M.Sc Cs- I

Sem I

Journal

Roll No.	4640
Name	Priyanka Arun Yadav
Subject	Machine Learning
Sub Code	PGCS1MLPP523
Sign	

4640
Priyanka Yadav

INDEX

4640
Priyanka Yadav

AIM: **Perform Linear Regression and compare MSE and RMSE**

THEORY:

Linear regression is a basic yet powerful statistical and machine learning method used to model the relationship between a dependent variable (the target) and one or more independent variables (the predictors). There are two primary types:

Simple Linear Regression (with one predictor)

Multiple Linear Regression (with multiple predictors)

1. Simple Linear Regression:

This technique models the relationship between two continuous variables—one independent variable and one dependent variable—by fitting a linear equation to observed data. The formula for simple linear regression is:

$y=\beta0+\beta1x+\epsilon$ y = β_0 + β_1x + ϵy=β0+β1x+ϵ

y: Dependent variable (response)

x: Independent variable (predictor)

β_0: Intercept (the value of y when x = 0)

β_1: Slope of the line (rate of change of y with respect to x)

ϵ: Error term (residual noise)

2. Multiple Linear Regression:

This method extends simple linear regression to predict the dependent variable based on two or more independent variables. The equation for multiple linear regression is:

$y=\beta0+\beta1x1+\beta2x2+\cdots+\beta nxn+\epsilon$ y = β_0 + β_1x_1 + β_2x_2 + \dots + β_nx_n + ϵy=β0+β1x1+β2x2 +⋯+βnxn+ϵ

y: Dependent variable

x_1, x_2, ..., x_n: Independent variables (predictors)

β_0: Intercept

β_1, β_2, ..., β_n: Coefficients (slopes) for each predictor

ϵ: Error term

Advantages of Simple Linear Regression:

Easy to Interpret: Visualizing and understanding the relationship between the two variables is straightforward.

Efficient: Less computationally intensive, especially with smaller datasets.

Good for Small Datasets: Works well when data size is small and the relationship is linear.

Disadvantages:

Limited Application: Only applicable when there is a linear relationship between two variables.

Inaccurate for Complex Data: It can't handle non-linear relationships well.

Advantages of Multiple Linear Regression:

Handles Multiple Variables: Can model more complex relationships by incorporating multiple predictors.

Increased Accuracy: Offers better predictive power with relevant independent variables.

Useful for Feature Selection: Helps assess the impact of each predictor.

Disadvantages:

Risk of Overfitting: Too many variables may result in an overly complex model that doesn't generalize well.

Collinearity Issues: High correlation between independent variables can make the model unstable.

More Computationally Intensive: Requires more resources than simple linear regression.

Applications:

Simple Linear Regression:

Predicting house prices based on square footage.

Forecasting stock prices based on historical trends.

Estimating sales from advertising expenditure.

Multiple Linear Regression:

Predicting sales based on multiple factors like advertising and demographics.

Forecasting economic indicators using factors like employment and inflation.

Predicting patient outcomes in healthcare based on age, BMI, and other factors.

not generalize well to new data.

Collinearity: If independent variables are highly correlated, it can make the model unstable and lead to misleading results.

Computationally expensive: More complex than simple linear regression and requires more computational resources.

4640
Priyanka Yadav

Simple Linear Regression Applications:

1. **Predicting house prices**: Based on one feature like square footage, predicting house prices.
2. **Stock price forecasting**: Estimating future stock prices based on past trends.
3. **Sales prediction**: Estimating sales based on advertising expenditure.

Multiple Linear Regression Applications:

1. **Marketing**: Predicting sales based on multiple features like advertising budget, promotion campaigns, and customer demographics.
2. **Health sector**: Predicting patient outcomes based on various factors like age, BMI, and blood pressure.
3. **Economics**: Forecasting economic indicators (like GDP) based on factors like employment rate, inflation, and interest rates.

CODE & OUTPUT:

4640
Priyanka Yadav

```
import pandas as pd
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score

iris=load_iris()
df=pd.DataFrame(data=iris.data, columns=iris.feature_names)
df['target']=iris.target
```

```
X=df[['sepal width (cm)']]
y=df['sepal length (cm)']
```

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

```
sir_model= LinearRegression()
sir_model.fit(X_train, y_train)
```

```
LinearRegression()
```

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

```
mse=mean_squared_error(y_test, y_pred)
r2= r2_score(y_test,y_pred)

print("Simple Linear Regression")
print("Mean Squared Error", mse)
print("R-squared:", r2)
```

```
Simple Linear Regression
Mean Squared Error 0.7035672420306047
R-squared: -0.01926874931997946
```


```
mse=mean_squared_error(y_test, y_pred)
r2= r2_score(y_test,y_pred)

print("Simple Linear Regression")
print("Mean Squared Error", mse)
print("R-squared:", r2)
```

```
Simple Linear Regression
Mean Squared Error 0.7035672420306047
R-squared: -0.01926874931997946
```


```
mse=mean_squared_error(y_test, y_pred)
r2= r2_score(y_test,y_pred)

print("Multiple Linear Regression")
print("Mean Squared Error", mse)
print("R-squared:", r2)
```

```
Multiple Linear Regression
Mean Squared Error 0.10212647866320405
R-squared: 0.852047790231016
```

```
X= df[['sepal width (cm)', 'petal length (cm)', 'petal width (cm)']]
y= df['sepal length (cm)']
```

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

mlr_model= LinearRegression()
mlr_model.fit(X_train, y_train)

y_pred= mlr_model.predict(X_test)
```


```
mse=mean_squared_error(y_test, y_pred)
r2= r2_score(y_test,y_pred)

print("Multiple Linear Regression")
print("Mean Squared Error", mse)
print("R-squared:", r2)
```

```
Multiple Linear Regression
Mean Squared Error 0.10212647866320405
R-squared: 0.852047790231016
```


CONCLUSION:

Hence we learned about Linear regression and its 2 types that is Simple and Multiple linear regression. We also calculated the Mean Squared Error(MSE) and Root Mean Squared Error(RMSE)

4640
Priyanka Yadav

AIM: for a given dataset, split the data into training and testing and fit the following on the training set:

(i) Linear Model using least squares

(ii) Ridge Regression Model

(iii)Lasso Model with explanation

THEORY: Ridge and Lasso regression are two popular **regularization techniques** used to enhance the performance of linear regression models, especially when dealing with multicollinearity or when the model is prone to overfitting. Both methods add a penalty to the loss function to constrain the coefficients of the regression model, but they do so in different ways.

1. Ridge Regression: Ridge Regression, also known as **L2 Regularization**, modifies the standard linear regression by adding a penalty equivalent to the square of the magnitude of the coefficients. This technique discourages large coefficients, thus preventing overfitting and improving the model's generalization capabilities.

The objective function for Ridge Regression is: $\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 + \lambda \sum_{j=1}^{p} \beta_j^2$

Where: yi = Actual value y^i= Predicted value βj = Coefficient for the j^th predictor

- p = Number of predictors λ = Regularization parameter (controls the strength of the penalty)

In matrix form: $(y - X\beta)^T(y - X\beta) + \lambda\beta^T\beta$

Advantages:

1. **Reduces Overfitting:** By penalizing large coefficients, Ridge regression minimizes overfitting, especially in datasets with multicollinearity.
2. **Handles Multicollinearity:** Effective in situations where predictors are highly correlated.
3. **Stable Estimates:** Produces more reliable estimates when predictors are numerous or highly correlated.
4. **Does Not Eliminate Predictors:** All predictors remain in the model, which can be beneficial when all variables are expected to have some influence.

Disadvantages:

1. **Does Not Perform Variable Selection:** Unlike Lasso, Ridge does not set coefficients exactly to zero, so it doesn't inherently perform feature selection.
2. **Interpretability:** The inclusion of all predictors can make the model harder to interpret if many variables are involved.

3. **Choice of λ\lambdaλ:** Selecting the optimal value of the regularization parameter requires careful tuning, typically via cross-validation.

Applications:

- **Predictive Modeling:** Enhancing the performance of regression models in various domains such as finance, biology, and engineering.
- **Handling Multicollinearity:** Situations where independent variables are highly correlated.
- **High-Dimensional Data:** Scenarios with a large number of predictors relative to the number of observations.

2. Lasso Regression: Lasso Regression, also known as **L1 Regularization**, extends linear regression by adding a penalty equal to the absolute value of the magnitude of coefficients. This approach not only helps in preventing overfitting but also performs **feature selection** by shrinking some coefficients exactly to zero, effectively removing irrelevant predictors from the model.

The objective function for Lasso Regression is: $\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 + \lambda \sum_{j=1}^{p} |\beta_j|$

Where: yi, y^i, βj , λ and p are defined as in Ridge Regression.

In matrix form: $(y - X\beta)^T(y - X\beta) + \lambda \sum_{j=1}^{p} |\beta_j|$

Advantages:

1. **Feature Selection:** Automatically selects significant predictors by shrinking less important ones to zero.
2. **Simpler Models:** Produces more interpretable models by reducing the number of variables.
3. **Reduces Overfitting:** Similar to Ridge, it helps prevent overfitting by constraining the size of the coefficients.
4. **Handles High-Dimensional Data:** Effective in scenarios where the number of predictors exceeds the number of observations.

Disadvantages:

1. **Performance with Highly Correlated Predictors:** When predictors are highly correlated, Lasso may arbitrarily select one and ignore the others, potentially missing important variables.
2. **Less Stable:** The selection of variables can be sensitive to small changes in the data.
3. **Requires Careful Tuning:** Choosing the optimal λ\lambdaλ is crucial and typically involves cross-validation.

Applications: Feature Selection: Reducing the dimensionality of data by selecting key variables.

- **Sparse Modeling:** Situations where only a subset of predictors is expected to be relevant.
- **Predictive Modeling:** Enhancing model performance in fields like genomics, finance, and image processing.

CODE & OUTPUT:

```
import pandas as pd
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression, Ridge, Lasso
from sklearn.metrics import mean_squared_error, r2_score

iris=load_iris()
df=pd.DataFrame(data=iris.data, columns=iris.feature_names)
df['target']=iris.target
```

```
ridge_model= Ridge(alpha=1.0)
ridge_model.fit(X_train, y_train)

y_pred_ridge=ridge_model.predict(X_test)

mse_ridge=mean_squared_error(y_test, y_pred_ridge)
r2_ridge= r2_score(y_test,y_pred_ridge)

print("\n Ridge Linear Regression")
print(f"Mean Squared Error (MSE): {mse_ridge:.2f}")
print(f"R-squared: {r2_ridge:.2f}")
```

```
 Ridge Linear Regression
Mean Squared Error (MSE): 0.10
R-squared: 0.86
```

```
lr_model= LinearRegression()
lr_model.fit(X_train, y_train)

y_pred= lr_model.predict(X_test)

mse_lr=mean_squared_error(y_test, y_pred)
r2_lr= r2_score(y_test,y_pred)

print("Linear Regression")
print(f"Mean Squared Error: {mse_lr:.2f}")
print(f"R-squared: {r2_lr:.2f}")
```

```
Linear Regression
Mean Squared Error: 0.10
R-squared: 0.85
```

```
lasso_model= Lasso(alpha=1.0)
lasso_model.fit(X_train, y_train)

y_pred_lasso=lasso_model.predict(X_test)

mse_lasso=mean_squared_error(y_test, y_pred_lasso)
r2_lasso= r2_score(y_test,y_pred_lasso)

print("\n Lasso Linear Regression")
print(f"Mean Squared Error (MSE): {mse_lasso:.2f}")
print(f"R-squared: {r2_lasso:.2f}")
```

```
 Lasso Linear Regression
Mean Squared Error (MSE): 0.52
R-squared: 0.24
```

CONCLUSION:

Ridge Regression (L2):

- **Purpose:** Prevent overfitting by penalizing large coefficients.
- **Key Feature:** Shrinks coefficients towards zero but does not eliminate any.
- **Best When:** All predictors are relevant, especially in the presence of multicollinearity.

Lasso Regression (L1): Purpose: Prevent overfitting and perform feature selection by penalizing the absolute size of coefficients.

- **Key Feature:** Can shrink some coefficients to exactly zero, effectively selecting a simpler model.
- **Best When:** You believe that only a subset of predictors is important, or you need feature selection.

4640
Priyanka Yadav

AIM: Fit a classification model using following:

1. logistic regression

2. Linear Discriminant Analysis

THEORY: Logistic Regression is a type of regression analysis used for predicting the outcome of a categorical dependent variable based on one or more predictor variables.The goal of logistic regression is to model the probability of an event occurring (e.g., 0 or 1, yes or no, etc.) based on a set of input variables.

For a binary classification problem, the probability that the output **Y** is 1 given the input features **X = (x_1, x_2, ..., x_n)** is modeled as:

$$P(Y = 1|X) = \sigma(z) = \frac{1}{1 + e^{-(\beta_0+\beta_1 x_1+\beta_2 x_2+\cdots+\beta_n x_n)}}$$ Where:

- **z** = $\beta_0 + \beta_1 x_1 + \beta_2 x_2 + \ldots + \beta_n x_n$ $\boldsymbol{\beta_0}$ is the intercept (bias term).
- $\boldsymbol{\beta_1, \beta_2, \ldots, \beta_n}$ are the coefficients (weights) for each feature.

Advantages: Simplicity and Interpretability: Easy to implement and understand. Coefficients indicate the direction and magnitude of feature influence.

Probabilistic Output: Provides probabilities for class membership, useful for ranking and thresholding.

Efficiency: Computationally less intensive, making it suitable for large datasets.

Feature Scaling Not Strictly Required: Performs reasonably well without feature scaling, though scaling can improve convergence.

Less Prone to Overfitting: Especially when the number of features is less compared to the number of observations. Regularization techniques (like L1 and L2) can further reduce overfitting.

Disadvantages: Linearity Assumption: Assumes a linear relationship between the independent variables and the log-odds of the dependent variable. May not capture complex patterns without proper feature engineering.

Sensitive to Outliers: Outliers can disproportionately influence the model coefficients**.**

Limited to Binary Classification (Basic Logistic Regression): Extensions like multinomial or ordinal logistic regression are required for multi-class problems.

Requires Large Sample Sizes: Especially when the number of features is large, to achieve reliable estimates.

Cannot Automatically Model Interactions: Interactions between variables must be explicitly included.

Applications: Medical and Healthcare: Predicting the presence or absence of a disease

4640
Priyanka Yadav

Finance and Banking: Credit scoring to assess the probability of a borrower defaulting on a loan. Fraud detection by identifying suspicious transactions.

Marketing and Sales: Customer churn prediction to identify customers likely to leave a service.

Social Sciences: Studying factors influencing binary outcomes like voting behavior.

Natural Language Processing (NLP): Text classification tasks, spam detection or sentiment analysis.

Linear Discriminant Analysis (LDA): Linear Discriminant Analysis is a dimensionality reduction technique used to find a linear combination of features that separates two or more classes. supervised learning method that aims to project high-dimensional data onto a lower-dimensional space, where the classes are well-separated.

Key characteristics of Linear Discriminant Analysis: 1. Supervised learning method. 2. Dimensionality reduction technique. 3. Finds a linear combination of features that separates classes. 4. Assumes normality and equal covariance of the classes. 5. Can be used for multi-class classification problems

CODE & OUTPUT:

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.metrics import accuracy_score, confusion_matrix

from sklearn.linear_model import LogisticRegression
```

```
market_data= pd.read_csv('https://raw.githubusercontent.com/sam16tyagi/Machine-Learning-techniques-in-python/master/logistic%20regression%20dataset-Socia

X= market_data.iloc[:,[2,3]].values
y=market_data.iloc[:,4].values
X_train, X_test, y_train, y_test= train_test_split(X,y, test_size=0.2, random_state=42)
```

```
y_pred=logistic_model.predict(X_test_scaled)

print("Logistic Regression Accuracy:", accuracy_score(y_test, y_pred))
print("Logistic Regression Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))
```

```
Logistic Regression Accuracy: 0.65
Logistic Regression Confusion Matrix:
[[52  0]
 [28  0]]
```

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

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

```
LogisticRegression
LogisticRegression()
```

```
lda= LinearDiscriminantAnalysis()
lda.fit(X_train_scaled, y_train)

y_pred=lda.predict(X_test_scaled)

print("LDA Accuracy:", accuracy_score(y_test, y_pred))
print("LDA Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))
```

```
LDA Accuracy: 0.8625
LDA Confusion Matrix:
[[50  2]
 [ 9 19]]
```

4640
Priyanka Yadav

AIM: Implementation of KNN

THEORY: K-nearest neighbours (KNN) is a type of supervised learning algorithm used for both regression and classification. KNN tries to predict the correct class for the test data by calculating the distance between the test data and all the training points. Then select the K number of points which is closest to the test data. The KNN algorithm calculates the probability of the test data belonging to the classes of 'K' training data and which class holds the highest probability will be selected. In the case of regression, the value is the mean of the 'K' selected training points.

Advantages of KNN

1. **Simplicity**:Easy to understand and implement, especially for beginners in machine learning.
2. **No Training Phase**:KNN is a lazy learner, meaning it doesn't require an explicit training phase. The algorithm simply stores the training data and makes predictions when queried.
3. **Versatility**:Can be used for both classification and regression tasks.
4. **Non-parametric**:Does not make any assumptions about the underlying data distribution. This makes KNN flexible and capable of modeling complex patterns in the data.
5. **Adaptability to New Data**:KNN can easily incorporate new data without needing to retrain the model entirely. Since all computations are deferred until prediction, the addition of new data is as simple as appending it to the dataset.

Disadvantages of KNN

1. **Computationally Expensive at Prediction**: Since KNN is a lazy learner, all the computation occurs at prediction time, which can be slow for large datasets, as it requires calculating distances between the query point and every training point.
2. **Memory-Intensive**: The algorithm needs to store the entire dataset, which can be a problem for large datasets, as it leads to high memory usage.
3. **Sensitive to the Choice of K:** The performance of KNN depends heavily on the value of K (the number of neighbors considered). Small values of K may lead to overfitting (too specific to the training data), while large values of K may cause underfitting (too generalized).
4. **Sensitive to Noisy Data and Outliers**: KNN can be heavily affected by noisy data and outliers, especially with small values of K. These points can disproportionately influence predictions.
5. **Distance Metric Dependency**: The choice of distance metric (e.g., Euclidean, Manhattan, Minkowski) can significantly impact the performance of the model. This can be problematic when dealing with high-dimensional data.

Applications of KNN

1. **Pattern Recognition**: One of the most common uses of KNN is in recognizing patterns, such as image, handwriting, and gesture recognition, where the distance between input vectors and stored patterns is a key factor.
2. **Text Classification**: In natural language processing, KNN is used for text classification tasks such as spam detection and document categorization. It works by comparing the text to previously classified documents and finding the closest match.
3. **Recommender Systems**: KNN can be used in recommender systems to suggest products or content by finding users with similar tastes based on their past behavior (collaborative filtering).
4. **Anomaly Detection**: In fields like cybersecurity, KNN can help identify unusual patterns or behaviors by finding the distance between current data points and known normal data points.
5. **Medical Diagnosis**: KNN can be applied to predict diseases based on patient symptoms or medical test results. For example, it can classify whether a tumor is benign or malignant based on historical patient data.

CODE & OUTPUT:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.datasets import make_blobs
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
```

```
X, y= make_blobs(n_samples= 500, n_features=2, centers=4, cluster_std=1.5, random_state=4)
```

```
#plt.style.use('seaborn')
plt.figure(figsize=(6,6))
plt.scatter(X[:,0], X[:,1], c=y, marker='*', s=100, cmap='viridis')
plt.title('KNN')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.show()
```

```
from sklearn.metrics import accuracy_score
print("Accuracy with k=5", accuracy_score(y_test, y_pred_5)*100)
print("Accuracy with k=1", accuracy_score(y_test, y_pred_1)*100)
```

```
Accuracy with k=5 93.60000000000001
Accuracy with k=1 90.4
```

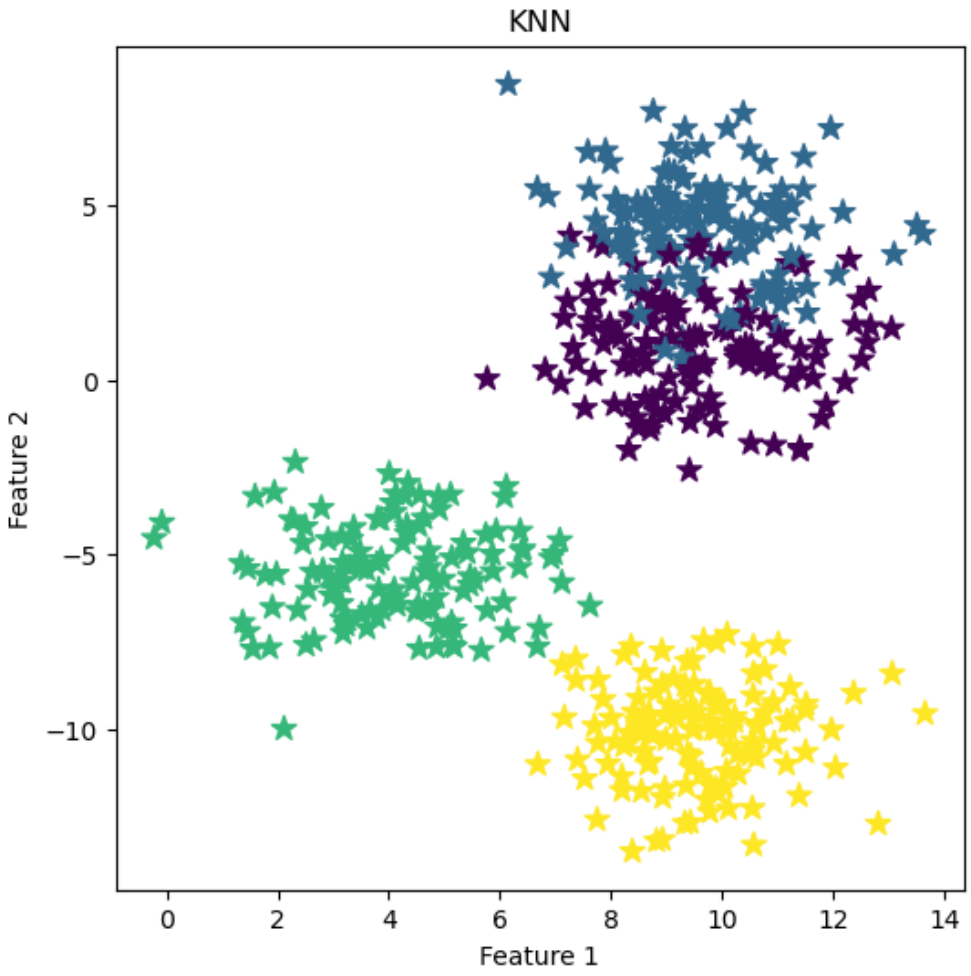
```
plt.subplot(1,2,1)
plt.scatter(X_test[:,0], X_test[:,1], c=y_pred_5, marker='*', s=100)
plt.title("Predicted Values with k=5", fontsize=10)
plt.subplot(1,2,2)
plt.show
plt.scatter(X_test[:,0], X_test[:,1], c=y_pred_1, marker='*', s=100)
plt.title("Predicted Values with k=1", fontsize=10)
plt.show()
```

4640
Priyanka Yadav

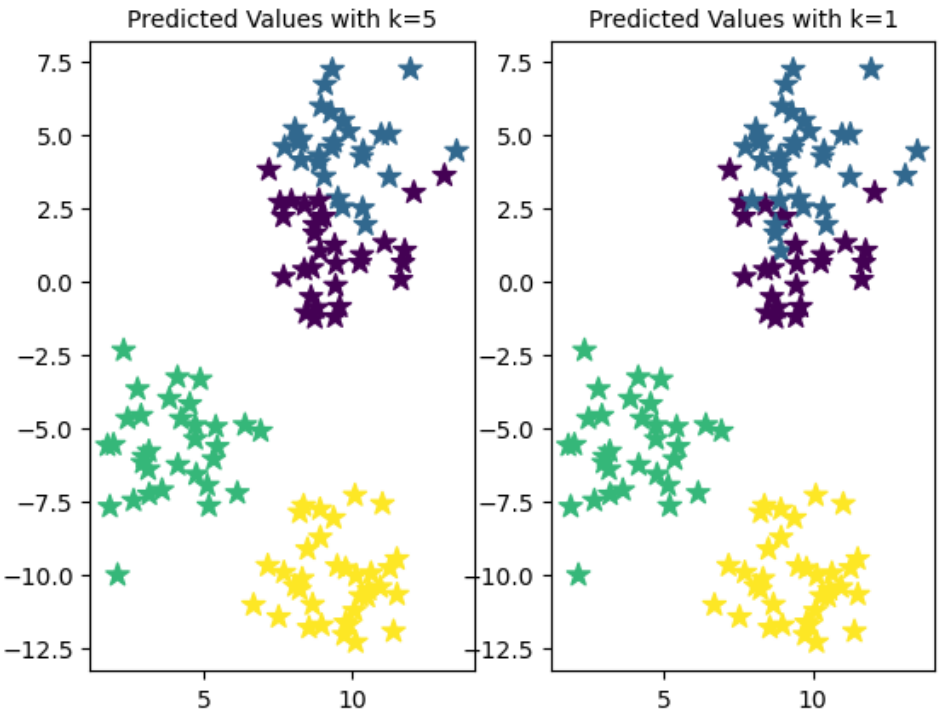
```
X_train, X_test, y_train, y_test= train_test_split(X,y, random_state=0)
```

```
knn5= KNeighborsClassifier(n_neighbors =5)
knn1= KNeighborsClassifier(n_neighbors =1)
```

```
knn5.fit(X_train, y_train)
knn1.fit(X_train, y_train)
y_pred_5=knn5.predict(X_test)
y_pred_1= knn1.predict(X_test)
```



4640
Priyanka Yadav



CONCLUSION:

Hence, successfully performed the Practical using KNN K- Nearest neighbour.

AIM: Perform the polynomial regression and make a plot of the resulting polynomial fit to the data.

THEORY: Polynomial Regression is an extension of linear regression that models the relationship between the independent variable(s) and the dependent variable as an nth-degree polynomial. This allows it to capture more complex patterns and non-linear relationships in the data.

Advantages of Polynomial Regression

1. **Ability to Model Non-Linear Relationships**: It can capture non-linear relationships between the features and the target variable by fitting higher-degree polynomials, which linear regression cannot handle effectively.
2. **Flexibility**: By adjusting the degree of the polynomial, the model can be made more flexible to capture the intricacies of the data, enabling it to fit curves of varying shapes.
3. **Simple to Understand and Implement**: It is a straightforward extension of linear regression, making it relatively easy to understand and implement using the same foundational principles.
4. **Well-Suited for Small Datasets**: It can be highly effective when dealing with small datasets where complex models like neural networks might be overkill or unnecessary.
5. **Higher Accuracy with Appropriate Degree**: If the relationship between the independent variables and the dependent variable is inherently non-linear, polynomial regression can provide a better fit and higher accuracy than linear regression.

4640
Priyanka Yadav

Disadvantages of Polynomial Regression

1. **Risk of Overfitting**: A high-degree polynomial may fit the training data too well (overfitting), capturing noise and outliers. This leads to poor generalization on unseen data, reducing the model's predictive power.
2. **Sensitive to Outliers:** It can be very sensitive to outliers. A few extreme data points can disproportionately affect the polynomial curve, leading to distorted predictions.
3. **Interpretability:** As the degree of the polynomial increases, the model becomes harder to interpret. The coefficients in a high-degree polynomial equation don't offer clear insights into the relationships between variables.
4. **Extrapolation is Unreliable**: It is generally good at interpolating between known data points but performs poorly when extrapolating outside the range of the training data. The model's predictions tend to deviate wildly for input values beyond the observed data.
5. **Prone to High Variance**: If not properly tuned, polynomial regression models with high-degree polynomials may have high variance, where the model is overly sensitive to small variations in the training data.

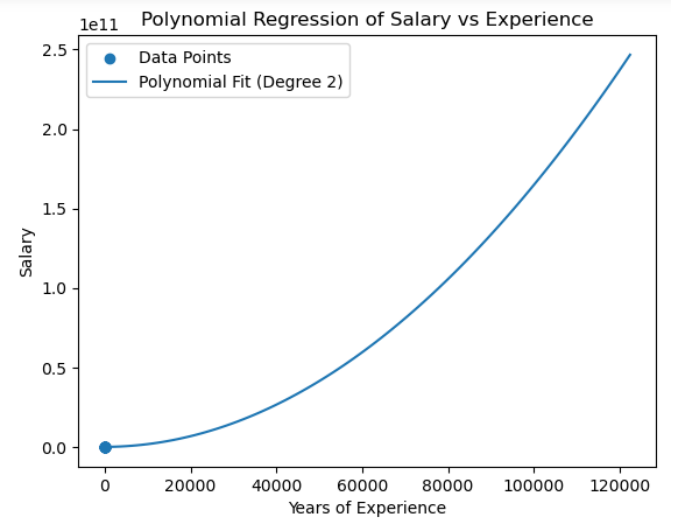
Applications of Polynomial Regression

1. **Economics and Finance**: It is often used to model non-linear relationships in economic data, such as demand curves, cost curves, or price-quantity relationships. For example, it can be applied to forecast GDP growth based on historical data.
2. **Physics and Engineering**: It is commonly applied in fields like physics and engineering, where relationships between variables often follow a polynomial pattern (e.g., acceleration vs. time in physics, stress vs. strain in materials engineering).
3. **Growth Curves in Biology**: In biological studies, polynomial regression can model the growth curves of organisms, allowing researchers to predict how variables like size or population change over time.
4. **Climate Modeling**: It can be used in climate science to model relationships between various environmental variables, such as temperature changes over time or the relationship between CO_2 levels and global temperature.
5. **Marketing and Sales**: In marketing, polynomial regression can be used to model the relationship between advertising spending and sales. As spending increases, sales often follow a non-linear curve, with diminishing returns at higher levels of spending.

CODE & OUTPUT:

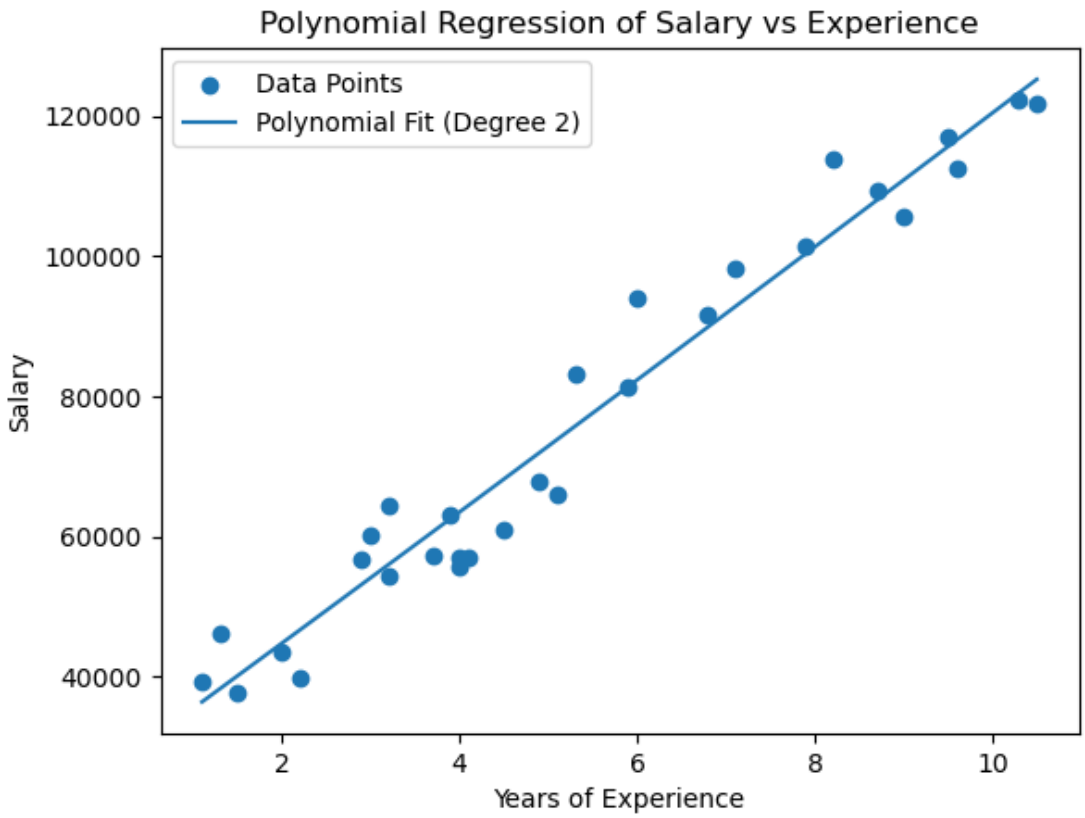
4640
Priyanka Yadav

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

data=pd.read_csv('https://raw.githubusercontent.com/contactsunny/data-science-examples/master/salaryData.csv')

degree=2

coeffs= np.polyfit(data['YearsExperience'], data['Salary'], degree)

poly_func= np.poly1d(coeffs)

x_fit=np.linspace(data['YearsExperience'].min(), data['Salary'].max(), 100)

y_fit=poly_func(x_fit)

plt.scatter(data['YearsExperience'], data['Salary'], label='Data Points')
plt.plot(x_fit, y_fit, label=f'Polynomial Fit (Degree {degree})')
plt.xlabel('Years of Experience')
plt.ylabel('Salary')
plt.title('Polynomial Regression of Salary vs Experience')
plt.legend()
plt.show()
```



4640
Priyanka Yadav

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression

data=pd.read_csv('https://raw.githubusercontent.com/contactsunny/data-science-examples/master/salaryData.csv')

poly_features= PolynomialFeatures(degree=2, include_bias=False)
X_poly= poly_features.fit_transform(data[['YearsExperience']])
y=data['Salary']

model= LinearRegression()

model.fit(X_poly,y)

x_fit= np.linspace(data['YearsExperience'].min(), data['YearsExperience'].max(), 100)
X_fit_poly= poly_features.transform(x_fit.reshape(-1, 1))

y_fit= model.predict(X_fit_poly)

plt.scatter(data['YearsExperience'],y, label='Data Points')
plt.plot(x_fit, y_fit, label=f'Polynomial Fit (Degree 2)')
plt.xlabel('Years of Experience')
plt.ylabel('Salary')
plt.title('Polynomial Regression of Salary vs Experience')
plt.legend()
plt.show()
```



CONCLUSION:

Hence, we successfully performed the polynomial regression and make a plot of the resulting polynomial fit to the data.

4640
Priyanka Yadav

AIM: Implementation of PCA.

THEORY: Principal Component Analysis (PCA) is a widely used dimensionality reduction technique that transforms a dataset into a set of orthogonal (uncorrelated) components while retaining as much variance as possible. It is particularly useful when dealing with high-dimensional datasets where relationships between features are complex.

Advantages of PCA

1. **Dimensionality Reduction**: PCA reduces the number of features (dimensions) in a dataset while retaining the most important information. This leads to simpler models, faster computation, and reduced storage requirements.
2. **Removes Redundancy**: By transforming correlated variables into uncorrelated principal components, PCA eliminates multicollinearity in the dataset, which can improve model performance and reduce overfitting.
3. **Data Visualization**: PCA makes it easier to visualize high-dimensional data in 2D or 3D by projecting the data onto its principal components. This helps in understanding patterns, clusters, and trends.
4. **Noise Reduction**: By keeping only the principal components that explain the most variance, PCA helps reduce noise in the data, leading to more robust models.
5. **Improves Computational Efficiency**: By reducing the number of features, PCA can significantly speed up the training of machine learning algorithms, especially those sensitive to high-dimensionality (e.g., k-NN, SVM).

Disadvantages of PCA

1. **Loss of Information**: While PCA aims to retain the most important information, it can result in the loss of some information, especially when many components are discarded. This might affect the accuracy of models trained on reduced data.
2. **Difficult to Interpret Principal Components**: The transformed principal components are linear combinations of the original features, making them difficult to interpret, especially when compared to the original features.
3. **Assumes Linearity**: PCA assumes that the directions of maximum variance capture the most important information in the data. However, it may fail to capture complex, non-linear relationships in the data.
4. **Sensitive to Scaling**: PCA is sensitive to the scale of features. If the features are not standardized (e.g., through normalization or scaling), PCA might give undue importance to variables with higher magnitudes, leading to biased results.

5. **Principal Components Do Not Always Have Meaningful Physical Interpretations**: The new principal components are linear combinations of the original variables and often lack clear or intuitive meaning, making the results harder to interpret in a practical sense.

Applications of PCA

1. **Dimensionality Reduction in Machine Learning**: PCA is widely used to reduce the number of features in datasets, especially before applying machine learning algorithms like SVM, k-NN, or neural networks to avoid the curse of dimensionality and improve model efficiency.
2. **Image Compression**: PCA is commonly used in image processing for reducing the dimensionality of image data while retaining the most critical information, leading to compression with minimal loss in visual quality.
3. **Facial Recognition**: PCA is used in facial recognition systems to reduce the high-dimensional data of facial images into a smaller set of principal components (called eigenfaces), making the recognition process faster and more efficient.
4. **Noise Filtering**: PCA is applied to filter out noise in datasets by keeping only the principal components that explain significant variance and discarding components associated with noise.
5. **Finance and Portfolio Management**: PCA is used to reduce the dimensionality of financial data, such as stock price movements, and to identify factors that explain the most variance in the market. It also helps in risk management and asset allocation.

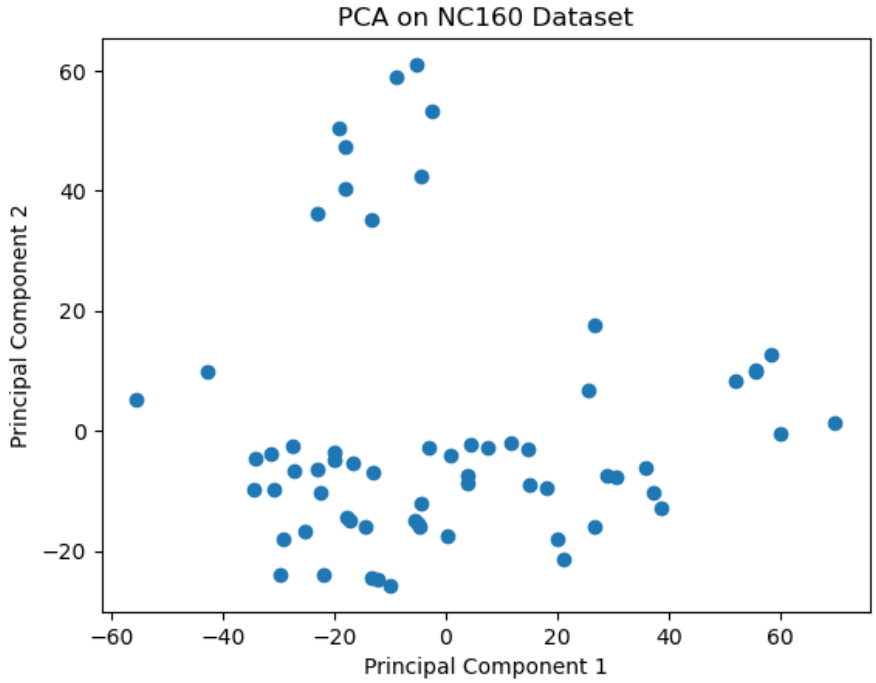
CODE & OUTPUT:

4640
Priyanka Yadav

```
import pandas as pd
import numpy as np
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt

# Load the dataset
df = pd.read_csv('https://raw.githubusercontent.com/vincentarelbundock/Rdatasets/master/csv/ISLR/NCI60.csv', index_col=[-1])
df.drop(columns=["rownames"], axis = 0, inplace = True)
print(df)
# Preprocess the data
scaler = StandardScaler()
df_scaled = scaler.fit_transform(df)

# Perform PCA
pca = PCA(n_components=2)
pca.fit(df_scaled)

# Transform the data
df_pca = pca.transform(df_scaled)

# Visualize the results
plt.scatter(df_pca[:, 0], df_pca[:, 1])
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.title('PCA on NC160 Dataset')
plt.show()
```

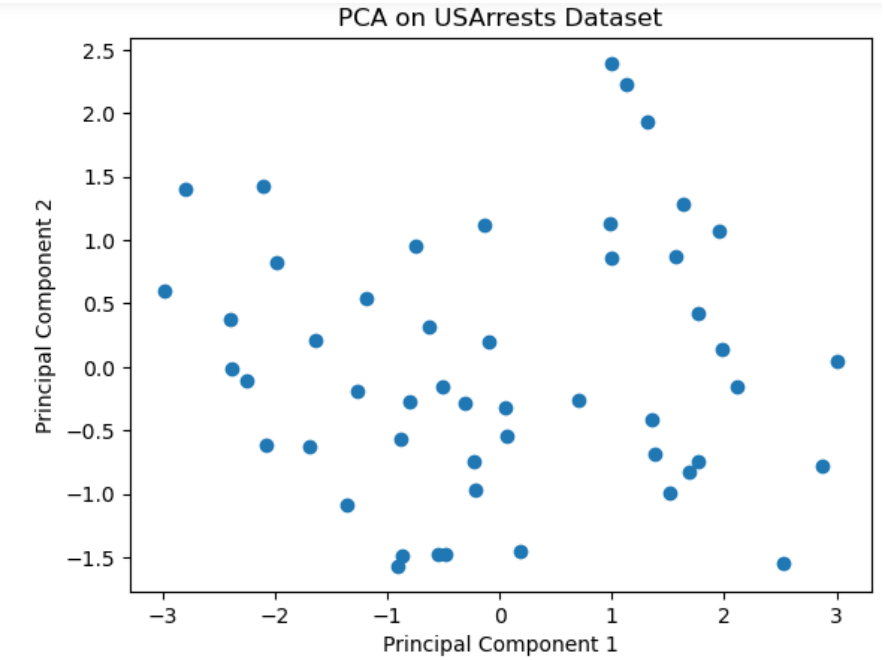
labs	data.1	data.2	data.3	data.4	data.5	data.6	\
CNS	0.300000	1.180000	0.550000	1.140000	-0.265000	-7.000000e-02	
CNS	0.679961	1.289961	0.169961	0.379961	0.464961	5.799610e-01	
CNS	0.940000	-0.040000	-0.170000	-0.040000	-0.605000	0.000000e+00	
RENAL	0.280000	-0.310000	0.680000	-0.810000	0.625000	-1.387779e-17	
BREAST	0.485000	-0.465000	0.395000	0.905000	0.200000	-5.000000e-03	
...	...	...	...	...	...	...	
MELANOMA	-0.030000	-0.480000	0.070000	-0.700000	-0.195000	4.100000e-01	
MELANOMA	-0.270000	0.630000	-0.100000	1.100000	1.045000	8.000000e-02	
MELANOMA	0.210000	-0.620000	-0.150000	-1.330000	0.045000	-4.000000e-01	
MELANOMA	-0.050000	0.140000	-0.090000	-1.260000	0.045000	-2.710505e-20	
MELANOMA	0.350000	-0.270000	0.020000	-1.230000	-0.715000	-3.400000e-01	

4640
Priyanka Yadav




```python
import pandas as pd
import numpy as np
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt

# Load the dataset
df = pd.read_csv('https://raw.githubusercontent.com/vincentarelbundock/Rdatasets/master/csv/datasets/USArrests.csv', index_col=0)
print(df)

# Preprocess the data
scaler = StandardScaler()
df_scaled = scaler.fit_transform(df) # Scale the dataframe without the state names column

# Perform PCA
pca = PCA(n_components=2)
pca.fit(df_scaled)

# Transform the data
df_pca = pca.transform(df_scaled)

# Visualize the results
plt.scatter(df_pca[:, 0], df_pca[:, 1])
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.title('PCA on USArrests Dataset')
plt.show()
```

4640
Priyanka Yadav



AIM: Support Vector Machines (SVM) in Machine Learning

THEORY: Support Vector Machines (SVM) is a popular supervised learning algorithm in machine learning that can be used for classification and regression tasks. The goal of SVM is to find the best hyperplane that separates the data into different classes with the maximum margin.

Advantages of SVM

1. **Effective in High-Dimensional Spaces**: SVM performs well when the number of features (dimensionality) is high, even when the number of samples is smaller than the number of features, making it well-suited for complex datasets.
2. **Robust to Overfitting (with Proper Regularization)**: By using the regularization parameter (C), SVM can be fine-tuned to avoid overfitting, especially in high-dimensional spaces. It generalizes well when the proper hyperparameters are selected.
3. **Works Well for Non-Linear Data (with Kernels)**: The kernel trick allows SVM to classify non-linearly separable data by mapping it into a higher-dimensional space, making it effective for complex decision boundaries.
4. **Maximizes Margin**: SVM focuses on finding the hyperplane that maximizes the margin (distance) between classes, making it less sensitive to data points close to the decision boundary and improving generalization.
5. **Effective in Handling Unstructured Data**: SVM is known to work well with unstructured and semi-structured data such as text, images, and videos, making it a popular choice in fields like natural language processing (NLP) and computer vision.

Disadvantages of SVM

1. **Difficult to Interpret**: SVM models are often considered "black box" algorithms, meaning that it's difficult to interpret how the model is making predictions, especially when using complex kernels.
2. **Sensitive to Parameter Selection**: The performance of SVM heavily depends on the selection of key parameters like the regularization parameter (C) and kernel parameters (e.g., γ for the RBF kernel). Poorly chosen parameters can lead to overfitting or underfitting.
3. **Computationally Expensive**: Training an SVM can be slow, especially for large datasets. The algorithm requires solving a quadratic optimization problem, which becomes computationally expensive as the dataset size increases.
4. **Memory-Intensive**: For large datasets, SVM can require significant memory to store support vectors, making it less suitable for large-scale applications without optimization techniques.
5. **Not Suitable for Very Large Datasets**: SVM may not scale well to extremely large datasets due to its high computational cost, particularly when training with non-linear kernels.

Applications of SVM

1. **Text Classification and NLP**: SVM is widely used in text classification tasks such as spam detection, sentiment analysis, and topic categorization. Its ability to handle high-dimensional data makes it ideal for document classification, where each word can be treated as a feature.
2. **Image Classification**: In computer vision, SVM is used for tasks like object detection, facial recognition, and handwritten digit classification (e.g., MNIST dataset). The kernel trick allows SVM to perform well in image recognition problems.
3. **Medical Diagnosis**: SVM has been used in medical fields to classify diseases such as cancer (e.g., tumor classification) or predict patient outcomes based on medical data. It can handle non-linear relationships between features, which are common in medical datasets.
4. **Bioinformatics**: SVM is used to classify proteins, genes, or sequences, making it useful for applications like gene expression analysis and protein classification.
5. **Handwriting Recognition**: SVM is applied in handwriting recognition systems, such as recognizing handwritten digits, characters, or letters, using features extracted from pixel values or shapes.

CODE & OUTPUT:

4640
Priyanka Yadav

```
: from sklearn import datasets
  from sklearn.model_selection import train_test_split
  from sklearn import svm
  from sklearn import metrics

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

  #split ds into training and test set
  X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.2, random_state=42)

  #create a SVM classifier
  classifier=svm.SVC()

  #train the classifier
  classifier.fit(X_train,y_train)

  #predict response for test dataset
  y_pred=classifier.predict(X_test)

  #model accuracy
  print("Accuracy", metrics.accuracy_score(y_test,y_pred))
```

```
Accuracy 1.0
```

CONCLUSION:

Hence, we successfully performed practical using Support Vector Machines (SVM) in Machine Learning.

AIM: Implementation of Classification Tree.

THEORY: Classification Trees (also known as Decision Trees for classification) are a type of supervised learning algorithm used to predict the label of a target variable by learning decision rules derived from the features. The tree structure consists of nodes representing features, branches representing decision rules, and leaves representing the outcomes or classes.

Advantages of Classification Trees

1. **Easy to Understand and Interpret**: It is simple to understand and visualize. The decision process is laid out in a tree structure, which makes it easy for non-experts to interpret how decisions are made.
2. **No Need for Feature Scaling**: Unlike algorithms like SVM or k-NN, decision trees do not require features to be normalized or scaled, which simplifies the preprocessing steps.
3. **Handles Both Numerical and Categorical Data**: can handle both continuous numerical features and categorical features without any specific transformations, making them flexible for a wide range of datasets.

4. **Can Handle Missing Values**: They handle missing data, as they can use surrogate splits or skip instances without affecting the overall prediction much.
5. **Non-Parametric**: It non-parametric, meaning they do not make any assumptions about the underlying distribution of the data. This makes them robust to a wide range of real-world data patterns.

Disadvantages of Classification Trees

1. **Prone to Overfitting**: Classification trees are prone to overfitting, especially if they are deep and contain too many branches. This results in poor generalization to unseen data.
2. **Unstable and Sensitive to Small Variations in Data**: Small changes in the dataset can lead to significant changes in the structure of the decision tree, leading to a completely different model. This makes decision trees less robust.
3. **Biased Toward Dominant Features**: Decision trees tend to be biased toward features with more levels (categories or distinct values) because they provide more opportunities for splitting, even if they are not the most important features.
4. **Not Suitable for Linear Relationships**: Decision trees struggle with linear relationships between features and target variables, which may be better captured by algorithms like logistic regression or SVM.
5. **Greedy Algorithm**: Decision trees use a greedy algorithm that makes locally optimal choices at each node (e.g., by maximizing information gain), but this doesn't guarantee that the final tree is globally optimal.

Applications of Classification Trees

1. **Customer Segmentation and Targeting**: In marketing, They can be used to segment customers based on features such as age, income, or purchase history to predict whether they will respond to a marketing campaign.
2. **Credit Scoring and Risk Assessment**: They are used in financial services to assess creditworthiness, predicting whether a loan applicant is likely to default based on factors like credit history, income, and employment status.
3. **Medical Diagnosis**: They are applied in healthcare to classify diseases based on symptoms and medical tests. For example, a classification tree can predict whether a patient has a certain disease based on diagnostic features like blood pressure, cholesterol levels, or age.
4. **Fraud Detection**: In fraud detection systems,it is used to classify transactions as legitimate or fraudulent based on features such as transaction amount, location, and time of purchase.
5. **Churn Prediction**: Telecom companies use it to predict customer churn by analyzing customer behavior data, such as call patterns, subscription plans, and payment history.

CODE & OUTPUT:

4640
Priyanka Yadav

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn.datasets import load_iris
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
```

```
iris= load_iris()
X= iris.data
y= iris.target
```

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

```
clf= DecisionTreeClassifier(random_state=42)
```

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

```
DecisionTreeClassifier(random_state=42)
```

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

```
accuracy= accuracy_score(y_test, y_pred)
print("Accuracy: ",accuracy)
print("Classification Report:")
print(classification_report(y_test, y_pred))
print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))
```

```
Accuracy:  1.0
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

Confusion Matrix:
[[10  0  0]
 [ 0  9  0]
 [ 0  0 11]]
```

CONCLUSION:

Hence, we successfully performed practical using Classification Tree.

AIM: For a given dataset, split the dataset into training and testing. Fit the Bagging Model on the training set and evaluate the performance on the test set.

THEORY: Bagging, short for Bootstrap Aggregating, is an ensemble learning technique used in machine learning to enhance the stability and accuracy of algorithms. It primarily aims to reduce variance and combat overfitting, especially in high-variance models like decision trees.

How Bagging works

The core idea behind is to train multiple models independently on different random subsets of the training data. Here's a breakdown of the process:

1. **Bootstrap Sampling:-** Random subsets of the original dataset are created by sampling with replacement. This means some instances may appear multiple times in a subset, while others may be omitted.

2. **Model Training**:- A separate model (often the same type, like a decision tree) is trained on each bootstrap sample. This helps to reduce variance, especially in models that are prone to overfitting.
3. **Aggregation**:- For regression tasks, the predictions of all the models are averaged. For classification tasks, a majority vote is typically used to determine the final class.

Advantages of Bagging

1. **Reduces Variance (Less Overfitting)**: Bagging helps to reduce the variance of the model by combining multiple independent learners, each trained on different samples of the data. This leads to better generalization and reduces the risk of overfitting, particularly for high-variance models like decision trees.
2. **Works Well with High Variance Models**: Models like decision trees are prone to overfitting, but bagging stabilizes them, making it especially effective when using algorithms like decision trees (e.g., Random Forest).
3. **Parallelization**: Since each model is trained independently on different bootstrapped datasets, bagging can be parallelized easily, leading to faster training when resources are available.
4. **Robust to Outliers**: Bagging makes models more robust to outliers because each individual learner is trained on a different subset of the data, meaning that outliers won't dominate the training process for all learners.
5. **Simple to Implement**: The bagging process is relatively simple to implement and understand. It doesn't require much tuning apart from choosing the number of models and selecting the base learner.

Disadvantages of Bagging

1. **Loss of Interpretability**: Bagging generates multiple models and averages them, which can make the final model difficult to interpret compared to a single decision tree.
2. **Requires More Computation**: Since multiple models are trained, bagging requires significantly more computation and memory compared to a single learner.
3. **Cannot Significantly Reduce Bias**: Bagging is most effective at reducing variance, but it doesn't help much in reducing bias. If the base model is biased (e.g., too simple), bagging may not improve the overall performance.

Applications of Bagging

1. **Random Forests**: One of the most popular applications of bagging is in Random Forests, where multiple decision trees are combined to improve the accuracy and robustness of predictions.

2. **Spam Detection**: Bagging is used in spam email detection to improve the robustness of classifiers like decision trees, reducing false positives and negatives.
3. **Credit Risk Prediction**: Bagging is employed in financial sectors to predict credit risk by reducing the variance of decision tree models, leading to more reliable predictions.
4. **Customer Churn Prediction**: Bagging can be used to improve customer churn prediction models by combining the results of several weak learners to improve accuracy and stability.

CODE & OUTPUT:

```
from sklearn.model_selection import train_test_split
from sklearn.ensemble import BaggingRegressor
from sklearn.metrics import mean_squared_error
from sklearn.preprocessing import StandardScaler
import pandas as pd
```

```
data=pd.read_csv('https://raw.githubusercontent.com/plotly/datasets/refs/heads/master/tesla-stock-price.csv')
```

```
X= data[['open','high','low']]
y= data['close']
```

```
X_train, X_test, y_train, y_test= train_test_split(X,y, test_size=0.4, random_state=42)
```

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

```
model= BaggingRegressor(n_estimators=100, random_state=42)
model.fit(X_train_scaled, y_train)
```

```
BaggingRegressor
BaggingRegressor(n_estimators=100, random_state=42)
```

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

```
mse= mean_squared_error(y_test, y_pred)
print(f"Mean Squared Error: {mse}")
```

```
Mean Squared Error: 17.100339701881076
```

AIM: For a given dataset, split the dataset into training and testing. Fit the Boosting Model on the training set and evaluate the performance on the test set.

THEORY: Boosting is another powerful ensemble learning technique in machine learning that aims to improve the accuracy of models by combining multiple weak learners to create a strong learner. Here's an overview of how boosting works:

1. **Weak Learners**:- Boosting primarily uses weak learners, which are models that perform slightly better than random chance. Common choices include decision stumps (one-level decision trees).

2. **Sequential Training**:- Unlike bagging, where models are trained independently, boosting trains models sequentially. Each model is trained to correct the errors made by the previous ones.
3. **Weight Adjustment**:- After each iteration, boosting adjusts the weights of the training instances. Misclassified instances are given more weight, so the next model focuses on correcting those errors.

Final Prediction:- The final prediction is made by combining the predictions of all the weak learners, typically through a weighted sum or majority vote.

Advantages of Boosting

1. **Reduces Bias and Variance**: Boosting reduces both bias and variance, improving the model's accuracy. While bagging mainly reduces variance, boosting can simultaneously reduce bias, making it suitable for complex models.
2. **Better Performance**: Boosting generally achieves better performance compared to bagging because it focuses on the hardest-to-predict cases, leading to more accurate predictions, particularly for difficult problems.
3. **Handles Weak Learners**: Boosting can combine weak learners (e.g., shallow trees) to create a strong predictive model, even if individual learners perform only slightly better than random guessing.
4. **Can Focus on Hard-to-Classify Instances**: By assigning higher weights to misclassified points, boosting ensures that the model improves its performance on the difficult instances in the data.
5. **Flexibility with Different Base Learners**: Boosting is not limited to decision trees. It can be applied with different types of base learners like SVMs, linear models, or neural networks.

Disadvantages of Boosting

1. **Prone to Overfitting (with Noisy Data)**: Boosting is sensitive to noisy data and outliers, as the algorithm will continue to focus on hard-to-classify points, which could be outliers. This can lead to overfitting if not controlled.
2. **Slow Training Process**: Since boosting builds models sequentially, with each model depending on the previous one, the training process can be slow compared to bagging, which can be parallelized.
3. **More Complex to Tune**: Boosting involves tuning several hyperparameters (e.g., learning rate, number of iterations, maximum depth), making it more complex and time-consuming to optimize compared to bagging.
4. **Not Suitable for Large Datasets**: Due to its iterative nature, boosting may struggle with very large datasets, where the training time could become prohibitive unless optimized carefully.

5. **Sensitive to Outliers**: Boosting algorithms are highly sensitive to noisy data and outliers because they tend to focus on the hardest examples, which might be noisy data points.

Applications of Boosting

1. **Gradient Boosting Machines (GBM)**: One of the most well-known applications of boosting is in GBM, which is widely used in predictive modeling tasks such as customer churn prediction, click-through rate (CTR) prediction, and fraud detection.
2. **AdaBoost (Adaptive Boosting)**: AdaBoost is used for binary classification problems like face detection, where the algorithm boosts the performance of weak classifiers and improves accuracy.
3. **XGBoost**: XGBoost, a highly efficient implementation of gradient boosting, is widely used in data science competitions and applications like recommendation systems, time series forecasting, and ranking problems.
4. **Medical Diagnosis**: Boosting techniques are used in healthcare applications to diagnose diseases, such as classifying patients based on medical history or genetic data, where accurate predictions are critical.
5. **Credit Scoring**: Boosting is applied in the financial sector for credit scoring systems, where it improves the accuracy of predicting loan defaults or customer creditworthiness.

CODE & OUTPUT:

```
from sklearn.model_selection import train_test_split
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.metrics import mean_squared_error
from sklearn.preprocessing import StandardScaler
import pandas as pd
```

```
data=pd.read_csv('https://raw.githubusercontent.com/plotly/datasets/refs/heads/master/tesla-stock-price.csv')
```

```
data['date']= pd.to_datetime(data['date'])
```

```
X= data[['open','high','low']]
y= data['close']
```

```
X_train, X_test, y_train, y_test= train_test_split(X,y, test_size=0.4, random_state=42)
```

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

4640
Priyanka Yadav

```
model= GradientBoostingRegressor(n_estimators=100, learning_rate=0.1, random_state=42)
model.fit(X_train_scaled, y_train)
```

```
GradientBoostingRegressor(random_state=42)
```

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

```
mse= mean_squared_error(y_test, y_pred)
print(f"Mean Squared Error: {mse}")
```

```
Mean Squared Error: 17.09685378425005
```

CONCLUSION:

Hence, we successfully performed practical using Boosting and calculating Accuracy.