Experiment No. 1A

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.

Dataset 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

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

Problem Analysis:

Upon critical Analysis of the given Problem, it can be seen that there's a direct Relationship between CGPA and Salary of the Students.

This type of problem can easily be dealt with by using Linear Regression.

We will use linear least squares which is an approach to fitting a mathematical or statistical model to data in cases where the idealized value provided by the model for any data point is expressed linearly in terms of the unknown parameters of the model.

The resulting fitted model can be used to summarize the data, to predict unobserved values from the same system, and to understand the mechanisms that may underlie the system.

- Machine Learning Task: Linear Regression
- Predictor: CGPA
- Response Variable: Salary
- Model Parameters: Mean
- Hyper Parameters: None

Pseudocode:

```
average_cgpa = average(CGPA)

average_sal = average(Salary)

numerator = Summation[(CGPA_i-
average_cgpa) * (Salary_i –
average_sal)]

numerator = Summation[square(CGPA_i-
average_cgpa)]

slope = numerator / denominator

b = average_sal - slope *
average_cgpa

Assemble the equation of a line

Predicted_sal = slope * salary + b
```

Now for each new prediction:

Multiply CGPA by weight(m) and add the bias(b) to get the predicted Salary(y_p).

Program:

```
"""                         Expt. 1A
Linear regression using linear least
squares fit method
"""
import matplotlib.pyplot as plt
from numpy import array, zeros
from statistics import mean
from sklearn.metrics import r2_score


# past data for training the regression
model
# Example Predictor is CGPA and Target is
Salary
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
average_cgpa, average_salary = mean(cgpa),
mean(salary)
numerator, denominator = 0, 0
```

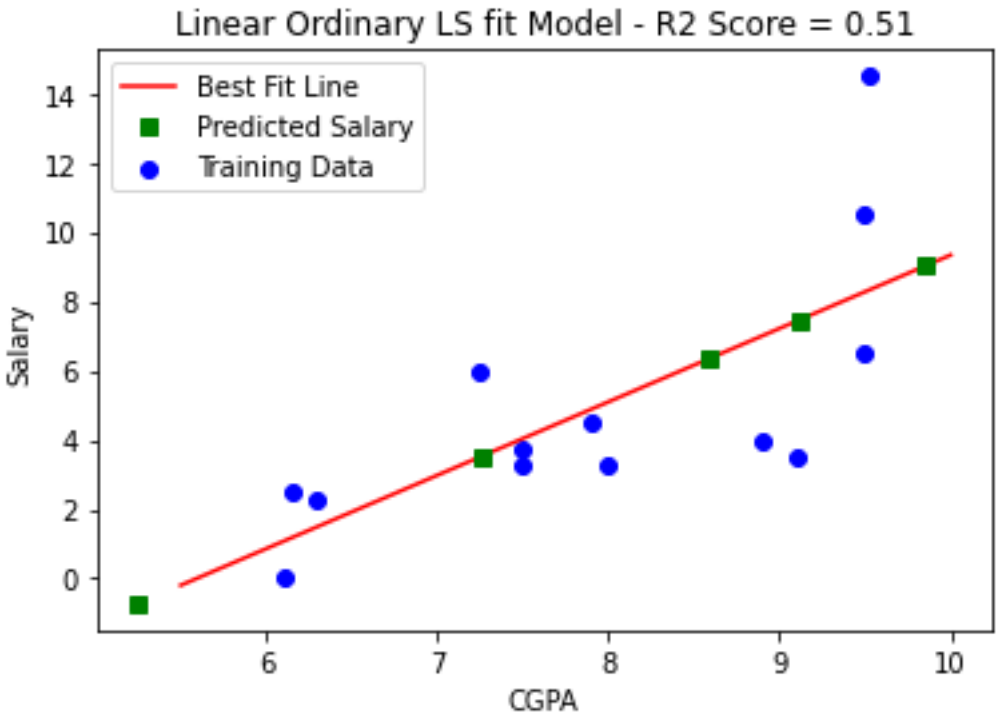
```
for i in range(len(cgpa)):
    numerator += (cgpa[i]-average_cgpa) *
(salary[i]-average_salary)
    denominator += (cgpa[i]-average_cgpa)
** 2

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')


#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:

```
Equation of the best fit line is:
y = 2.12 * x + -11.85
R2 Score: 0.51
Predicted Salary for the Student 1 with CGPA 9.11 is Rs 7.45 Lac
Predicted Salary for the Student 2 with CGPA 5.25 is Rs 0.00 Lac
Predicted Salary for the Student 3 with CGPA 8.58 is Rs 6.33 Lac
Predicted Salary for the Student 4 with CGPA 7.26 is Rs 3.53 Lac
Predicted Salary for the Student 5 with CGPA 9.85 is Rs 9.02 Lac
```

Graphs:



Conclusion:

We have Successfully Predicted the salaries of the Students with given CGPAs with Linear regression using linear least squares fit method.

Experiment No. 1B

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

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.

Dataset 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

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

Problem Analysis:

Upon critical Analysis of the given Problem, it can be seen that there's a direct Relationship between CGPA and Salary of the Students.

This type of problem can easily be dealt with by using Linear Regression.

Ordinary least squares (OLS) is a type of linear least squares method for estimating the unknown parameters in a linear regression model.

OLS chooses the parameters of a linear function of a set of explanatory variables by the principle of least squares: minimizing the sum of the squares of the differences between the observed dependent variable (values of the variable being observed) in the given dataset and those predicted by the linear function of the independent variable.

- Machine Learning Task: Linear Regression
- Predictor: CGPA
- Response Variable: Salary
- Model Parameters: Mean
- Hyper Parameters: None

Pseudocode:

```
ols = Linear_Regression()
ols.fit(CGPA, Salary)
for every predictor_salary:
   ols.predict(cgpa)
```

OLS Working:

$Y = \beta 0 + \Sigma j=1..p\ \beta jXj + \varepsilon$

where Y is the dependent variable, β0, is the intercept of the model, X j corresponds to the jth explanatory variable of the model (j= 1 to p), and e is the random error with expectation 0 and variance σ^2.

The vector of the predicted values can be written as follows:

$y = X\ (X'\ DX)-1\ X'Dy$

Program:

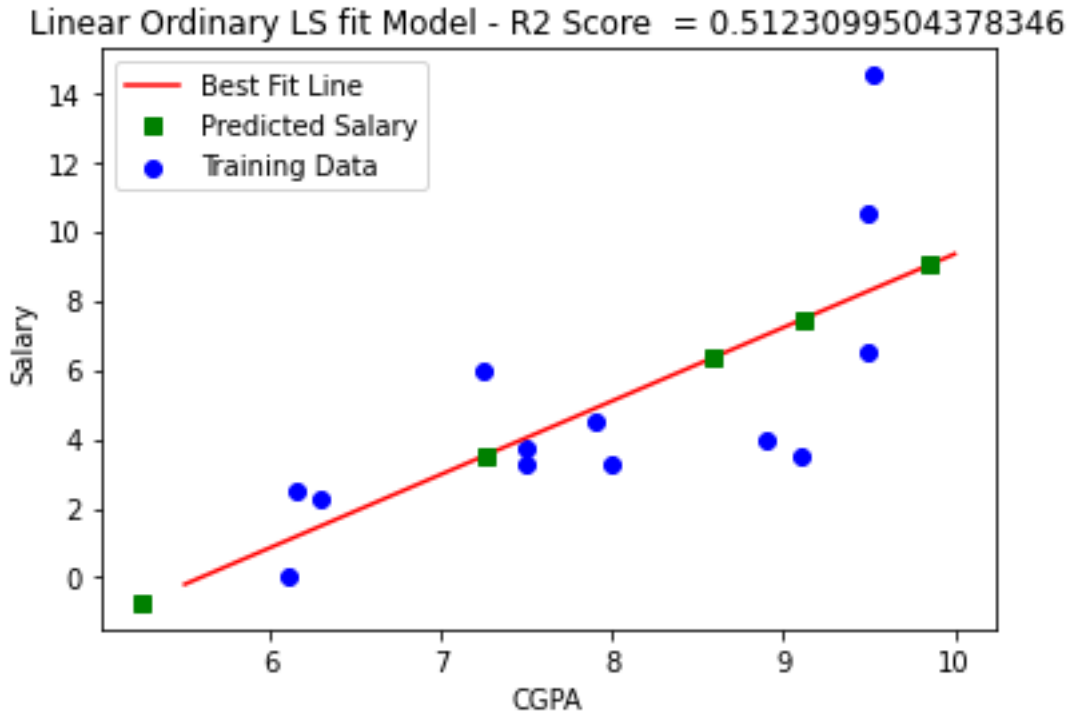
```
"""                Expt. 1B
Linear regression with Ordinary least
squares method
using Scikit-learn
"""
import matplotlib.pyplot as plt
from numpy import array
from sklearn.linear_model import
LinearRegression
from sklearn.metrics import
mean_squared_error as mse

# 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 Plotting
the 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  predicted
salary
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 else 0
    print('Predicted Salary for the Student
{i+1} with CGPA '
      + f'{predictor_cgpa[i][0]} is Rs
{response_salary[i][0]} Lac')
```

Output:

```
Predicted Salary for the Student 1 with
CGPA 9.11 is Rs 7.45 Lac

Predicted Salary for the Student 2 with
CGPA 5.25 is Rs 0.00 Lac

Predicted Salary for the Student 3 with
CGPA 8.58 is Rs 6.33 Lac

Predicted Salary for the Student 4 with
CGPA 7.26 is Rs 3.53 Lac

Predicted Salary for the Student 5 with
CGPA 9.85 is Rs 9.02 Lac
```

Graphs:



Conclusion:

We have Successfully Predicted the salaries of the Students with given CGPAs with Linear regression using Ordinary least squares fit method.

Experiment No. 2

Aim:

Implementing linear classifier using Linear Discriminant Function.

Problem Statement:

Predict, for loan approval, whether a customer is a low risk customer or a high risk customer, given his income (in lakhs) and savings (in thousands) using Linear Discriminant Function.

Dataset Description:

Low Risk Customers and High Risk Customers have two columns i.e. their yearly Income in Lakhs and Savings in thousands represented in Vectors.

Lower Risk Customers have Higher Savings as compared to their salaries, whereas Higher Risk

Customers have Lower Savings as compared to their salaries.

Data for Low Risk Customers

Income (in Lakhs)	Savings (in Thousands)
7.0	0.7
5.8	0.62
5.5	0.5
4.8	0.6
7.5	0.6
3.9	0.5
3.5	0.6
5.3	0.7
9.2	0.6
4.5	0.7

Data for Low Risk Customers

Income (in Lakhs)	Savings (in Thousands)
3.0	0.05
6.5	0.2
4.5	0.12
9.5	0.25
4.8	0.03
6.0	0.1
4.0	0.15
6.2	0.02
5.4	0.21
7.0	0.16

Problem Analysis:

Upon critical Analysis of the given Problem, it can be seen that there's a Relationship between Income and Savings of the Customers.

This type of problem can easily be dealt with by using Linear Discriminant Analysis(LDA).

Linear discriminant analysis (LDA) is a generalization of Fisher's linear discriminant, a method used in statistics and other fields, to find a linear combination of features that characterizes or separates two or more classes of objects or events.

- Machine Learning Task: Classification
- Predictor: Income and Savings
- Response Variable: Category whether Low or High Risk Customer
- Model Parameters: Mean
- Hyper Parameters: Differential Weights and Bias

Pseudocode:

```
avgInL = mean(low_risk_cust->In)
avgSaL = mean(low_risk_cust->Sa)
avgInH = mean(high_risk_cust->In)
avgSaH = mean(high_risk_cust->Sa)
W = [avgInL–avgInH, avgSaL-avgInH]
b = 0.5 * [square(norm(low_center))
      – square(norm(high_center))]
g = dot_product(W, new_Data) + b
g > 0 ? :
   new_Data → low_risk
else:
   new_Data → high_risk
slope = (center2y – center1x)/
         (center2x – center1x)
intercept = centery – centerx *slope
g(x) → slope * x_limits + intercept
```

Program:

```
"""                    Expt. 2
Implementing linear classifier using non-
parametric training approach
"""

import matplotlib.pyplot as plt
from numpy import array, dot
from statistics import mean
from numpy.linalg import norm

# First value in each vector is income(lacs)
# Second is saving amount(lacs) per year

low_risk_customers = array([
    [7,0.7],[5.8,0.62],[5.5,0.5],[4.8,0.6],
        [7.5,0.6],[3.9,0.5],[3.5,0.6],
        [5.3,0.7],[9.2,0.6],[4.5,.7]])

high_risk_customers = array([
    [3,.05],[6.5,0.2],[4.5,0.12],[9.5,0.25],
        [4.8,0.03],[6,0.1],[4,0.15],
        [6.2,0.02],[5.4,.21],[7,0.16]])


#plot the low_risk training instances(points)
plt.scatter(low_risk_customers[:,0],low_risk_c
ustomers[:,1],color = 'blue')

#plot the high_risk training instances(points)
plt.scatter(high_risk_customers[:,0],high_risk
_customers[:,1],color = 'red')
```

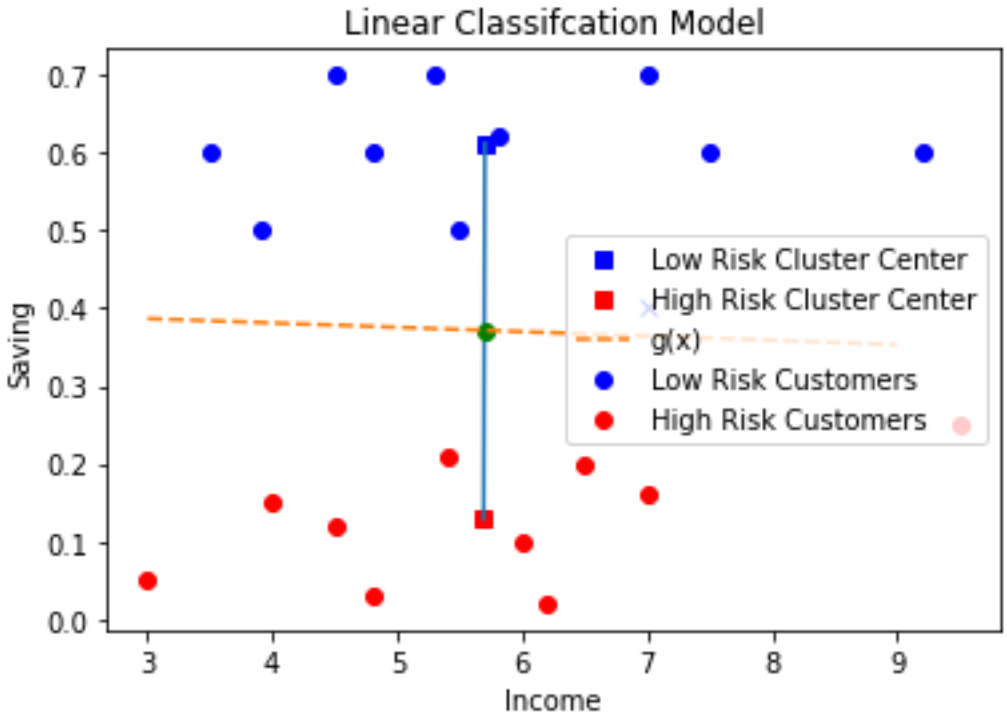
```
#Computing cluster centers for both the classes
low_risk_avg_income = mean(low_risk_customers[:,0])
low_risk_avg_salary = mean(low_risk_customers[:,1])
high_risk_avg_income =
mean(high_risk_customers[:,0])
high_risk_avg_salary =
mean(high_risk_customers[:,1])

low_risk_cluster_center =
array([low_risk_avg_income,low_risk_avg_salary])
high_risk_cluster_center =
array([high_risk_avg_income,high_risk_avg_salary])

#plotting cluster center in green
plt.plot(low_risk_cluster_center[0],low_risk_cluste
r_center[1],'gs')
plt.plot(high_risk_cluster_center[0],high_risk_clus
ter_center[1],'gs')

#Plotting line joining low_risk_cluster_center and
high_risk_cluster_center(centroids)
Cx =
[low_risk_cluster_center[0],high_risk_cluster_cente
r[0]]
Cy =
[low_risk_cluster_center[1],high_risk_cluster_cente
r[1]]
plt.plot(Cx,Cy)

# #Finding g(x) of the decison line
W = low_risk_cluster_center -
high_risk_cluster_center
norm1 = norm(low_risk_cluster_center)
norm2 = norm(high_risk_cluster_center)
b = 0.5 * ((norm2 ** 2) - (norm1 ** 2))
```

```
plt.xlabel("Income")
plt.ylabel("Saving")
plt.title("Linear Classifcation Model")

# Applying the model on new data to
predict class label
X = [7, 0.4]
g = dot(W,X)+b
if(g>0):
    print("Low risk customer ")
    plt.plot(X[0],X[1],'bx')
else:
    print("High risk customer ")
    plt.plot(X[0],X[1],'rx')

# Plotting Perpendicular Bisector
center = array([sum(Cx), sum(Cy)])/2
plt.plot(center[0], center[1], 'go')
slope = (Cy[0] - Cy[1]) / (Cx[0] -
Cx[1])
slope = -1 / slope
intercept = center[1] - slope *
center[0]
Px = array([3.0, 9.0])
Py = slope * Px + intercept
plt.plot(Px, Py, '--')
```

Output:

```
Low risk customer
```

Graphs:



Conclusion:

We have Successfully Predicted the category of the Customers with given Income and Savings by implementing linear classifier using Linear Discriminant Function.

Experiment No. 3

Aim:

Program for Classification using KNN algorithm from Scikit learn library.

Problem Statement:

A hobby botanist is interested in distinguishing the species of some iris flowers that she has found. She has collected some measurements associated with each iris: the length and width of the petals and the length and width of the sepals, all measured in centimetres. She also has the measurements of some irises that have been previously identified by an expert botanist as belonging to the species setosa, versicolor, or virginica.

For these measurements, she can be certain of which species each iris

belongs to. Let’s assume that these are the only species our hobby botanist will encounter in the wild.

Our goal is to build a machine learning model that can learn from the measurements of these irises whose species is known, so that we can predict the species for a new iris.

Dataset Description:

Keys of iris_dataset:

'feature_names'	'target'	'frame'	'DESCR'
'target_names	'filename'	'data'	

Data Set Characteristics:

Number of Instances: 150 (50 in each of three classes)

Number of Attributes: 4 numeric, predictive attributes and the class

Attribute Information:

- sepal length in cm
- sepal width in cm
- petal length in cm
- petal width in cm
- class:
  - Iris-Setosa
  - Iris-Versicolour
  - Iris-Virginica

Feature Names:

'sepal length (cm)'	'sepal width (cm)'
'petal length (cm)'	'petal width (cm)

Target Names:

'setosa'	'versicolor'	'virginica'

Shape of data: (150, 4)

Problem Analysis:

Upon critical Analysis of the given Problem, it can be seen that there's a Relationship between Lengths and Widths of Petals and Sepals and the category of iris.

This type of problem can easily be dealt with by using K Nearest Neighbours for Classification.

In k-NN classification, the output is a class membership. An object is classified by a plurality vote of its neighbours, with the object being assigned to the class most common among its k nearest neighbours (k is a positive integer, typically small).

If k = 1, then the object is simply assigned to the class of that single nearest neighbour.

- Machine Learning Task: Classification

- Predictor:

'sepal length (cm)'	'sepal width (cm)'
'petal length (cm)'	'petal width (cm)

- Response Variable: EITHER OF THE THREE FLOWER SPECIES:

'setosa'	'versicolor'	'virginica'

- Model Parameters: Mean, median
- Hyper Parameters:
    - Value of K
    - Distance Metric (like 'euclidean','minkowski','manhattan’,etc.)

Pseudocode:

```
data = load_iris_data()

train, test = split(data)

knn = KNN(k = 1 and metric = ‘Euclidean’)

knn.fit(train)

knn.predict(new_data)
```

Pseudo Code for KNN

- Load the data
- Initialize the value of k
- Calculate the distance between test data and each row of training data.
- Sort the calculated distances in ascending order based on distance values
- Get top k rows from the sorted array
- Get the most frequent class of these rows as predicted class

Program:

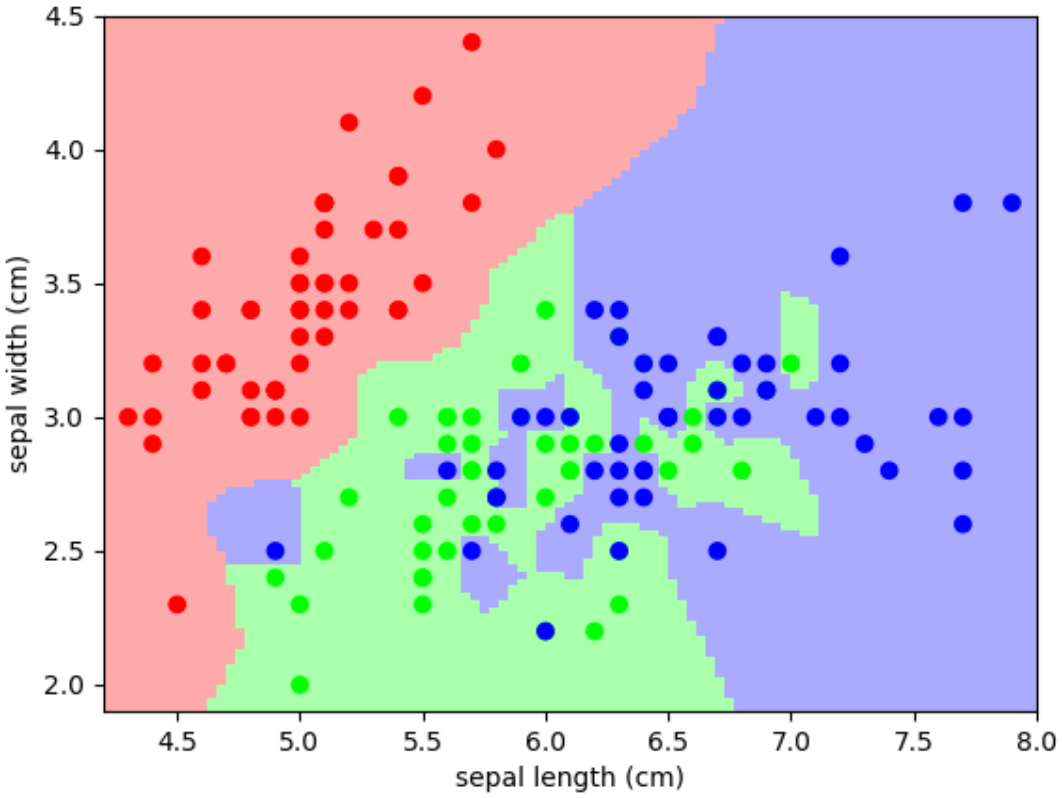
```
import numpy as np
#Loading the dataset
from sklearn.datasets import load_iris
from sklearn.model_selection import
train_test_split
from sklearn.neighbors import
KNeighborsClassifier

data = load_iris()

X_train, X_test, Y_train, Y_test =
train_test_split(
        data.data, data.target,

stratify=data.target,test_size=0.25)

# Hyper Parameters
metr='euclidean'
#other
options:{'euclidean','minkowski','manhatt
an'},etc.
K=1

#Create KNN Classifiers
knn =
KNeighborsClassifier(n_neighbors=K,metric
=metr);
```

```
#Train the classifier model using the
training set
knn.fit(X_train, Y_train)

#Predict the response for test data
x_new(dimensions of the new iris flower
to be classified)
x_new = np.array([[5, 2.9, 1, .2]])
y_pred = knn.predict(x_new)

print(f"\nTest Data: {x_new}
Prediction: {y_pred}")
print(f"Predicted target name:
{data['target_names'][y_pred]}")

# MODEL EVALUATION: Predict the
responses for test dataset and
calculate model accuracy
# Model Accuracy: how often is the
classifier correct?
Y_pred = knn.predict(X_test)
print(f"\nTest set predictions:\n
{Y_pred}")
print(f"\nK={K}, Distance metric =
{metr} \n%Accuracy on Test set =
{np.mean(Y_pred == Y_test)*100:.2f}")
```

Output:

```
Test Data: [[5.  2.9 1.  0.2]] Prediction: [0]

Predicted target name: ['setosa']

Test set predictions:

 [1 2 1 0 1 1 0 1 2 1 1 2 0 0 0 1 2 0 1 2 2 2 2 1 0
1 2 0 0 0 0 0 2 1 0 1 2

 1]

K=1, Distance metric = euclidean

%Accuracy on Test set = 94.74
```

Graphs:



Conclusion:

We have Successfully Predicted the category of the iris flower with given sepal length, sepal width, petal length, petal width by KNN for Classification.

Experiment No. 4

Aim:

Program for Regression using KNN algorithm implemented from scratch.

Problem Statement:

Given a set of features that describe a house in Boston, design an optimal KNN regression model that can predict the house price for any given house.

Dataset Description:

The Boston Housing Dataset is a derived from information collected by the U.S. Census Service concerning housing in the area of Boston MA. There are 506 observations with 13 features (independent variables) like:

the number of rooms(rm),
crime_rate(crim),
air pollution variable(nox),
cost of public services in each community(tax),
pupil-teacher ratio(ptratio),etc.

The dependent/target variable is house price that is given in thousand dollars.

Problem Analysis:

Upon critical Analysis of the given Problem, it can be seen that there's a Relationship between the price of the house and the different features given in the dataset description.

This type of problem can easily be dealt with by using K Nearest Neighbours for Regression.

In k-NN regression, the output is the property value for the object. This value is the average of the values of k nearest neighbours.

KNN regression is a non-parametric method that, in an intuitive manner, approximates the association between independent variables and the continuous outcome by averaging the observations in the same neighbourhood.

Machine Learning Task:

Regression

Predictor:

- CRIM - per capita crime rate by town
- ZN - proportion of residential land zoned for lots over 25,000 sq.ft.
- INDUS - proportion of non-retail business acres per town.
- CHAS - Charles River dummy variable (1 if tract bounds river; 0 otherwise)

- NOX - nitric oxides concentration (parts per 10 million)
- RM - average number of rooms per dwelling
- AGE - proportion of owner-occupied units built prior to 1940
- DIS - weighted distances to five Boston employment centres
- RAD - index of accessibility to radial highways
- TAX - full-value property-tax rate per $10,000
- PTRATIO - pupil-teacher ratio by town
- B - 1000(Bk - 0.63)^2 where Bk is the proportion of blacks by town
- LSTAT - % lower status of the population
- MEDV - Median value of owner-occupied homes in $1000's

Response Variable:

The *price* of house in Thousand Dollars.

Model Parameters:

Final cluster centers

Hyper Parameters:

- Value of K
- Distance Metric (like 'euclidean','minkowski','manhattan’,etc.)
- Seed/initial values for cluster centers

Pseudocode:

```
fun knn_reg():return average of first k values
of sorted target vector

fun edist(): return sqrt(sum((p1-p2)**2))

fun elbow(): return elbow point of the graph

dataset = load_boston()

split data into training and testing

for k from 1 to sqrt(no. of rows in dataset):

    calculate error for each value of k
```

Pseudo Code for KNN

- Load the data
- Initialize the value of k
- Calculate the distance between test data and each row of training

data.

- Sort the calculated distances in ascending order based on distance

values

- Get top k rows from the sorted array
- Get the mean of these rows as predicted value.

Program:

```
from sklearn.datasets import
load_boston
from sklearn.model_selection import
train_test_split as SPLIT
from numpy import sqrt, sum,
argsort, arange, array, average
from sklearn.metrics import
mean_squared_error as MSE
import matplotlib.pyplot as plt

boston = load_boston() # Loading the
Boston dataset
x = boston.data[:,:]
y = boston.target
print(x.shape, y.shape)
# fixed random state ensures same
train and test sets across various
runs
# Size : 30% test && 70% train
xtrain, xtest, ytrain, ytest =
SPLIT(x, y, test_size=0.3,
random_state=102)

##function to find Euclidean
distance
def edist(v1,v2):
    return sqrt(sum((v1-v2)**2))
```

```
def explore():
    ## Exploring the dataset
characteristics and having glimpse of
data
    # printing the sizes of training
and testing data sets
    print(xtrain.shape,ytrain.shape)
    print(xtest.shape,ytest.shape)
    # Print the information contained
within the dataset
    print("\nKeys of iris_dataset:
\n{}".format(boston.keys()))
    print(boston['DESCR'][:500] +
"\n...")
    #Print the feature names
    print("\nFeature names:
\n{}".format(boston['feature_names']))
    #Printing the  Few Rows
    print("\nFirst five rows of
data:\n{}".format(boston['data'][:5]))
    #Print the class values few
datapoints

print("\nTarget:\n{}".format(boston['t
arget'][:5]))
    #Print the dimensions of data
    print("\nShape of data:
{}".format(boston['data'].shape))
```

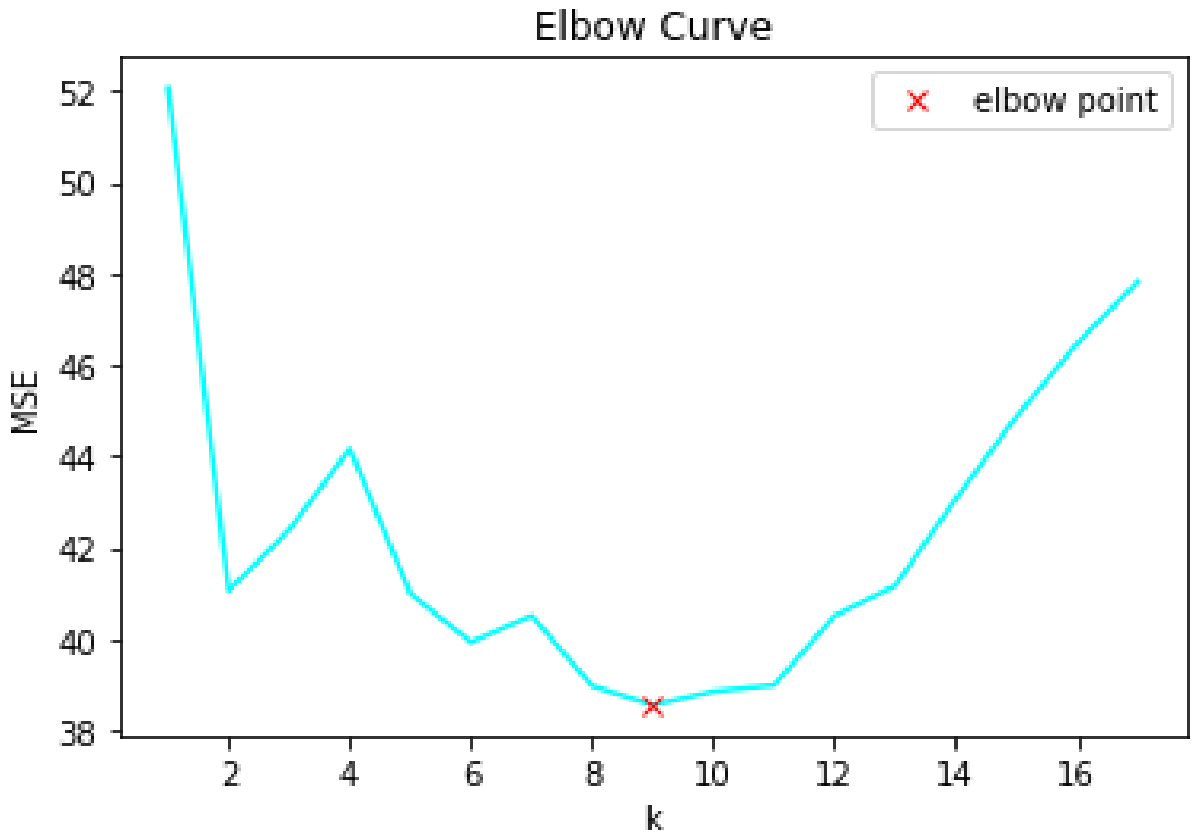
```
##Function to find mean squared error for the
entire test dataset
def knn_mse(k):
    preds = array([knn_reg(xtest[i] , k) for
i in range(xtest.shape[0])])
    return MSE(ytest , preds)


##function to predict values using knn for
given test data tx
def knn_reg(tx , k):
    #Find distances between new data and all
the training data
    distances = array([edist(xtrain[i], tx)
for i in range(xtrain.shape[0])])

    #sort the distances in ascending order
    inds = argsort(distances)
    distances = distances[inds]
    tr_y_sorted = ytrain[inds] #sorted values
of target variable

    #predicted value is the average of first
k values of target vector
    return average(tr_y_sorted[:k])


## function to find elbow plot
def find_elbow():
    inds=argsort(mse_val)
    for i in inds:
        diff1=mse_val[i-1]-mse_val[i]
        diff2=mse_val[i]-mse_val[i+1]
        if(diff1>0 and diff2<0):
            break
    eb1=i+1
    return(eb1)
```

```
if __name__ == '__main__':
    explore()

    ##Finding MSEs for different values of k
    maxk=int(sqrt(xtrain.shape[0])) #maximum
value of k
    mse_val = [] #to store rmse values for
different k
    for k in range(1,maxk):
        error= knn_mse(k)
        mse_val.append(error) #store rmse values
        print(f'MSE value for k = {k} is:
{error:.3f}')

    ##plotting the elbow curve
    k=arange(1,maxk)
    plt.xlabel("k")
    plt.ylabel("MSE")
    plt.title("Elbow Curve")
    plt.plot(k,mse_val, 'cyan')
    ##finding the k for the elbow point
    ke=find_elbow()
    print(f"Best Value of k using elbow curve is
{ke}")
    plt.plot(ke,mse_val[ke-1],'rx', label =
'elbow point')
    plt.legend()

    # Now model is ready to predict the cost for
new house with
    # given features in xnew vector and ke as k
    xnew=array([2.7310e-02, 0.0000e+00,
7.0700e+00, 0.0000e+00,
                    4.6900e-01, 6.4210e+00,
7.8900e+01, 4.9671e+00,
                    2.0000e+00, 2.4200e+02,
1.7800e+01, 3.9690e+02 ,9.1400e+00])
    hcost = knn_reg(xnew , ke)
    print(f"Predicted price of the given house
is {hcost:.2f} thousand dollars")
```

Output:

```
MSE value for k = 1 is: 52.070
MSE value for k = 2 is: 41.057
MSE value for k = 3 is: 42.389
MSE value for k = 4 is: 44.167
MSE value for k = 5 is: 40.997
MSE value for k = 6 is: 39.929
MSE value for k = 7 is: 40.512
MSE value for k = 8 is: 38.973
MSE value for k = 9 is: 38.557
MSE value for k = 10 is: 38.854
MSE value for k = 11 is: 38.988
MSE value for k = 12 is: 40.509
MSE value for k = 13 is: 41.166
MSE value for k = 14 is: 43.070
MSE value for k = 15 is: 44.868
MSE value for k = 16 is: 46.468
MSE value for k = 17 is: 47.821
Best Value of k using elbow curve is 9
Predicted price of the given house is 24.00 thousand dollars
```

Graphs:



Conclusion:

We have Successfully Predicted the price of the house with given features by KNN for Regression.

Experiment No. 5

Aim:

KNN for regression implemented using KNN algorithm from Scikit learn library.

Problem Statement:

Given a set of features that describe a house in Boston, design an optimal KNN regression model that can predict the house price for any given house.

Dataset Description:

The Boston Housing Dataset is a derived from information collected by the U.S. Census Service concerning housing in the area of Boston MA. There are 506 observations with 13 features (independent variables) like:

the number of rooms(rm),
crime_rate(crim),
air pollution variable(nox),
cost of public services in each community(tax),
pupil-teacher ratio(ptratio),etc.

The dependent/target variable is house price that is given in thousand dollars.

Problem Analysis:

Upon critical Analysis of the given Problem, it can be seen that there's a Relationship between the price of the house and the different features given in the dataset description.

This type of problem can easily be dealt with by using K Nearest Neighbours for Regression.

In k-NN regression, the output is the property value for the object. This value is the average of the values of k nearest neighbours.

KNN regression is a non-parametric method that, in an intuitive manner, approximates the association between independent variables and the continuous outcome by averaging the observations in the same neighbourhood.

Machine Learning Task:

Regression

Predictor:

- CRIM - per capita crime rate by town
- ZN - proportion of residential land zoned for lots over 25,000 sq.ft.
- INDUS - proportion of non-retail business acres per town.
- CHAS - Charles River dummy variable (1 if tract bounds river; 0 otherwise)

- NOX - nitric oxides concentration (parts per 10 million)
- RM - average number of rooms per dwelling
- AGE - proportion of owner-occupied units built prior to 1940
- DIS - weighted distances to five Boston employment centres
- RAD - index of accessibility to radial highways
- TAX - full-value property-tax rate per $10,000
- PTRATIO - pupil-teacher ratio by town
- B - 1000(Bk - 0.63)^2 where Bk is the proportion of blacks by town
- LSTAT - % lower status of the population
- MEDV - Median value of owner-occupied homes in $1000's

Response Variable:

The *price* of house in Thousand Dollars.

Model Parameters:

Final cluster centers

Hyper Parameters:

- Value of K
- Distance Metric (like 'euclidean','minkowski','manhattan’,etc.)
- Seed/initial values for cluster centers

Pseudocode:

```
fun elbow(): return elbow point of the graph
dataset = load_boston()
split data into training and testing
for k from 1 to sqrt(no. of rows in dataset):
    model = KNeighborsRegressor(k)
    model.fit(data_train)
    pred = model.predict(data_test)
```

Pseudo Code for KNN

- Load the data
- Initialize the value of k
- Calculate the distance between test data and each row of training

data.

- Sort the calculated distances in ascending order based on distance

values

- Get top k rows from the sorted array
- Get the mean of these rows as predicted value.

Program:

```
from sklearn.datasets import load_boston
from sklearn import neighbors
from sklearn.model_selection import
train_test_split
import numpy as np
from sklearn.metrics import
mean_squared_error
import matplotlib.pyplot as plt
import math

#Loading the Boston dataset
boston = load_boston()
x=boston.data[:,:]
y=boston.target
print(x.shape,y.shape)

tsize=0.30 #30% of total data is used for
testing and 70% used for training

##splitting the dataset into training and
testing sets,
#(parameter random state is fixed at some
integer, to ensure the same train
#and test sets across various runs)
xtrain,xtest,ytrain,ytest=train_test_spli
t(x,y,test_size=tsize,random_state=102)
```

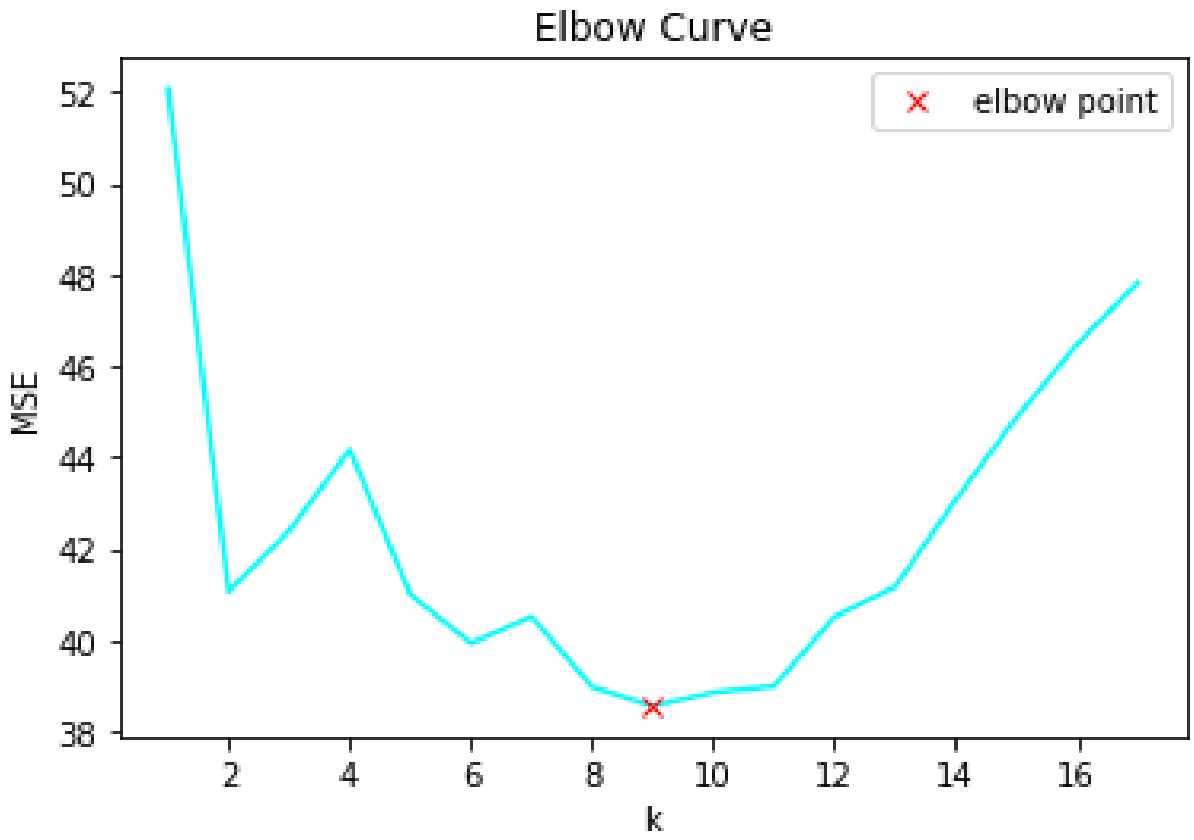
```
###---Code using Scikit learn library for KNN
regression
##Finding MSEs for different values of k
maxk=int(math.sqrt(xtrain.shape[0])) #maximum
value of k
mse_val = [] #to store rmse values for
different k
for K in range(1,maxk):
    model =
neighbors.KNeighborsRegressor(n_neighbors = K)

    model.fit(xtrain, ytrain)  #fit the model
    pred=model.predict(xtest) #make prediction
on test set
    error = mean_squared_error(ytest,pred)
#calculate rmse
    mse_val.append(error) #store rmse values
    print('MSE value for k= ' , K , 'is:',
error)

##function to find elbow point
def find_elbow():
    inds=np.argsort(mse_val)
    for i in inds:
        diff1=mse_val[i-1]-mse_val[i]
        diff2=mse_val[i]-mse_val[i+1]
        if(diff1>0 and diff2<0):
            break
    eb1=i+1
    return(eb1)
```

```
##plotting the elbow curve
k=np.arange(1,maxk)
xl="k"
yl="MSE"
plt.xlabel(xl)
plt.ylabel(yl)
plt.title("Elbow Curve")
plt.plot(k,mse_val)

##finding the k for the elbow point
ke=find_elbow()
print("Best Value of k using elbow curve is
",ke)
plt.plot(ke,mse_val[ke-1],'rx')
plt.annotate("  elbow point", (ke,mse_val[ke-
1]))
## Now with the best k(i.e. ke) predict the
cost for the new house with given features
xnew=np.array([2.7310e-02, 0.0000e+00,
7.0700e+00, 0.0000e+00, 4.6900e-01,
                6.4210e+00, 7.8900e+01,
4.9671e+00, 2.0000e+00, 2.4200e+02,
                    1.7800e+01, 3.9690e+02
,9.1400e+00])
model =
neighbors.KNeighborsRegressor(n_neighbors =
ke)
model.fit(xtrain, ytrain)  #fit the model
xnew=xnew.reshape(1,-1)
hcost=model.predict(xnew)
print("Predicted price of the given house is
{:.2f}".format(hcost[0]),"thousand dollars")
```

Output:

```
MSE value for k = 1 is: 52.070
MSE value for k = 2 is: 41.057
MSE value for k = 3 is: 42.389
MSE value for k = 4 is: 44.167
MSE value for k = 5 is: 40.997
MSE value for k = 6 is: 39.929
MSE value for k = 7 is: 40.512
MSE value for k = 8 is: 38.973
MSE value for k = 9 is: 38.557
MSE value for k = 10 is: 38.854
MSE value for k = 11 is: 38.988
MSE value for k = 12 is: 40.509
MSE value for k = 13 is: 41.166
MSE value for k = 14 is: 43.070
MSE value for k = 15 is: 44.868
MSE value for k = 16 is: 46.468
MSE value for k = 17 is: 47.821
Best Value of k using elbow curve is 9
Predicted price of the given house is 24.00 thousand dollars
```

Graphs:



Conclusion:

We have Successfully Predicted the price of the house with given features by KNN for Regression.

Experiment No. 6

Aim:

Naïve Bayes Classifier using Scikit learn library.

Problem Statement:

Wine classification is a very famous multi-class classification problem.

Build a ML model to classify the type of wine.

Dataset Description:

The Wine dataset is the result of a chemical analysis of wines grown in the same region in Italy but derived from three different cultivars. The Dataset comprises of 13 features: alcohol, malic_acid, ash, alcalinity_of_ash, magnesium, flavanoids, hue, color_intensity,

od280/od315_of_diluted_wines,nonflavanoid_phenols, total_phenols, proanthocyanins, proline and type of wine cultivar.

This data has three type of wine Class_0, Class_1, and Class_2.

Class	3
Samples per class	[59,71,48]
Samples total	178
Dimensionality	13
Features	real, positive

Problem Analysis:

Upon critical Analysis of the given Problem, it can be seen that there’s a Relationship between the class of wine and the features of the wine.

This type of problem can easily be dealt with by using Gaussian Naïve Bayes Classifier

Naïve Bayes classifiers are a family of simple "probabilistic classifiers" based on applying Bayes' theorem with

strong(naïve)independence assumptions between the features. They are among the simplest Bayesian network models, but coupled with kernel density estimation, they can achieve higher accuracy levels.

In Gaussian naïve Bayes We deal with continuous data, a typical assumption is that the continuous values associated with each class are distributed according to a normal (or Gaussian) distribution.

For example, suppose the training data contains a continuous attribute x. We first segment the data by the class, and then compute the mean and variance of x in each class.

Machine Learning Task:

Classification

Predictor:

- Alcohol
- Malic acid
- Ash
- Alcalinity of ash
- Magnesium
- Total phenols
- Flavanoids
- Nonflavanoid phenols
- Proanthocyanins
- Color intensity
- Hue
- OD280/OD315 of diluted wines
- Proline

Response Variable: EITHER OF THE THREE Classes:

Class_0, Class_1, Class_2

Model Parameters:

Mean, Standard Deviation, Probability Density Equation

Hyper Parameters:

None

Pseudocode:

```
Input:
   Training dataset T


Step:
   Read training dataset T
   Calculate mean and S.D. of
   predictor variables of each
   class
   For i from 1 to len(dataset):
      Calculate probability fi using
      gauss density equation in each
      class.
   Calculate likelihood for each
   class
   Get the greatest likelihood


Output:
   A class of testing dataset
```

Program:

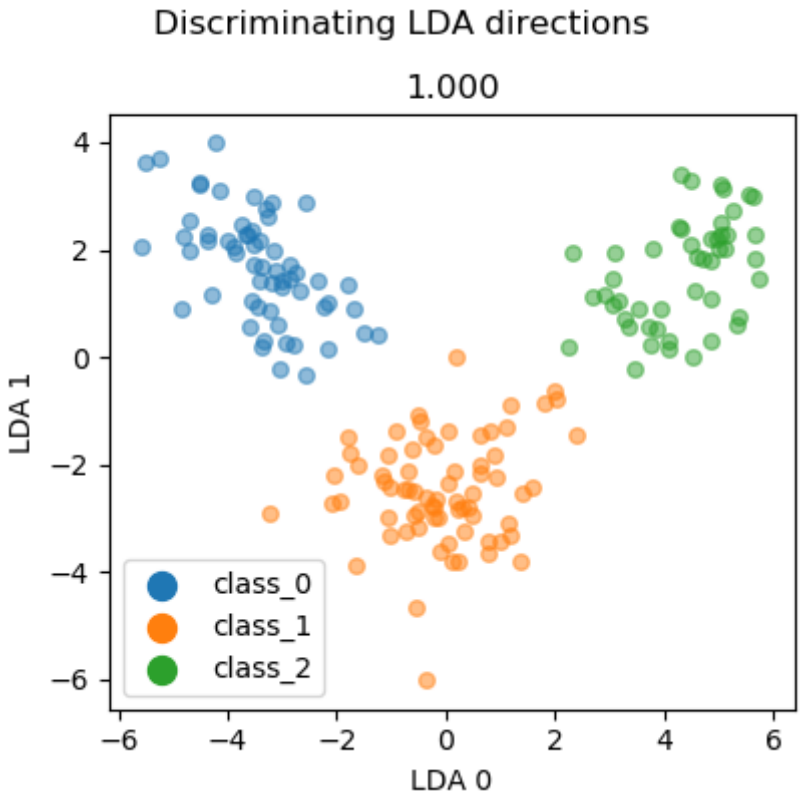
```
# Import scikit-learn dataset library
from sklearn.datasets import load_wine
wine = load_wine()

from sklearn.model_selection import
train_test_split as SPLIT
X_train, X_test, y_train, y_test =
SPLIT(wine.data, wine.target,
test_size=0.3,random_state=1)
# 70% training and 30% test

def explore():
    # print the names of the 13 features
    print ("Features: ", wine.feature_names)

    # print the label type of wine(class_0,
class_1, class_2)
    print ("Labels: ", wine.target_names)

    # print data(feature)shape
    wine.data.shape
    # print the wine data features (top 5
records)
    print (wine.data[0:5])
    print (wine.target)
    #Count number of observation in each class
    for i in set(wine.target):
        print('Class', i, ' -> ',
list(wine.target).count(i))
```

```
explore()

# Import Gaussian Naive Bayes model
from sklearn.naive_bayes import
GaussianNB as GNB

# Train the model using the training sets
# and Predict the response for test
dataset
y_pred = GNB().fit(X_train,
y_train).predict(X_test)

# Import scikit-learn metrics module for
accuracy calculation
from sklearn import metrics

# Model Accuracy, how often is the
classifier correct?
print(f"Accuracy:
{100*metrics.accuracy_score(y_test,
y_pred):.3f}%")

import matplotlib.pyplot as plt
from dabl import plot
from dabl.utils import data_df_from_bunch

plot(data_df_from_bunch(wine), 'target')
plt.show()
```

Output:

```
Class 0  ->  59
Class 1  ->  71
Class 2  ->  48
Accuracy: 98.148%
```

Graphs:



Conclusion:

We have Successfully Predicted the class of the wine with given features width by Naïve Bayes Classifier.

Experiment No. 7

Aim:

C-means clustering using Scikit-learn

Problem Statement:

BB supermarket mall maintains the customer data using membership cards. They have some basic data about the customers like Customer ID, name, age, gender, number of visits to the supermarket in a year, and the average spending per visit.

They want to understand their customers so that the sense can be given to the marketing team so that they can plan the strategy accordingly. As a data scientist your task is to identify the customer groups and interpret them appropriately after analysing the clusters.

This can be later given to the marketing team to increase the customer base and the profit of the supermarket.

Dataset Description:

The BB Supermarket dataset consists of customers like Customer ID, name, age, gender, number of visits to the supermarket in a year(Nvisits), and the average spending per visit(purchaseAmount).

Out of these Nvisits and purchaseAmount can be used for categorization of customers, rest do not play a very important role.

Class	4
Samples Total	28
Dimensionality	2
Features	ID, name, age, gender, Nvisits, purchaseAmount

Problem Analysis:

Upon critical Analysis of the given Problem, it can be seen that number of visits and purchase amount play an important role in determining the category of customer.

This type of problem can easily be dealt with by using K Means Clustering.

K-Means Clustering is an unsupervised learning algorithm that is used to solve the clustering problems in machine learning or data science.

k-means clustering is a method of vector quantization, originally from signal processing, that aims to partition n observations into k clusters in which each observation belongs to the cluster with the nearest mean (cluster centers or cluster centroid), serving as a prototype of the cluster.

Machine Learning Task:

Clustering

Predictor:

- Nvisits
- Purchase Amount

Response Variable:

Class of Customers:
['Moderate', 'Basic', 'Prime',
'Window Shopper']

Model Parameters:

Cluster Centroids

Hyper Parameters:

- Number of Clusters
- Number of iterations

Pseudocode:

```
data <- customer behaviour

kmeans = KMeans().fit(data)

labels = kmeans.predict(X)

plt.plot(cluster_centers)

plt.legend(cluster_interpretation)
```

Pseudo Code for K-Means Clustering:

1. Choose the number of clusters(K) and obtain the data points
2. Place the centroids c_1, c_2, ... c_k randomly
3. Repeat steps 4 and 5 until convergence or until the end of a fixed number of iterations
4. for each data point x_i: - find the nearest centroid(c_1, c_2 .. c_k) - assign the point to that cluster
5. for each cluster j = 1..k - new centroid = mean of all points assigned to that cluster
6. End

Program:

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

# X contains training data samples from a
synthetic dataset
#x = [Nvisits,purchaseAmount]
X =
[[10,20],[9,0],[6,5],[7,25],[11,0],[12,10],[8,11]
,

[4,150],[5,120],[7,100],[5,200],[6,120],[4,220],[
6,300],

[7,404],[5,388],[6,225],[8,350],[8,236],[4,167],[
9,400],

[9,699],[5,553],[8,450],[9,817],[10,1010],[11,825
],[11,700]]

# Number of clusters k (you may find best k by
using elbow point)
k = 4
model = KMeans(n_clusters=k,random_state=0)

# Fitting the input data
kmeans = model.fit(X) #model is trained and
stored in kmeans

# Centroid values
Xarr = np.array(X)
centroids = np.around(kmeans.cluster_centers_,
decimals=2)
print(f"Cluster Centers are: {centroids}")
```

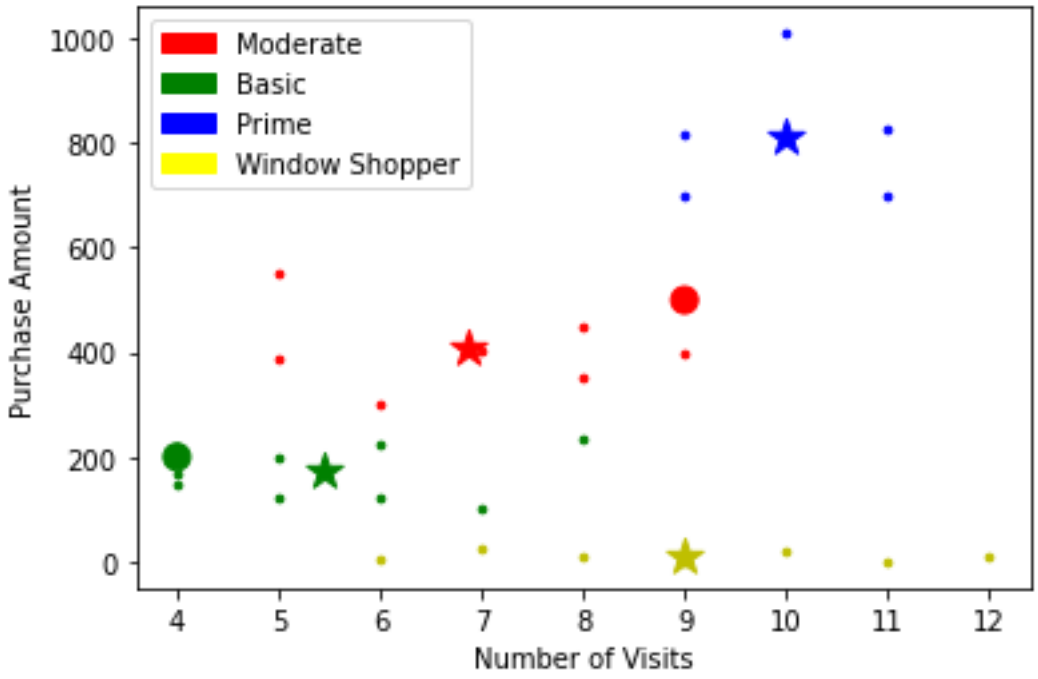
```
# Predicting the cluster labels
labels = kmeans.predict(X)
print(f"Cluster Labels are: {labels}")

colors = ['r', 'g', 'b', 'y', 'c', 'm']
fig, ax = plt.subplots()
plt.xlabel('Number of Visits')
plt.ylabel('Purchase Amount')

for i in range(k):
    #plot all points labelled i during
each iteration with different color
    points = np.array([X[j] for j in
range(len(X)) if labels[j] == i])
    ax.scatter(points[:, 0], points[:,
1], s=7, c=colors[i])
    #plot centroid of that cluster
    ax.scatter(centroids[i, 0],
centroids[i, 1], marker='*', s=200,
c=colors[i])

# predicting class of input pattern
[4,200]
xnew = np.array([[4,200],[9,500]])
newlabels = kmeans.predict(xnew)
```

```
for i in range(len(xnew)):
    print(f"Pattern {xnew[i]} belongs
to cluster #{newlabels[i]}")
ax.scatter(xnew[:, 0], xnew[:, 1],
s=100, c=[colors[i] for i in
newlabels])

# Cluster Analysis:Give One possible
interpretation of the four clusters in
your own words
import matplotlib.patches as mpatches
red_patch = mpatches.Patch(color='red',
label='Moderate')
green_patch =
mpatches.Patch(color='green',
label='Basic')
blue_patch =
mpatches.Patch(color='blue',
label='Prime')
yellow_patch =
mpatches.Patch(color='yellow',
label='Window Shopper')

plt.legend(handles=[red_patch,
green_patch, blue_patch, yellow_patch])
```

Output:

```
Cluster Centres are:
[[6.86 406.43] [5.44 170.89] [10. 810.2] [9. 10.14]]
Cluster Labels are: [3 3 3 3 3 3 3 1 1 1 1 1 1 0 0 0 1 0
1 1 0 2 0 0 2 2 2 2]
Pattern [4 200] belongs to cluster #1
Pattern [9 500] belongs to cluster #0
```

Graphs:



Conclusion:

We have Successfully Clustered the given customer in different categories by k-means clustering.

Experiment No. 8

Aim:

Decision Trees using Scikit-learn

Problem Statement:

Predicting whether a person is infected from COVID-19 or not.

Dataset Description:

The dataset reports given a person has/has not one of the three symptoms: 'Fever', 'Cough', 'Breathing Issue' and if he is suffering from COVID-19 or not.

Class	2
Samples Total	13
Dimensionality	3
Features	‘Fever’, ‘Cough’, ‘Breathing Issue’

Problem Analysis:

Upon critical Analysis of the given Problem, it can be seen that there's a Relationship between the person being COVID-19 infected and if he has symptoms like Fever, Cough, or Breathing Issue as given in the dataset description.

This type of problem can easily be dealt with by using Decision Trees for Classification.

A decision tree is a decision support tool that uses a tree-like model of decisions and their possible consequences, including chance event outcomes, resource costs, and utility.

It is one way to display an algorithm that only contains conditional control statements.

Decision tree builds classification or regression models in the form of a tree structure.

It breaks down a dataset into smaller and smaller subsets while at the same time an associated decision tree is incrementally developed.

The final result is a tree with decision nodes and leaf nodes.

A decision node has two or more branches. Leaf node represents a classification or decision. The topmost decision node in a tree which corresponds to the best predictor called root node.

Decision trees can handle both categorical and numerical data.

The core algorithm for building decision trees called ID3 which uses Entropy and Information Gain to construct a decision tree.

A decision tree can easily be transformed to a set of rules by mapping from the root node to the leaf nodes one by one.

Machine Learning Task:

Classification

Predictor:

If a person has or has not symptoms like: *Fever, Cough, Breathing Issue*

Response Variable:

Whether a person is *infected* by COVID-19 or not.

Model Parameters:

Nodes and Edges

Hyper Parameters:

- Criterion: {“gini”, “entropy”}
- Max_depth: maximum depth of tree
- Min_impurity_split: Threshold for early stopping in tree growth. A node will split if its impurity is above the threshold, otherwise it is a leaf.

Pseudocode:

```
Data: X<-Input, Y<-Target

model = DecisionTreeClassifier()

clf = model.fit(X,Y)

prediction = clf.predict(sample)

print(prediction)

plot_decision_tree()
```

Algorithm for Decision Tree:

1. Place the best attribute of the dataset at the root of the tree.
2. Split the training set into subsets. Subsets should be made in such a way that each subset contains data with the same value for an attribute.
3. Repeat step 1 and step 2 on each subset until you find leaf nodes in all the branches of the tree.

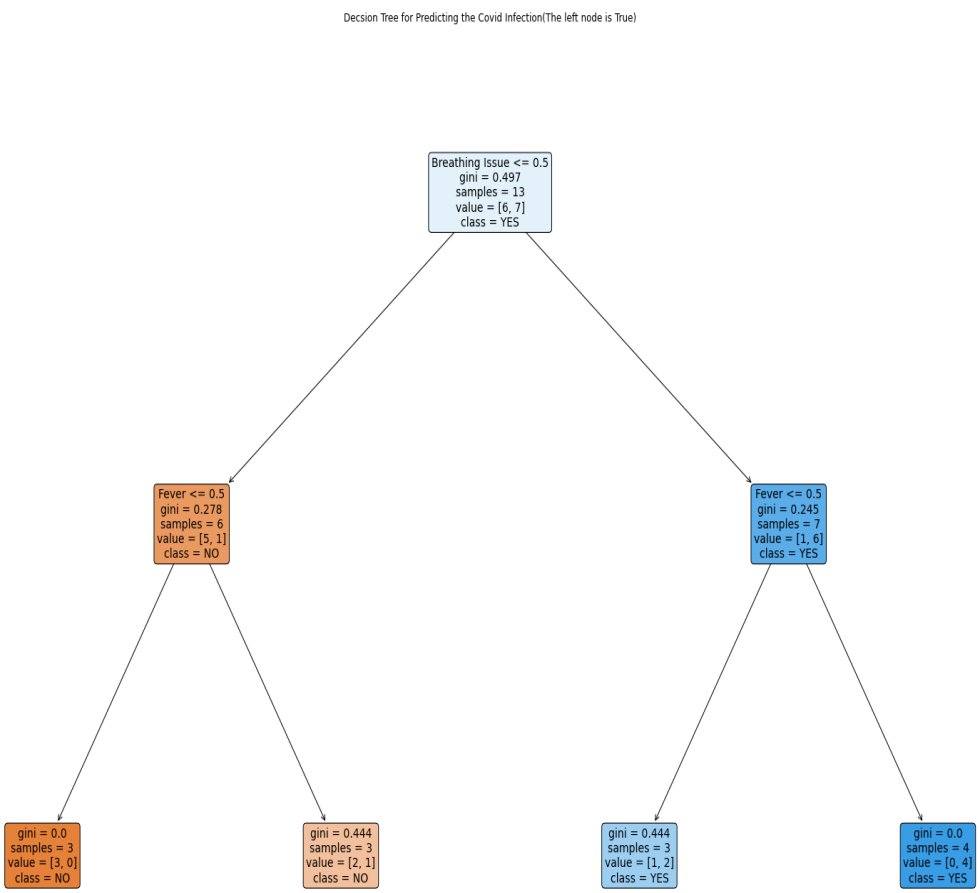
Program:

```
feature_names = ['Fever', 'Cough', 'Breathing
Issue']
X = [ [0,0,0], [1,1,1], [1,1,0],
      [1,0,1], [1,1,1], [0,1,0],
      [1,0,1], [0,1,1], [1,1,0],
      [0,1,0],[0,1,1],[0,1,1],[1,1,0] ]
# Labels for the training data
Y = ['NO', 'YES', 'NO', 'YES',
     'YES', 'NO', 'YES', 'YES',
     'YES', 'NO', 'YES', 'NO', 'NO']
# Finding Unique Labels
labels = list(set(Y))
# Create a decision tree classier model
from sklearn.tree import DecisionTreeClassifier
as DTClf
clf = DTClf(max_depth =3).fit(X,Y)

# Now predict and print the class for the unknown
patient(example)
pred = clf.predict([[1,1,0]])
print(f"Person infected? {pred[0]}")
# plotting tree for COVID-19 classication
import matplotlib.pyplot as plt
# plt the figure,
plt.figure(figsize=(30,20))
from sklearn.tree import plot_tree as PT
PT(clf, class_names=labels,
   feature_names=feature_names,
   rounded = True,filled = True,fontsize=14)
plt.title('Decsion Tree for Predicting the Covid'
          + ' Infection(The left node is True)')
plt.show()
```

Output:

```
Person infected? NO
```

Graphs:



Conclusion:

We have Successfully Predicted if the person is infected from COVID-19 or not given features by Decision Trees for Classification.

Experiment No. 9

Aim:

SVM Classification using Scikit learn library.

Problem Statement:

A hobby botanist is interested in distinguishing the species of some iris flowers that she has found. She has collected some measurements associated with each iris: the length and width of the petals and the length and width of the sepals, all measured in centimetres. She also has the measurements of some irises that have been previously identified by an expert botanist as belonging to the species setosa, versicolor, or virginica.

For these measurements, she can be certain of which species each iris belongs to. Let’s assume that these are the only species our hobby botanist will encounter in the wild.

Our goal is to build a machine learning model that can learn from the measurements of these irises whose species is known, so that we can predict the species for a new iris.

Dataset Description:

Keys of iris_dataset:

'feature_names'	'target'	'frame'	'DESCR'
'target_names	'filename'	'data'	

Data Set Characteristics:

Number of Instances: 150 (50 in each of three classes)

Number of Attributes: 4 numeric, predictive attributes and the class

Attribute Information:

- sepal length in cm
- sepal width in cm
- petal length in cm
- petal width in cm
- class:
  - Iris-Setosa
  - Iris-Versicolour
  - Iris-Virginica

Feature Names:

'sepal length (cm)'	'sepal width (cm)'
'petal length (cm)'	'petal width (cm)

Target Names:

'setosa'	'versicolor'	'virginica'

Shape of data: (150, 4)

Problem Analysis:

Upon critical Analysis of the given Problem, it can be seen that there's a Relationship between Lengths and

Widths of Petals and Sepals and the category of iris.

This type of problem can easily be dealt with by using Support Vector Machine for Classification.

Support Vector Machines(SVMs) find a decision hyperplane between different classes of data using support vectors. A margin is a separation of decision line to the closest class points. Our aim is to find max-margin hyperplane. If the hyperspace is non-linear, we choose a kernel with non-linear basis function to map the non-linear space to new higher dimensional linear space, which then may be separated using linear classifier. Kernel could be linear, polynomial, exponential, etc.

There are two important hyper parameters of SVM; C and gamma. For large values of C, the optimization will choose a smaller-margin hyperplane.

With low value of gamma, points far away from plausible hyperplane are considered in calculation for the hyperplane.

- Machine Learning Task: *Classification*
- Predictor:

'sepal length (cm)'	'sepal width (cm)'
'petal length (cm)'	'petal width (cm)

- Response Variable: EITHER OF THE THREE FLOWER SPECIES:

'setosa'	'versicolor'	'virginica'

- Model Parameters:
  Hyperplane, Support Vector, Margin

- Hyper Parameters:
  - C – Regularization Parameter
  - Kernel: {‘linear’, ‘sigmoid’, ‘precomputed’, ‘poly’, ‘rbf’}
  - Gamma: {‘scale’, ‘auto’}

Pseudocode(Holdout):

```
data = load_iris_data()
train, test = split(data)
model = SVC(kernel, gamma, C)
clf = model.fit(train)
pred = clf.predict(test)
y_pred = clf.pred(X_new)
```

Pseudocode(K-Fold):

```
X, y = load_iris_data(X_Y)
model = SVC(kernel, gamma, C)
cross_val = StratifiedKFold()
scores = cross_val_score(model, X,
y, scoring='accuracy', cv=cv)
scores.mean()
```

Program(Holdout):

```
def explore():
    # Print the information contained within the
dataset
    print(f"\nKeys of iris_dataset:
\n{data.keys()}")
    print(data['DESCR'][:500] + "\n...")

    #Print the feature names
    print(f"\nFeature names:
\n{data['feature_names']}")

    #Print the classes
    print(f"\nTarget names:
{data['target_names']}")

    #Printing first Few Rows
    print(f"\nFirst five rows of
data:\n{data['data'][:5]}")

    #Print the class values few datapoints
    print(f"\nTarget:\n{data['target'][:5]}")

    #Print the dimensions of data
    print(f"\nShape of data:
{data['data'].shape}")

import numpy as np
from sklearn.datasets import load_iris
from sklearn.model_selection import
train_test_split as SPLIT
data = load_iris()
explore()
```

```
# Split dataset into training set and test
set using stratification
X_train, X_test, Y_train, Y_test =
SPLIT(data.data, data.target,

stratify=data.target,test_size=0.2)
# 20% data for testing
### The code for building the model and
getting the accuracy ###
from sklearn.svm import SVC

# Create the SVC model with hyperparameters
kernel, C, gamma = "linear", 0.55, 0.0005

model = SVC(kernel=kernel, C=C,
gamma=gamma)

#Train the classifier model using the
training set
clf = model.fit(X_train,Y_train)

# MODEL EVALUATION: Predict the responses
for test dataset and calculate model
accuracy
# Model Accuracy: how often is the
classifier correct?
Y_pred = clf.predict(X_test)
print(f"Test set predictions: {Y_pred}")
print(f"Kernel={kernel},%Accuracy on Test
set = {np.mean(Y_pred == Y_test)*100:.2f}")
```

```
# display the confusion matrix
from sklearn.metrics import confusion_matrix as
CMat
print('Confusion Matrix is:\n',CMat(Y_test,
Y_pred))

# Predict the response for test data x_new
# (dimensions of the new iris flower to be
classified)
x_new = np.array([[5, 2.9, 1, .2]])
y_pred = clf.predict(x_new)

print(f"New Test Data: {x_new}\nPrediction:Class
{y_pred}")
print(f"Predicted target name:
{data['target_names'][y_pred][0]}")
```

Output:

```
Test set predictions: [1 0 2 1 0 0 2 0 2 0 2 2 0 2 1
2 1 1 2 2 0 0 0 2 2 1 1 1 0 2]

Kernel=linear,%Accuracy on Test set = 93.33

Confusion Matrix is:

 [[10  0  0]

 [ 0  8  2]

 [ 0  0 10]]

New Test Data: [[5.  2.9 1.  0.2]]

Prediction:Class [0]

Predicted target name: setosa
```

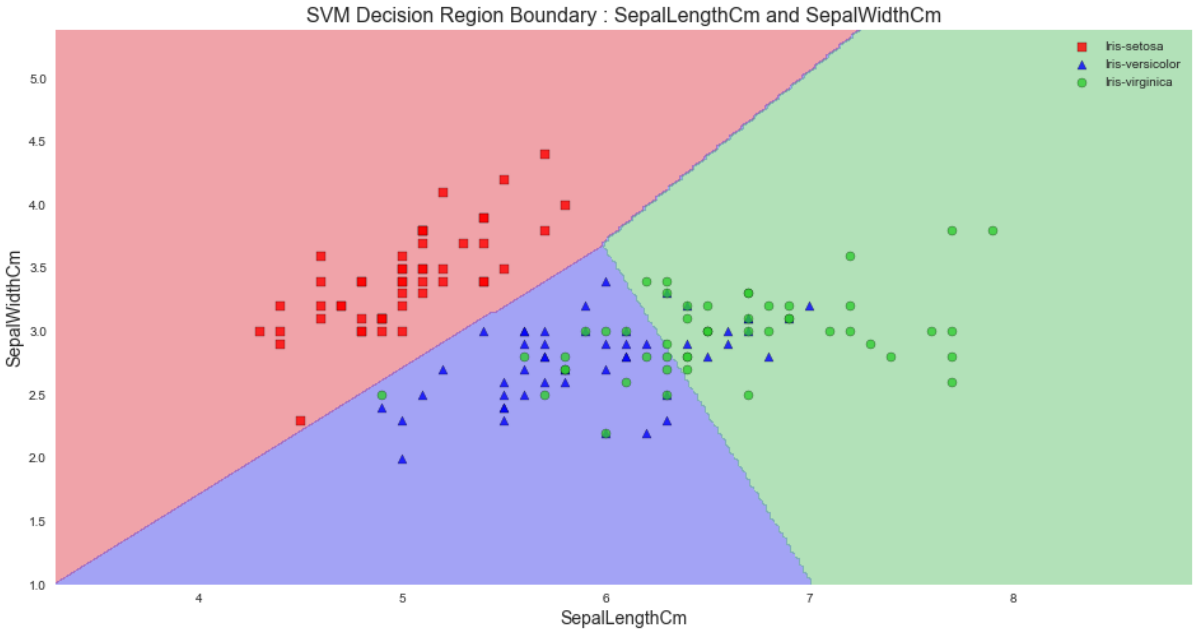
Program(K-Fold):

```
 from sklearn.datasets import load_iris
X, y = load_iris(return_X_y=True)

# Creating Linear SVC Model and Setting
hyperparameters
kernel, C, gamma = "linear", 0.55, 0.001
# gives mean acuracy 0.99 and std deviation
0.01

from sklearn.svm import SVC
model = SVC(kernel=kernel, C=C, gamma=gamma)
from sklearn.model_selection import
StratifiedKFold, cross_val_score
# Cross-Validation procedure
k = 5
cv = StratifiedKFold(n_splits=k ,shuffle=True)
# calling cross validation function
# evaluating score on each train-test fold
scores = cross_val_score(model, X, y,
scoring='accuracy', cv=cv)
print(scores)
print(f"Mean accuracy with {kernel} kernel is
{scores.mean()*100:.2f}%"
      + f" with a standard deviation of
{scores.std():.2f}")
```

Output:

```
[0.96666667  1.  0.96666667 1.  1.]

Mean accuracy with linear kernel is 98.67% with a
standard deviation of 0.02
```

Graphs:



Conclusion:

We have Successfully Predicted the category of the iris flower with given sepal length, sepal width, petal length, petal width by SVM for Classification.

Experiment No. 10

Aim: Implementing a Neural Network based estimation using Scikit learn.

Problem Statement:

Wine classification is a very famous multi-class classification problem.

Build a ML model to classify the type of wine.

Dataset Description:

The Wine dataset is the result of a chemical analysis of wines grown in the same region in Italy but derived from three different cultivars. The Dataset comprises of 13 features: alcohol, malic_acid, ash, alcalinity_of_ash, magnesium, flavanoids, hue, color_intensity, od280/od315_of_diluted_wines,nonflavanoid_phenols, total_phenols, proanthocyanins, proline and type of wine cultivar.

This data has three type of wine Class_0, Class_1, and Class_2.

Class	3
Samples per class	[59,71,48]
Samples total	178
Dimensionality	13
Features	real, positive

Problem Analysis:

Upon critical Analysis of the given Problem, it can be seen that there's a Relationship between the class of wine and the features of the wine.

This type of problem can easily be dealt with by using Multi-Layer Perceptron Classifier.

Multi-layer perceptron (MLP) is a supplement of feed forward neural network. It consists of three types of layers—the input layer, output layer and hidden layer. The input layer receives the input signal to be processed. The required task such as

prediction and classification is performed by the output layer.

An arbitrary number of hidden layers that are placed in between the input and output layer are the true computational engine of the MLP.

Similar to a feed forward network in a MLP the data flows in the forward direction from input to output layer. The neurons in the MLP are trained with the back propagation learning algorithm.

MLPs are designed to approximate any continuous function and can solve problems which are not linearly separable. The major use cases of MLP are pattern classification, recognition, prediction and approximation.

Machine Learning Task:

Classification

Predictor:

- *Alcohol*
- *Malic acid*
- *Ash*
- *Alcalinity of ash*
- *Magnesium*
- *Total phenols*
- *Flavanoids*
- *Nonflavanoid phenols*
- *Proanthocyanins*
- *Color intensity*
- *Hue*
- *OD280/OD315 of diluted wines*
- *Proline*

Response Variable: EITHER OF THE THREE Classes:

Class_0, Class_1, Class_2

Model Parameters:

Weights (Learned During Training)

Hyper Parameters:

- *Bias Inputs*
- *Number of Layers*
- *Number of Neurons in each Layer*
- *Activation Function*
- *Learning rule/Algorithm*
- *Learning Rate*
- *Epochs*
- *Batch Size*

Pseudocode:

```
Input: Training dataset T

Step:

    Read training dataset T

    Split T into Training and Testing
    Sets

    Scale Data Using Scaler

    mlp = MLPClassifier()

    mlp.fit(train_data)

    pred = mlp.predict(test_data)

    print(classification_report(),
    confusion_matrix(), accuracy())

Output: A class of testing dataset
```

Program:

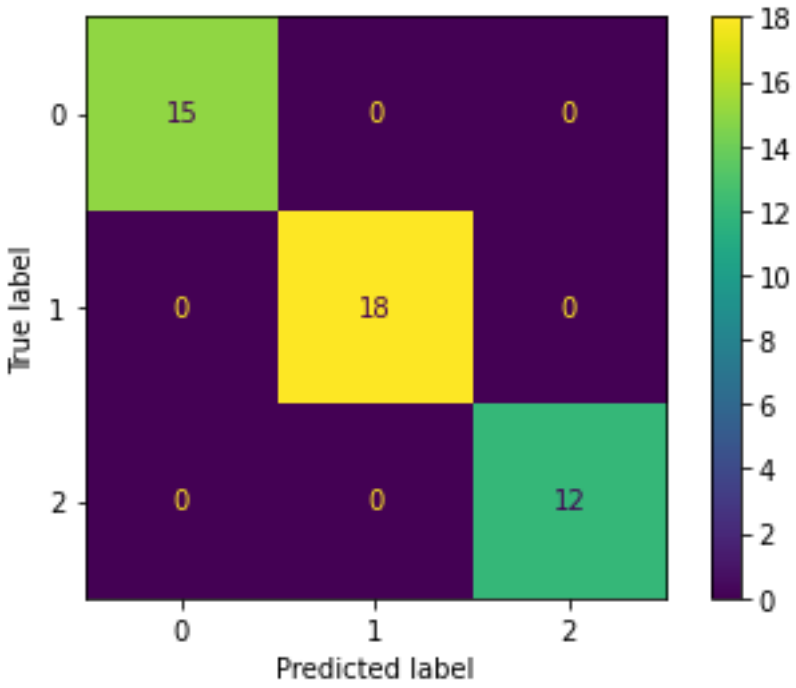
```
from sklearn.datasets import load_wine
wine = load_wine()

# printing class distribution of wine dataset
import numpy as np
print(f'Classes: {np.unique(wine.target)}')
print(f'Class distribution of the dataset:
{np.bincount(wine.target)}')

# from sklearn.cross_validation import
train_test_split(Hold out method)
from sklearn.model_selection import
train_test_split as SPLIT
X_train, X_test, y_train, y_test =
SPLIT(wine.data, wine.target,
test_size=0.25,stratify=wine.target,
random_state=123)

# printing class distribution of test dataset
print(f'Classes: {np.unique(y_test)}')
print(f'Class distribution for test data:
{np.bincount(y_test)}')

# MLP is sensitive to feature scaling, hence
performing scaling
# Options: MinmaxScaler and Standardscaler
# from sklearn.preprocessing import
MinMaxScaler
# scaler = MinMaxScaler()
from sklearn.preprocessing import
StandardScaler as SS
X_train_stdsc = SS().fit_transform(X_train)
X_test_stdsc = SS().fit_transform(X_test)
```

```
# Setting of hyperparameters of the network
from sklearn.neural_network import MLPClassifier as
MLP
mlp =
MLP(hidden_layer_sizes=(10,),learning_rate_init=0.00
1,max_iter=5000)

# Calculating Training Time : more neurons, more
time
from time import time
start = time()
# Train the model using the scaled training sets
mlp.fit(X_train_stdsc, y_train)
end = time()
print(f'Training Time: {(end-start)*1000:.3f}ms')

# Predict the response for test dataset
y_pred = mlp.predict(X_test_stdsc) # scaled

# Import scikit-learn metrics module for evaluating
model performance
from sklearn.metrics import
classification_report,confusion_matrix,accuracy_scor
e

# Model Accuracy, how often is the classifier
correct?
print(f'Accuracy:{accuracy_score(y_test, y_pred)}')

# display the confusion matrix
print('Confusion Matrix
is:\n',confusion_matrix(y_test, y_pred))
print('Classification
Report:\n',classification_report(y_test, y_pred))

from sklearn.metrics import plot_confusion_matrix as
PCM
PCM(mlp, X_test_stdsc, y_test)
```

Output:

```
Classes: [0 1 2]
Class distribution of the dataset: [59 71 48]
Classes: [0 1 2]
Class distribution for test data: [15 18 12]
Training Time: 1060.992ms
Accuracy: 1.0
Confusion Matrix is:
 [[15  0  0]
 [ 0 18  0]
 [ 0  0 12]]
Classification Report:
              precision recall f1-score support

           0    1.0       1.0    1.0       15
           1    1.0       1.0    1.0       18
           2    1.0       1.0    1.0       12
    accuracy                      1.0       45
   macro avg    1.0       1.0    1.0       45
weighted avg    1.0       1.0    1.0       45
```

Graphs:



Conclusion:

We have Successfully Predicted the class of the wine with given features width by Multi-Layer Perceptron Classifier.

Experiment No. 11

Aim: Experiment on classification using transfer learning (Deep learning).

Problem Statement:

Image Classification using GoogleNet Convolutional Neural Network.

Model Description:

GoogleNet is a Convolutional Neural Network that is 144 layers deep.

It is trained on more than 14 million hand-annotated images

We can load a pretrained version of this trained network.

The network trained on ImageNet classifies images into 1000 object categories, such as keyboard, mouse, pencil, and many animals.

The pretrained networks has an image input size of 224x224x3.

GoogleNet has 6.7977 million parameters(weights)

Class	1000
Depth	144 layers
Trained on	14 Million Images
Image input size	224 X 224 X 3
No. of Weights	6.7977 Million

Problem Analysis:

CNNs are regularized versions of multilayer perceptrons. Multilayer perceptrons usually mean fully connected networks, that is, each neuron in one layer is connected to all neurons in the next layer. The "full connectivity" of these networks make them prone to overfitting data. Typical ways of regularization, or preventing overfitting, include: penalizing parameters during training

(such as weight decay) or trimming connectivity (skipped connections, dropout, etc.) CNNs take a different approach towards regularization: they take advantage of the hierarchical pattern in data and assemble patterns of increasing complexity using smaller and simpler patterns embossed in their filters. Therefore, on a scale of connectivity and complexity, CNNs are on the lower extreme.

Convolutional networks were inspired by biological processes in that the connectivity pattern between neurons resembles the organization of the animal visual cortex. Individual cortical neurons respond to stimuli only in a restricted region of the visual field known as the receptive field. The receptive fields of different neurons partially overlap such that they cover the entire visual field. CNNs use relatively little pre-processing compared to

other image classification algorithms. This means that the network learns to optimize the filters (or kernels) through automated learning, whereas in traditional algorithms these filters are hand-engineered. This independence from prior knowledge and human intervention in feature extraction is a major advantage.

Machine Learning Task:

Classification

Predictor: Image of size 224 X 224 X3

Response Variable: *One of 1000 object categories, such as keyboard, mouse, pencil, and many animals.*

Model Parameters:

Weights (Learned During Training)

Hyper Parameters:

- *Bias Inputs*
- *Number of Layers*
- *Number of Neurons in each Layer*
- *Activation Function*
- *Learning rule/Algorithm*
- *Learning Rate*
- *Epochs*
- *Batch Size*
- *Dropout for regularization*

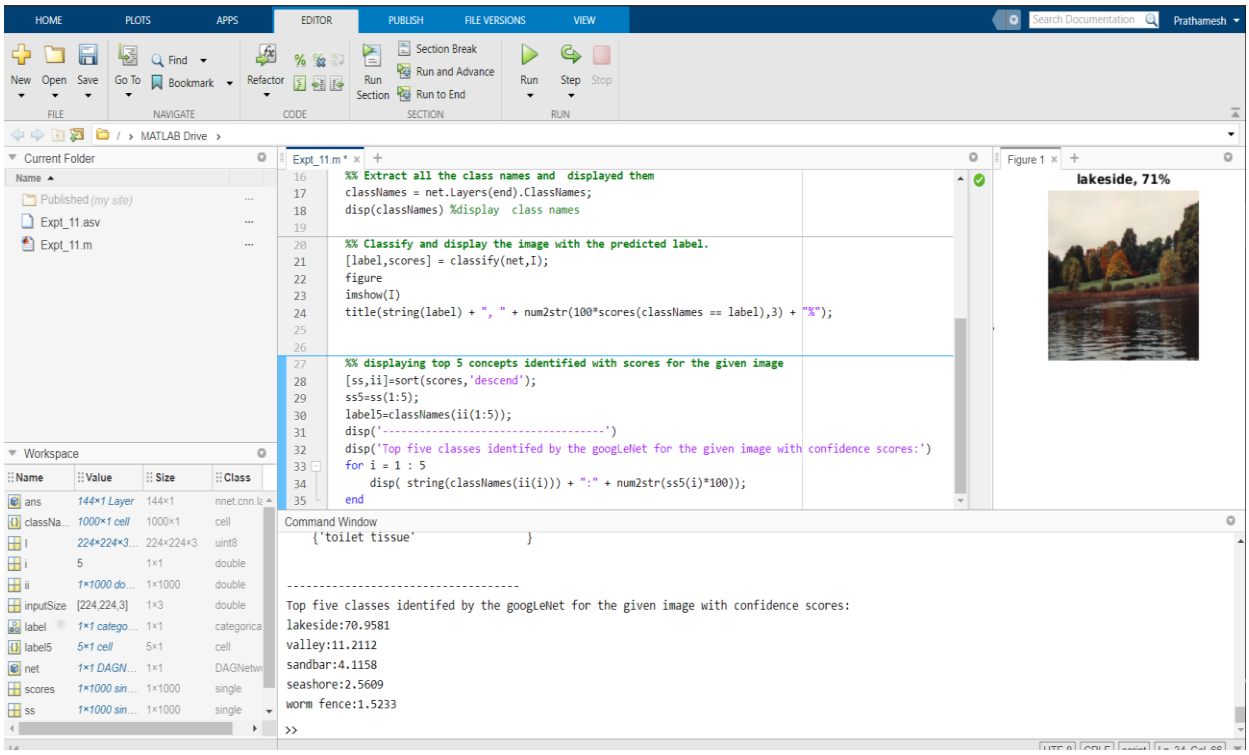
Pseudocode:

```
Input: net = Pretrained CNN named googlenet
Step:
    load the pretrained CNN
    load the image and resize
    label, scores = classify(net, image)
    confidence = scores[classNames==label]
    print(label, confidence)
Output: A class and it’s confidence level.
```

Program:

```
%% Load the pretrained CNN named google net
net = googlenet;

%% See details of the architecture
net.Layers

%% Read and Resize Image

% The image that you want to classify must have the same size as
the input size of the network.
% For GoogLeNet, the network input size is the InputSize property
of the image input layer.
%I = imread("football.jpg");
I = imread("autumn.tif")
inputSize = net.Layers(1).InputSize;
I = imresize(I,inputSize(1:2));

%% Extract all the class names and  displayed them
classNames = net.Layers(end).ClassNames;
disp(classNames) %display  class names

%% Classify and display the image with the predicted label.
[label,scores] = classify(net,I);
figure
imshow(I)
title(string(label) + ", " + num2str(100*scores(classNames ==
label),3) + "%");


%% displaying top 5 concepts identified with scores for the given
image
[ss,ii]=sort(scores,'descend');
ss5=ss(1:5);
label5=classNames(ii(1:5));
disp('-------------------------------------')
disp('Top five classes identifed by the googLeNet for the given
image with confidence scores:')
for i = 1 : 5
    disp( string(classNames(ii(i))) + ":" + num2str(ss5(i)*100));
end
```

Output:



Conclusion:

We have Successfully Predicted the class of the image by using GoogleNet Convolutional Neural Network.