

B. TECH (CS) (R23)

III -I

Machine Learning

Lab Manual

S No	Title of the Program
1	Compute Central Tendency Measures: Mean, Median, Mode Measure of Dispersion: Variance, Standard Deviation.
2	Apply the following Pre-processing techniques for a given dataset. a. Attribute selection b. Handling Missing Values c. Discretization d. Elimination of Outliers
3	Apply KNN algorithm for classification and regression
4	Demonstrate decision tree algorithm for a classification problem and perform parameter tuning for better results
5	Demonstrate decision tree algorithm for a regression problem
6	Apply Random Forest algorithm for classification and regression
7	Demonstrate Naïve Bayes Classification algorithm.
8	Apply Support Vector algorithm for classification
9	Demonstrate simple linear regression algorithm for a regression problem
10	Apply Logistic regression algorithm for a classification problem
11	Demonstrate Multi-layer Perceptron algorithm for a classification problem
12	Implement the K-means algorithm and apply it to the data you selected. Evaluate
13	Demonstrate the use of Fuzzy C-Means Clustering
14	Demonstrate the use of Expectation Maximization based clustering algorithm

Experiment-1

1. Compute Central Tendency Measures: Mean, Median, Mode Measure of Dispersion: Variance, Standard Deviation.

Aim:

To Compute Central Tendency Measures: Mean, Median, Mode Measure of Dispersion: Variance, Standard Deviation.

Procedure:

Central Tendency Measures:

1. Mean:

The sum of all values divided by the number of values. It represents the average value in a dataset.

Formula: Mean (μ) = $\Sigma x / N$, where Σx is the sum of all values and N is the number of values.

2. Median:

The middle value in a sorted dataset. If there's an even number of values, the median is the average of the two middle values.

To find: Arrange the data in ascending order and locate the middle value.

3. Mode:

The value that appears most frequently in a dataset.

To find: Count the occurrences of each value and identify the value with the highest frequency.

Measures of Dispersion:

1. Variance:

A measure of how spread out the data is from the mean. It's the average of the squared differences from the mean.

Formula: Variance (σ^2) = $\Sigma (x - \mu)^2 / N$, where x is each value, μ is the mean, and N is the number of values.

2. Standard Deviation:

The square root of the variance. It provides a measure of the spread of data in the original units.

Formula: Standard Deviation (σ) = $\sqrt{\text{Variance}}$.

It is often used to describe the typical distance between data points and the mean.

Program:

```
import statistics

def compute_statistics(data):
    print("Data:", data)
    # Central Tendency
    mean = statistics.mean(data)
    median = statistics.median(data)
    try:
```

```
mode = statistics.mode(data)
except statistics.StatisticsError:
mode = "No unique mode"
# Dispersion
variance = statistics.variance(data)
std_dev = statistics.stdev(data)
# Display results
print(f"\nCentral Tendency Measures:")
print(f"Mean : {mean}")
print(f"Median : {median}")
print(f"Mode : {mode}")
print(f"\nDispersion Measures:")
print(f"Variance : {variance}")
print(f"Standard Deviation : {std_dev}")
# Example usage
data = [10, 20, 20, 30, 40, 50, 60]
compute_statistics(data)
```

Output:

```
Data: [10, 20, 20, 30, 40, 50, 60]
Central Tendency Measures:
Mean : 32.857142857142854
Median : 30
Mode : 20
Dispersion Measures:
Variance : 323.8095238095238
Standard Deviation : 17.994708216848746
```

Experiment-2

2. Apply the following Pre-processing techniques for a given dataset. a. Attribute selection b. Handling Missing Values c. Discretization d. Elimination of Outliers

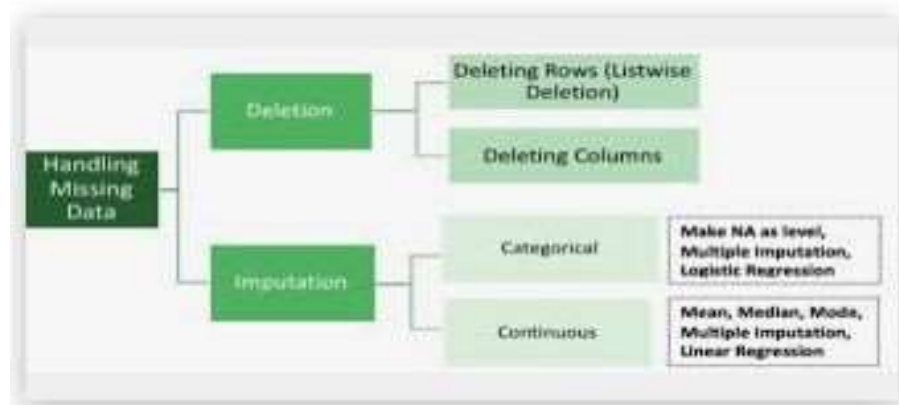
Aim:

To Apply the following Pre-processing techniques for a given dataset.

a. Attribute selection b. Handling Missing Values c. Discretization d. Elimination of Outliers

Procedure:

- ❖ **Attribute selection**, also known as **feature selection**, is the process of selecting a subset of relevant features (attributes) for use in model construction. It helps reduce dimensionality, improve model performance, and remove irrelevant or redundant data.
- ❖ **Handle missing data** two ways to use:



Deletion:

Delete the row with a missing value(s): It is a listwise deletion process in which we remove the entire row from the dataset which contains a lot of missing values.

Delete the column with a missing value(s): It is a listwise deletion process in which we remove the entire column from the dataset which contains a lot of missing values.

Imputing:

Imputation is a technique used for replacing the missing data with some substitute value to retain most of the data/information of the dataset.

Imputing with some other values: This is an important technique used in Imputation as it can handle both the Numerical and Categorical variables. This technique states that we group the missing values in a column and assign them to a new value that is near value or previous value or max value or min value.

Imputing with mean or medium values: One of the techniques is mean imputation in which the missing values are replaced with the mean value or medium value

- ❖ **Discretization** is the process of converting continuous data or numerical values into discrete categories or bins. This technique is often used in data analysis and machine learning to simplify

complex data and make it easier to analyse and work with. Instead of dealing with exact values, discretization groups the data into ranges and helps algorithms perform better especially in classification tasks. There are several types of discretization techniques used in data analysis to convert continuous data into discrete categories but mainly binning is used.

- ❖ **Removing outliers** from a dataset is the process of eliminating data points that significantly deviate from the general trend or distribution of the data. This is often done to improve the accuracy and reliability of data analysis and machine learning models, as outliers can disproportionately influence results.

Program:

```
import pandas as pd
import numpy as np
from sklearn.impute import SimpleImputer
from sklearn.feature_selection import SelectKBest, f_classif
from sklearn.preprocessing import KBinsDiscretizer
from scipy import stats
import seaborn as sns
import matplotlib.pyplot as plt

# Sample dataset (replace with your dataset)
data = {
    'Age': [25, 27, 29, np.nan, 31, 120, 23, 25],
    'Income': [50000, 54000, 58000, np.nan, 62000, 64000, 1000000, 58000],
    'Education': [1, 2, 2, 3, 1, 3, 2, 2],
    'Target': [0, 1, 0, 1, 0, 1, 0, 1]
}
df = pd.DataFrame(data)
print("Original Data:\n", df)

# --- a. Attribute Selection (Select best 2 features based on ANOVA F-test) ---
X = df.drop(columns=['Target'])
y = df['Target']

# Impute missing values temporarily for selection
imputer = SimpleImputer(strategy='mean')
X_imputed = pd.DataFrame(imputer.fit_transform(X), columns=X.columns)
selector = SelectKBest(score_func=f_classif, k=2)
X_selected = selector.fit_transform(X_imputed, y)
selected_features = X.columns[selector.get_support()]
print("\nSelected Features:\n", selected_features)
```

```

# Keep only selected features
df = df[selected_features.tolist() + ['Target']]

# --- b. Handling Missing Values ---

# Impute using mean strategy
imputer = SimpleImputer(strategy='mean')
df[df.columns] = imputer.fit_transform(df)
print("\nAfter Handling Missing Values:\n", df)

# --- c. Discretization ---

# Discretize 'Age' into 3 bins (uniform)
discretizer = KBinsDiscretizer(n_bins=3, encode='ordinal', strategy='uniform')
df['Age_binned'] = discretizer.fit_transform(df[['Age']])
print("\nAfter Discretization of Age:\n", df)

# --- d. Elimination of Outliers ---

# Use Z-score method to remove outliers (threshold = 3)
z_scores = np.abs(stats.zscore(df.select_dtypes(include=[np.number])))
df = df[(z_scores < 3).all(axis=1)]

print("\nAfter Eliminating Outliers:\n", df)

```

Output:

Original Data:

	Age	Income	Education	Target
0	25.0	50000.0	1	0
1	27.0	54000.0	2	1
2	29.0	58000.0	2	0
3	NaN	NaN	3	1
4	31.0	62000.0	1	0
5	120.0	64000.0	3	1
6	23.0	1000000.0	2	0
7	25.0	58000.0	2	1

Selected Features:

```
Index(['Age', 'Education'], dtype='object')
```

After Handling Missing Values:

	Age	Education	Target
0	25.0	1.0	0.0
1	27.0	2.0	1.0
2	29.0	2.0	0.0

3	40.0	3.0	1.0
4	31.0	1.0	0.0
5	120.0	3.0	1.0
6	23.0	2.0	0.0
7	25.0	2.0	1.0

After Discretization of Age:

Age	Education	Target	Age_binned
-----	-----------	--------	------------

0	25.0	1.0	0.0	0.0
1	27.0	2.0	1.0	0.0
2	29.0	2.0	0.0	0.0
3	40.0	3.0	1.0	0.0
4	31.0	1.0	0.0	0.0
5	120.0	3.0	1.0	2.0
6	23.0	2.0	0.0	0.0
7	25.0	2.0	1.0	0.0

After Eliminating Outliers:

Age	Education	Target	Age_binned
-----	-----------	--------	------------

0	25.0	1.0	0.0	0.0
1	27.0	2.0	1.0	0.0
2	29.0	2.0	0.0	0.0
3	40.0	3.0	1.0	0.0
4	31.0	1.0	0.0	0.0
5	120.0	3.0	1.0	2.0
6	23.0	2.0	0.0	0.0
7	25.0	2.0	1.0	0.0

Experiment-3

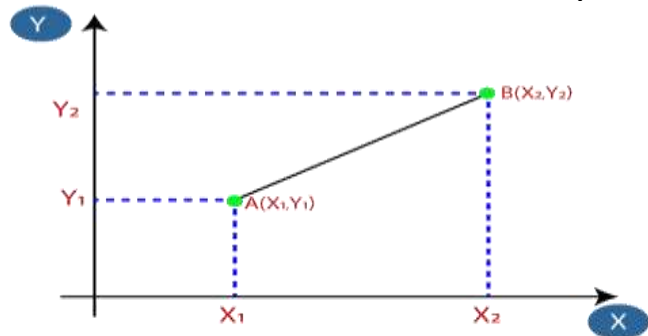
3. Apply KNN algorithm for classification and regression

Aim: To Apply KNN algorithm for classification and regression

Procedure:

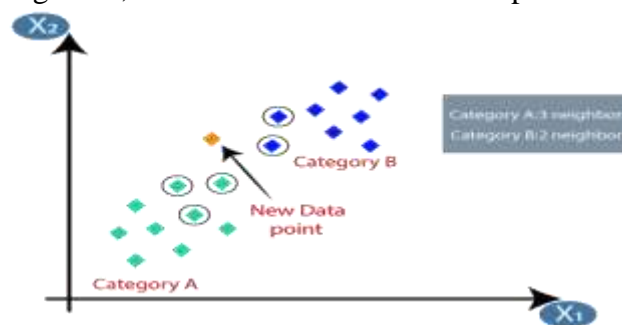
The K-NN working can be explained on the basis of the below algorithm:

- ❖ **Step-1:** Select the number K of the neighbors i.e; Firstly, we will choose the number of neighbors, near to new data points.
- ❖ **Step-2:** Calculate the Euclidean distance of new data point to neighbor data points.



○ Euclidean Distance between A₁ and B₂ = $\sqrt{(X_2 - X_1)^2 + (Y_2 - Y_1)^2}$

- ❖ **Step-3:** Among these k neighbors, count the number of the data points in each category.



- ❖ **Step-4:** 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:
- ❖ **Step-5:** As we can see the 3 nearest neighbors are from category A, hence this new data point must belong to category A.

Program:

```
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score
from sklearn.datasets import fetch_california_housing
from sklearn.neighbors import KNeighborsRegressor
from sklearn.metrics import mean_squared_error, r2_score

# Load dataset
iris = load_iris()
X, y = iris.data, iris.target

# Split dataset
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
# Apply KNN
knn_clf = KNeighborsClassifier(n_neighbors=3)
knn_clf.fit(X_train, y_train)
# Predict
y_pred = knn_clf.predict(X_test)
# Evaluate
print("KNN Classification Accuracy:", accuracy_score(y_test, y_pred))
#*****
# Load dataset
housing = fetch_california_housing()
X, y = housing.data, housing.target
# Split dataset
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
# Apply KNN
knn_reg = KNeighborsRegressor(n_neighbors=5)
knn_reg.fit(X_train, y_train)
# Predict
y_pred = knn_reg.predict(X_test)
# Evaluate
print("KNN Regression MSE:", mean_squared_error(y_test, y_pred))
print("KNN Regression R2 Score:", r2_score(y_test, y_pred))
```

Output:

KNN Classification Accuracy: 1.0

KNN Regression MSE: 1.136942049088978

KNN Regression R2 Score: 0.1337849088797427

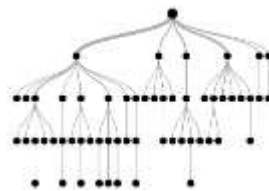
Experiment-4

4. Demonstrate decision tree algorithm for a classification problem and perform parameter tuning for better results

Aim: Demonstrate decision tree algorithm for a classification problem and perform parameter tuning for better results

Procedure:

A decision tree, which has a hierarchical structure made up of root, branches, internal, and leaf nodes, is a non-parametric supervised learning approach used for classification and regression applications. It is a tool that has applications spanning several different areas. These trees can be used for classification as well as regression problems. The name itself suggests that it uses a flowchart like a tree structure to show the predictions that result from a series of feature-based splits. It starts with a root node and ends with a decision made by leaves.



Types of Decision Tree

- **ID3:** This algorithm measures how mixed up the data is at a node using something called [entropy](#). It then chooses the feature that helps to clarify the data. This is an improved version of ID3 that can handle missing data and continuous attributes.
- **CART:** This algorithm uses a different measure called Gini impurity to decide how to split the data. It can be used for both classification (sorting data into categories) and regression (predicting continuous values) tasks.

Steps:

1. Load and explore the dataset
2. Train a Decision Tree Classifier
3. Evaluate the model
4. Tune hyperparameters using Grid Search
5. Re-evaluate the tuned model

Program:

```
import pandas as pd
from sklearn.datasets import load_iris
from sklearn.tree import DecisionTreeClassifier, plot_tree
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.metrics import classification_report, accuracy_score
import matplotlib.pyplot as plt
# Load dataset
iris = load_iris()
```

```

X = pd.DataFrame(iris.data, columns=iris.feature_names)
y = pd.Series(iris.target)
# Split 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)
# Initial Decision Tree Model
clf = DecisionTreeClassifier(random_state=42)
clf.fit(X_train, y_train)
y_pred = clf.predict(X_test)
print("Initial Model Accuracy:", accuracy_score(y_test, y_pred))
print("\nClassification Report:\n", classification_report(y_test, y_pred))
# Plot the tree
plt.figure(figsize=(12, 8))
plot_tree(clf, feature_names=iris.feature_names, class_names=iris.target_names, filled=True)
plt.title("Initial Decision Tree")
plt.show()
# Parameter tuning using GridSearchCV
param_grid = {
    'criterion': ['gini', 'entropy'],
    'max_depth': [2, 3, 4, 5, None],
    'min_samples_split': [2, 3, 4],
    'min_samples_leaf': [1, 2, 3]
}
grid_search = GridSearchCV(DecisionTreeClassifier(random_state=42),
    param_grid,
    cv=5,
    scoring='accuracy',
    n_jobs=-1)
grid_search.fit(X_train, y_train)
# Best model
best_clf = grid_search.best_estimator_
y_pred_best = best_clf.predict(X_test)
print("\nTuned Model Accuracy:", accuracy_score(y_test, y_pred_best))
print("\nBest Parameters:", grid_search.best_params_)
print("\nClassification Report:\n", classification_report(y_test, y_pred_best))
# Plot the best decision tree
plt.figure(figsize=(12, 8))
plot_tree(best_clf, feature_names=iris.feature_names, class_names=iris.target_names,
    filled=True)
plt.title("Tuned Decision Tree")
plt.show()

```

Output:

Initial Model Accuracy: 1.0

Classification Report:

	precision	recall	f1-score	support
--	-----------	--------	----------	---------

0	1.00	1.00	1.00	19
---	------	------	------	----

1	1.00	1.00	1.00	13
---	------	------	------	----

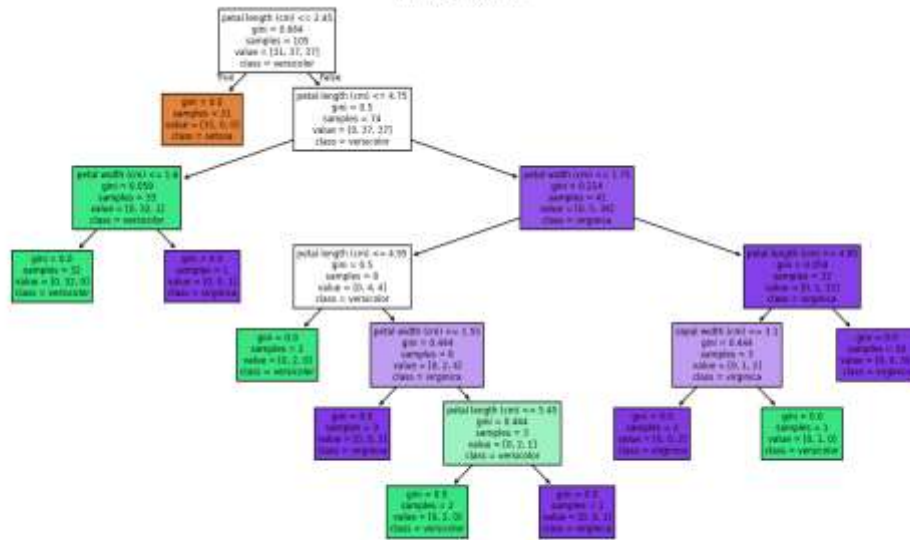
2	1.00	1.00	1.00	13
---	------	------	------	----

accuracy			1.00	45
----------	--	--	------	----

macro avg	1.00	1.00	1.00	45
-----------	------	------	------	----

weighted avg	1.00	1.00	1.00	45
--------------	------	------	------	----

Initial Decision Tree



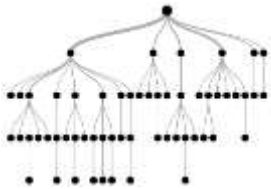
Experiment-5

5 .Demonstrate decision tree algorithm for a regression problem

Aim: Demonstrate decision tree algorithm for a regression problem

Procedure:

A decision tree, which has a hierarchical structure made up of root, branches, internal, and leaf nodes, is a non-parametric supervised learning approach used for classification and regression applications. It is a tool that has applications spanning several different areas. These trees can be used for classification as well as regression problems. The name itself suggests that it uses a flowchart like a tree structure to show the predictions that result from a series of feature-based splits. It starts with a root node and ends with a decision made by leaves.



Types of Decision Tree

- **ID3:** This algorithm measures how mixed up the data is at a node using something called [entropy](#). It then chooses the feature that helps to clarify the data. This is an improved version of ID3 that can handle missing data and continuous attributes.
- **CART:** This algorithm uses a different measure called Gini impurity to decide how to split the data. It can be used for both classification (sorting data into categories) and regression (predicting continuous values) tasks.

Program:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.datasets import fetch_california_housing
from sklearn.tree import DecisionTreeRegressor, plot_tree
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, r2_score

# Load dataset
data = fetch_california_housing()
X = pd.DataFrame(data.data, columns=data.feature_names)
y = data.target

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Initialize and train Decision Tree Regressor
regressor = DecisionTreeRegressor(random_state=42)
regressor.fit(X_train, y_train)

# Predict on test data
y_pred = regressor.predict(X_test)

# Evaluate the model
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)
print(f"Mean Squared Error: {mse:.4f}")
print(f"R² Score: {r2:.4f}")

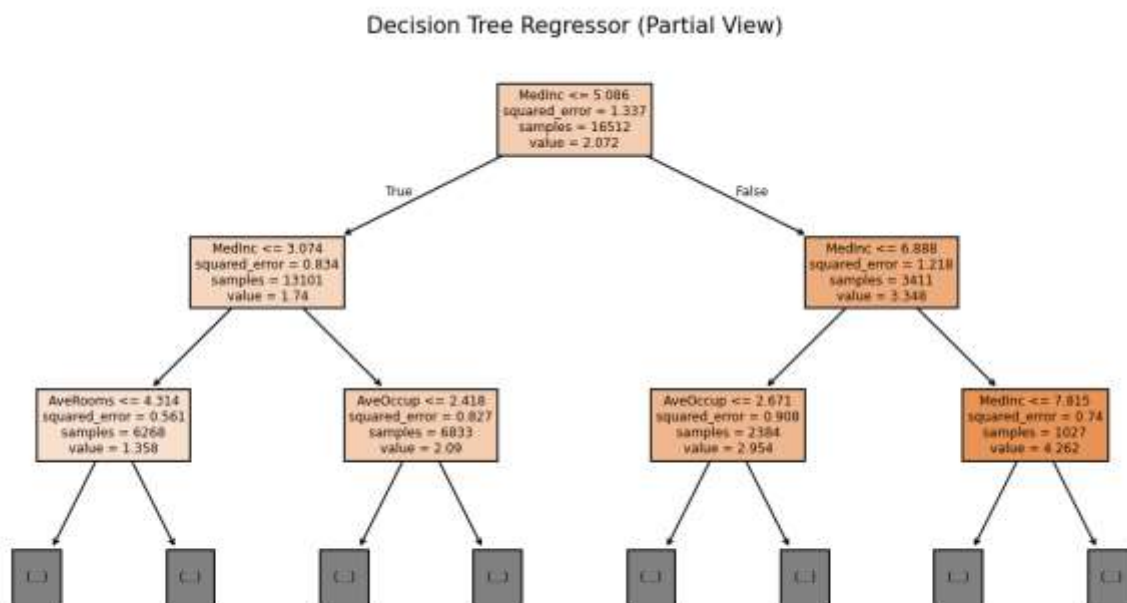
# Plot a portion of the tree (for visualization purposes)
```

```
plt.figure(figsize=(12, 6))
plot_tree(regressor, feature_names=data.feature_names, max_depth=2, filled=True)
plt.title("Decision Tree Regressor (Partial View)")
plt.show()
```

Output:

Mean Squared Error: 0.4952

R² Score: 0.6221



Experiment-6

6. Apply Random Forest algorithm for classification and regression

Aim: Apply Random Forest algorithm for classification and regression

Procedure:

The following steps explain the working Random Forest Algorithm:

Step 1: Select random samples from a given data or training set.

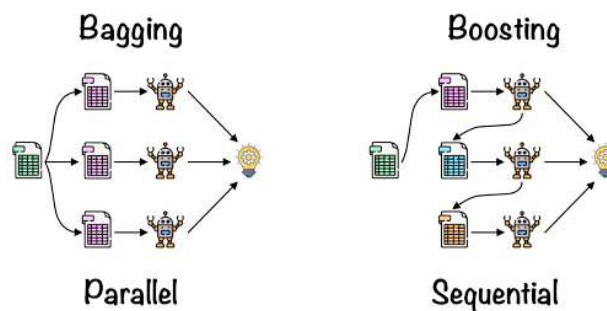
Step 2: This algorithm will construct a decision tree for every training data.

Step 3: Voting will take place by averaging the decision tree.

Step 4: Finally, select the most voted prediction result as the final prediction result.

This combination of multiple models is called Ensemble. Ensemble uses two methods:

1. Bagging: Creating a different training subset from sample training data with replacement is called Bagging. The final output is based on majority voting.
2. Boosting: Combining weak learners into strong learners by creating sequential models such that the final model has the highest accuracy is called Boosting. Example: ADA BOOST, XG BOOST.



Program:

```
import numpy as np
import pandas as pd
from sklearn.datasets import load_iris, fetch_california_housing
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier, RandomForestRegressor
from sklearn.metrics import accuracy_score, classification_report, mean_squared_error, r2_score
# -----
# Random Forest Classification
# -----
print("=== RANDOM FOREST CLASSIFICATION ===")
# Load dataset
iris = load_iris()
X_class, y_class = iris.data, iris.target
# Split into train and test
Xc_train, Xc_test, yc_train, yc_test = train_test_split(X_class, y_class, test_size=0.3, random_state=42)
# Create and train model
```



```

rf_classifier = RandomForestClassifier(n_estimators=100, random_state=42)
rf_classifier.fit(Xc_train, yc_train)
# Predict and evaluate
y_pred_class = rf_classifier.predict(Xc_test)
print("Accuracy:", accuracy_score(yc_test, y_pred_class))
print("Classification Report:\n", classification_report(yc_test, y_pred_class))
# -----
# Random Forest Regression
# -----
print("\n=== RANDOM FOREST REGRESSION ===")
# Load dataset
housing = fetch_california_housing()
X_reg, y_reg = housing.data, housing.target
# Split into train and test
Xr_train, Xr_test, yr_train, yr_test = train_test_split(X_reg, y_reg, test_size=0.3, random_state=42)
# Create and train model
rf_regressor = RandomForestRegressor(n_estimators=100, random_state=42)
rf_regressor.fit(Xr_train, yr_train)
# Predict and evaluate
y_pred_reg = rf_regressor.predict(Xr_test)
print("Mean Squared Error:", mean_squared_error(yr_test, y_pred_reg))
print("R^2 Score:", r2_score(yr_test, y_pred_reg))

```

Output:

```

=== RANDOM FOREST CLASSIFICATION ===
Accuracy: 1.0
Classification Report:
precision  recall  f1-score  support
0         1.00    1.00    1.00     19
1         1.00    1.00    1.00     13
2         1.00    1.00    1.00     13

accuracy                1.00    45
macro avg              1.00    1.00    1.00    45
weighted avg           1.00    1.00    1.00    45
=== RANDOM FOREST REGRESSION ===
Mean Squared Error: 0.25638991335459355
R^2 Score: 0.8046612733369111

```

Experiment-7

7. Demonstrate Naïve Bayes Classification algorithm.

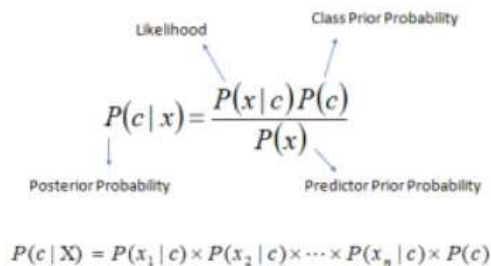
Aim: Demonstrate Naïve Bayes Classification algorithm.

Procedure:

The Naive Bayes algorithm is a supervised machine learning algorithm used for classification tasks. It's based on Bayes' theorem

Naive Bayes is known to outperform even highly sophisticated classification methods.

Bayes theorem provides a way of computing posterior probability $P(c|x)$ from $P(c)$, $P(x)$ and $P(x|c)$. Look at the equation below:



The diagram shows the equation $P(c|x) = \frac{P(x|c)P(c)}{P(x)}$ with arrows pointing from labels to the corresponding parts of the equation. 'Likelihood' points to $P(x|c)$, 'Class Prior Probability' points to $P(c)$, 'Posterior Probability' points to $P(c|x)$, and 'Predictor Prior Probability' points to $P(x)$.

$$P(c|x) = \frac{P(x|c)P(c)}{P(x)}$$
$$P(c|X) = P(x_1|c) \times P(x_2|c) \times \dots \times P(x_n|c) \times P(c)$$

Above,

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

Program:

```
# Import libraries

from sklearn.datasets import load_iris

from sklearn.model_selection import train_test_split

from sklearn.naive_bayes import GaussianNB

from sklearn.metrics import accuracy_score, classification_report, confusion_matrix

# Load the Iris dataset

iris = load_iris()

X = iris.data

y = iris.target
```

```

# Split dataset into training and testing sets (80% train, 20% test)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Initialize Gaussian Naive Bayes classifier

nb_classifier = GaussianNB()

# Train the model

nb_classifier.fit(X_train, y_train)

# Make predictions

y_pred = nb_classifier.predict(X_test)

# Evaluate the model

print("Accuracy:", accuracy_score(y_test, y_pred))

print("\nConfusion Matrix:\n", confusion_matrix(y_test, y_pred))

print("\nClassification Report:\n", classification_report(y_test, y_pred))

```

Output:

Accuracy: 1.0

Confusion Matrix:

```
[[10  0  0]
```

```
[ 0  9  0]
```

```
[ 0  0 11]]
```

Classification Report:

	precision	recall	f1-score	support
--	-----------	--------	----------	---------

0	1.00	1.00	1.00	10
---	------	------	------	----

1	1.00	1.00	1.00	9
---	------	------	------	---

2	1.00	1.00	1.00	11
---	------	------	------	----

accuracy			1.00	30
----------	--	--	------	----

macro avg	1.00	1.00	1.00	30
-----------	------	------	------	----

weighted avg	1.00	1.00	1.00	30
--------------	------	------	------	----

Experiment-8

8 . Apply Support Vector algorithm for classification

Aim: Apply Support Vector algorithm for classification

Procedure: A support vector machine (SVM) is a type of supervised learning algorithm used in machine learning to solve classification and regression tasks. SVMs are particularly good at solving binary classification problems, which require classifying the elements of a data set into two groups. SVMs aim to find the best possible line, or *decision boundary*, that separates the data points of different data classes. This boundary is called a *hyperplane* when working in high-dimensional feature spaces. The idea is to maximize the margin, which is the distance between the hyperplane and the closest data points of each category, thus making it easy to distinguish data classes

Program:

```
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 classification_report, accuracy_score, confusion_matrix

# Load dataset

iris = datasets.load_iris()

X = iris.data

y = iris.target

# Split dataset (80% train, 20% test)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Standardize features

scaler = StandardScaler()

X_train = scaler.fit_transform(X_train)

X_test = scaler.transform(X_test)

# Apply SVM classifier

svm_model = SVC(kernel='rbf', C=1.0, gamma='scale') # You can try other kernels like 'linear', 'poly',
'sigmoid'

svm_model.fit(X_train, y_train)

# Predict on test set
```

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

```
# Evaluation
```

```
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
```

```
print("\nClassification Report:\n", classification_report(y_test, y_pred))
```

```
print("Accuracy Score:", accuracy_score(y_test, y_pred))
```

Output:

Confusion Matrix:

```
[[10 0 0]
```

```
 [ 0 9 0]
```

```
 [ 0 0 11]]
```

Classification Report:

	precision	recall	f1-score	support
--	-----------	--------	----------	---------

0	1.00	1.00	1.00	10
---	------	------	------	----

1	1.00	1.00	1.00	9
---	------	------	------	---

2	1.00	1.00	1.00	11
---	------	------	------	----

accuracy			1.00	30
----------	--	--	------	----

macro avg	1.00	1.00	1.00	30
-----------	------	------	------	----

weighted avg	1.00	1.00	1.00	30
--------------	------	------	------	----

Accuracy Score: 1.0

Experiment-9

9. Demonstrate simple linear regression algorithm for a regression problem.

Aim: Demonstrate simple linear regression algorithm for a regression problem

Procedure:

Simple linear regression is a relation between one input variable and one output variable.

❖ Ex: Relation between income and expenditure.

Relation between experience and salary.

The relationship shown by a Simple Linear Regression model is linear or a sloped straight line, hence it is called Simple Linear Regression.

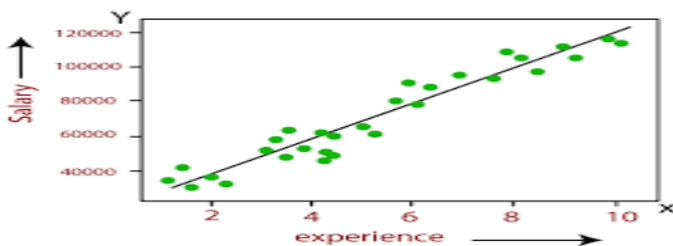
$$Y=ax+b$$

Here y: Output variable /dependent variable.

x: Input variable/independent variable.

a: Intercept of the regression line.

b: Slope of regression line.



Program:

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import load_diabetes
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, r2_score
# Load dataset (using diabetes dataset for regression)
data = load_diabetes()
X = data.data[:, np.newaxis, 2] # Using only one feature (BMI) for simple linear regression
y = data.target
# Split data into training/testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
# Create linear regression model
model = LinearRegression()
```

```

# Train the model
model.fit(X_train, y_train)

# Make predictions
y_pred = model.predict(X_test)

# Print model coefficients
print(f"Coefficient (slope): {model.coef_[0]}")

print(f"Intercept: {model.intercept_}")

print(f"Mean Squared Error: {mean_squared_error(y_test, y_pred):.2f}")

print(f"R2 Score: {r2_score(y_test, y_pred):.2f}")

# Plot outputs
plt.scatter(X_test, y_test, color='blue', label='Actual')

plt.plot(X_test, y_pred, color='red', linewidth=2, label='Predicted Line')

plt.title("Simple Linear Regression (BMI vs Disease Progression)")

plt.xlabel("BMI")

plt.ylabel("Disease Progression")

plt.legend()

plt.show()

```

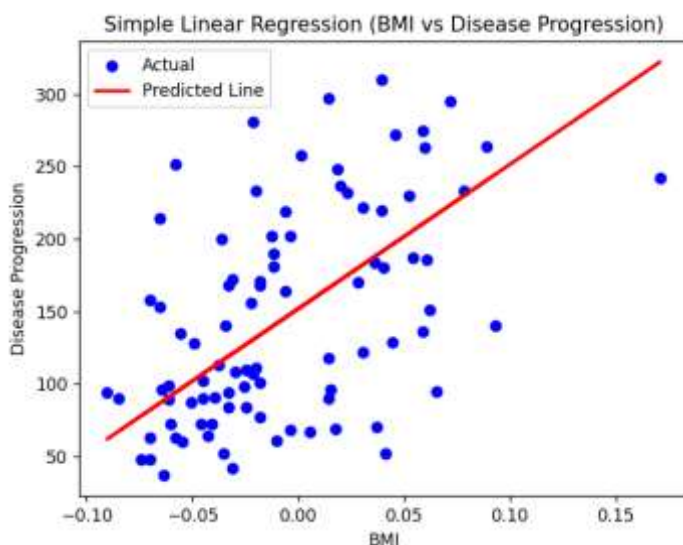
Output:

Coefficient (slope): 998.5776891375593

Intercept: 152.00335421448167

Mean Squared Error: 4061.83

R² Score: 0.23



Experiment-10

10. Apply Logistic regression algorithm for a classification problem

Aim: Apply Logistic regression algorithm for a classification problem

Procedure:

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.

- ❖ Prediction value is categorical i.e; Exam result is pass or fail, Email is spam or not.
- ❖ Logistic regression is used for binary classification.

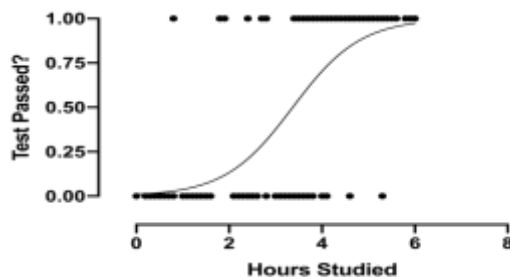
$$\text{Log}(y/(1-y))=c+b_1x_1+b_2x_2+b_3x_2\text{.....}$$

Here y: Output variable /dependent variable.

x1,x2,x3: Input variables/independent variables.

C: Constants value

b: Intercept of the regression line.



Program:

```
from sklearn.datasets import load_iris
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, accuracy_score, confusion_matrix

# Load dataset
iris = load_iris()
X = iris.data
y = iris.target

# For binary classification, use only two classes (e.g., Setosa vs Versicolor)
X = X[y != 2]
y = y[y != 2]

# Split into training and testing data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Create and train Logistic Regression model
```



```
model = LogisticRegression()
model.fit(X_train, y_train)

# Make predictions
y_pred = model.predict(X_test)

# Evaluate model
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
print("\nClassification Report:\n", classification_report(y_test, y_pred))
print("Accuracy Score:", accuracy_score(y_test, y_pred))
```

Output:

Confusion Matrix:

```
[[12  0]
```

```
[ 0  8]]
```

Classification Report:

	precision	recall	f1-score	support	
0	1.00	1.00	1.00	12	
1	1.00	1.00	1.00	8	
accuracy			1.00	20	
macro avg		1.00	1.00	1.00	20
weighted avg		1.00	1.00	1.00	20

Accuracy Score: 1.0

Experiment-11

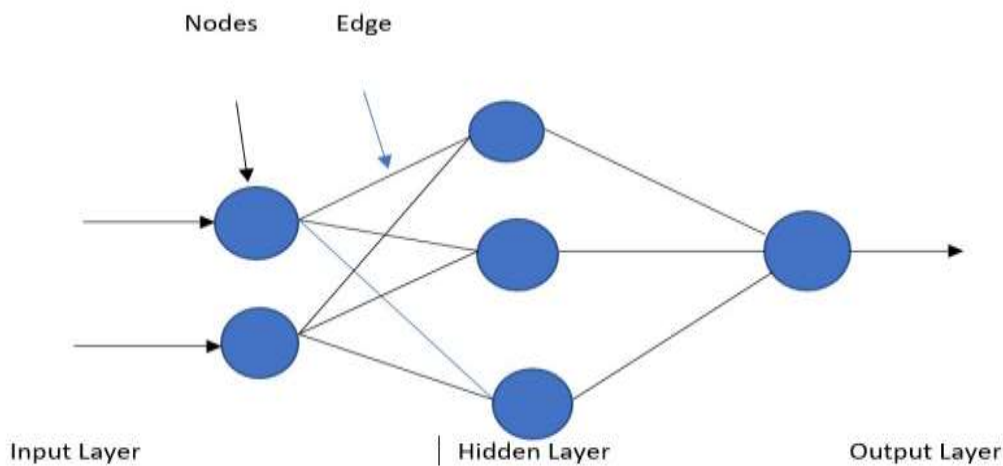
11 Demonstrate Multi-layer Perceptron algorithm for a classification problem

Aim: Demonstrate Multi-layer Perceptron algorithm for a classification problem

Procedure: An MLP is a type of feedforward artificial neural network with multiple layers, including an input layer, one or more hidden layers, and an output layer. Each layer is fully connected to the next. In this article, we will understand MultiLayer Perceptron Neural Network, an important concept of deep learning and neural networks.

Artificial Neural Network has three layers:

- Input Layer
- Hidden Layer
- Output Layer



There are nodes and edges in the Neural Network.

- A node is any object or individual.
- Edges are connectivity between two edges. These are directional and Nondirectional

Program:

```
# Import necessary libraries
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import classification_report, accuracy_score, confusion_matrix

# Load dataset
iris = load_iris()
X, y = iris.data, iris.target
```

```

# Split dataset into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Standardize features
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Initialize MLPClassifier
mlp = MLPClassifier(hidden_layer_sizes=(10, 10), max_iter=1000, random_state=42)

# Train the model
mlp.fit(X_train, y_train)

# Make predictions
y_pred = mlp.predict(X_test)

# Evaluate the model
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
print("\nClassification Report:\n", classification_report(y_test, y_pred))
print("Accuracy Score:", accuracy_score(y_test, y_pred))

```

Output:

Confusion Matrix:

```
[[19  0  0]
```

```
[ 0 13  0]
```

```
[ 0  0 13]]
```

Classification Report:

```
precision  recall  f1-score  support
```

```
0      1.00      1.00      1.00      19
```

```
1      1.00      1.00      1.00      13
```

```
2      1.00      1.00      1.00      13
```

```
accuracy                1.00      45
```

```
macro avg      1.00      1.00      1.00      45
```

```
weighted avg      1.00      1.00      1.00      45
```

Accuracy Score: 1.0

Experiment-12

12. Implement the K-means algorithm and apply it to the data you selected. Evaluate performance by measuring the sum of the Euclidean distance of each example from its class center. Test the performance of the algorithm as a function of the parameters K

Aim: Implement the K-means algorithm and apply it to the data you selected. Evaluate performance by measuring the sum of the Euclidean distance of each example from its class center. Test the performance of the algorithm as a function of the parameters K

Procedure:

Kmeans algorithm is an iterative algorithm that tries to partition the dataset into K pre-defined distinct non-overlapping subgroups (clusters) where each data point belongs to **only one group**. It tries to make the intra-cluster data points as similar as possible while also keeping the clusters as different (far) as possible. It assigns data points to a cluster such that the sum of the squared distance between the data points and the cluster's centroid (arithmetic mean of all the data points that belong to that cluster) is at the minimum. The less variation we have within clusters, the more homogeneous (similar) the data points are within the same cluster.

The way kmeans algorithm works is as follows:

1. Specify number of clusters K .
2. Initialize centroids by first shuffling the dataset and then randomly selecting K data points for the centroids without replacement.
3. Keep iterating until there is no change to the centroids. i.e assignment of data points to clusters isn't changing.
 - Compute the sum of the squared distance between data points and all centroids.
 - Assign each data point to the closest cluster (centroid).
 - Compute the centroids for the clusters by taking the average of the all data points that belong to each cluster.

Program:

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler
```

```

# Load the dataset (you can choose any dataset)
data = load_iris()
X = data.data

# Standardize the data
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

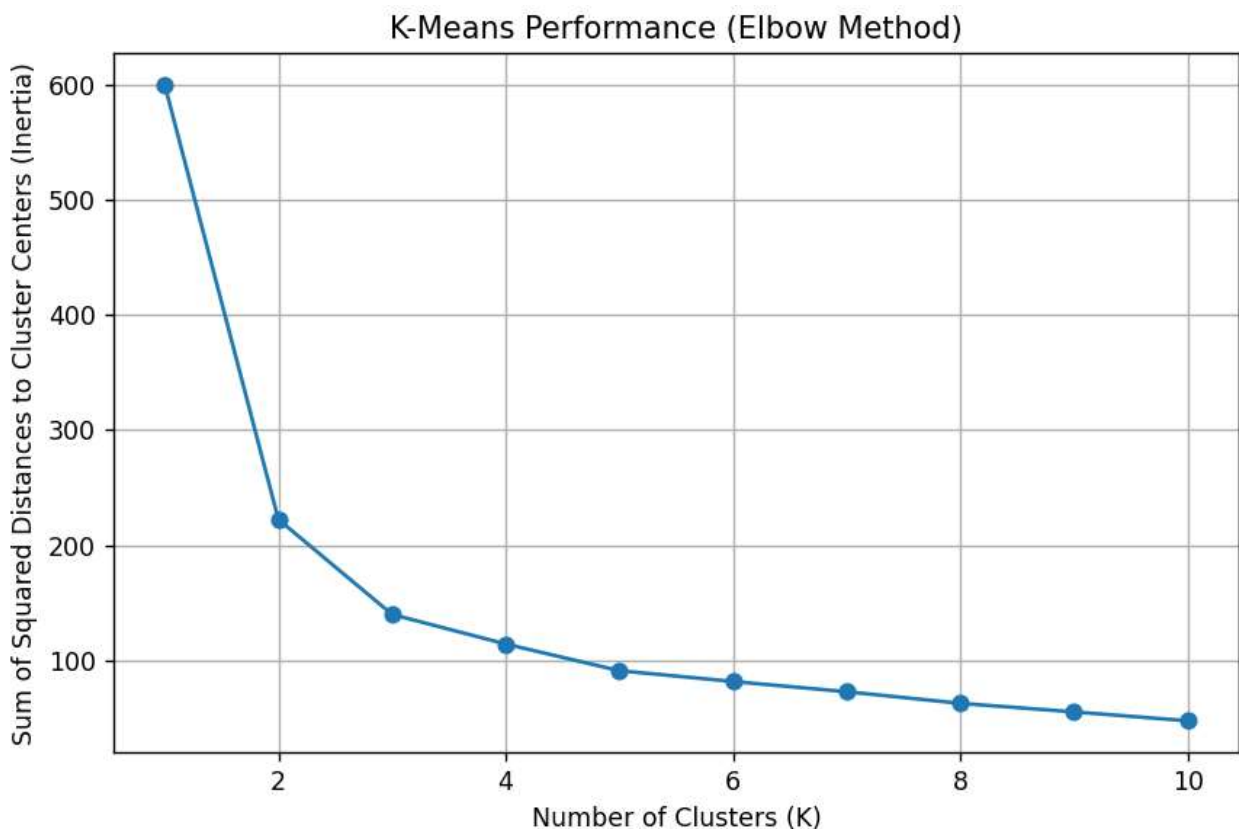
# Range of K values to test
k_values = range(1, 11)
distortions = [] # Sum of Euclidean distances from points to their cluster center

for k in k_values:
    kmeans = KMeans(n_clusters=k, random_state=42, n_init=10)
    kmeans.fit(X_scaled)
    distortions.append(kmeans.inertia_) # inertia_ is the sum of squared distances

# Plotting distortion vs K (Elbow Method)
plt.figure(figsize=(8, 5))
plt.plot(k_values, distortions, marker='o')
plt.title('K-Means Performance (Elbow Method)')
plt.xlabel('Number of Clusters (K)')
plt.ylabel('Sum of Squared Distances to Cluster Centers (Inertia)')
plt.grid(True)
plt.show()

```

Output:



Experiment-13

13 Demonstrate the use of Fuzzy C-Means Clustering

Aim: Demonstrate the use of Fuzzy C-Means Clustering

Procedure:

Fuzzy C Means is a soft clustering technique in which every data point is assigned a cluster along with the probability of it being in the cluster

Run the FCM Algorithm

1. **Initialization:** Randomly choose and initialize cluster centroids from the data set and specify a fuzziness parameter (m) to control the degree of fuzziness in the clustering.
2. **Membership Update:** Calculate the degree of membership for each data point to each cluster based on its distance to the cluster centroids using a distance metric (ex: Euclidean distance).
3. **Centroid Update:** Update the centroid value and recalculate the cluster centroids based on the updated membership values.
4. **Convergence Check:** Repeat steps 2 and 3 until a specified number of iterations is reached or the membership values and centroids converge to stable values

Program:

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

# Generate sample data
n_samples = 300
n_features = 2
n_clusters = 3
X, y_true = make_blobs(n_samples=n_samples, centers=n_clusters, n_features=n_features,
random_state=42)

# Transpose the data for skfuzzy
X_transposed = X.T

# Apply Fuzzy C-Means
cntr, u, u0, d, jm, p, fpc = fuzz.cluster.cmeans(
X_transposed, c=n_clusters, m=2.0, error=0.005, maxiter=1000, init=None)

# Predict cluster membership
cluster_membership = np.argmax(u, axis=0)

# Plot the results
plt.figure(figsize=(8, 5))
colors = ['r', 'g', 'b']
```

```

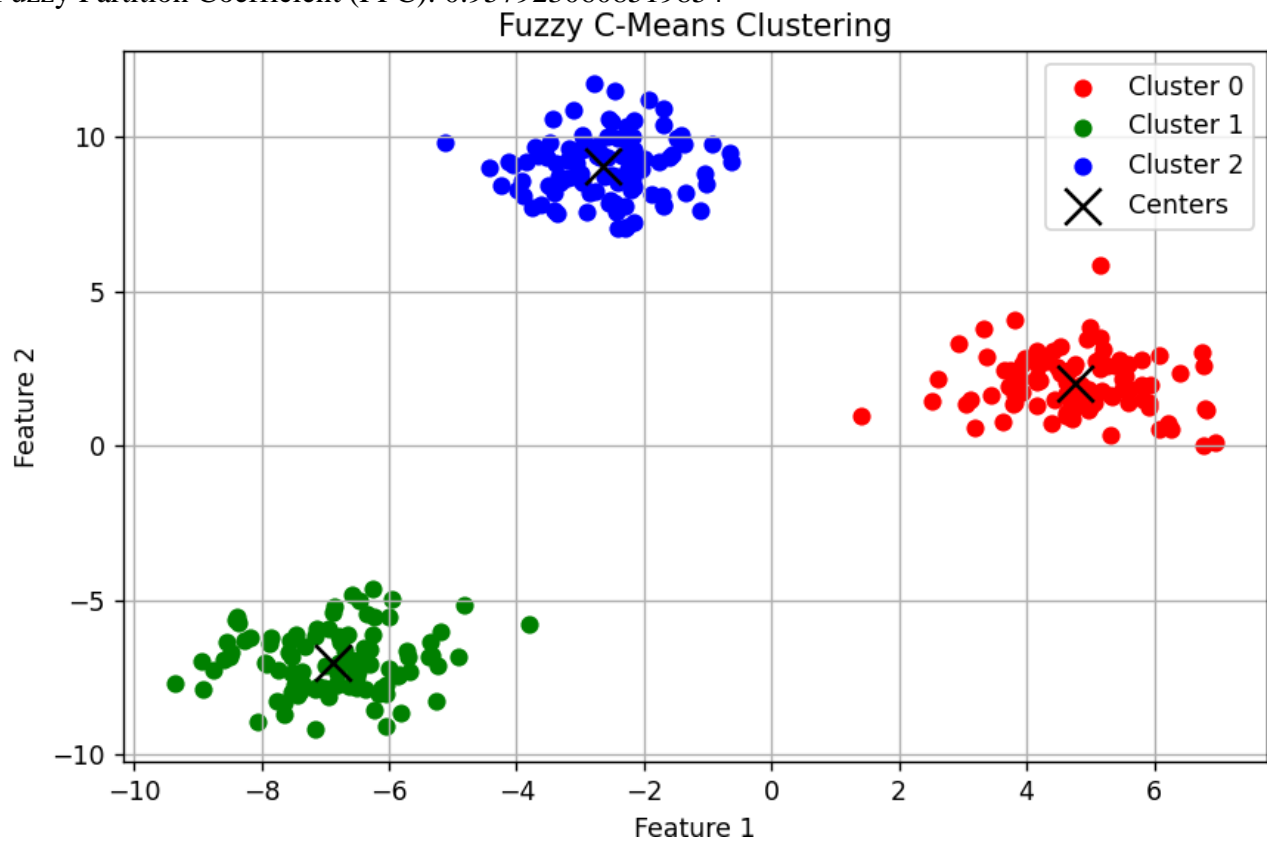
for j in range(n_clusters):
plt.scatter(X[cluster_membership == j, 0], X[cluster_membership == j, 1], c=colors[j], label=f'Cluster {j}')

# Plot cluster centers
plt.scatter(cnr[:, 0], cnr[:, 1], marker='x', s=200, c='black', label='Centers')
plt.title("Fuzzy C-Means Clustering")
plt.xlabel("Feature 1")
plt.ylabel("Feature 2")
plt.legend()
plt.grid(True)
plt.show()
# Print Fuzzy Partition Coefficient
print("Fuzzy Partition Coefficient (FPC):", fpc)

```

Output:

Fuzzy Partition Coefficient (FPC): 0.9579230608519854



Experiment-14

14. Demonstrate the use of Expectation Maximization based clustering algorithm

Aim: To Demonstrate the use of Expectation Maximization based clustering algorithm

Procedure:

Expectation-Maximization (EM) based clustering algorithm using the **Gaussian Mixture Model (GMM)** from sklearn. This is one of the most common applications of EM in clustering.

This outlines a process of using Gaussian Mixture Models (GMM) for clustering, evaluating the results, and visualizing the outcome. The core steps involve creating synthetic data, applying GMM, visualizing the clusters, and assessing their quality using the Silhouette score.

- **Step 1:** Creates synthetic 2D data with 3 cluster centers.
- **Step 2:** Uses GaussianMixture from sklearn.mixture, which internally implements EM.
- **Step 3:** Plots the clustered points colored by predicted cluster.
- **Step 4:** Calculates **Silhouette Score** to evaluate clustering quality.

Program:

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.mixture import GaussianMixture
from sklearn.datasets import make_blobs
from sklearn.metrics import silhouette_score

# Step 1: Generate synthetic data
X, y_true = make_blobs(n_samples=300, centers=3, cluster_std=0.60, random_state=42)

# Step 2: Apply EM using Gaussian Mixture Model
gmm = GaussianMixture(n_components=3, random_state=42)
gmm.fit(X)
labels = gmm.predict(X)

# Step 3: Plotting the clustered data
plt.figure(figsize=(8, 6))
plt.scatter(X[:, 0], X[:, 1], c=labels, s=40, cmap='viridis')
plt.title("EM Clustering using Gaussian Mixture Model")
plt.xlabel("Feature 1")
plt.ylabel("Feature 2")
plt.grid(True)
plt.show()

# Step 4: Print performance metric
sil_score = silhouette_score(X, labels)
print("Silhouette Score: {:.3f}".format(sil_score))
```


Output:

