Practical 1(A)

Name :Vinay partap Singh

Course Name – MLT Lab

Section – B

Course Code – CSE-2424

Roll No. – 177

Reg No. – 20010232

Aim:- Linear regression using linear least squares fit method.

Problem Statement :-

Predict the Salary of a Student (in Lakhs) during Campus Placement, given his CGPA (on a Scale of 0 – 10) and Previous Year's Placement Records.

Theory :-

Linear regression is a statistical method used to model the relationship between a dependent variable (usually denoted as "Y") and one or more independent variables (often denoted as "X"). Linear regression aims to find the best-fitting linear equation that describes the relationship between these variables. The method typically used to achieve this is called the linear least squares fit.

The linear regression equation takes the form:

$Y = \beta 0 + \beta 1 X1 + \beta 2 X2 + ... + \beta n * Xn + \varepsilon$

Here, Y is the dependent variable, X1, X2, ..., Xn are the independent variables, β0 is the intercept, β1, β2, ..., βn are the coefficients (slopes) for each independent variable, and ε represents the error term.

The goal is to estimate the coefficients (β0, β1, β2, ..., βn) in a way that minimizes the sum of the squared differences between the predicted values (Y) and the actual data points. This is done by solving a system of equations using the method of least squares.

The formulas for calculating the coefficients are:

$\beta 1 = \Sigma((Xi - \bar{X})(Yi - \bar{Y})) / \Sigma((Xi - \bar{X})^2)$ $\beta 0 = \bar{Y} - \beta 1 * \bar{X}$

Where Σ represents summation, Xi and Yi are the data points, $\bar{X}$ and $\bar{Y}$ are the means of the independent and dependent variables, respectively.

Linear regression is widely used in various fields for tasks such as prediction, modeling, and understanding relationships between variables due to its simplicity and interpretability.

Data Description:- Data has two Columns referring to CGPA and Salary of students from previous years placed through College Campus Placement. CGPA Ranges from 6.1 to 9.52 and Salary Ranges from 0 Lakhs to 14.5 Lakhs.

CGPA	Salary(in Lakhs)
6.1	0.0
6.15	2.5
6.3	2.25
7.24	6.0
7.5	3.3
7.5	3.75
7.9	4.5
8.0	3.3
8.9	4.0
9.1	3.5
9.5	6.5
9.5	10.5

9.52	14.5
Mean(CGPA) = 7.94	Mean(Salary)= 4.9

Program :-

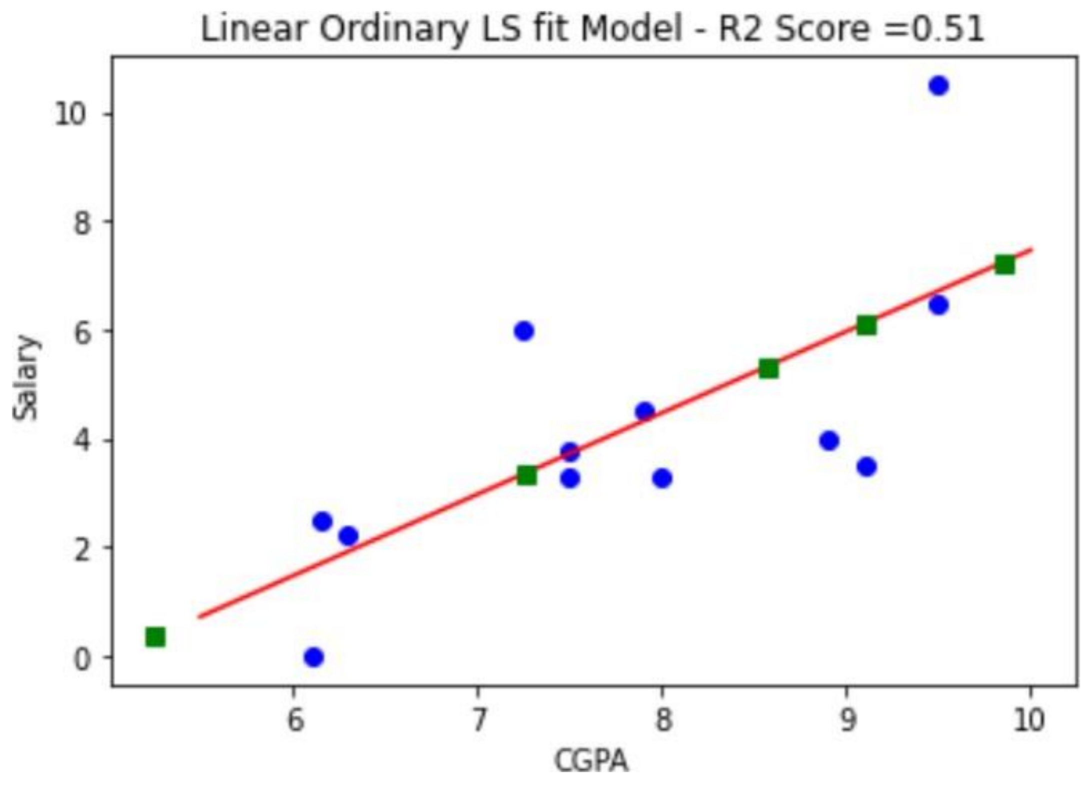
```python
import matplotlib.pyplot as plt
from numpy import array, zeros,mean,percentile,where
from sklearn.metrics import r2_score
cgpa = array(
[6.1, 6.15, 6.30, 7.24,
7.50, 7.50, 7.90, 8.0,
8.9, 9.1, 9.5, 9.5, 9.52])
salary = array(
[0.0, 2.50, 2.25, 6.00,
3.30, 3.75, 4.5, 3.30,
4.0, 3.5, 6.5, 10.5, 14.5])
# Finding slope and y_intercept
# for the best fit line using formulae
cgpa=cgpa[cgpa<=percentile(cgpa,95)]
salary=salary[salary<=percentile(salary,95)]
average_cgpa, average_salary = mean(cgpa), mean(salary)
numerator, denominator = 0, 0
for i in range(len(cgpa)):
    numerator += (cgpa[i]-average_cgpa) * (salary[i]-average_salary)
    denominator += (cgpa[i]-average_cgpa) ** 2
slope = numerator / denominator
y_intercept = average_salary - slope * average_cgpa
print('Equation of the best fit line is: y = '+ f'{slope:.2f} * x +
{y_intercept:.2f}')
# Computing R square value
predicted_salary = zeros((len(cgpa), 1))
for i in range(len(cgpa)):
    predicted_salary[i] = slope * cgpa[i] + y_intercept
score_r2 = round(r2_score(salary, predicted_salary), 2)
print('R2 Score:', score_r2)

#plot the training instances(points)
plt.scatter(cgpa, salary, color = 'blue')


#Plotting the best fit line
plt.xlabel("CGPA")
plt.ylabel("Salary")
plt.title(f"Linear Ordinary LS fit Model - R2 Score ={score_r2}")
plt.plot([5.5, 10.0], slope * array([5.5, 10.0]) + y_intercept,'red')
```

```python
#For the given CGPAs find the predicted salary
predictor_cgpa = array([9.11, 5.25, 8.58, 7.26, 9.85])
response_salary = slope * predictor_cgpa + y_intercept
for i in range(len(predictor_cgpa)):
    response_salary[i] = response_salary[i] if response_salary[i]> 0 else 0
    plt.plot(predictor_cgpa[i],  response_salary[i],  'gs')
    print(f'Predicted Salary for the Student {i+1} with CGPA '+
f'{predictor_cgpa[i]} is Rs {response_salary[i]:.2f}Lac')
```

Output :-

```
R2 Score: 0.51
Predicted Salary for the Student 1 with CGPA 9.11 is Rs 6.13Lac
Predicted Salary for the Student 2 with CGPA 5.25 is Rs 0.34Lac
Predicted Salary for the Student 3 with CGPA 8.58 is Rs 5.33Lac
Predicted Salary for the Student 4 with CGPA 7.26 is Rs 3.35Lac
Predicted Salary for the Student 5 with CGPA 9.85 is Rs 7.24Lac
```

Graphs :-



Conclusion :- In this experiment we learnt about how to create a linear regression model using linear least square fit method and use to predict salary from cgpa.

Practical No. 1(B)

Aim :- Linear regression with Ordinary least squares method using Scikit-learn.

Problem Statement :-

Predict the Salary of a Student (inLakhs) during Campus Placement, given his CGPA (on a Scale of 0 – 10) and Previous Year's Placement Records..

Theory :-

Linear regression with Ordinary Least Squares (OLS) is a fundamental machine learning technique for modeling the relationship between a dependent variable (Y) and one or more independent variables (X). Scikit-Learn, a popular Python library, provides a convenient way to perform OLS linear regression.

In OLS, we aim to find the line (for simple linear regression) or hyperplane (for multiple linear regression) that minimizes the sum of squared differences between the observed and predicted values. The model is represented as:

Y = β0 + β1*X1 + β2*X2 + ... + βn*Xn + ε

- Y: Dependent variable
- X1, X2, ..., Xn: Independent variables
- β0: Intercept
- β1, β2, ..., βn: Coefficients
- ε: Error term

The OLS method calculates the coefficients (β0, β1, β2, ..., βn) by minimizing the residual sum of squares (RSS), which is the sum of the squared differences between the observed and predicted values:

RSS = Σ(Yi - Ŷi)^2

Scikit-Learn's LinearRegression class makes it straightforward to perform OLS linear regression. You can fit the model to your data, obtain coefficients, make predictions, and assess model performance with various evaluation metrics.

Data Description:- Data has two Columns referring to CGPA and Salary of students from previous years placed through College Campus Placement. CGPA Ranges from 6.1 to 9.52 and Salary Ranges from 0 Lakhs to 14.5 Lakhs.

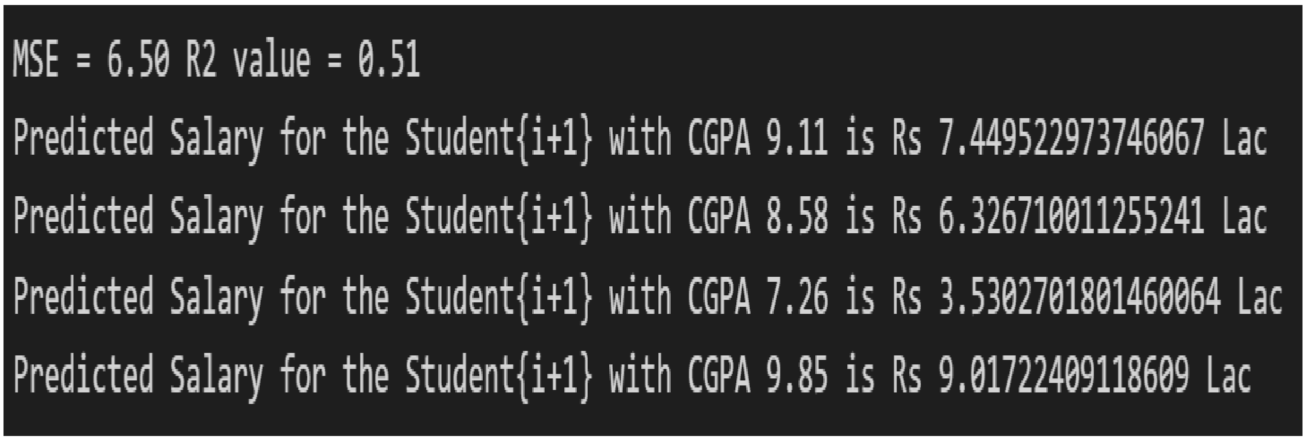
CGPA	Salary(in Lakhs)
6.1	0.0
6.15	2.5
6.3	2.25
7.24	6.0
7.5	3.3
7.5	3.75
7.9	4.5
8.0	3.3
8.9	4.0
9.1	3.5
9.5	6.5
9.5	10.5
9.52	14.5
Mean(CGPA) = 7.94	Mean(Salary)= 4.9

Program :-

```python
import matplotlib.pyplot as plt
from numpy import array,percentile
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error as mse
# past data for training the regression model
```

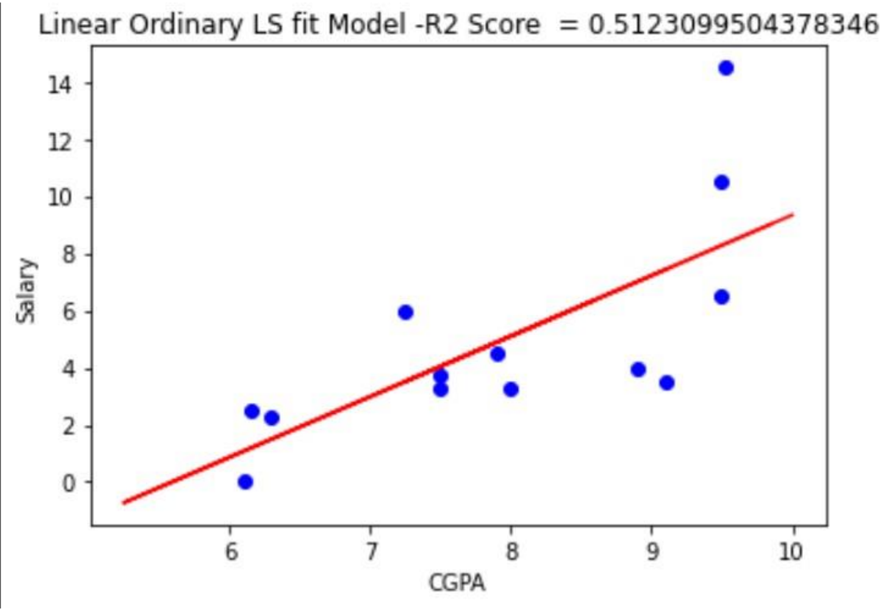
```
cgpa = array([6.1, 6.15, 6.30, 7.24,7.50,7.50, 7.9, 8.0,
8.9, 9.1,9.5, 9.5, 9.52]).reshape(-1,1)
salary = array([0.0, 2.50, 2.25, 6.00,3.30,3.75, 4.5, 3.3,
4.0, 3.5,6.5, 10.5, 14.5]).reshape(-1,1)
# plot the training instances(points)
plt.scatter(cgpa, salary, color='blue')
# Defining and fitting the model
ols = LinearRegression()
ols.fit(cgpa, salary)
#Visualizing the linear model and Plottingthe best fit
line
plt.xlabel("CGPA")
plt.ylabel("Salary")
plt.title("Linear Ordinary LS fit Model -R2 Score "+f" =
{ols.score(cgpa, salary)}")
plt.plot([5.5, 10.0],ols.predict(array([5.5,
10.0]).reshape(-1,1)), 'red')
print(f'MSE = {mse(salary,ols.predict(cgpa)):.2f} '+ f'R2
value = {ols.score(cgpa,salary):.2f}')
#For the given CGPAs find the predictedsalary
predictor_cgpa = array([9.11, 5.25, 8.58,7.26,
9.85]).reshape(-1,1)
response_salary =ols.predict(predictor_cgpa)
plt.plot(predictor_cgpa, response_salary,'red')
for i in range(len(predictor_cgpa)):
    response_salary[i] = response_salary[i]
if response_salary[i] > 0:
    print('Predicted Salary for the Student{i+1} with CGPA
'+ f'{predictor_cgpa[i][0]} is
Rs{response_salary[i][0]} Lac')
```

Output :-



```
MSE = 6.50 R2 value = 0.51

Predicted Salary for the Student{i+1} with CGPA 9.11 is Rs 7.449522973746067 Lac

Predicted Salary for the Student{i+1} with CGPA 8.58 is Rs 6.326710011255241 Lac

Predicted Salary for the Student{i+1} with CGPA 7.26 is Rs 3.5302701801460064 Lac

Predicted Salary for the Student{i+1} with CGPA 9.85 is Rs 9.01722409118609 Lac
```

Graph :-



Conclusion – In this experiment we learnt about Linear regression using Sickit-learn library.

Practical No 1(C)

Aim:- **Linear Regression using gradient descent algorithm.**

Problem Statement :-

Predict the random uniform distributed points y = 2 * X + 4 + np.random.randn(100) * 0.1 from 100 random inputs in column X.

Theory :-

Linear regression is a fundamental machine learning technique used for predicting a continuous output variable (Y) based on one or more input features (X). The gradient descent algorithm is a commonly used optimization technique for training linear regression models.

In linear regression, we seek to find the best-fit line represented by the equation:

$Y = \beta_0 + \beta_1 X + \varepsilon$

Where:

- Y is the predicted output.
- X is the input feature.
- β_0 is the y-intercept.
- β_1 is the slope of the line.
- ε is the error term.

The goal is to minimize the mean squared error (MSE) between the predicted values and the actual data points. Gradient descent helps us find the optimal values of β_0 and β_1 by iteratively updating them:

$\beta_0 = \beta_0 - \alpha * (1/n) * \Sigma(Y_i - (\beta_0 + \beta_1 X_i))$

$\beta_1 = \beta_1 - \alpha * (1/n) * \Sigma((Y_i - (\beta_0 + \beta_1 X_i)) * X_i)$

Where:

- α (alpha) is the learning rate, controlling the step size in each iteration.
- n is the number of data points.

We compute the gradient of the loss function with respect to β0 and β1 and adjust these coefficients accordingly. The process continues until convergence or a predefined number of iterations.

Gradient descent helps linear regression models find the best-fit line that minimizes prediction errors and is a fundamental optimization algorithm in machine learning.

Data Description:- Data has two Columns X and Y where x are 100 random points and y are 100 uniform points.

Program :-

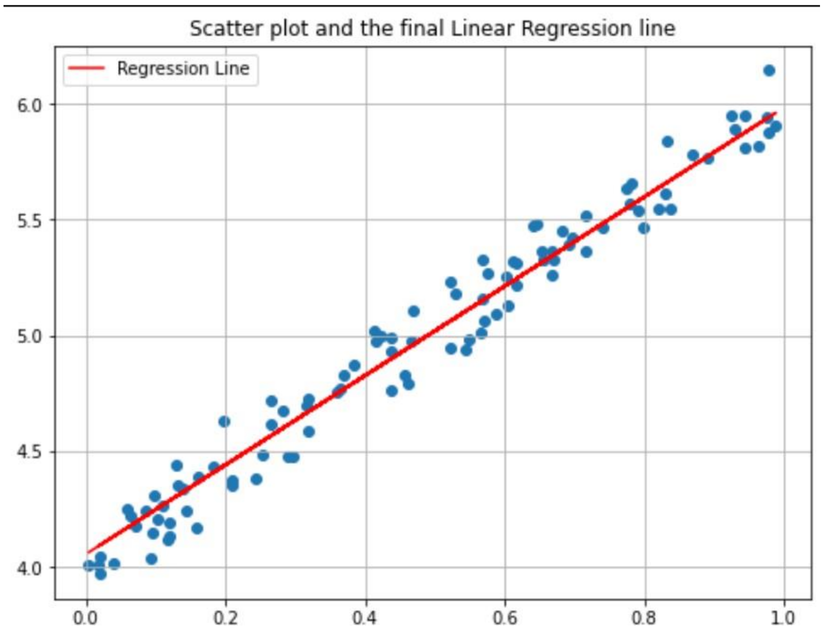
```
import numpy as np

# Generate sample data
# Generate an array of 100 uniformly distributed random numbers in the
interval [0,1)
np.random.seed(0)
X = np.random.rand(100)
y = 2 * X + 4 + np.random.randn(100) * 0.1 # randn() generates 100 normally
distributed numbers around 0

# Initialize parameters
c = np.random.randn() # y-intercept
m = np.random.randn() # slope

alpha = 0.01 # learning_rate
num_iterations = 2000
mse_thresh = .01
n=len(X)
# Gradient Descent

for i in range(num_iterations):
    # Calculate predictions
    ni = i
    y_pred = c + m * X #calulate predicted y vector
    # Calculate gradients
    D_m = 2*sum(X * (y_pred - y))/n # Derivative wrt m
    D_c = 2*sum(y_pred - y)/n  # Derivative wrt c
```

```
        # Update parameters
        m = m - alpha * D_m # Update m
        c = c - alpha * D_c # Update c

    # Calculate Mean Squared Error
        mse = np.mean((y_pred - y) ** 2)
        if mse<=mse_thresh:
            print(f"\nConverged in {i} iterations.")
            break
    else:
        print(f"\nNot Converged to the desired MSE of {mse_thresh} after
    {num_iterations} iterations.")

    # Calculate coefficient of determination R square
    from sklearn.metrics import r2_score
    R2 = r2_score(y,y_pred)

    # Print estimated parameters
    print("Estimated Parameters:")
    print("Intercept (c):", c)


   print("Slope (m):", m)
   print("MSE = ",mse)
   print("R squared value = ",R2)


   #  Plot  scatter  plot  and  regression  line
   plt.rcParams['figure.figsize']   =   (8,   6)
   plt.scatter(X, y)
   plt.plot(X, y_pred, color='red', label='Regression Line') plt.title('Scatter plot
   and the final Linear Regression line')plt.legend()
   plt.grid(True)
   plt.show()
```

Output :-

```
Not Converged to the desired MSE of 0.01 after 2000 iterations.
Estimated Parameters:
Intercept (c): 4.053909351506216
Slope (m): 1.9311523386854845
MSE =  0.01025489546540647
R squared value =  0.9698653293475056
```

Graph :-



Conclusion :- In this experiment we learnt about Linear regression using gradient descent method.

Practical No. 2

Aim:- Implementing multiclass linear classifier using Linear discriminant functions.

Problem Statement –

Iris flower classification using linear discriminant functions.

Theory –

A multiclass linear classifier using Linear Discriminant Functions is a machine learning model used for classification tasks. It's based on discriminant analysis, which aims to find a decision boundary that best separates different classes in the feature space. This classifier is often used when there are more than two classes to distinguish.

Mathematically, let's consider there are K classes. For each class k, we calculate a discriminant function, $g_k(x)$, which represents the likelihood of the input x belonging to class k. These functions are typically linear in nature:

$g_k(x) = w_k^T * x + b_k$

Where:

- $g_k(x)$ is the discriminant function for class k.
- w_k is the weight vector associated with class k.
- b_k is the bias term for class k.

To make a prediction for a new input x, we evaluate all K discriminant functions and assign the input to the class with the highest score:

Predicted class = $\mathrm{argmax}(g_k(x))$

The weight vectors (w_k) and bias terms (b_k) can be estimated from the training data using techniques like Linear Discriminant Analysis (LDA) or other optimization methods.

This approach is effective for linearly separable data, and it's a foundation for more complex classifiers like Support Vector Machines and neural networks when the data isn't linearly separable.

Data Description–

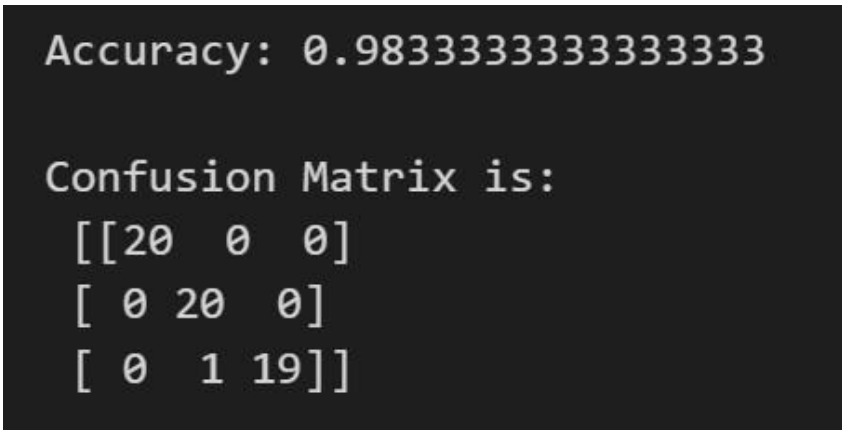
- **Number of Instances:** 150
- **Number of Features:** 4
- **Feature Names:**
  - Sepal Length (in cm)
  - Sepal Width (in cm)
  - Petal Length (in cm)
  - Petal Width (in cm)
- **Target Variable:**
  - Three classes of iris plants to classify:
    - Iris Setosa
    - Iris Versicolor
    - Iris Virginica
- **Data Type:** Multivariate

Program –

```
import numpy as np
from sklearn import datasets
from sklearn.model_selection import train_test_split

# Load the Iris dataset
iris = datasets.load_iris()
```

```python
X = iris.data
y = iris.target

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y,stratify =
iris.target,test_size=0.4, random_state=10)

def Compute_linear_discriminant(cp, x): #cp is class prototype, x is new input
    return np.dot(cp, x) - 0.5 * np.dot(cp, cp)

def calculate_prototypes(X_train, y_train):
    class_prototypes = []

    for class_label in np.unique(y_train):
        # make an array of all the samples belonging to a given class_label
        class_samples = X_train[y_train == class_label]
        #find the mean of these class label
        class_prototype = np.mean(class_samples, axis=0)
        class_prototypes.append(class_prototype)
    return np.array(class_prototypes)


def ld_classifier(test_sample, class_prototypes):
    predicted_class = None
    max_discriminant = float('-inf')

    for i in range(len(class_prototypes)):
        discriminant = Compute_linear_discriminant(class_prototypes[i],
test_sample)
        if discriminant > max_discriminant:
            max_discriminant = discriminant
            predicted_class = i

    return predicted_class

# Calculate class prototypes from the training data
class_prototypes = calculate_prototypes(X_train, y_train)

# Make predictions for all the test samples
predictions = []
for test_sample in X_test:
    predicted_class = ld_classifier(test_sample, class_prototypes)
    predictions.append(predicted_class)

# Performance evaluation of classifier
from sklearn.metrics import confusion_matrix
accuracy = np.mean(np.array(predictions) == y_test)
print("Accuracy:", accuracy)
print('\nConfusion Matrix is:\n',confusion_matrix(y_test, predictions)
```

Output –



```
Accuracy: 0.9833333333333333

Confusion Matrix is:
 [[20  0  0]
 [ 0 20  0]
 [ 0  1 19]]
```

Conclusion :- In this experiment we learnt about multiclass linear classifier using Linear discriminant functions and used on iris dataset.

Practical NO. 3

Aim:- Polynomial regression using scikit learning.

Problem Statement :-

Predicting product Sales Based on amount spend on Advertising.

Theory :-

Polynomial regression is a powerful extension of linear regression in machine learning, allowing us to model relationships that are not linear but instead follow a polynomial curve. It's particularly useful when data points seem to follow a nonlinear pattern.

In linear regression, we model the relationship between a dependent variable (Y) and one or more independent variables (X) as a straight line:

$$Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \ldots + \beta_k X_k + \varepsilon$$

Here, $\beta_0, \beta_1, \beta_2, \ldots, \beta_k$ are the coefficients, $X_1, X_2, \ldots, X_k$ are the independent variables, and ε is the error term.

Polynomial regression, on the other hand, extends this by introducing higher-order terms of the independent variable(s) to capture nonlinear patterns. The equation becomes:

$$Y = \beta_0 + \beta_1 X + \beta_2 X^2 + \ldots + \beta_k X^k + \varepsilon$$

Here, $X^2, X^3, \ldots, X^k$ represent the squared, cubed, and higher-order terms of X, allowing us to model curvatures in the data.
To perform polynomial regression in scikit-learn, follow these steps:

1. Import the necessary libraries.
2. Prepare your data.
3. Create a PolynomialFeatures object to transform your input features into polynomial features.
4. Fit a linear regression model to the polynomial features.
5. Predict values using the model.

Scikit-learn provides convenient tools like **PolynomialFeatures** and **LinearRegression** to simplify the implementation. Adjusting the degree of the polynomial allows you to find the right balance between model complexity and fit to the data, preventing overfitting or underfitting. It's a valuable tool for capturing non-linear relationships in your data.

Data Description:-

Amount spend on Advertising in Lac and product Sales in million.

Program :-

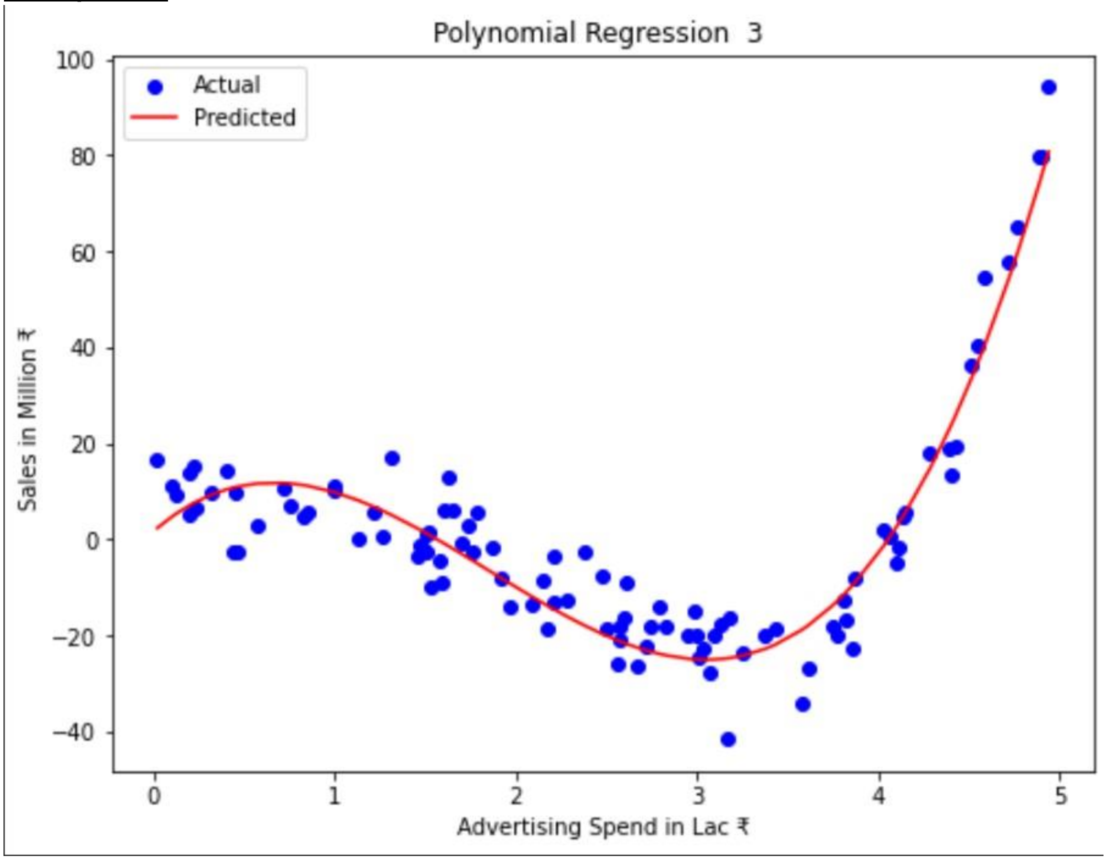
```
import numpy as np
import matplotlib.pyplot as plt
from         sklearn.model_selection         import
train_test_split from  sklearn.preprocessing  import
PolynomialFeatures from sklearn.linear_model  import
LinearRegression
from sklearn.metrics import mean_squared_error, r2_score

# Generate synthetic
datanp.random.seed(10)
X = np.sort(5 * np.random.rand(100, 1), axis=0) # amount spend on Advertising
in Lac
y = X**4 - 4.5 * X**3 + 1.5 * X**2 - 2*X + 10 + 2*np.random.normal(0, 3,
(100,1))# product Sales in million

# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
random_state=0)

# Create polynomial features
degree = 3
poly = PolynomialFeatures(degree)
X_poly =
poly.fit_transform(X_train)

# Train a linear regression model on the polynomial
featuresmodel = LinearRegression()
model.fit(X_poly, y_train)

# Make predictions on the test set
X_test_poly =
poly.transform(X_test)y_pred =
model.predict(X_test_poly)
```

```
# Calculate the Mean Squared Error and R-squared valuemse =

mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)
print("for polynomial of degree:", degree)print("Mean
Squared Error:", mse)
print("R-squared value:", r2)
print(model.coef_)
# Plot the results
plt.scatter(X, y, color='blue', label='Actual')
plt.plot(X, model.predict(poly.transform(X)), color='red', label='Predicted')
plt.title(f'Polynomial Regression {degree} ')
plt.xlabel('Advertising Spend in Lac ₹')
plt.ylabel('Sales in Million ₹ ') plt.legend()
plt.show()
```

Output –

```
for polynomial of degree: 3
Mean Squared Error: 45.17386476019422
R-squared value: 0.9288523111029762
[[  0.          32.80243577 -30.1649506    5.41971317]]
```

Graph:-



Conclusion :- In this experiment we learnt about the polynomial regression and how it can fit non-linearly on data.