**DEPARTMENT OF CSE - ARTIFICIAL INTELLIGENCE & MACHINE LEARNING**

**III B Tech I Sem**

**MACHINE LEARNING LAB**

Experiment-1:Implement and demonstrate the FIND-S algorithm for finding the most specific hypothesis based on a given set of training data samples. Read the training data from a .CSV file.

Experiment-2: For a given set of training data examples stored in a .CSV file, implement and demonstrate the Candidate- Elimination algorithm to output a description of the set of all hypotheses consistent with the training examples.

Experiment-3:Write a program to demonstrate the working of the decision tree based ID3 algorithm. Use an appropriate data set for building the decision tree and apply this knowledge to classify a new sample.

Experiment-4:Exercises to solve the real-world problems using the following machine learning methods: a) LinearRegression b) Logistic Regression c) Binary Classifier

Experiment-5: Develop a program for Bias, Variance, Remove duplicates , Cross Validation

Experiment-6: Write a program to implement Categorical Encoding, One-hot Encoding

Experiment-7:Build an Artificial Neural Network by implementing the Back propagation algorithm and test the same

using appropriate data sets.

Experiment-8:Write a program to implement k-Nearest Neighbor algorithm to classify the iris data set. Print both correctand wrong predictions.

Experiment-9: Implement the non-parametric Locally Weighted Regression algorithm in order to fit data

Experiment-10:Assuming a set of documents that need to be classified, use the naïve Bayesian Classifier model to perform his task. Built-in Java classes/API can be used to write the program. Calculate the accuracy, precision, andrecall for your data set.

Experiment-11: Apply EM algorithm to cluster a Heart Disease Data Set. Use the same data set forclustering using k-Means algorithm. Compare the results of these two algorithms and comment on thequality of clustering. You can add Java/Python ML library classes/API in the program.

Experiment-12: Exploratory Data Analysis for Classification using Pandas or Matplotlib.

Experiment-13:Write a Python program to construct a Bayesian network considering medical data. Use this model to demonstrate the diagnosis of heart patients using standard Heart Disease Data Set

Experiment-14: Write a program to Implement Support Vector Machines and Principle Component Analysis

Experiment-15: Write a program to Implement Principle Component Analysis

**Experiment-1**

**Aim:**Implement and demonstrate the FIND-S algorithm for finding the most specific hypothesis based on a given set of training data samples. Read the training data from a .CSV file.

Program code:

import csv

a = []

with open('1.csv', 'r') as csvfile:

for row in csv.reader(csvfile):

a.append(row)

print(a)

print("\n The total number of training instances are : ",len(a))

num\_attribute = len(a[0])-1

hypothesis = ['0']\*num\_attribute

print("\n The initial hypothesis is : \n", hypothesis)

for i in range(0, len(a)):

if a[i][num\_attribute] == 'yes':

for j in range(0, num\_attribute):

if hypothesis[j] == '0' or hypothesis[j] == a[i][j]:

hypothesis[j] = a[i][j]

else:

hypothesis[j] = '?'

print("\n The hypothesis for the training instance {} is: \n" .format(i+1),hypothesis)

print("\n The Maximally specific hypothesis for the training instances is: \n",hypothesis )

OUTPUT:

[['sunny', 'warm', 'normal', 'strong', 'warm', 'same', 'yes'], ['sunny', 'warm', 'high', 'strong', 'warm', 'same', 'yes'],

['rainy', 'cold', 'high', 'strong', 'warm', 'change', 'no'], ['sunny', 'warm', 'high', 'strong', 'cool', 'change', 'yes']]

The total number of training instances are : 4

The initial hypothesis is :

['0', '0', '0', '0', '0', '0']

The hypothesis for the training instance 1 is:

['sunny', 'warm', 'normal', 'strong', 'warm', 'same']

The hypothesis for the training instance 2 is:

['sunny', 'warm', '?', 'strong', 'warm', 'same']

The hypothesis for the training instance 3 is:

['sunny', 'warm', '?', 'strong', 'warm', 'same']

The hypothesis for the training instance 4 is:

['sunny', 'warm', '?', 'strong', '?', '?']

The Maximally specific hypothesis for the training instances is:

['sunny', 'warm', '?', 'strong', '?', '?']

**EXPERIMENT-2**

**Aim**: For a given set of training data examples stored in a .CSV file, implement and demonstrate the Candidate- Elimination algorithm to output a description of the set of all hypotheses consistent with the training examples

Program code:

import csv

with open("trainingexamples.csv") as f:

csv\_file = csv.reader(f)

data = list(csv\_file)

specific = data[1][:-1]

general = [['?' for i in range(len(specific))] for j in range(len(specific))]

for i in data:

if i[-1] == "Yes":

for j in range(len(specific)):

if i[j] != specific[j]:

specific[j] = "?"

general[j][j] = "?"

elif i[-1] == "No":

for j in range(len(specific)):

if i[j] != specific[j]:

general[j][j] = specific[j]

else:

general[j][j] = "?"

print("\nStep " + str(data.index(i)+1) + " of Candidate Elimination Algorithm")

print(specific)

print(general)

gh = [] # gh = general Hypothesis

for i in general:

for j in i:

if j != '?':

gh.append(i)

break

print("\nFinal Specific hypothesis:\n", specific)

print("\nFinal General hypothesis:\n", gh)

OUTPUT: Step 1 of Candidate Elimination Algorithm

['Sunny', 'Warm', 'Normal', 'Strong', 'Warm', 'Same']

[['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]

Step 2 of Candidate Elimination Algorithm

['Sunny', 'Warm', 'Normal', 'Strong', 'Warm', 'Same']

[['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]

Step 3 of Candidate Elimination Algorithm

['Sunny', 'Warm', '?', 'Strong', 'Warm', 'Same']

[['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]

Step 4 of Candidate Elimination Algorithm

['Sunny', 'Warm', '?', 'Strong', 'Warm', 'Same']

[['Sunny', '?', '?', '?', '?', '?'], ['?', 'Warm', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', 'Same']]

Step 5 of Candidate Elimination Algorithm

['Sunny', 'Warm', '?', 'Strong', '?', '?']

[['Sunny', '?', '?', '?', '?', '?'], ['?', 'Warm', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]

Final Specific hypothesis:

['Sunny', 'Warm', '?', 'Strong', '?', '?']

Final General hypothesis:

[['Sunny', '?', '?', '?', '?', '?'], ['?', 'Warm', '?', '?', '?', '?']]

**EXPERIMENT-3**

**Aim** :Write a program to demonstrate the working of the decision tree based ID3 algorithm. Use an appropriate data set for building the decision tree and apply this knowledge to classify a new sample.

Program code:

import numpy as np

import math

import csv

def read\_data(filename):

with open(filename, 'r') as csvfile:

datareader = csv.reader(csvfile, delimiter=',')

headers = next(datareader)

metadata = []

traindata = []

for name in headers:

metadata.append(name)

for row in datareader:

traindata.append(row)

return (metadata, traindata)

class Node:

def \_\_init\_\_(self, attribute):

self.attribute = attribute

self.children = []

self.answer = ""

def \_\_str\_\_(self):

return self.attribute

def subtables(data, col, delete):

dict = {}

items = np.unique(data[:, col])

count = np.zeros((items.shape[0], 1), dtype=np.int32)

for x in range(items.shape[0]):

for y in range(data.shape[0]):

if data[y, col] == items[x]:

count[x] += 1

for x in range(items.shape[0]):

dict[items[x]] = np.empty((int(count[x]), data.shape[1]), dtype="|S32")

pos = 0

for y in range(data.shape[0]):

if data[y, col] == items[x]:

dict[items[x]][pos] = data[y]

pos += 1

if delete:

dict[items[x]] = np.delete(dict[items[x]], col, 1)

return items, dict

def entropy(S):

items = np.unique(S)

if items.size == 1:

return 0

counts = np.zeros((items.shape[0], 1))

sums = 0

for x in range(items.shape[0]):

counts[x] = sum(S == items[x]) / (S.size \* 1.0)

for count in counts:

sums += -1 \* count \* math.log(count, 2)

return sums

def gain\_ratio(data, col):

items, dict = subtables(data, col, delete=False)

total\_size = data.shape[0]

entropies = np.zeros((items.shape[0], 1))

intrinsic = np.zeros((items.shape[0], 1))

for x in range(items.shape[0]):

ratio = dict[items[x]].shape[0]/(total\_size \* 1.0)

entropies[x] = ratio \* entropy(dict[items[x]][:, -1])

intrinsic[x] = ratio \* math.log(ratio, 2)

total\_entropy = entropy(data[:, -1])

iv = -1 \* sum(intrinsic)

for x in range(entropies.shape[0]):

total\_entropy -= entropies[x]

return total\_entropy / iv

def create\_node(data, metadata):

if (np.unique(data[:, -1])).shape[0] == 1:

node = Node("")

node.answer = np.unique(data[:, -1])[0]

return node

gains = np.zeros((data.shape[1] - 1, 1))

for col in range(data.shape[1] - 1):

gains[col] = gain\_ratio(data, col)

split = np.argmax(gains)

node = Node(metadata[split])

metadata = np.delete(metadata, split, 0)

items, dict = subtables(data, split, delete=True)

for x in range(items.shape[0]):

child = create\_node(dict[items[x]], metadata)

node.children.append((items[x], child))

return node

def empty(size):

s = ""

for x in range(size):

s += " "

return s

def print\_tree(node, level):

if node.answer != "":

print(empty(level), node.answer)

return

print(empty(level), node.attribute)

for value, n in node.children:

print(empty(level + 1), value)

print\_tree(n, level + 2)

metadata, traindata = read\_data("datasets.csv")

data = np.array(traindata)

node = create\_node(data, metadata)

print\_tree(node, 0)

OUTPUT

Day

D1

b'No'

D10

b'Yes'

D2

b'No'

D3

b'Yes'

D4

b'Yes'

D5

b'Yes'

D6

b'No'

D7

b'Yes'

D8

b'No'

D9

b'Yes'

<ipython-input-13-9006b06be895>:38: DeprecationWarning: Conversion of an array with ndim > 0 to a scalar is deprecated, and will error in future. Ensure you extract a single element from your array before performing this operation. (Deprecated NumPy 1.25.)

dict[items[x]] = np.empty((int(count[x]), data.shape[1]), dtype="|S32")

<ipython-input-13-9006b06be895>:62: DeprecationWarning: Conversion of an array with ndim > 0 to a scalar is deprecated, and will error in future. Ensure you extract a single element from your array before performing this operation. (Deprecated NumPy 1.25.)

sums += -1 \* count \* math.log(count, 2)

**EXPERIMENT-4**

**Aim:**

**PROGRAM**

To solve the real-world problems using the machine learning methods. Linear Regression and

Logistic Regression

Dataset: std\_marks.csv-constructed on own by using students lab internal and external marks.

Program code:

import pandas as pd

from sklearn import linear\_model

import matplotlib.pyplot as plt

from sklearn.metrics import mean\_squared\_error

from sklearn.model\_selection import train\_test\_split

data=pd.read\_csv(r"E:\sudhakar\std\_marks.csv")

print('First 5 rows of the data set are:')

print(data.head())

dim=data.shape

print('Dimensions of the data set are',dim)

print('Statistics of the data are:')

print(data.describe())

print('Correlation matrix of the data set is:')

print(data.corr())

x\_set=data[['internal']]

print('First 5 rows of features set are:')

print(x\_set.head())

y\_set=data[['external']]

print('First 5 rows of features set are:')

print(y\_set.head())

x\_train,x\_test, y\_train, y\_test = train\_test\_split(x\_set,y\_set, test\_size = 0.3)

model=linear\_model.LinearRegression()

model.fit(x\_train,y\_train)

print('Regression coefficient is',float(model.coef\_))

print('Regression intercept is',float(model.intercept\_))

y\_pred=model.predict(x\_test)

y\_preds=[]

for i in y\_pred:

7 y\_preds.append(float(i))

print('Predicted values for test data are:')

print(y\_preds)

print('mean squared error is ',mean\_squared\_error(y\_test,y\_pred))

plt.scatter(x\_test,y\_test,color='blue',label='actual y values')

plt.plot(x\_test,y\_pred,color='red',linewidth=3,label='predicted regression line')

plt.ylabel('y value')

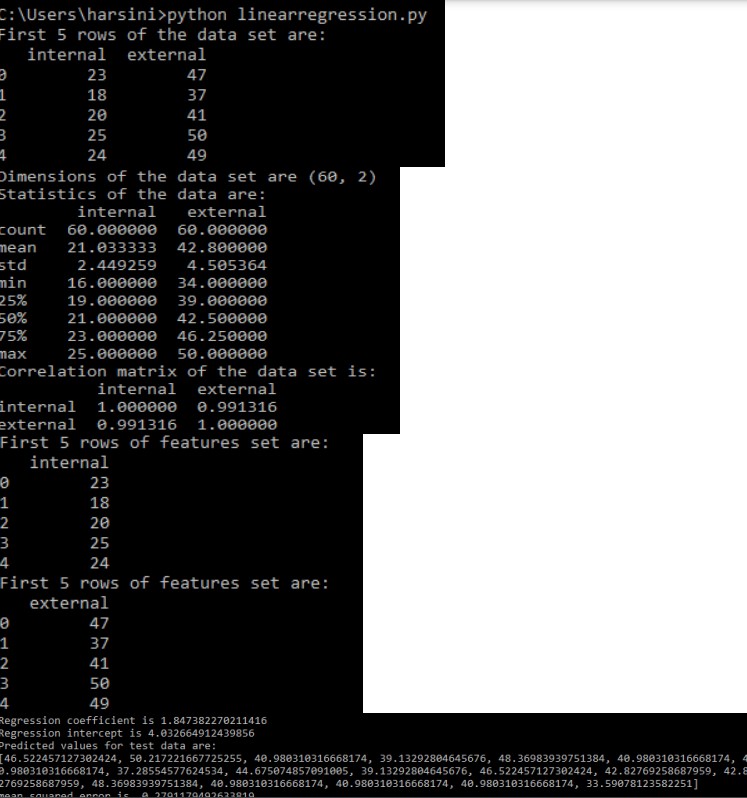
plt.xlabel('x value')

plt.title('simple linear regression')

plt.legend(loc='best')

plt.show()

Output screen shots:



B)

import warnings

warnings.filterwarnings("ignore")

import pandas as pd

import seaborn as sns

from sklearn.model\_selection import train\_test\_split

from sklearn.linear\_model import LogisticRegression

from sklearn.metrics import confusion\_matrix

from sklearn.metrics import classification\_report

from sklearn.metrics import accuracy\_score

from sklearn.metrics import recall\_score

from sklearn.metrics import precision\_score

from sklearn.preprocessing import StandardScaler

data=pd.read\_csv(r"E:\sudhakar\heart.csv")

print('The first 5 rows of the data set are:')

print(data.head())

dim=data.shape

print('Dimensions of the data set are',dim)

print('Statistics of the data are:')

print(data.describe())

print('Correlation matrix of the data set is:')

print(data.corr())

class\_lbls=data['target'].unique()

class\_labels=[]

for x in class\_lbls:

class\_labels.append(str(x))

print('Class labels are:')

print(class\_labels)

sns.countplot(data['target'])

col\_names=data.columns

feature\_names=col\_names[:-1]

feature\_names=list(feature\_names)

print('Feature names are:')

print(feature\_names)

x\_set = data.drop(['target'], axis=1)

print('First 5 rows of features set are:')

print(x\_set.head())

y\_set=data[['target']]

print('First 5 rows of features set are:')

print(y\_set.head())

scaler=StandardScaler()

x\_train,x\_test, y\_train, y\_test = train\_test\_split(x\_set,y\_set, test\_size = 0.3)

scaler.fit(x\_train)

x\_train=scaler.transform(x\_train)

model = LogisticRegression()

model.fit(x\_train, y\_train)

x\_test=scaler.transform(x\_test)

y\_pred=model.predict(x\_test)

print('Predicted class labels for test data are:')

print(y\_pred)

print("Accuracy:",accuracy\_score(y\_test, y\_pred))

print("Precision:",precision\_score(y\_test, y\_pred))

print("Recall:",recall\_score(y\_test, y\_pred))

print(classification\_report(y\_test,y\_pred,target\_names=class\_labels))

cm=confusion\_matrix(y\_test,y\_pred)

df\_cm = pd.DataFrame(cm, columns=class\_labels, index = class\_labels)

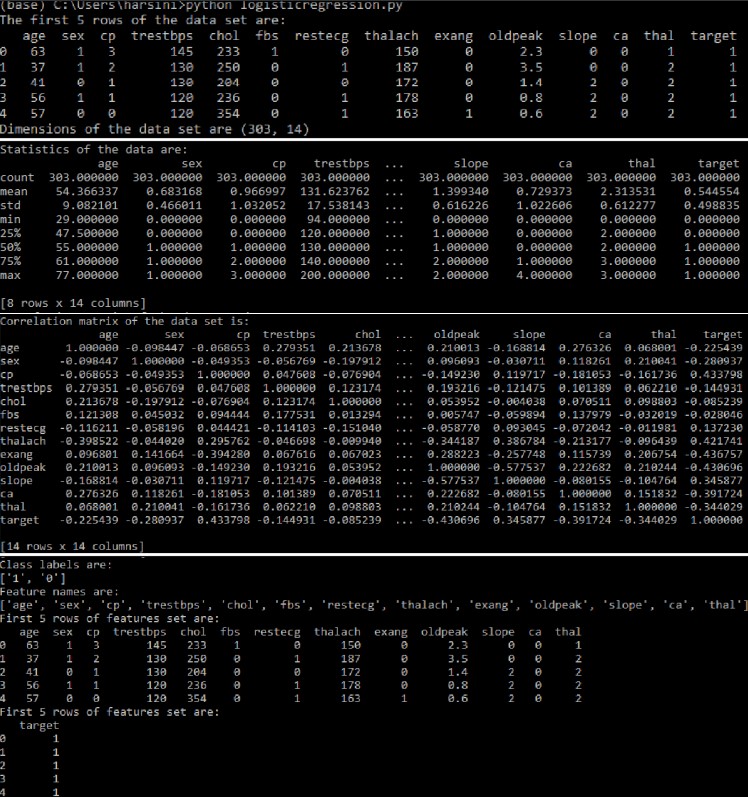
df\_cm.index.name = 'Actual'

df\_cm.columns.name = 'Predicted'

sns.set(font\_scale=1.5)

sns.heatmap(df\_cm, annot=True,cmap="Blues",fmt='d')

OUTPUT



**EXPERIMENGT-5**

**Aim:** Develop a program for Bias, Variance, Remove duplicates , Cross Validation

**PROGRAM**

from pandas import read\_csv

from sklearn.model\_selection import train\_test\_split

from sklearn.linear\_model import LinearRegression

from mlxtend.evaluate import bias\_variance\_decomp

# load dataset

url = 'https://raw.githubusercontent.com/jbrownlee/Datasets/master/housing.csv'

dataframe = read\_csv(url, header=None)

# separate into inputs and outputs

data = dataframe.values

X, y = data[:, :-1], data[:, -1]

# split the data

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.33, random\_state=1)

# define the model

model = LinearRegression()

# estimate bias and variance

mse, bias, var = bias\_variance\_decomp(model, X\_train, y\_train, X\_test, y\_test, loss='mse', num\_rounds=200, random\_seed=1)

# summarize results

print('MSE: %.3f' % mse)

print('Bias: %.3f' % bias)

print('Variance: %.3f' % var)

**OUTPUT**

MSE: 22.418

Bias: 20.744

Variance: 1.674

**EXPERIMENT-6**

**Aim;** Write a program to implement Categorical Encoding, One-hot Encoding

**PROGRAM**

import pandas as pd

# Sample categorical data

data = {'fruit': ['apple', 'orange', 'banana', 'apple']}

df = pd.DataFrame(data)

# One Hot Encoding using Pandas get\_dummies

encoded\_df = pd.get\_dummies(df, columns=['fruit'])

print(encoded\_df)

**OUTPUT**

fruit\_apple fruit\_banana fruit\_orange

0 True False False

1 False False True

2 False True False

3 True False False

**Experiment -7**

**Aim**: To build ANN by implementing the back propogation algorithm and test the same using appropriate datasets

Program code:

import numpy as np

import matplotlib.pyplot as plt

# Define the sigmoid activation function and its derivative

def sigmoid(x):

return 1 / (1 + np.exp(-x))

def sigmoid\_derivative(x):

return x \* (1 - x)

# XOR dataset

X = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])

y = np.array([[0], [1], [1], [0]])

# Neural network architecture

input\_size = 2

hidden\_size = 4

output\_size = 1

# Initialize weights and biases

weights\_input\_hidden = np.random.rand(input\_size, hidden\_size)

bias\_hidden = np.zeros((1, hidden\_size))

weights\_hidden\_output = np.random.rand(hidden\_size, output\_size)

bias\_output = np.zeros((1, output\_size))

# Training parameters

epochs = 10000

learning\_rate = 0.1

# Lists to store loss values for visualization

loss\_history = []

# Training loop

for epoch in range(epochs):

# Forward pass

hidden\_layer\_input = np.dot(X, weights\_input\_hidden) + bias\_hidden

hidden\_layer\_output = sigmoid(hidden\_layer\_input)

output\_layer\_input = np.dot(hidden\_layer\_output, weights\_hidden\_output) + bias\_output

output\_layer\_output = sigmoid(output\_layer\_input)

# Calculate loss

loss = np.mean(np.square(y - output\_layer\_output))

loss\_history.append(loss)

# Backpropagation

output\_delta = (y - output\_layer\_output) \* sigmoid\_derivative(output\_layer\_output)

hidden\_layer\_loss = output\_delta.dot(weights\_hidden\_output.T)

hidden\_layer\_delta = hidden\_layer\_loss \* sigmoid\_derivative(hidden\_layer\_output)

# Update weights and biases

weights\_hidden\_output += hidden\_layer\_output.T.dot(output\_delta) \* learning\_rate

bias\_output += np.sum(output\_delta, axis=0, keepdims=True) \* learning\_rate

weights\_input\_hidden += X.T.dot(hidden\_layer\_delta) \* learning\_rate

bias\_hidden += np.sum(hidden\_layer\_delta, axis=0, keepdims=True) \* learning\_rate

# Test the trained neural network

test\_input = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])

predictions = sigmoid(np.dot(sigmoid(np.dot(test\_input, weights\_input\_hidden) + bias\_hidden), weights\_hidden\_output) + bias\_output)

print(predictions)

# Visualize the training loss

plt.plot(range(epochs), loss\_history)

plt.xlabel('Epochs')

plt.ylabel('Loss')

plt.title('Training Loss')

plt.show()

A graph with a line

Description automatically generated

**EXPERIMENT-8**

**Aim:**Write a program to implement k-Nearest Neighbor algorithm to classify the iris data set. Print both correct and wrong predictions.

Program Code:

# Import necessary libraries

from sklearn import datasets

from sklearn.model\_selection import train\_test\_split

from sklearn.neighbors import KNeighborsClassifier

from sklearn.metrics import accuracy\_score

# Load the Iris dataset

iris = datasets.load\_iris()

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, test\_size=0.3, random\_state=42)

# Initialize the k-NN classifier with a specified k value (e.g., k=3)

knn = KNeighborsClassifier(n\_neighbors=3)

# Fit the classifier to the training data

knn.fit(X\_train, y\_train)

# Make predictions on the test data

y\_pred = knn.predict(X\_test)

# Print correct and wrong predictions

correct\_predictions = 0

wrong\_predictions = 0

for i in range(len(y\_test)):

if y\_pred[i] == y\_test[i]:

print(f"Correct Prediction: Actual Class {y\_test[i]}, Predicted Class {y\_pred[i]}")

correct\_predictions += 1

else:

print(f"Wrong Prediction: Actual Class {y\_test[i]}, Predicted Class {y\_pred[i]}")

wrong\_predictions += 1

# Calculate and print the accuracy of the k-NN classifier

accuracy = accuracy\_score(y\_test, y\_pred)

print(f"Accuracy: {accuracy \* 100:.2f}%")

print(f"Correct Predictions: {correct\_predictions}")

print(f"Wrong Predictions: {wrong\_predictions}")

Output:

Correct Prediction: Actual Class 1, Predicted Class 1

Correct Prediction: Actual Class 0, Predicted Class 0

Correct Prediction: Actual Class 2, Predicted Class 2

Correct Prediction: Actual Class 1, Predicted Class 1

Correct Prediction: Actual Class 1, Predicted Class 1

Correct Prediction: Actual Class 0, Predicted Class 0

Correct Prediction: Actual Class 1, Predicted Class 1

Correct Prediction: Actual Class 2, Predicted Class 2

Correct Prediction: Actual Class 1, Predicted Class 1

Correct Prediction: Actual Class 1, Predicted Class 1

Correct Prediction: Actual Class 2, Predicted Class 2

Correct Prediction: Actual Class 0, Predicted Class 0

Correct Prediction: Actual Class 0, Predicted Class 0

Correct Prediction: Actual Class 0, Predicted Class 0

Correct Prediction: Actual Class 0, Predicted Class 0

Correct Prediction: Actual Class 1, Predicted Class 1

Correct Prediction: Actual Class 2, Predicted Class 2

Correct Prediction: Actual Class 1, Predicted Class 1

Correct Prediction: Actual Class 1, Predicted Class 1

Correct Prediction: Actual Class 2, Predicted Class 2

Correct Prediction: Actual Class 0, Predicted Class 0

Correct Prediction: Actual Class 2, Predicted Class 2

Correct Prediction: Actual Class 0, Predicted Class 0

Correct Prediction: Actual Class 2, Predicted Class 2

Correct Prediction: Actual Class 2, Predicted Class 2

Correct Prediction: Actual Class 2, Predicted Class 2

Correct Prediction: Actual Class 2, Predicted Class 2

Correct Prediction: Actual Class 2, Predicted Class 2

Correct Prediction: Actual Class 0, Predicted Class 0

Correct Prediction: Actual Class 0, Predicted Class 0

Correct Prediction: Actual Class 0, Predicted Class 0

Correct Prediction: Actual Class 0, Predicted Class 0

Correct Prediction: Actual Class 1, Predicted Class 1

Correct Prediction: Actual Class 0, Predicted Class 0

Correct Prediction: Actual Class 0, Predicted Class 0

Correct Prediction: Actual Class 2, Predicted Class 2

Correct Prediction: Actual Class 1, Predicted Class 1

Correct Prediction: Actual Class 0, Predicted Class 0

Correct Prediction: Actual Class 0, Predicted Class 0

Correct Prediction: Actual Class 0, Predicted Class 0

Correct Prediction: Actual Class 2, Predicted Class 2

Correct Prediction: Actual Class 1, Predicted Class 1

Correct Prediction: Actual Class 1, Predicted Class 1

Correct Prediction: Actual Class 0, Predicted Class 0

Correct Prediction: Actual Class 0, Predicted Class 0

Accuracy: 100.00%

Correct Predictions: 45

Wrong Predictions: 0

**EXPERIMENT -9**

**Aim:**To implement non-parametric locally weighted regression are in order to figure the data points select appropriate dataset

Program Code:

import numpy as np

import matplotlib.pyplot as plt

# Generate synthetic data (sine wave with noise)

np.random.seed(0)

X = np.sort(5 \* np.random.rand(80, 1), axis=0)

y = np.sin(X).ravel() + np.random.rand(80)

def locally\_weighted\_regression(test\_point, X, y, tau):

    m, n = X.shape

    weights = np.zeros((m, m))

    for i in range(m):

        xi = X[i]

        weights[i, i] = np.exp(-((test\_point - xi) \*\* 2) / (2 \* tau \*\* 2))

    theta = np.linalg.pinv(X.T @ (weights @ X)) @ (X.T @ (weights @ y))

    return test\_point \* theta

# Predict for a set of test points

tau = 0.1  # Bandwidth parameter

X\_test = np.linspace(0, 5, 100)

y\_pred = [locally\_weighted\_regression(test\_point, X, y, tau) for test\_point in X\_test]

# Plot the data and predictions

plt.scatter(X, y, c='r', label='Data points')

plt.plot(X\_test, y\_pred, c='b', label='LWR Predictions')

plt.xlabel('X')

plt.ylabel('y')

plt.legend()

plt.show()

A graph with red dots

Description automatically generated

**Experiment-10:**

**Aim:**Assuming a set of documents that need to be classified, use the naïve Bayesian Classifier model to perform this task. Built-in Java classes/API can be used to write the program. Calculate the accuracy, precision, and recall for your data set.

Program code:

from sklearn.datasets import fetch\_20newsgroups

from sklearn.model\_selection import train\_test\_split

from sklearn.feature\_extraction.text import CountVectorizer

from sklearn.naive\_bayes import MultinomialNB

from sklearn import metrics

# Load the 20 newsgroups dataset

newsgroups = fetch\_20newsgroups(subset='all', remove=('headers', 'footers', 'quotes'))

# Split the dataset into training and testing sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(newsgroups.data, newsgroups.target, test\_size=0.2, random\_state=42)

# Vectorize the text data

vectorizer = CountVectorizer()

X\_train\_vec = vectorizer.fit\_transform(X\_train)

X\_test\_vec = vectorizer.transform(X\_test)

# Train the Naive Bayes Classifier

nb\_classifier = MultinomialNB()

nb\_classifier.fit(X\_train\_vec, y\_train)

# Make predictions

y\_pred = nb\_classifier.predict(X\_test\_vec)

# Calculate Accuracy, Precision, and Recall

accuracy = metrics.accuracy\_score(y\_test, y\_pred)

precision = metrics.precision\_score(y\_test, y\_pred, average='weighted')

recall = metrics.recall\_score(y\_test, y\_pred, average='weighted')

# Print the results

print(f"Accuracy: {accuracy:.4f}")

print(f"Precision: {precision:.4f}")

print(f"Recall: {recall:.4f}")

Output:

Accuracy: 0.6175

Precision: 0.6960

Recall: 0.6175

EXPERIMENT-11

Aim:

Program code: from sklearn.cluster import KMeans

from sklearn.mixture import GaussianMixture

import sklearn.metrics as metrics

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

names = ['Sepal\_Length','Sepal\_Width','Petal\_Length','Petal\_Width', 'Class']

dataset = pd.read\_csv("8-dataset.csv", names=names)

X = dataset.iloc[:, :-1]

label = {'Iris-setosa': 0,'Iris-versicolor': 1, 'Iris-virginica': 2}

y = [label[c] for c in dataset.iloc[:, -1]]

plt.figure(figsize=(14,7))

colormap=np.array(['red','lime','black'])

# REAL PLOT

plt.subplot(1,3,1)

plt.title('Real')

plt.scatter(X.Petal\_Length,X.Petal\_Width,c=colormap[y])

# K-PLOT

model=KMeans(n\_clusters=3, random\_state=0).fit(X)

plt.subplot(1,3,2)

plt.title('KMeans')

plt.scatter(X.Petal\_Length,X.Petal\_Width,c=colormap[model.labels\_])

print('The accuracy score of K-Mean: ',metrics.accuracy\_score(y, model.labels\_))

print('The Confusion matrixof K-Mean:\n',metrics.confusion\_matrix(y, model.labels\_))

# GMM PLOT

gmm=GaussianMixture(n\_components=3, random\_state=0).fit(X)

y\_cluster\_gmm=gmm.predict(X)

plt.subplot(1,3,3)

plt.title('GMM Classification')

plt.scatter(X.Petal\_Length,X.Petal\_Width,c=colormap[y\_cluster\_gmm])

print('The accuracy score of EM: ',metrics.accuracy\_score(y, y\_cluster\_gmm))

print('The Confusion matrix of EM:\n ',metrics.confusion\_matrix(y, y\_cluster\_gmm))

Output

The accuracy score of K-Mean: 0.24

The Confusion matrixof K-Mean:

[[ 0 50 0]

[48 0 2]

[14 0 36]]

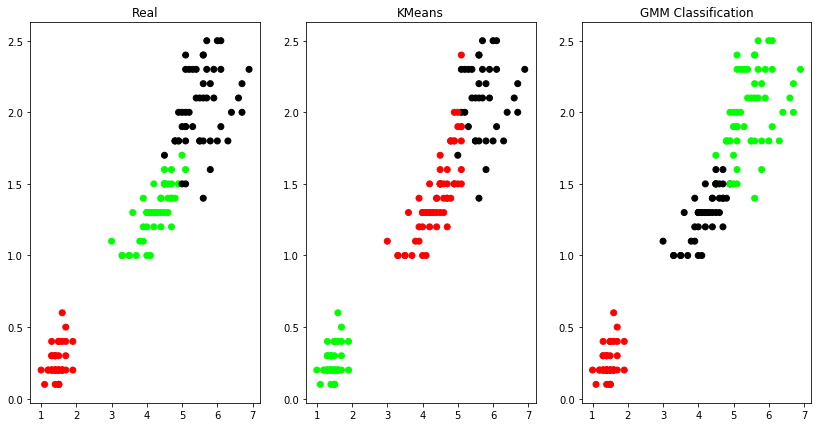
The accuracy score of EM: 0.36666666666666664

The Confusion matrix of EM:

[[50 0 0]

[ 0 5 45]

[ 0 50 0]]



**EXPERIMENT-12**

**Aim:** Exploratory Data Analysis for Classification using Pandas or Matplotlib.

Program code:

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

# Load the Heart Disease Data Set

url = "https://archive.ics.uci.edu/ml/machine-learning-databases/heart-disease/processed.cleveland.data"

column\_names = ["age", "sex", "cp", "trestbps", "chol", "fbs", "restecg", "thalach", "exang", "oldpeak", "slope", "ca", "thal", "target"]

heart\_data = pd.read\_csv(url, header=None, names=column\_names, na\_values="?")

# Display the first few rows of the dataset

print(heart\_data.head())

# Summary statistics of numerical features

print(heart\_data.describe())

# Info about the dataset, including data types and missing values

print(heart\_data.info())

# Distribution of the target variable

sns.countplot(x='target', data=heart\_data)

plt.title('Distribution of Target Variable')

plt.show()

# Pairplot to visualize relationships between numerical features

sns.pairplot(heart\_data, hue='target')

plt.suptitle('Pairplot of Numerical Features', y=1.02)

plt.show()

# Correlation heatmap to check for feature correlations

corr\_matrix = heart\_data.corr()

plt.figure(figsize=(12, 8))

sns.heatmap(corr\_matrix, annot=True, cmap='coolwarm', fmt=".2f")

plt.title('Correlation Heatmap')

plt.show()

# Boxplots for individual features

features = ["age", "trestbps", "chol", "thalach", "oldpeak"] # Select a subset of features

for feature in features:

sns.boxplot(x='target', y=feature, data=heart\_data)

plt.title(f'Boxplot of {feature} by Target')

plt.show()

Output:

age sex cp trestbps chol fbs restecg thalach exang oldpeak \

0 63.0 1.0 1.0 145.0 233.0 1.0 2.0 150.0 0.0 2.3

1 67.0 1.0 4.0 160.0 286.0 0.0 2.0 108.0 1.0 1.5

2 67.0 1.0 4.0 120.0 229.0 0.0 2.0 129.0 1.0 2.6

3 37.0 1.0 3.0 130.0 250.0 0.0 0.0 187.0 0.0 3.5

4 41.0 0.0 2.0 130.0 204.0 0.0 2.0 172.0 0.0 1.4

slope ca thal target

0 3.0 0.0 6.0 0

1 2.0 3.0 3.0 2

2 2.0 2.0 7.0 1

3 3.0 0.0 3.0 0

4 1.0 0.0 3.0 0

age sex cp trestbps chol fbs \

count 303.000000 303.000000 303.000000 303.000000 303.000000 303.000000

mean 54.438944 0.679868 3.158416 131.689769 246.693069 0.148515

std 9.038662 0.467299 0.960126 17.599748 51.776918 0.356198

min 29.000000 0.000000 1.000000 94.000000 126.000000 0.000000

25% 48.000000 0.000000 3.000000 120.000000 211.000000 0.000000

50% 56.000000 1.000000 3.000000 130.000000 241.000000 0.000000

75% 61.000000 1.000000 4.000000 140.000000 275.000000 0.000000

max 77.000000 1.000000 4.000000 200.000000 564.000000 1.000000

restecg thalach exang oldpeak slope ca \

count 303.000000 303.000000 303.000000 303.000000 303.000000 299.000000

mean 0.990099 149.607261 0.326733 1.039604 1.600660 0.672241

std 0.994971 22.875003 0.469794 1.161075 0.616226 0.937438

min 0.000000 71.000000 0.000000 0.000000 1.000000 0.000000

25% 0.000000 133.500000 0.000000 0.000000 1.000000 0.000000

50% 1.000000 153.000000 0.000000 0.800000 2.000000 0.000000

75% 2.000000 166.000000 1.000000 1.600000 2.000000 1.000000

max 2.000000 202.000000 1.000000 6.200000 3.000000 3.000000

thal target

count 301.000000 303.000000

mean 4.734219 0.937294

std 1.939706 1.228536

min 3.000000 0.000000

25% 3.000000 0.000000

50% 3.000000 0.000000

75% 7.000000 2.000000

max 7.000000 4.000000

<class 'pandas.core.frame.DataFrame'>

RangeIndex: 303 entries, 0 to 302

Data columns (total 14 columns):

# Column Non-Null Count Dtype

--- ------ -------------- -----

0 age 303 non-null float64

1 sex 303 non-null float64

2 cp 303 non-null float64

3 trestbps 303 non-null float64

4 chol 303 non-null float64

5 fbs 303 non-null float64

6 restecg 303 non-null float64

7 thalach 303 non-null float64

8 exang 303 non-null float64

9 oldpeak 303 non-null float64

10 slope 303 non-null float64

11 ca 299 non-null float64

12 thal 301 non-null float64

13 target 303 non-null int64

dtypes: float64(13), int64(1)

memory usage: 33.3 KB

None

A graph with blue squares

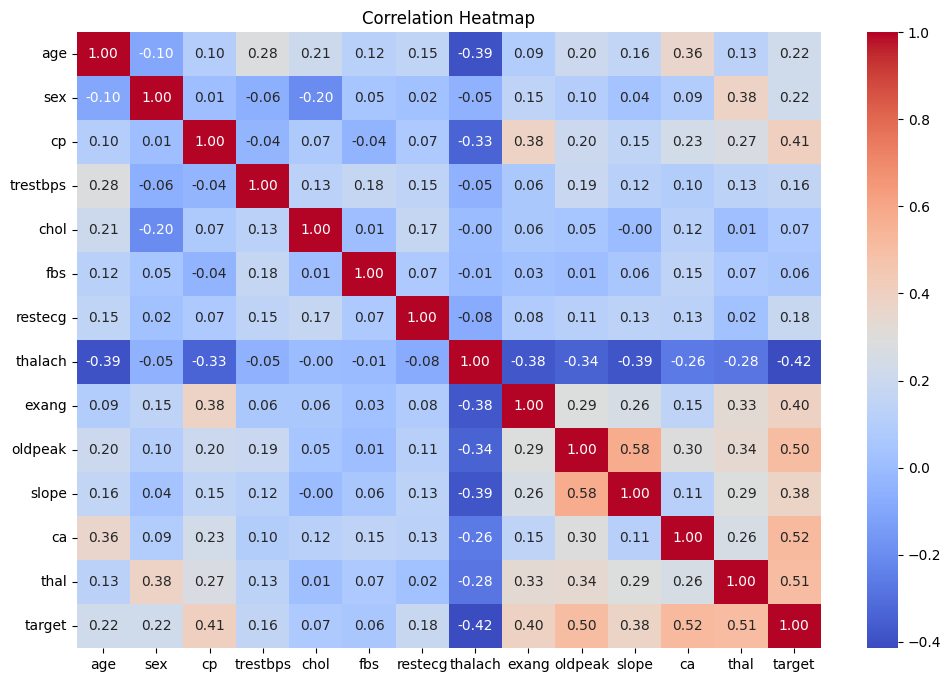
Description automatically generated

A graph of a box diagram

Description automatically generated with medium confidenceA screenshot of a graph

Description automatically generated

A chart with blue boxes and white text

Description automatically generated

A graph with blue squares and numbers

Description automatically generatedA diagram of a box plot

Description automatically generated

A graph of blue boxes

Description automatically generated

**EXPERIMENT-13**

**Aim:** Write a Python program to construct a Bayesian network considering medical data. Use this model to demonstrate the diagnosis of heart patients using standard Heart Disease Data Set

**Program code:**

**EXPERIMENT-14**

**Aim:**Write a program to Implement Support Vector Machines and Principle Component Analysis

Program code:

import numpy as np

import matplotlib.pyplot as plt

from sklearn import datasets

from sklearn.model\_selection import train\_test\_split

from sklearn.preprocessing import StandardScaler

from sklearn.decomposition import PCA

from sklearn.svm import SVC

from sklearn.metrics import accuracy\_score

# Load a sample dataset (Iris dataset)

data = datasets.load\_iris()

X = data.data

y = data.target

# Split the 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=42)

# Step 1: Standardize the data (mean=0, variance=1)

scaler = StandardScaler()

X\_train\_std = scaler.fit\_transform(X\_train)

X\_test\_std = scaler.transform(X\_test)

# Step 2: Apply PCA for dimensionality reduction

pca = PCA(n\_components=2)

X\_train\_pca = pca.fit\_transform(X\_train\_std)

X\_test\_pca = pca.transform(X\_test\_std)

# Step 3: Train an SVM classifier on the reduced data

svm = SVC(kernel='linear', C=1.0, random\_state=42)

svm.fit(X\_train\_pca, y\_train)

# Step 4: Make predictions on the test data

y\_pred = svm.predict(X\_test\_pca)

# Calculate accuracy

accuracy = accuracy\_score(y\_test, y\_pred)

print(f'Accuracy: {accuracy:.2f}')

# Visualize the decision boundary

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

# Plot the training data

plt.scatter(X\_train\_pca[:, 0], X\_train\_pca[:, 1], c=y\_train, cmap='viridis', label='Training Data')

plt.xlabel('Principal Component 1')

plt.ylabel('Principal Component 2')

# Plot decision regions

x\_min, x\_max = X\_test\_pca[:, 0].min() - 1, X\_test\_pca[:, 0].max() + 1

y\_min, y\_max = X\_test\_pca[:, 1].min() - 1, X\_test\_pca[:, 1].max() + 1

xx, yy = np.meshgrid(np.arange(x\_min, x\_max, 0.01), np.arange(y\_min, y\_max, 0.01))

Z = svm.predict(np.c\_[xx.ravel(), yy.ravel()])

Z = Z.reshape(xx.shape)

plt.contourf(xx, yy, Z, alpha=0.8, cmap='coolwarm')

plt.title('SVM Decision Boundary')

plt.legend()

plt.show()

OUTPUT:

A graph of a diagram

Description automatically generated with medium confidence

**Experiment-15**

**Aim:**Write a program to Implement Principle Component Analysis

Program code:

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

# Sample data (replace with your dataset)

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

[4, 5, 6],

[7, 8, 9],

[10, 11, 12]])

# Step 1: Standardize the data (mean=0, variance=1)

mean = np.mean(data, axis=0)

std\_dev = np.std(data, axis=0)

data\_standardized = (data - mean) / std\_dev

# Step 2: Calculate the covariance matrix

cov\_matrix = np.cov(data\_standardized, rowvar=False)

# Step 3: Calculate the eigenvalues and eigenvectors of the covariance matrix

eigenvalues, eigenvectors = np.linalg.eig(cov\_matrix)

# Step 4: Sort eigenvalues in descending order and arrange corresponding eigenvectors

sorted\_indices = np.argsort(eigenvalues)[::-1]

eigenvalues = eigenvalues[sorted\_indices]

eigenvectors = eigenvectors[:, sorted\_indices]

# Step 5: Select the top k eigenvectors to reduce dimensions (k should be chosen based on explained variance)

k = 2

selected\_eigenvectors = eigenvectors[:, :k]

# Step 6: Project the data onto the new feature space

reduced\_data = np.dot(data\_standardized, selected\_eigenvectors)

# Visualize the results

sns.set(style="whitegrid")

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

# Plot the original data

plt.subplot(121)

plt.scatter(data[:, 0], data[:, 1], label="Original Data")

plt.xlabel("Feature 1")

plt.ylabel("Feature 2")

plt.title("Original Data")

# Plot the reduced data after PCA

plt.subplot(122)

plt.scatter(reduced\_data[:, 0], reduced\_data[:, 1], label="Reduced Data")

plt.xlabel("Principal Component 1")

plt.ylabel("Principal Component 2")

plt.title("Reduced Data after PCA")

plt.tight\_layout()

plt.show()

Output:

