G H Raisoni College of Engineering and Management, Wagholi, Pune

Department of AI & AIML Engineering

LAB MANUAL

Subject: MACHINE LEARNING (23UAIPCP2404)

Class: SY (CSE AI B – 2023 Course)

Examination Scheme:
Practical (EXT): 25 Marks
Total: 25 Marks

Prepared By: DR. VAISHALI YOGESH BAVISKAR

LIST OF EXPERIMENTS

Sr. No.	Title of Experiment	CO
1	Tools and Libraries used for Machine Learning	CO1
2	Generate a proper 2-D data set of N points. Split the data set into Training Data set and Test Data set. i) Perform linear regression analysis with Least Squares Method. ii) Plot the graphs for Training MSE and Test MSE and comment on Curve Fitting and Generalization Error. iii) Verify the Effect of Data Set Size and Bias-Variance Tradeoff. iv) Apply Cross Validation and plot the graphs for errors. v) Apply Subset Selection Method and plot the graphs for errors. vi) Describe your findings in each case	CO3
3	Study and Implement Multiple Linear Regression	CO2
4	Implement Logistic Regression using any dataset.	CO2
5	Study and Implement Decision Tree.	CO2
6	Implement Naïve Bayes Classifier on data set of your choice. Test and Compare for Accuracy and Precision.	CO3
7	Implement K Nearest Neighbour (KNN) classification	CO2
8	Implement Support Vector Machine algorithm.	CO2
9	Given the following data, which specify classifications for nine combinations of VAR1 and VAR2 predict a classification for a case where VAR1=0.906 and VAR2=0.606, using the result of k-means clustering with 3 means (i.e. 3 centroids) VAR1 VAR2 CLASS 1.713 1.586 0 0.180 1.786 1 0.353 1.240 1 0.940 1.566 0 1.486 0.759 1 1.266 1.106 0 1.540 0.419 1 0.459 1.799 1 0.773 0.186 1	CO2
10	Unsupervised Learning: Implement K-Means Clustering and Hierarchical clustering on proper data set of your choice. Compare their Convergence	CO3
11	Study and Implement Bagging/Boosting using Random Forests.	CO4

	Virtual Lab Assignments	
12	Linear and Multiple Linear Regression https://vlab.spit.ac.in/ai/#/experiments/10	CO2
13	Logistic Regression Algorithm https://vlab.spit.ac.in/ai/#/experiments/11	CO2
14	K-Nearest Neighbor Algorithm https://vlab.spit.ac.in/ai/#/experiments/4	CO2
15	Support Vector Machine Algorithm https://vlab.spit.ac.in/ai/#/experiments/5	CO2
16	K- Means Clustering Algorithm https://vlab.spit.ac.in/ai/#/experiments/3	CO2
17	Random Forest Algorithm https://vlab.spit.ac.in/ai/#/experiments/12	CO2
	Content Beyond Syllabus	
18	To classify simple binary data using a quantum circuit as a feature map in a hybrid quantum-classical approach.	CO4

ASSIGNMENT NO. 1

TITLE

Tools and Libraries used for Machine Learning

OBJECTIVE

- To familiarize students with essential tools and libraries used in Machine Learning (ML) workflows
- Provide hands-on experience with their usage.

PLATFORM REQUIRED

Operating System: Windows or Linux

Software/Tools: Anaconda, Spyder, Jypter Notebook, Google Colab

THEORY

1. Python Programming Language

Python is the most widely used programming language for Machine Learning due to its simplicity, readability, and extensive libraries.

Applications:

- Writing ML algorithms.
- Data preprocessing.
- Integration with libraries and frameworks.

Example:

```
print("Hello, Machine Learning!")
```

2. NumPy

NumPy is a library for numerical computations. It provides support for arrays, matrices, and high-level mathematical functions.

Applications

- Handling large datasets.
- Performing mathematical operations like matrix multiplication.

Example:

```
import numpy as np

array = np.array([1, 2, 3])

print(array * 2)
```

3. Pandas

Pandas is a library for data manipulation and analysis. It allows easy handling of tabular data with rows and columns.

Applications

- Reading and writing datasets.
- Data cleaning and transformation.

Example

```
import pandas as pd

data = pd.read_csv("data.csv")

print(data.head())
```

4. Matplotlib

Matplotlib is a plotting library for creating static, interactive, and animated visualizations.

Applications

- Data visualization.
- Plotting graphs like line charts, scatter plots, etc.

Example

```
import matplotlib.pyplot as plt

plt.plot([1, 2, 3], [4, 5, 6])

plt.show()
```

5. Seaborn

Built on Matplotlib, Seaborn provides a high-level interface for drawing attractive and informative statistical graphics.

Applications

- Visualizing distributions and relationships between data variables.

Example

```
import seaborn as sns

sns.scatterplot(x=[1, 2, 3], y=[4, 5, 6])
```

6. scikit-learn

scikit-learn is a library for ML providing simple and efficient tools for data mining and analysis.

Applications

- Building ML models like regression, classification, clustering, etc.
- Feature engineering and cross-validation.

Example

```
from sklearn.linear_model import LinearRegression

model = LinearRegression()

model.fit([[1], [2], [3]], [4, 5, 6])

print(model.predict([[4]]))
```

7. TensorFlow and Keras

TensorFlow is an open-source library for numerical computation and ML. Keras is a high-level API for TensorFlow.

Applications

- Building and training neural networks.
- Deep learning tasks like image and speech recognition.

Example

```
import tensorflow as tf
from tensorflow.keras import Sequential
from tensorflow.keras.layers import Dense
model = Sequential([Dense(10, activation='relu'), Dense(1)])
model.compile(optimizer='adam', loss='mse')
```

8. PyTorch

PyTorch is an open-source machine learning framework based on Torch.

Applications

- Building neural networks with dynamic computation graphs.
- Research and deployment of ML models.

Example

```
import torch
x = torch.tensor([1.0, 2.0])
print(x * 2)
```

9. Jupyter Notebook

An open-source web application for creating and sharing documents with live code,

equations, and visualizations.

Applications

- Interactive coding.
- Data visualization and storytelling.

10. OpenCV

OpenCV is a library for real-time computer vision.

Applications

- Image and video analysis.
- Feature extraction and object detection.

Example

```
import cv2

img = cv2.imread("image.jpg")

cv2.imshow("Image", img)

cv2.waitKey(0)
```

QUESTIONS

1. Practice with NumPy- Create a NumPy array with random numbers., Perform matrix multiplication.
2. Data Handling with Pandas- Load a dataset using Pandas, Clean the dataset by handling missing values.
3. Data Visualization- Use Matplotlib and Seaborn to create scatter and bar plots for a dataset.

CONCLUSION

Thus, this assignment introduces essential ML tools and libraries, enabling students to perform data preprocessing, visualization, and model building efficiently.

ASSIGNMENT NO. 2

TITLE

A company wants to analyze the relationship between an employee's experience and their salary. You are tasked with building a simple linear regression model to predict the salary of an employee based on the number of years of experience they have.

PROBLEM STATEMET

Analyze the Relationship Between Years of Experience and Salary Using Linear Regression. Develop a linear regression model to predict the salary of an employee based on their years of experience. The analysis includes loading the dataset, splitting the data into training and testing sets, training the model, evaluating performance using regression metrics, and visualizing the results.

OBJECTIVE

- Load a dataset containing employee years of experience and salary data.
- Train a simple linear regression model to predict salary based on years of experience.
- Split the dataset into training and testing subsets.
- Evaluate model performance using metrics such as MSE and R^2.
- Visualize the results of the model on both training and testing datasets.
- Interpret findings and analyze the impact of years of experience on salary.

PLATFORM REQUIRED

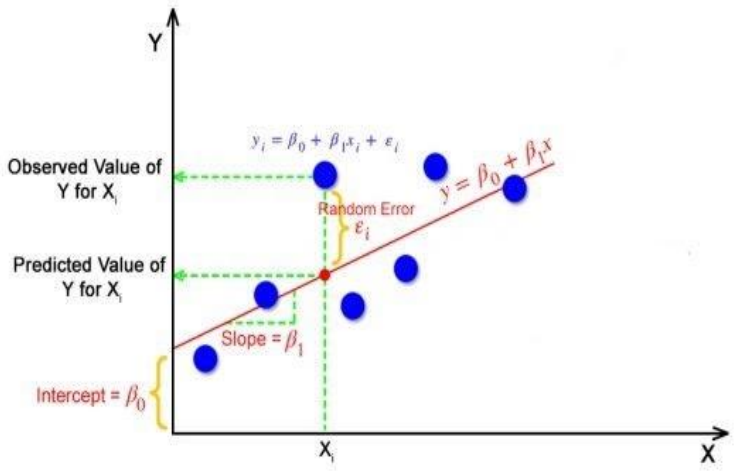
- **Operating System:** Windows or Linux
- **Software/Tools:** Anaconda, Spyder, Jypter Notebook, Google Colab

THEORY

Linear Regression

Linear Regression is a supervised learning algorithm used for predicting a continuous target variable based on input features. The model establishes a relationship between the dependent variable (target) and independent variables (features) using a straight line,

represented as:



Where:

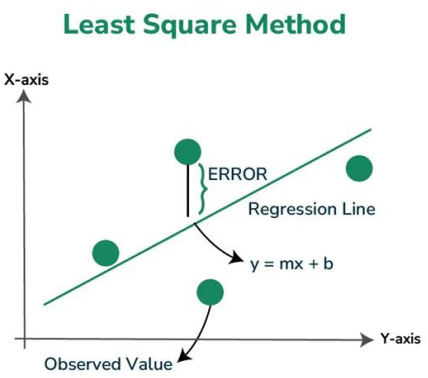
- Target variable: The variable whose values are to be predicted by other variables. It is also known as the dependent variable.
- Input feature: Also known as the explanatory or predictor variable.
- Intercept: Also known as the y-intercept, it is where the line of best fit intersects the y-axis. It represents the starting point of the data.
- Slope: Represents the change in y for any 1 unit change in x.
- Error term: Represents the margin of error within a statistical model. It is the sum of the deviations within the regression line.

Least Squares Method minimizes the sum of squared differences between the observed and predicted values

Least Square method

Least Squares method is a statistical technique used to find the equation of best-fitting curve or line to a set of data points by minimizing the sum of the squared differences between the observed values and the values predicted by the model.

This method aims at minimizing the sum of squares of deviations as much as possible. The line obtained from such a method is called a regression line or line of best fit.



Least Square Method formula is used to find the best-fitting line through a set of data points. For a simple linear regression, which is a line of the form y=mx+c, where y is the dependent variable, x is the independent variable, a is the slope of the line, and b is the y-intercept, the formulas to calculate the slope (m) and intercept (c) of the line are derived from the following equations:

Slope (m) Formula: m = n(∑xy)−(∑x)(∑y) / n(∑x2)−(∑x)2

Intercept (c) Formula: c = (∑y)−a(∑x) / n

Where:

- n is the number of data points,
- ∑xy is the sum of the product of each pair of x and y values,
- ∑x is the sum of all x values,
- ∑y is the sum of all y values,
- ∑x2 is the sum of the squares of x values.

Evaluation Metrics for Regression Model

Mean Absolute Error (MAE)

Mean absolute error, or L1 loss, stands out as one of the simplest and easily comprehensible loss functions and evaluation metrics. It computes by averaging the absolute differences between predicted and actual values across the dataset. Mathematically, it represents the arithmetic mean of absolute errors, focusing solely on their magnitude, irrespective of direction. A lower MAE indicates superior model accuracy.

$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n} | y_i - \hat{y}_i |$$

where

y_i = actual value

y_hat_i = predicted value

n = sample size

Mean Squared Error (MSE)

MSE is one of the most common regression loss functions and an important error metric. In Mean Squared Error, also known as L2 loss, we calculate the error by squaring the difference between the predicted value and actual value and averaging it across the dataset.

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

Root Mean Squared Error (RMSE)

Root Mean Square Error in Machine Learning (RMSE) is a popular metric used in machine learning and statistics to measure the accuracy of a predictive model. It quantifies the differences between predicted values and actual values, squaring the errors, taking the mean, and then finding the square root. RMSE provides a clear understanding of the model's performance, with lower values indicating better predictive accuracy relative root mean square error. It is computed by taking the square root of MSE. RMSE is also called the Root Mean Square Deviation. It measures the average magnitude of the errors and is concerned with the deviations from the actual value. RMSE value with zero indicates that the model has a perfect fit. The lower the RMSE, the better the model and its predictions. A higher relative root mean square error in machine learning indicates that there is a large deviation from the residual to the ground truth. RMSE can be used with different features as it helps in figuring out if the feature is improving the model's prediction or not.

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}$$

STEP BY STEP ALGORITHM

1. Import necessary 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
from sklearn.metrics import mean_squared_error, r2_score
```

2. Load the dataset

```
dataset = pd.read_csv('Salary_data.csv')
print("Dataset Preview:")
 print(dataset.head())
```

3. Split the dataset

```
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.20, random_state=42)
print("\nTraining and Test Split Completed.")
```

4. Train the linear regression model

```
regressor = LinearRegression()
regressor.fit(X_train, Y_train)
print("\nModel Training Completed.")
```

5. Make predictions

```
Y_train_pred = regressor.predict(X_train)
Y_test_pred = regressor.predict(X_test)
```

6. Evaluate the model

```
train_mse = mean_squared_error(Y_train, Y_train_pred)
test_mse = mean_squared_error(Y_test, Y_test_pred)
train_r2 = r2_score(Y_train, Y_train_pred)
test_r2 = r2_score(Y_test, Y_test_pred)
```

7. Visualizing the training set results

```
plt.scatter(X_train, Y_train, color='blue', label='Training Data')
plt.plot(X_train, Y_train_pred, color='red', label='Regression Line')
plt.title('Salary vs Experience (Training Set)')
plt.xlabel('Years of Experience')
plt.ylabel('Salary')
plt.legend()
plt.show()
```

8. Visualize the test set results

```
plt.scatter(X_test, Y_test, color='green', label='Test Data')
plt.plot(X_train, Y_train_pred, color='red', label='Regression Line')
plt.title('Salary vs Experience (Test Set)')
plt.xlabel('Years of Experience')
plt.ylabel('Salary')
plt.legend()
plt.show()
```

9. Display Regression Coefficients

```
print(f"\nRegression Coefficients:")
print(f"Intercept (b0): {regressor.intercept_[0]:.2f}")
print(f"Slope (b1): {regressor.coef_[0][0]:.2f}")
```

QUESTIONS

1. What is linear Regression?
2. What is least square method in linear regression
3. What is Linear Regression, and how is it applied in this problem?
4. Explain the importance of the slope and intercept in the regression equation.
5. Interpret the model's performance using MAE, MSE and RMSE values.
6. How does the test dataset ensure the generalization of the model?
7. Explain the significance of visualizing regression results.

CONCLUSION

This assignment demonstrates the practical application of linear regression to analyze and predict employee salaries based on years of experience. Key concepts such as data preprocessing, model evaluation, and visualization have been implemented, providing insights into the relationship between the independent and dependent variables.

ASSIGNMENT NO. 3

TITLE

Study and Implement Multiple Linear Regression

PROBLEM STATEMENT

Implement Multiple Linear Regression using a dataset that contains information about 50 startups and evaluate the model.

i) Perform One-Hot Encoding on categorical data.

ii) Split the dataset into Training and Test sets.

iii) Train the model using Linear Regression and predict outcomes.

iv) Evaluate model performance using various metrics and visualize results.

v) Analyze predicted values and their differences with actual values.

OBJECTIVE

- Import and preprocess the dataset for linear regression analysis.
- Apply One-Hot Encoding for categorical variables.
- Train and test the Multiple Linear Regression model.
- Evaluate model accuracy using various metrics.
- Visualize and interpret the results.

PLATFORM REQUIRED

Operating System: Windows or Linux

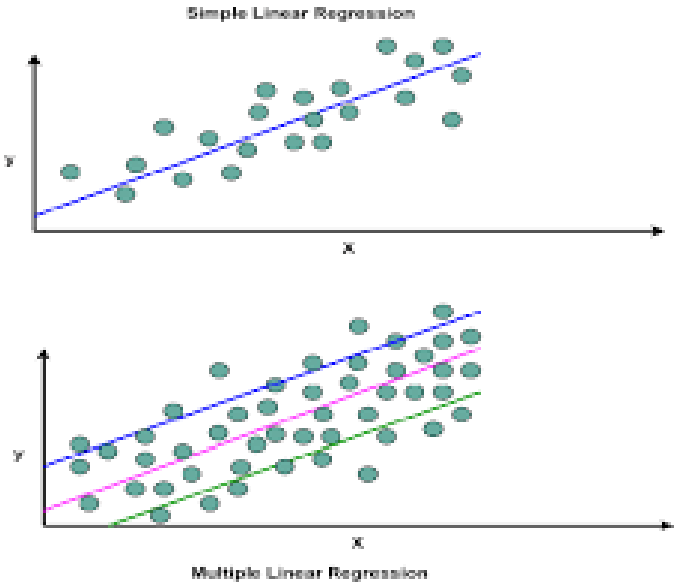
Software/Tools: Anaconda, Spyder, Jupyter Notebook, Google Colab

THEORY

Multiple Linear Regression

Multiple Linear Regression is a supervised learning algorithm used to model the relationship between one dependent variable and two or more independent variables.

One-Hot Encoding is used for converting categorical data into numerical form by creating binary columns for each category. It ensures that machine learning algorithms can process the data correctly.



Multiple Linear Regression Formula

$$yi = \beta 0 + \beta 1xi1 + \beta 2xi2 + ... + \beta pxip + \epsilon$$

- **yi** is the dependent or predicted variable
- **β0** is the y-intercept, i.e., the value of y when both xi and x2 are 0.
- **β1** and **β2** are the regression coefficients representing the change in y relative to a one-unit change in **xi1** and **xi2**, respectively.
- **βp** is the slope coefficient for each independent variable
- **ϵ** is the model's random error (residual) term.

STEP BY STEP ALGORITHM

1. Import Necessary Libraries

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
```

```
from sklearn.metrics import r2_score, mean_absolute_error, mean_squared_error
from sklearn.preprocessing import OneHotEncoder
from sklearn.compose import make_column_transformer
```

2. Download the dataset and load it using pandas

```
startup_df = pd.read_csv('50_Startups.csv') # Load dataset

print(startup_df.head()) # View first few rows
```

3. View Dataset Information

```
print(startup_df.head())  # View first few rows
print(startup_df.info())  # Dataset summary
print(f"Dataset contains {startup_df.shape[0]} rows and {startup_df.shape[1]} columns.")
```

4. View column names and statistical details.

```
print(startup_df.columns)
print(startup_df.describe())
```

5. Define Input and Output Variables

```
x = startup_df.drop(columns=['Profit']) # Independent variables (excluding target)

y = startup_df['Profit'] # Dependent variable
```

6. Use One-Hot Encoding for the categorical "State" column.

```
col_trans = make_column_transformer(
    (OneHotEncoder(handle_unknown='ignore'), ['State']),  # Encode "State"
    remainder='passthrough'  # Keep numeric columns unchanged
)
```

7. Split the Dataset

```
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=0) # Apply
transformations only after splitting

x_train = col_trans.fit_transform(x_train)

x_test = col_trans.transform(x_test)
```

8. Train the Model

```
linreg = LinearRegression()  # Initialize model
```

```
linreg.fit(x_train, y_train)  # Train the model
```

9. Predict Test Results

```
y_pred = linreg.predict(x_test) # Generate predictions
```

10. Predict outcomes for the test set.

```
y_pred = linreg.predict(x_test)
print(y_pred)
```

11. Evaluate the Model

```
r2 = r2_score(y_test, y_pred)

mae = mean_absolute_error(y_test, y_pred)

mse = mean_squared_error(y_test, y_pred)

rmse = np.sqrt(mse)

print(f"R² Score: {r2:.2f}")

print(f"Mean Absolute Error (MAE): {mae:.2f}")

print(f"Mean Squared Error (MSE): {mse:.2f}")

print(f"Root Mean Squared Error (RMSE): {rmse:.2f}")
```

12. Visualize Results
13. Scatter plot for actual vs. predicted values.

```
plt.figure(figsize=(8, 6))
plt.scatter(y_test, y_pred, color='blue', alpha=0.6, label='Predicted vs Actual')
plt.plot([min(y_test), max(y_test)], [min(y_test), max(y_test)], color='black',
linestyle='dashed', label='Perfect Fit')
plt.xlabel('Actual Profit')
plt.ylabel('Predicted Profit')
plt.title('Actual vs Predicted Values')
plt.legend()
plt.show()
```

14. Regression plot.

```
plt.figure(figsize=(8, 6))
```

```
sns.regplot(x=y_test, y=y_pred, ci=None, color='red')
plt.title('Regression Plot')
plt.xlabel('Actual Profit')
plt.ylabel('Predicted Profit')
plt.show()
```

15. Analyze Predicted Values
16. Create a DataFrame to compare actual and predicted values.

```
pred_df = pd.DataFrame({
   'Actual Value': y_test.values,
   'Predicted Value': y_pred,
   'Difference': y_test.values - y_pred
})

print(pred_df)
```

QUESTIONS

1. What is Multiple Linear Regression?
2. Why is One-Hot Encoding necessary for categorical variables?
3. What is the significance of splitting the dataset into Training and Test sets?
4. How do metrics like R^2, MAE, MSE, and RMSE help in evaluating a model?
5. What observations can you make from the scatter plot and regression plot?
6. Analyze the differences between actual and predicted values. What does it indicate about the model's performance?

CONCLUSION

In this assignment, we implemented a Multiple Linear Regression model, handled categorical data using One-Hot Encoding, and evaluated the model using R^2 score, MAE, MSE, and RMSE. The visualizations helped us understand the linear relationship and accuracy of predictions. The difference between actual and predicted values was minimal, reflecting a high-performing model.

ASSIGNMENT NO. 4

TITLE

Implement Logistic Regression using any dataset

PROBLEM STATEMENT

Using a dataset containing information about age, salary, and purchased status, implement Logistic Regression to predict whether a customer will make a purchase.

i) Preprocess the dataset for analysis.

ii) Split the dataset into Training and Test sets.

iii) Train the Logistic Regression model.

iv) Evaluate model performance using various metrics.

v) Visualize and interpret the results.

OBJECTIVE

- Import and preprocess the dataset for logistic regression analysis.
- Train and test the Logistic Regression model.
- Evaluate model accuracy using metrics such as accuracy score, classification report, and confusion matrix.
- Visualize and interpret the results.

PLATFORM REQUIRED

Operating System: Windows or Linux

Software/Tools: Anaconda, Spyder, Jupyter Notebook, Google Colab

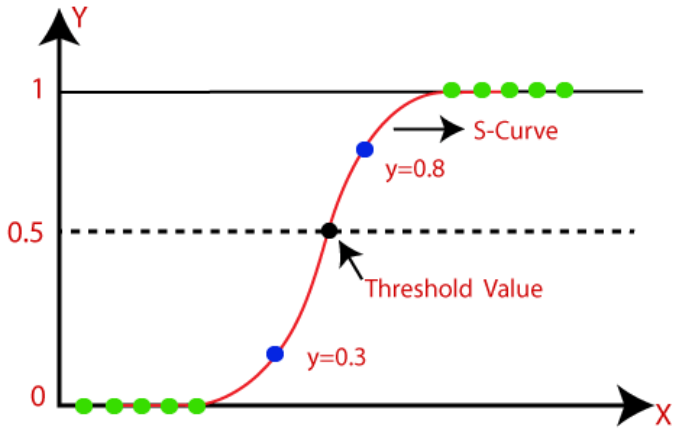
THEORY

Logistic Regression

Logistic regression is one of the most popular Machine Learning algorithms, which comes under the Supervised Learning technique. It is used for predicting the categorical dependent variable using a given set of independent variables. Logistic regression predicts the output of a categorical dependent variable. Therefore the outcome must be a categorical or discrete value. It can be either Yes or No, 0 or 1, true or False, etc. but instead of giving the exact value as 0 and 1, it gives the probabilistic values which lie between 0 and 1.

Logistic Regression is much similar to the Linear Regression except that how they are used. Linear Regression is used for solving Regression problems, whereas Logistic regression is used for solving the classification problems. In Logistic regression, instead of fitting a regression line, we fit an "S" shaped logistic function, which predicts two maximum values (0 or 1).

The curve from the logistic function indicates the likelihood of something such as whether the cells are cancerous or not, a mouse is obese or not based on its weight, etc. Logistic Regression is a significant machine learning algorithm because it has the ability to provide probabilities and classify new data using continuous and discrete datasets. Logistic Regression can be used to classify the observations using different types of data and can easily determine the most effective variables used for the classification. The below image is showing the logistic function:



Logistic Function (Sigmoid Function)

The sigmoid function is a mathematical function used to map the predicted values to probabilities. It maps any real value into another value within a range of 0 and 1. The value of the logistic regression must be between 0 and 1, which cannot go beyond this limit, so it forms a curve like the "S" form. The S-form curve is called the Sigmoid function or the logistic function.

In logistic regression, we use the concept of the threshold value, which defines the probability of either 0 or 1. Such as values above the threshold value tends to 1, and a value below the threshold values tends to 0.

Logistic Regression Equation

The Logistic regression equation can be obtained from the Linear Regression equation. The mathematical steps to get Logistic Regression equations are given below:

We know the equation of the straight line can be written as:

$$y= b_0 + b_1x_1 + b_2x_2 + b_3x_3 + \cdots + b_nx_n$$

In Logistic Regression y can be between 0 and 1 only, so for this let's divide the above equation by (1-y):

$$\frac{y}{1-y}$$; 0 for y= 0, and infinity for y=1

But we need range between -[infinity] to +[infinity], then take logarithm of the equation it will become:

$$log\left[\frac{y}{1-y}\right] = b_0 + b_1x_1 + b_2x_2 + b_3x_3 + \cdots + b_nx_n$$

The above equation is the final equation for Logistic Regression.

STEP BY STEP ALGORITHM

1. Import Necessary Libraries

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import StandardScaler
```

```
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns
```

1. Load the dataset

```
df = pd.read_csv('purchase_data.csv')
print(df.head())
```

2. Check the structure of the dataset

```
print("Dataset contains {} rows and {} columns".format(df.shape[0], df.shape[1]))
```

3. View column names and statistical details

```
print(df.columns)
print(df.describe())
```

4. Preprocess the dataset – remove missing or duplicate values

```
df.drop_duplicates(inplace=True)
df.dropna(inplace=True)
```

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

```
X = df[['Age', 'Salary']]
y = df['Purchased']
```

6. Split the data into Training (80%) and Test (20%) sets

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20, random_state=42)
```

7. Scale the dataset using Standard Scaler

```
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
```

```
X_test_scaled = scaler.transform(X_test)
```

8. Build and Train the Logistic Regression Model
9. Initialize and train the model

```
model = LogisticRegression()

model.fit(X_train_scaled, y_train)
```

10. Evaluate the Model
11. Compute accuracy score for the training set

```
train_acc = model.score(X_train_scaled, y_train)

print("Training Accuracy: {:.2f}%".format(train_acc * 100))
```

12. Predict outcomes for the test set

```
y_pred = model.predict(X_test_scaled)
```

13. Compute accuracy score for the test set

```
test_acc = accuracy_score(y_test, y_pred)

print("Test Accuracy: {:.2f}%".format(test_acc * 100))
```

14. Generate a classification report

```
print(classification_report(y_test, y_pred))
```

15. Plot a confusion matrix

```
cm = confusion_matrix(y_test, y_pred)

plt.figure(figsize=(8, 6))

sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')

plt.title("Confusion Matrix")

plt.ylabel("Actual Values")
```

```
plt.xlabel("Predicted Values")

plt.show()
```

16. Visualize Results
17. Plot the decision boundary

```
from matplotlib.colors import ListedColormap

X_set, y_set = X_train_scaled, y_train

X1, X2 = np.meshgrid(np.arange(start=X_set[:, 0].min() - 1, stop=X_set[:, 0].max() + 1,
    step=0.01),

 np.arange(start=X_set[:, 1].min() - 1, stop=X_set[:, 1].max() + 1, step=0.01))

plt.contourf(X1, X2, model.predict(np.array([X1.ravel(), X2.ravel()]).T).reshape(X1.shape),

alpha=0.75, cmap=ListedColormap(('red', 'green')))

plt.xlim(X1.min(), X1.max())

plt.ylim(X2.min(), X2.max())

for i, j in enumerate(np.unique(y_set)):

   plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1],

            c=ListedColormap(('red', 'green'))(i), label=j)

plt.title('Logistic Regression Decision Boundary')

plt.xlabel('Age')

plt.ylabel('Salary')

plt.legend()

plt.show()
```

QUESTIONS

1. What is Logistic Regression, and how does it work?
2. Why is it necessary to scale features in Logistic Regression?
3. What are the roles of the training and test sets in model evaluation?
4. How does a confusion matrix help in evaluating model performance?
5. How can the accuracy of Logistic Regression be improved for better predictions?

CONCLUSION

In this assignment, we implemented Logistic Regression to predict purchase decisions based on age and salary. The dataset was preprocessed, scaled, and split into training and test sets. The model was trained and achieved significant accuracy on both training and test data. Visualizations like the confusion matrix and decision boundary helped understand the model's performance, highlighting its strengths and areas for improvement.

ASSIGNMENT NO. 5

TITLE

Study and Implement Decision Tree.

PROBLEM STATEMENT

Implement a Decision Tree Classification Algorithm using a dataset (user_data.csv) containing user information. Visualize the decision tree structure, evaluate the model's performance, and interpret the results.

OBJECTIVE

- Import and preprocess the dataset for classification analysis.
- Split the dataset into Training and Test sets.
- Train the model using a Decision Tree Classifier.
- Visualize the decision tree structure.
- Predict outcomes on the test set and evaluate the model performance.
- Visualize the training and test set decision boundaries
- Interpret and analyze the results

PLATFORM REQUIRED

Operating System: Windows or Linux

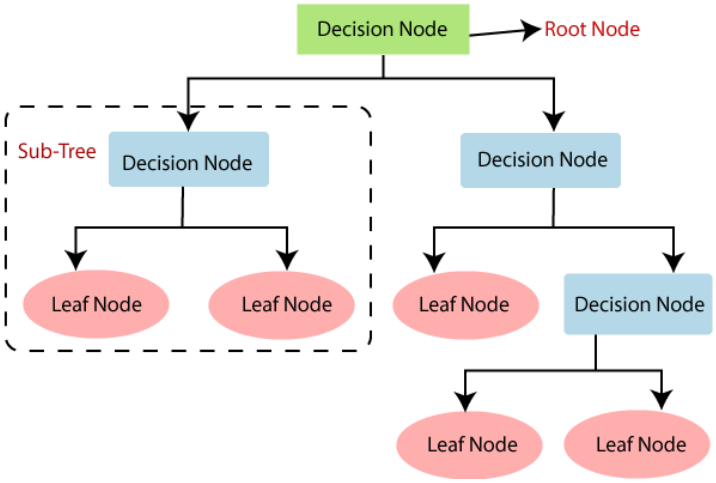
Software/Tools: Anaconda, Spyder, Jupyter Notebook, Google Colab

THEORY

Decision Tree Algorithm

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.

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. The decisions or the test are performed on the basis of features of the given dataset.



- Root Node: Root node is from where the decision tree starts. It represents the entire dataset, which further gets divided into two or more homogeneous sets.
- Leaf Node: Leaf nodes are the final output node, and the tree cannot be segregated further after getting a leaf node.
- Splitting: Splitting is the process of dividing the decision node/root node into sub-nodes according to the given conditions.
- Branch/Sub Tree: A tree formed by splitting the tree.
- Pruning: Pruning is the process of removing the unwanted branches from the tree.
- Parent/Child node: The root node of the tree is called the parent node, and other nodes are called the child nodes.

Attribute Selection Measures

While implementing a Decision tree, the main issue arises that how to select the best attribute for the root node and for sub-nodes. So, to solve such problems there is a technique which is called as Attribute selection measure or ASM. By this measurement, we can easily select the best attribute for the nodes of the tree. There are two popular techniques for ASM, which are:

1. Information Gain
2. Gini Index

1. Information Gain

Information gain is the measurement of changes in entropy after the segmentation of a dataset based on an attribute. It calculates how much information a feature provides us about a class. According to the value of information gain, we split the node and build the decision tree. A decision tree algorithm always tries to maximize the value of information gain, and a node/attribute having the highest information gain is split first. It can be

calculated using the below formula:

Information Gain= Entropy(S)- [(Weighted Avg) *Entropy(each feature)

Entropy

Entropy is a metric to measure the impurity in a given attribute. It specifies randomness in data. Entropy can be calculated as:

Entropy(s)= -P(yes)log2 P(yes)- P(no) log2 P(no)

2. **Gini Index**

Gini index is a measure of impurity or purity used while creating a decision tree in the CART(Classification and Regression Tree) algorithm. An attribute with the low Gini index should be preferred as compared to the high Gini index. It only creates binary splits, and the CART algorithm uses the Gini index to create binary splits.

Gini index can be calculated using the below formula:

$$\text{Gini Index} = 1 - \sum_j P_j^2$$

STEP BY STEP ALGORITHM

1. Import necessary libraries

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier, plot_tree
from sklearn.metrics import confusion_matrix, accuracy_score
    from matplotlib.colors import ListedColormap
```

2. Load and View Dataset

```
# Load the dataset

data_set = pd.read_csv('user_data.csv')  # Replace with the correct dataset path

    print(data_set.head())  # View the first few rows of the dataset
```

3. Extract Features and Target Variables

```
# Extract features and target variables

x = data_set.iloc[:, [2, 3]].values  # Features: Age and Estimated Salary

    y = data_set.iloc[:, 4].values       # Target: Purchased
```

4. Split the Dataset into Training and Test Sets

```
# Split the dataset into training and test sets

    x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.25, random_state=0)
```

5. Feature Scaling (Optional)

```
# Perform feature scaling for better visualization

from sklearn.preprocessing import StandardScaler

sc = StandardScaler()

x_train = sc.fit_transform(x_train)

    x_test = sc.transform(x_test)
```

6. Train the Decision Tree Classifier

```
# Train the Decision Tree Classifier

classifier = DecisionTreeClassifier(criterion='entropy', random_state=0)

    classifier.fit(x_train, y_train)
```

7. Visualize the Decision Tree Structure

```
# Visualize the Decision Tree
plt.figure(figsize=(12, 8))
plot_tree(classifier,
          feature_names=['Age', 'Estimated Salary'],
          class_names=['Not Purchased', 'Purchased'],
          filled=True,
          rounded=True,
          fontsize=10)
plt.title("Decision Tree Visualization")
     plt.show()
```

8. Predict Test Set Results

```
# Predict outcomes on the test set
y_pred = classifier.predict(x_test)
     print("Predicted Values:", y_pred)
```

9. Evaluate the Model Performance

```
# Confusion Matrix
cm = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:\n", cm)
# Accuracy Score
accuracy = accuracy_score(y_test, y_pred)
```

```
print("Accuracy:", accuracy)
```

10. Visualize the Training Set Results

```
# Visualize the training set results

x_set, y_set = x_train, y_train

x1, x2 = np.meshgrid(np.arange(start=x_set[:, 0].min() - 1, stop=x_set[:, 0].max() + 1,
    step=0.01),

                np.arange(start=x_set[:, 1].min() - 1, stop=x_set[:, 1].max() + 1, step=0.01))

plt.contourf(x1, x2, classifier.predict(np.array([x1.ravel(), x2.ravel()]).T).reshape(x1.shape),

          alpha=0.75, cmap=ListedColormap(('purple', 'green')))

plt.xlim(x1.min(), x1.max())

plt.ylim(x2.min(), x2.max())

for i, j in enumerate(np.unique(y_set)):

   plt.scatter(x_set[y_set == j, 0], x_set[y_set == j, 1],

            c=ListedColormap(('purple', 'green'))(i), label=j)

plt.title('Decision Tree Algorithm (Training set)')

plt.xlabel('Age')

plt.ylabel('Estimated Salary')

plt.legend()

    plt.show()
```

11. Visualize the Test Set Results

```
# Visualize the test set results
```

```
x_set, y_set = x_test, y_test

x1, x2 = np.meshgrid(np.arange(start=x_set[:, 0].min() - 1, stop=x_set[:, 0].max() + 1,
    step=0.01),

                np.arange(start=x_set[:, 1].min() - 1, stop=x_set[:, 1].max() + 1, step=0.01))

plt.contourf(x1, x2, classifier.predict(np.array([x1.ravel(), x2.ravel()]).T).reshape(x1.shape),

          alpha=0.75, cmap=ListedColormap(('purple', 'green')))

plt.xlim(x1.min(), x1.max())

plt.ylim(x2.min(), x2.max())

for i, j in enumerate(np.unique(y_set)):

   plt.scatter(x_set[y_set == j, 0], x_set[y_set == j, 1],

            c=ListedColormap(('purple', 'green'))(i), label=j)

plt.title('Decision Tree Algorithm (Test set)')

plt.xlabel('Age')

plt.ylabel('Estimated Salary')

plt.legend()

    plt.show()
```

QUESTIONS

1. What is the purpose of the Decision Tree algorithm in classification?
2. What do the nodes and edges in the decision tree visualization represent?
3. Analyze the confusion matrix. How many predictions were correct and incorrect?
4. How does feature scaling affect the decision boundary visualization?
5. Compare the training and test set visualizations. Do you notice overfitting or underfitting?
6. How does the accuracy score reflect the model’s performance?

CONCUSION

In this assignment, we implemented a Decision Tree Classification Algorithm, visualized its structure, and analyzed the decision boundaries. The decision tree visualization provided insights into the splits and decision-making process of the model. Metrics like accuracy and confusion matrix allowed for a quantitative evaluation of the model's predictions.

ASSIGNMENT NO. 6

TITLE

Implement Naïve Bayes Classifier on data set of your choice. Test and Compare for Accuracy and Precision.

PROBLEM STATEMENT

Implement a Naive Bayes Classifier Algorithm using a dataset (user_data.csv) containing user information. Visualize the decision boundaries for the training and test sets, evaluate the model's performance using metrics, and analyze the results.

OBJECTIVE

- Import and preprocess the dataset for classification analysis.
- Split the dataset into Training and Test sets.
- Train the model using the Naive Bayes Classification Algorithm.
- Predict outcomes for the test set.
- Evaluate model performance using the confusion matrix.
- Visualize the training and test set results.
- Analyze the decision boundaries of the Naive Bayes model.

PLATFORM REQUIRED

Operating System: Windows or Linux

Software/Tools: Anaconda, Spyder, Jupyter Notebook, Google Colab

THEORY

Naïve Bayes Classifier Algorithm

Naïve Bayes algorithm is a supervised learning algorithm, which is based on Bayes theorem and used for solving classification problems. It is mainly used in text classification that includes a high

dimensional training dataset. 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. It is a probabilistic classifier, which means it predicts on the basis of the probability of an object.

Some popular examples of Naïve Bayes Algorithm are spam filtration, Sentimental analysis, and classifying articles.

The Naïve Bayes algorithm is comprised of two words Naïve and Bayes, Which can be described as:

Naïve: It is called Naïve because it assumes that the occurrence of a certain feature is independent of the occurrence of other features. Such as if the fruit is identified on the bases of color, shape, and taste, then red, spherical, and sweet fruit is recognized as an apple. Hence each feature individually contributes to identify that it is an apple without depending on each other.

Bayes: It is called Bayes because it depends on the principle of Bayes' Theorem**.**

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:

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

Where,

P(A|B) is Posterior probability: Probability of hypothesis A on the observed event B.

P(B|A) is Likelihood probability: Probability of the evidence given that the probability of a hypothesis is true.

P(A) is Prior Probability: Probability of hypothesis before observing the evidence.

P(B) is Marginal Probability: Probability of Evidence.

Types of Naïve Bayes Model

There are three types of naïve bayes model, which are given below:

- **Gaussian –** The Gaussian model assumes that features follow a normal distribution. This means if predictors take continuous values instead of discrete, then the model assumes that these values are sampled from the Gaussian distribution.
- **Multinomial -** The Multinomial Naïve Bayes classifier is used when the data is multinomial distributed. It is primarily used for document classification problems, it means a particular document belongs to which category such as Sports, Politics, education, etc. The classifier uses the frequency of words for the predictors.
- **Bernoulli -** The Bernoulli classifier works similar to the Multinomial classifier, but the predictor variables are the independent Booleans variables. Such as if a particular word is present or not in a document. This model is also famous for document classification tasks.

ALGORITHM

1. Data preprocessing
2. Import necessary libraries

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

3. Load the dataset and extract features

```
dataset = pd.read_csv('user_data.csv')  # Replace with the correct dataset path
x = dataset.iloc[:, [2, 3]].values  # Features: Age and Estimated Salary
y = dataset.iloc[:, 4].values       # Target: Purchased
```

4. Split the dataset into training and test set

```
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.25, random_state=0)
```

5. Apply feature scaling for better performance

```
from sklearn.preprocessing import StandardScaler

sc = StandardScaler()

x_train = sc.fit_transform(x_train)

x_test = sc.transform(x_test)
```

6. Fit a Gaussian Naive Bayes classifier to the training set

```
from sklearn.naive_bayes import GaussianNB

classifier = GaussianNB()

classifier.fit(x_train, y_train)
```

7. Use the trained model to predict outcomes for the test set.

```
y_pred = classifier.predict(x_test)
```

8. Evaluate model performance using a confusion matrix.

```
from sklearn.metrics import confusion_matrix, accuracy_score

cm = confusion_matrix(y_test, y_pred)

accuracy = accuracy_score(y_test, y_pred)
```

9. Calculate the accuracy score.

```
print("Confusion Matrix:\n", cm)

print("Accuracy:", accuracy)
```

10. Visualize decision boundaries for both training and test sets.

```
# Visualize training set results

from matplotlib.colors import ListedColormap

x_set, y_set = x_train, y_train
```

```
X1, X2 = np.meshgrid(np.arange(start=x_set[:, 0].min() - 1, stop=x_set[:, 0].max() + 1, step=0.01), np.arange(start=x_set[:, 1].min() - 1, stop=x_set[:, 1].max() + 1, step=0.01))

plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(), X2.ravel()]).T).reshape(X1.shape), alpha=0.75, cmap=ListedColormap(('purple', 'green')))

plt.xlim(X1.min(), X1.max())

plt.ylim(X2.min(), X2.max())

for i, j in enumerate(np.unique(y_set)):

plt.scatter(x_set[y_set == j, 0], x_set[y_set == j, 1], c=ListedColormap(('purple', 'green'))(i), label=j)
plt.title('Naive Bayes (Training set)')

plt.xlabel('Age')

 plt.ylabel('Estimated Salary')

plt.legend()

plt.show()

# Visualize testing set results

x_set, y_set = x_test, y_test

X1, X2 = np.meshgrid(np.arange(start=x_set[:, 0].min() - 1, stop=x_set[:, 0].max() + 1, step=0.01), np.arange(start=x_set[:, 1].min() - 1, stop=x_set[:, 1].max() + 1, step=0.01))

plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(), X2.ravel()]).T).reshape(X1.shape), alpha=0.75, cmap=ListedColormap(('purple', 'green')))

plt.xlim(X1.min(), X1.max())

plt.ylim(X2.min(), X2.max())

for i, j in enumerate(np.unique(y_set)): plt.scatter(x_set[y_set == j, 0], x_set[y_set == j, 1], c=ListedColormap(('purple', 'green'))(i), label=j)

plt.title('Naive Bayes (Test set)')

plt.xlabel('Age')

plt.ylabel('Estimated Salary')

plt.legend() plt.show()
```

QUESTIONS

1. What are the key assumptions of the Naive Bayes algorithm?
2. Why is feature scaling necessary in this implementation?
3. Analyze the confusion matrix. How many predictions were correct and incorrect?
4. Compare the decision boundaries for training and test sets. What do you observe?
5. How does the accuracy score reflect the model's performance?
6. Can this algorithm handle categorical features? If yes, how?

CONCLUSION

In this assignment, we implemented the Naive Bayes Classification Algorithm using the user_data.csv dataset. The classifier performed well, as evidenced by the confusion matrix and accuracy score. Visualizations of decision boundaries provided insight into the classifier's behavior. Despite its simplicity, Naive Bayes is a powerful tool for classification tasks, especially with Gaussian-distributed data.

ASSIGNMENT NO. 7

TITLE

Implement K Nearest Neighbor Classification

PROBLEM STATEMENT

Implement a K-Nearest Neighbors (KNN) Classification Algorithm using a dataset (user_data.csv) containing user information. Visualize decision boundaries, evaluate the model's performance, and interpret the results.

OBJECTIVE

- Import and preprocess the dataset for classification analysis.
- Split the dataset into Training and Test sets.
- Train the model using a K-Nearest Neighbors (KNN) Classifier.
- Predict outcomes on the test set and evaluate model performance.
- Visualize decision boundaries for the training and test sets.
- Interpret and analyze the results.

PLATFORM REQUIRED

Operating System: Windows or Linux

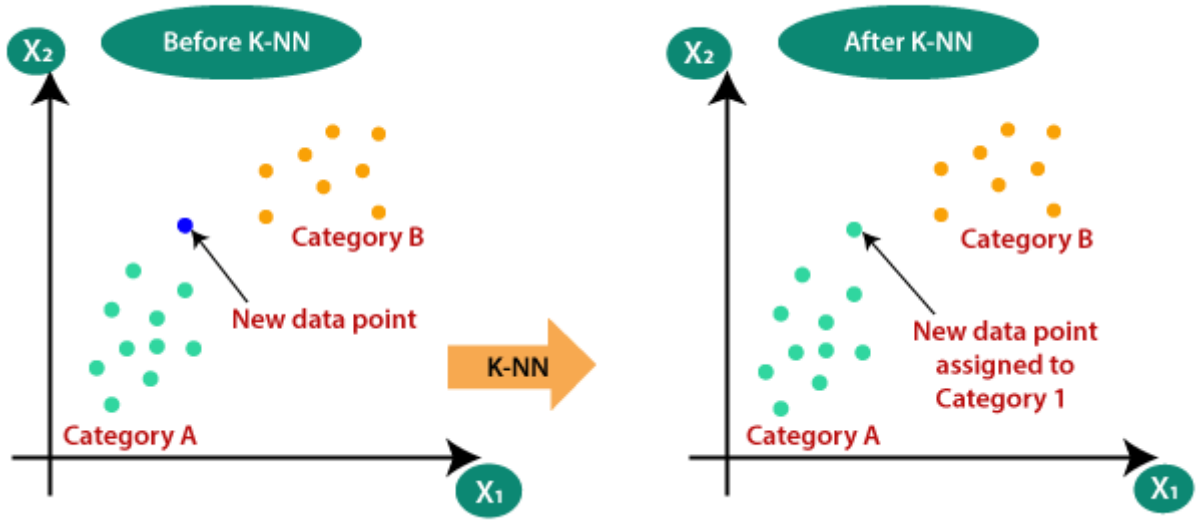
Software/Tools: Anaconda, Spyder, Jupyter Notebook, Google Colab

THEORY

K-Nearest Neighbor(KNN) Algorithm for Machine Learning

- K-Nearest Neighbour is one of the simplest Machine Learning algorithms based on Supervised Learning technique.
- 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.

- 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.
- K-NN algorithm can be used for Regression as well as for Classification but mostly it is used for the Classification problems.
- K-NN is a non-parametric algorithm, which means it does not make any assumption on underlying data.
- 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.
- 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.



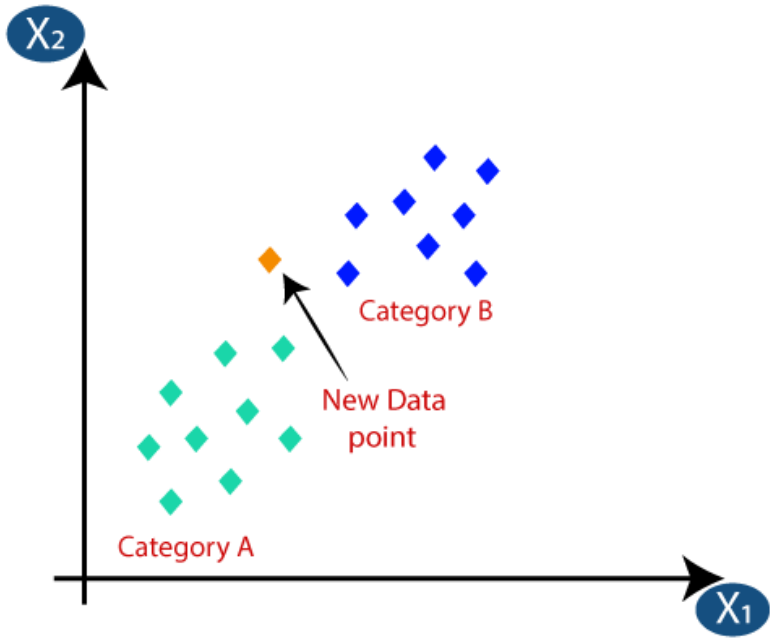
Suppose there are two categories, i.e., Category A and Category B, and we have a new data point x1, so this data point will lie in which of these categories. To solve this type of problem, we need a K-NN algorithm. With the help of K-NN, we can easily identify the category or class of a particular dataset.

KNN ALGORITHM

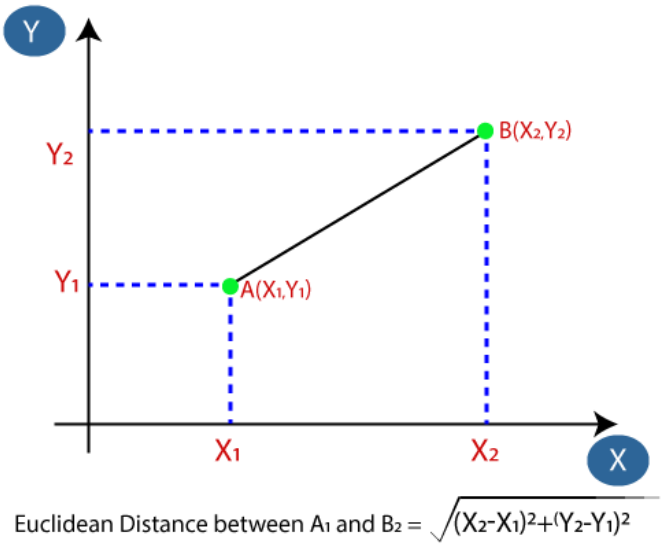
1. Select the number K of the neighbors
2. Calculate the Euclidean distance of K number of neighbors
3. Take the K nearest neighbors as per the calculated Euclidean distance.

4. Among these k neighbors, count the number of the data points in each category.
5. Assign the new data points to that category for which the number of the neighbor is maximum.

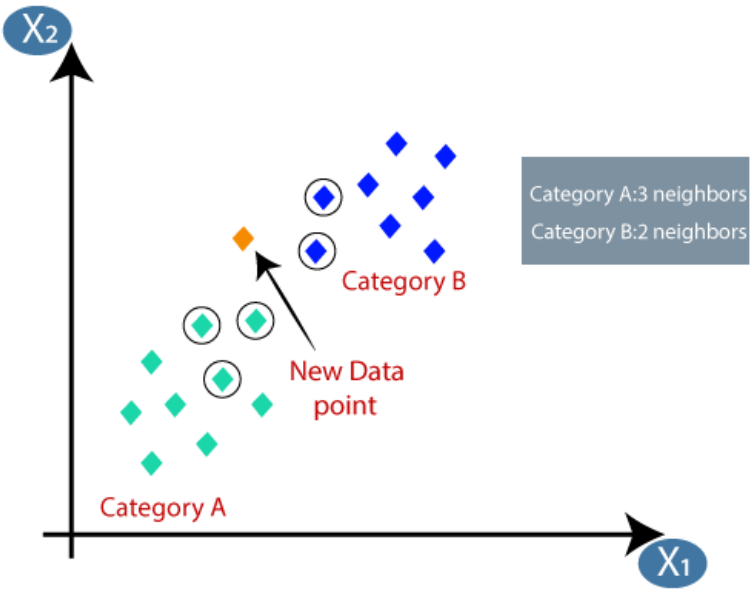
Suppose we have a new data point and we need to put it in the required category. Consider the below image:



- Firstly, we will choose the number of neighbors, so we will choose the k=5
- Next, we will calculate the Euclidean distance between the data points. The Euclidean distance is the distance between two points, which we have already studied in geometry. It can be calculated as:



By calculating the Euclidean distance we got the nearest neighbors, as three nearest neighbors in category A and two nearest neighbors in category B. Consider the below image:



As we can see the 3 nearest neighbors are from category A, hence this new data point must belong to category A.

IMPLEMENTATION OF KNN ALGORITHM

1. Import Necessary Libraries

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
```

2. Import the dataset

```
dataset = pd.read_csv('user_data.csv')
```

3. Extracting independent and dependant variables

```
X = dataset.iloc[:, [2, 3]].values
y = dataset.iloc[:, 4].values
```

4. Splitting the dataset into training and test sets

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=0)
```

5. Feature Scaling

```
from sklearn.preprocessing import StandardScaler

sc = StandardScaler()

X_train = sc.fit_transform(X_train)

X_test = sc.transform(X_test)
```

6. Training the K-NN model

```
from sklearn.neighbors import KNeighborsClassifier

classifier = KNeighborsClassifier(n_neighbors=5, metric='minkowski', p=2)

classifier.fit(X_train, y_train)
```

7. Predicting the test set results

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

8. Evaluating the model

```
from sklearn.metrics import confusion_matrix, accuracy_score

cm = confusion_matrix(y_test, y_pred)

accuracy = accuracy_score(y_test, y_pred)

print("Confusion Matrix:\n", cm)

print("Accuracy:", accuracy)
```

9. Visualizing the training set results

```
from matplotlib.colors import ListedColormap

X_set, y_set = X_train, y_train

X1, X2 = np.meshgrid(np.arange(start=X_set[:, 0].min() - 1, stop=X_set[:, 0].max() + 1, step=0.01),
np.arange(start=X_set[:, 1].min() - 1, stop=X_set[:, 1].max() + 1, step=0.01))

plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(), X2.ravel()]).T).reshape(X1.shape),
alpha=0.75, cmap=ListedColormap(('red', 'green')))

plt.xlim(X1.min(), X1.max())

plt.ylim(X2.min(), X2.max())

for i, j in enumerate(np.unique(y_set)):

plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1], c=ListedColormap(('red', 'green'))(i), label=j)
plt.title('K-NN (Training set)')
```

```
plt.xlabel('Age')

plt.ylabel('Estimated Salary')

plt.legend()

plt.show()
```

10. Visualizing the test set results

```
X_set, y_set = X_test, y_test

X1, X2 = np.meshgrid(np.arange(start=X_set[:, 0].min() - 1, stop=X_set[:, 0].max() + 1, step=0.01), np.arange(start=X_set[:, 1].min() - 1, stop=X_set[:, 1].max() + 1, step=0.01))

plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(), X2.ravel()]).T).reshape(X1.shape), alpha=0.75, cmap=ListedColormap(('red', 'green')))

plt.xlim(X1.min(), X1.max())

plt.ylim(X2.min(), X2.max())

for i, j in enumerate(np.unique(y_set)):

plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1], c=ListedColormap(('red', 'green'))(i), label=j)
plt.title('K-NN (Test set)')

plt.xlabel('Age')

plt.ylabel('Estimated Salary')

plt.legend()

plt.show()
```

CONCLUSION

In this assignment, the K-Nearest Neighbor algorithm was implemented on the user_data.csv dataset. The classifier's performance was evaluated using accuracy and the confusion matrix. Visualizations of decision boundaries provided insight into the behavior of the KNN algorithm. The results demonstrated the importance of feature scaling, parameter tuning, and dataset quality in achieving optimal performance.

ASSIGNMENT NO. 8

TITLE

Implement Support Vector Machine Algorithm

PROBLEM STATEMENT

Implement the Support Vector Machine (SVM) algorithm on the Iris dataset. Evaluate the model's performance using four different kernels (linear, polynomial, radial basis function (RBF), and sigmoid). Compare the accuracy and decision boundaries of each kernel and analyze their behavior on the dataset.

OBJECTIVE

- Import and preprocess the Iris dataset for classification analysis.
- Split the dataset into training and test sets.
- Train the model using the Support Vector Machine (SVM) algorithm with different kernel functions.
- Predict outcomes on the test set and evaluate the model's performance.
- Visualize decision boundaries for each kernel type.
- Compare the accuracy and interpret the results.

PLATFORM REQUIRED

Operating System: Windows or Linux

Software/Tools: Anaconda, Spyder, Jupyter Notebook, Google Colab

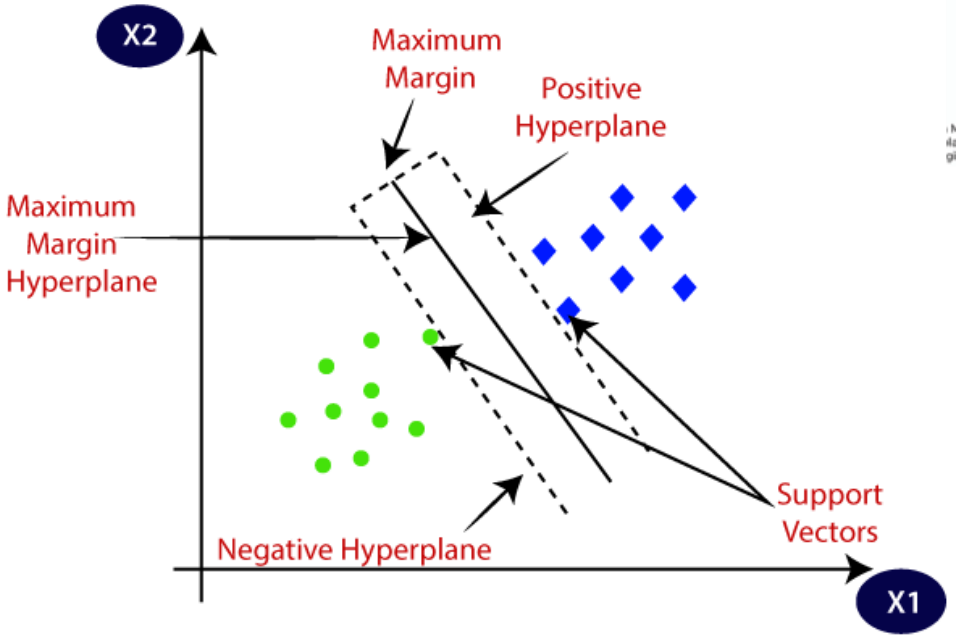
THEORY

Support Vector Machine (SVM) Algorithm

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

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

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



- **Support Vectors:** These are the points that are closest to the hyperplane. A separating line will be defined with the help of these data points.
- **Margin:** It is the distance between the hyperplane and the observations closest to the hyperplane (support vectors). In SVM large margin is considered a good margin. There are two types of margins hard margin and soft margin. I will talk more about these two in the later section.

Types of SVM

1. Linear SVM
2. Non Linear SVM

Linear SVM

Linear SVM is used for linearly separable data, which means if a dataset can be classified into two classes by using a single straight line, then such data is termed as linearly separable data, and classifier is used called as Linear SVM classifier.

Non Linear SVM

Non-Linear SVM is used for non-linearly separated data, which means if a dataset cannot be

classified by using a straight line, then such data is termed as non-linear data and classifier used is called as Non-linear SVM classifier.

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.

Popular kernel functions in SVM

The SVM kernel is a function that takes low-dimensional input space and transforms it into higher-dimensional space, ie it converts nonseparable problems to separable problems.

It is mostly useful in non-linear separation problems. Simply put the kernel, does some extremely complex data transformations and then finds out the process to separate the data based on the labels or outputs defined.

$$\text{Linear}: K(w,b) = w^T x + b$$
$$\text{Polynomial}: K(w,x) = (\gamma w^T x + b)^N$$
$$\text{Gaussian RBF}: K(w,x) = \exp(-\gamma ||x_i - x_j||^n$$
$$\text{Sigmoid}: K(x_i, x_j) = \tanh(\alpha x_i^T x_j + b)$$

STEP BY STEP ALGORITHM

1. Import Necessary Libraries

   ```
   import numpy as np
   import matplotlib.pyplot as plt
   import pandas as pd
   from sklearn import datasets
   from sklearn.model_selection import train_test_split
   from sklearn.preprocessing import StandardScaler
   from sklearn.svm import SVC
   from sklearn.metrics import accuracy_score, confusion_matrix
   from mlxtend.plotting import plot_decision_regions
   ```

2. Load the Iris Dataset

   ```
   iris = datasets.load_iris()
   X = iris.data[:, :2]  # Taking first two features for visualization
   ```

```
y = iris.target
```

3. Splitting the dataset into training and test sets

```
y_test = train_test_split(X, y, test_size=0.25, random_state=0)
```

4. Feature Scaling

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

5. Training the SVM model with different kernels

```
kernels = ['linear', 'poly', 'rbf', 'sigmoid']
models = {}
for kernel in kernels:
    model = SVC(kernel=kernel)
    model.fit(X_train, y_train)
    models[kernel] = model
```

6. Predicting the test set results and evaluating performance

```
for kernel, model in models.items():
    y_pred = model.predict(X_test)
    accuracy = accuracy_score(y_test, y_pred)
    cm = confusion_matrix(y_test, y_pred)
    print(f"Kernel: {kernel}")
    print("Confusion Matrix:\n", cm)
    print("Accuracy:", accuracy, "\n")
```

7. Visualizing decision boundaries

```
    for kernel, model in models.items():
plt.figure(figsize=(8, 6))
plot_decision_regions(X_train, y_train, clf=model, legend=2)
plt.title(f"Decision Boundary - {kernel} Kernel")
plt.xlabel(iris.feature_names[0])
plt.ylabel(iris.feature_names[1])
plt.show()
```

```
for kernel, model in models.items():
    plt.figure(figsize=(8, 6))
    plot_decision_regions(X_train, y_train, clf=model, legend=2)
    plt.title(f"Decision Boundary - {kernel} Kernel")
    plt.xlabel(iris.feature_names[0])
    plt.ylabel(iris.feature_names[1])
    plt.show()
```

QUESTIONS

1. What is SVM used for?
2. What are the different types of kernels in SVM?
3. Why do we need feature scaling in SVM?
4. How does the linear kernel work in SVM?
5. What is the difference between RBF and polynomial kernels?
6. What role do support vectors play in SVM?
7. Why is the sigmoid kernel not commonly used?
8. What are the advantages of using the RBF kernel?
9. How does SVM find the optimal hyperplane?
10. How does the choice of kernel impact classification accuracy?

CONCLUSION

In this assignment, the Support Vector Machine (SVM) algorithm is implemented on the Iris dataset using four different kernels: linear, polynomial, RBF, and sigmoid. The models were trained, tested, and evaluated based on accuracy and decision boundaries. The visualization of decision boundaries helped in understanding the classification behavior of each kernel. The results highlight the importance of choosing an appropriate kernel function based on data distribution for achieving optimal classification performance.

ASSIGNMENT NO. 9

TITLE

Implementation of K-Means Clustering Algorithm

PROBLEM STATEMENT

Implement the K-Means clustering algorithm to classify a dataset based on given features. Use a dataset with two variables (VAR1 and VAR2) and classify data points into three clusters (k=3). Visualize the clustering results and predict the classification for a new data point (VAR1=0.906, VAR2=0.606) based on the trained model.

VAR1	VAR2	CLASS
1.713	1.586	0
0.180	1.786	1
0.353	1.240	1
0.940	1.566	0
1.486	0.759	1
1.266	1.106	0
1.540	0.419	1
0.459	1.799	1
0.773	0.186	1

OBJECTIVE

- Load and preprocess the given dataset.
- Implement K-Means clustering with k = 3.
- Visualize the clusters and centroids.
- Predict the cluster for a new point using the trained model.
- Analyze the effectiveness of clustering based on the visualization.

PLATFORM REQUIRED

Operating System: Windows or Linux

Software/Tools: Anaconda, Spyder, Jupyter Notebook, Google Colab

THEORY

K-Means Clustering Algorithm

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

What is K-Means Algorithm?

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

It is an iterative algorithm that divides the unlabeled dataset into k different clusters in such a way that each dataset belongs only one group that has similar properties. It allows us to cluster the data into different groups and a convenient way to discover the categories of groups in the unlabeled dataset on its own without the need for any training. It is a centroid-based algorithm, where each cluster is associated with a centroid. The main aim of this algorithm is to minimize the sum of distances between the data point and their corresponding clusters.

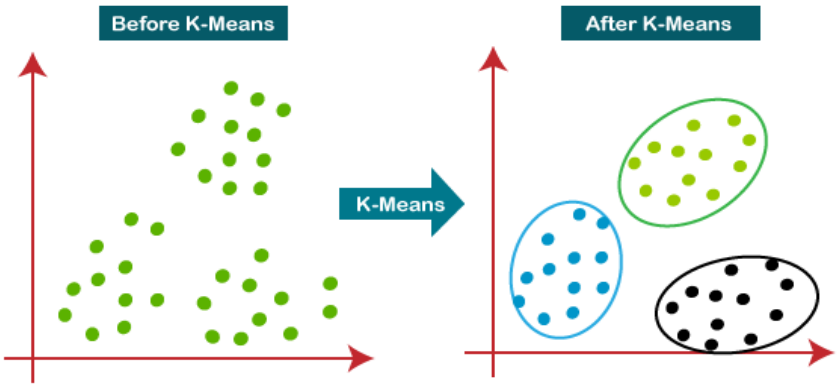
The algorithm takes the unlabeled dataset as input, divides the dataset into k-number of clusters, and repeats the process until it does not find the best clusters. The value of k should be predetermined in this algorithm.

The k-means clustering algorithm mainly performs two tasks:

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

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

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



Working of K-Means Algorithm

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

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

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

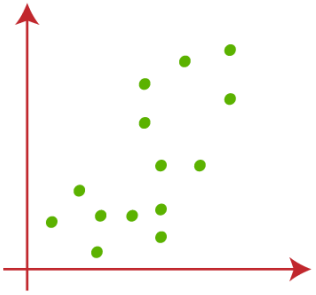
Step 4 : Calculate the variance and place a new centroid of each cluster.

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

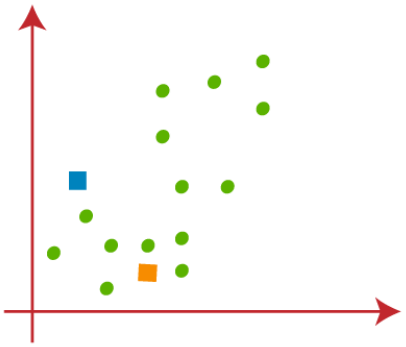
Step 6 : If any reassignment occurs, then go to step-4 else go to FINISH.

Step 7 : The model is ready.

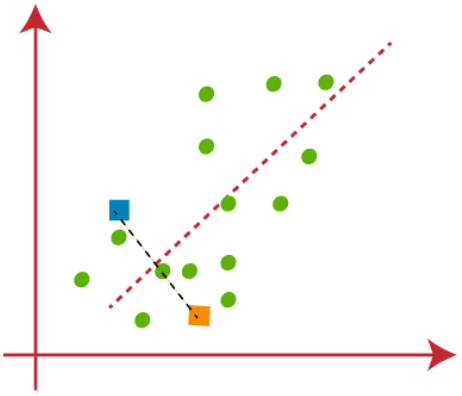
Suppose we have two variables M1 and M2. The x-y axis scatter plot of these two variables is given below:



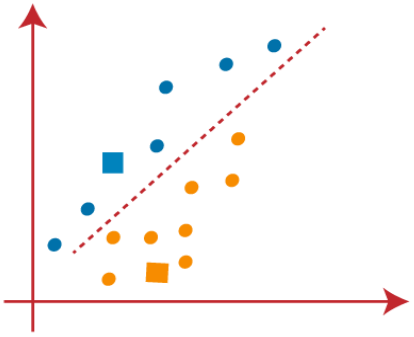
- Let's take number k of clusters, i.e., K=2, to identify the dataset and to put them into different clusters. It means here we will try to group these datasets into two different clusters.
- We need to choose some random k points or centroid to form the cluster. These points can be either the points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



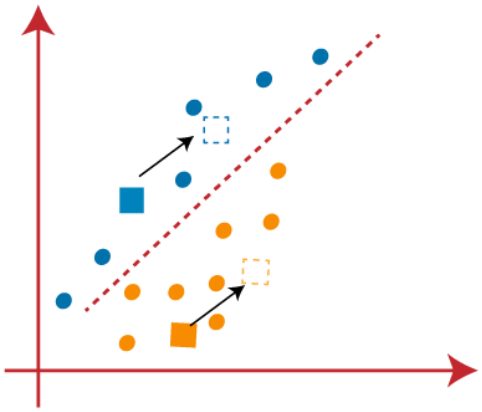
Now we will assign each data point of the scatter plot to its closest K-point or centroid. We will compute it by applying some mathematics that we have studied to calculate the distance between two points. So, we will draw a median between both the centroids.



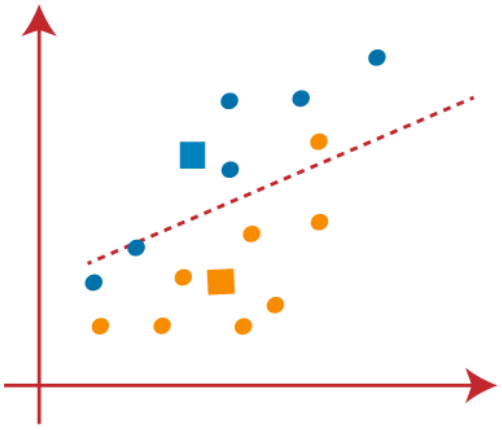
From the above image, it is clear that points left side of the line is near to the K1 or blue centroid, and points to the right of the line are close to the yellow centroid. Let's color them as blue and yellow for clear visualization.



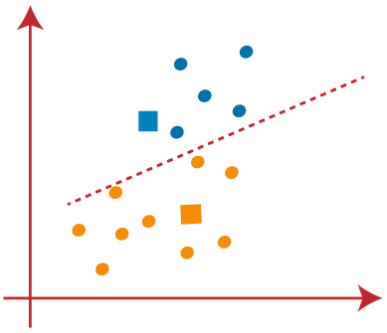
As we need to find the closest cluster, so we will repeat the process by choosing **a new centroid**. To choose the new centroids, we will compute the center of gravity of these centroids, and will find new centroids as below:



Next, we will reassign each datapoint to the new centroid. For this, we will repeat the same process of finding a median line. The median will be like below image



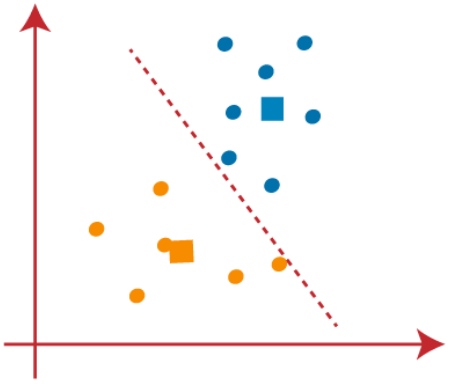
From the above image, we can see, one yellow point is on the left side of the line, and two blue points are right to the line. So, these three points will be assigned to new centroids.



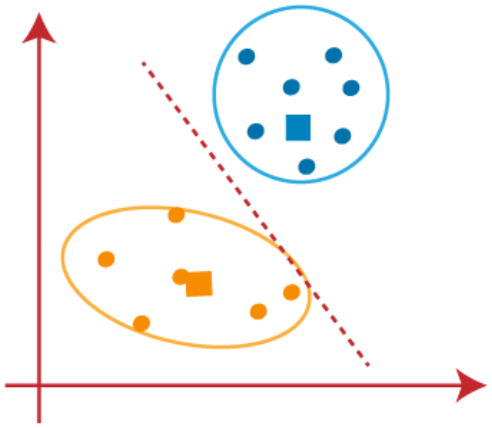
As reassignment has taken place, so we will again go to the step-4, which is finding new centroids or K-points.

We will repeat the process by finding the center of gravity of centroids, so the new centroids will be as shown in the below image:

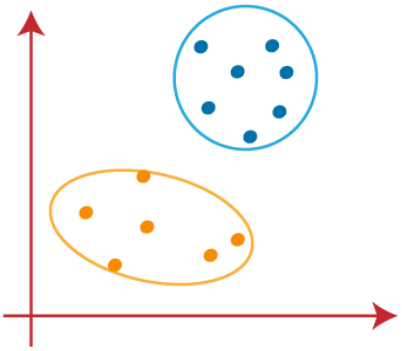
As we got the new centroids so again will draw the median line and reassign the data points. So, the image will be:



We can see in the above image; there are no dissimilar data points on either side of the line, which means our model is formed. Consider the below image:



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



ALGORITHM

1. Import the required libraries

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
```

2. Input the dataset manually - Create a dictionary or DataFrame with values of VAR1 and VAR2

```
data = {
```

'VAR1': [1.713, 0.180, 0.353, 0.940, 1.486, 1.266, 1.540, 0.459, 0.773],

'VAR2': [1.586, 1.786, 1.240, 1.566, 0.759, 1.106, 0.419, 1.799, 0.186]

}

df = pd.DataFrame(data)

3. Extract features for clustering - Convert the VAR1 and VAR2 columns into a NumPy array X

X = df[['VAR1', 'VAR2']].values

4. Apply K-Means clustering:

   - Initialize the KMeans object with k = 3 clusters
   - Fit the model to the feature matrix X
   - Predict cluster labels for all data points
   - Extract and store cluster centroids

```
kmeans = KMeans(n_clusters=3, random_state=0)
kmeans.fit(X)
labels = kmeans.labels_
centroids = kmeans.cluster_centers_
```

5. Visualize the clustering result:

   - Use matplotlib.pyplot.scatter() to plot the data points
   - Color-code points based on their assigned cluster
   - Plot the centroids using a different marker (e.g., red 'X')
   - Label axes and add title/legend

```
plt.scatter(X[:, 0], X[:, 1], c=labels, cmap='viridis')
plt.scatter(centroids[:, 0], centroids[:, 1], c='red', marker='X', label='Centroids')
plt.title('K-Means Clustering (k=3)')
plt.xlabel('VAR1')
plt.ylabel('VAR2')
plt.legend()
plt.grid(True)
```

```
plt.show()
```

6. Predict the cluster for a new data point

- Input the new data point (e.g., [0.906, 0.606])
- Use kmeans.predict() to find its assigned cluster
- Display the predicted cluster number for the new point

```
new_point = np.array([[0.906, 0.606]])
predicted_cluster = kmeans.predict(new_point)
print("New data point [0.906, 0.606] belongs to cluster:", predicted_cluster[0])
```

CONCLUSION

In this assignment, the K-Means clustering algorithm was successfully applied to a dataset with two features. The data points were grouped into three clusters and the model was used to predict the cluster assignment of a new point. The visualization of clusters helped in understanding the separation and grouping of data based on feature similarity. This demonstrated the practical application of unsupervised learning.

ASSIGNMENT NO. 10

TITLE

Unsupervised Learning: Implement K-Means Clustering and Hierarchical clustering on proper data set of your choice. Compare their Convergence

PROBLEM STATEMENT

Apply K-Means and Hierarchical Clustering algorithms on a dataset of your choice (e.g., customer segmentation, mall customer dataset, or iris dataset). Visualize the clustering results, compare how both methods form clusters, and analyze their convergence behavior.

OBJECTIVE

- To understand and apply Unsupervised Learning techniques.
- Implement and visualize K-Means Clustering and Hierarchical Clustering.
- Compare the convergence process and cluster formation of both algorithms.
- Analyze the strengths and weaknesses of both clustering methods.

PLATFORM REQUIRED

Operating System: Windows or Linux

Software/Tools: Anaconda, Spyder, Jupyter Notebook, Google Colab

THEORY

K-Means Clustering Algorithm

- A partitioning algorithm that divides data into k clusters.
- Minimizes intra-cluster variance and updates centroids iteratively.
- Convergence occurs when centroids stop changing significantly.

Hierarchical Clustering

- A tree-based (dendrogram) algorithm that builds clusters by:
    - Agglomerative approach (bottom-up)
    - Divisive approach (top-down)

- Does not require specifying the number of clusters in advance.
- Convergence occurs when all data is combined into one cluster or desired number is reached.
- Convergence Comparison

    - K-Means converges faster (less computationally expensive).
    - Hierarchical clustering gives more interpretable results (dendrogram).
    - K-Means requires k to be known; Hierarchical does not.

Working of K-Means and Hierarchical Clustering Algorithm

K Means Clustering

Step 1 : Choose the number of clusters k

Step 2 : Randomly initialize centroids

Step 3 : Assign each point to the nearest centroid

Step 4 : Recalculate centroids based on current cluster members.

Step 5 : Repeat steps 3 and 4 until convergence (no change in centroids)

Hierarchical Clustering (Agglomerative)

Step 1 : Treat each point as its own cluster

Step 2 : Compute pairwise distances between all clusters

Step 3 : Merge the two closest clusters

Step 4 : Update the distance matrix

Step 5 : Repeat steps 2–4 until one single cluster remains

Step 6: Cut the dendrogram at desired cluster level

ALGORITHM

K – Means Clustering Algorithm

1. Import the libraries

import pandas as pd

import matplotlib.pyplot as plt

```
from sklearn.cluster import KMeans

from sklearn.preprocessing import StandardScaler

from scipy.cluster.hierarchy import dendrogram, linkage

from sklearn.datasets import load_iris

import seaborn as sns
```

2. Import the dataset Preprocess the features (e.g., feature scaling).

```
data = load_iris()

df = pd.DataFrame(data.data, columns=data.feature_names)

X = StandardScaler().fit_transform(df)
```

3. Apply K- means clustering

- Choose the number of clusters k (e.g., k = 3).
- Initialize k centroids randomly.
- Assign each data point to the nearest centroid using Euclidean distance.
- Compute the new centroids by taking the mean of all points assigned to each cluster.
- Repeat steps 5 and 6 until centroids do not change significantly (i.e., convergence).
- Output final cluster assignments.

```
kmeans = KMeans(n_clusters=3, random_state=42)

kmeans_labels = kmeans.fit_predict(X)
```

4. Visualize the clusters using scatter plots.

```
plt.figure(figsize=(6, 5))

sns.scatterplot(x=X[:, 0], y=X[:, 1], hue=kmeans_labels, palette='viridis')

plt.title("K-Means Clustering (k=3)")

plt.xlabel("Feature 1")

plt.ylabel("Feature 2")
```

```
plt.grid(True)

plt.show()
```

Hierarchical Clustering Algorithm

5. Apply Hierarchical clustering

   - Treat each data point as a separate cluster (initially, n clusters).
   - Compute the distance matrix (e.g., using Euclidean distance).
   - Find the two closest clusters based on a linkage method (e.g., Ward's method).
   - Merge these two clusters into one.
   - Update the distance matrix based on the new cluster structure.
   - Repeat steps until only one cluster remains or desired number of clusters is reached.

```
linked = linkage(X, method='ward')
```

6. Plot the dendrogram to visualize hierarchical relationships.

```
plt.figure(figsize=(8, 4))

dendrogram(linked, orientation='top', distance_sort='descending', show_leaf_counts=False)

plt.title("Hierarchical Clustering Dendrogram")

 plt.show()
```

CONCLUSION

In this assignment, K-Means and Hierarchical clustering algorithms were successfully implemented on the selected dataset. K-Means converged faster but required specifying the number of clusters in advance. Hierarchical clustering provided a clear visual hierarchy using the dendrogram, though it was computationally more intensive. This comparison highlighted the strengths and limitations of both techniques, aiding in selecting the right method based on data size and interpretability.

ASSIGNMENT NO. 11

TITLE

Study and Implement Bagging/Boosting using Random Forests.

PROBLEM STATEMENT

Implement a Random Forest classifier, an ensemble method based on Bagging, to classify a dataset using decision trees. Train the model, evaluate its performance using accuracy and confusion matrix, and visualize feature importance or decision regions. Analyze how bagging improves performance and reduces overfitting compared to a single decision tree.

OBJECTIVE

- Understand the concept of Bagging and Random Forests.
- Apply Random Forest Classification on a suitable dataset.
- Evaluate the performance using confusion matrix and accuracy.
- Visualize and interpret feature importance and classification results.
- Understand the difference between single decision trees and ensemble methods.

PLATFORM REQUIRED

Operating System: Windows or Linux

Software/Tools: Anaconda, Spyder, Jupyter Notebook, Google Colab

THEORY

Bagging (Bootstrap Aggregating)

- An ensemble technique that trains multiple models on random subsets of the data (with replacement) and averages their predictions.
- Reduces variance and avoids overfitting.

Random Forest

- A Bagging-based ensemble method that builds multiple decision trees and combines their outputs using majority voting (for classification).
- Each tree is trained on a bootstrapped sample of the dataset and uses random subsets of features.

- Provides feature importance, handles large datasets, and works well for both classification and regression.

Working of Random Forest Classifier

Random Forest is an ensemble learning method based on Bagging (Bootstrap Aggregating). It combines multiple decision trees to improve classification (or regression) performance. Each tree is trained on a random sample of the data (with replacement), and during prediction, majority voting is used to classify new data points.

Step 1 : Random sampling: From the dataset (e.g., the Iris dataset), it creates multiple random subsets (bootstrapped datasets) for training individual trees.

Step 2 : Feature randomness: At each split in a tree, only a random subset of features is considered for splitting—this reduces correlation between trees.

Step 3 : Tree creation: Multiple decision trees are trained independently using the subsets.

Step 4 : Majority voting: For classification, each tree gives a prediction, and the final output is decided by majority vote across all trees.

Step 5 : Model evaluation: Accuracy and confusion matrix are used to assess performance.

Step 6: Feature importance: Based on how often and how significantly each feature contributes to tree decisions.

ALGORITHM

1. Import the libraries

```
import pandas as pd

from sklearn.datasets import load_iris

from sklearn.model_selection import train_test_split

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import confusion_matrix, accuracy_score

import matplotlib.pyplot as plt
```

```
import seaborn as sns
```

2. Load the Iris Dataset

```
data = load_iris()
```

3. Convert the dataset into a pandas DataFrame (X) for features and Store the target labels in variable y

```
X = pd.DataFrame(data.data, columns=data.feature_names)

y = data.target
```

4. Split the dataset into training and testing sets

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
```

5. Train Random Forest Model

```
clf = RandomForestClassifier(n_estimators=100, random_state=42)

clf.fit(X_train, y_train)
```

6. Predict the test set results and evaluate the model

```
y_pred = clf.predict(X_test)

cm = confusion_matrix(y_test, y_pred)

accuracy = accuracy_score(y_test, y_pred)
```

7. Print the results

```
print("Confusion Matrix:\n", cm)

print("Accuracy Score:", accuracy)
```

8. Evaluate the Feature Importance

```
importances = clf.feature_importances_

sns.barplot(x=importances, y=X.columns)
```

```
plt.title("Feature Importance")

plt.show()
```

CONCLUSION

In this assignment, we successfully implemented the Random Forest Classifier, an ensemble technique using Bagging. The model showed strong performance on the selected dataset and demonstrated better generalization than individual trees. Visualizing feature importance helped identify key predictors. This assignment highlights how ensemble methods like Random Forest improve accuracy and robustness of predictions.