerativeClustering
import numpy as np
# randomly chosen dataset
X = np.array([[1, 2], [1, 4], [1, 0], [4, 2], [4, 4], [4, 0]])
# here we need to mention the number of clusters
# otherwise the result will be a single cluster
# containing all the data
clustering = AgglomerativeClustering(n_clusters=2).fit(X)
# print the class labels
print(clustering.labels_)
```

OUTPUT:
```
[1, 1, 1, 0, 0, 0]
```