MLA0413-DEEP LEARNING FOR COMPLEX DATA MINING

NAME: SAANIYA SHADAAP

REG NO: 192225004

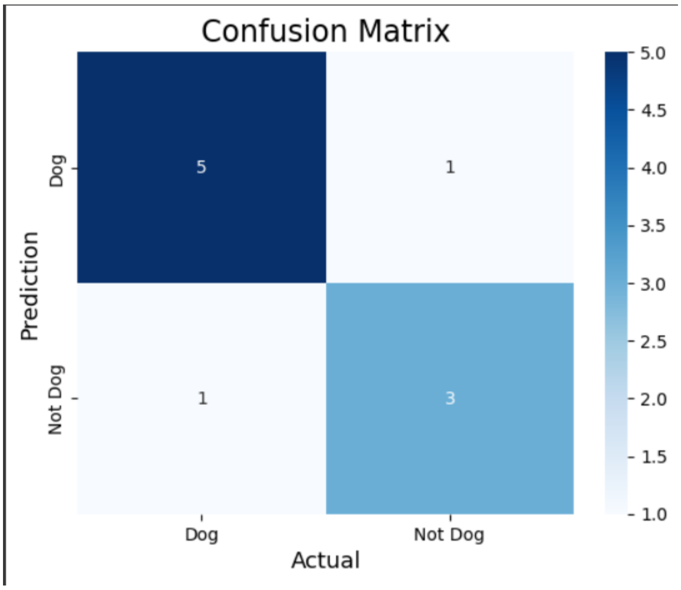
EXPERIMENT:1(A)

AIM: To demonstrate confusion matrix using python

PROGRAM:

```
import numpy as np
from sklearn.metrics import confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt
actual    = np.array(
  ['Dog','Dog','Dog','Not Dog','Dog','Not Dog','Dog','Dog','Not
Dog','Not Dog'])
predicted = np.array(
  ['Dog','Not Dog','Dog','Not Dog','Dog','Dog','Dog','Dog','Not
Dog','Not Dog'])
cm = confusion_matrix(actual,predicted)
sns.heatmap(cm,
            annot=True,
            fmt='g', cmap = 'Blues',
            xticklabels=['Dog','Not Dog'],
            yticklabels=['Dog','Not Dog'])
plt.ylabel('Prediction',fontsize=13)
plt.xlabel('Actual',fontsize=13)
plt.title('Confusion Matrix',fontsize=17)
plt.show()
```

OUTPUT :



EXPERIMENT:1(B)

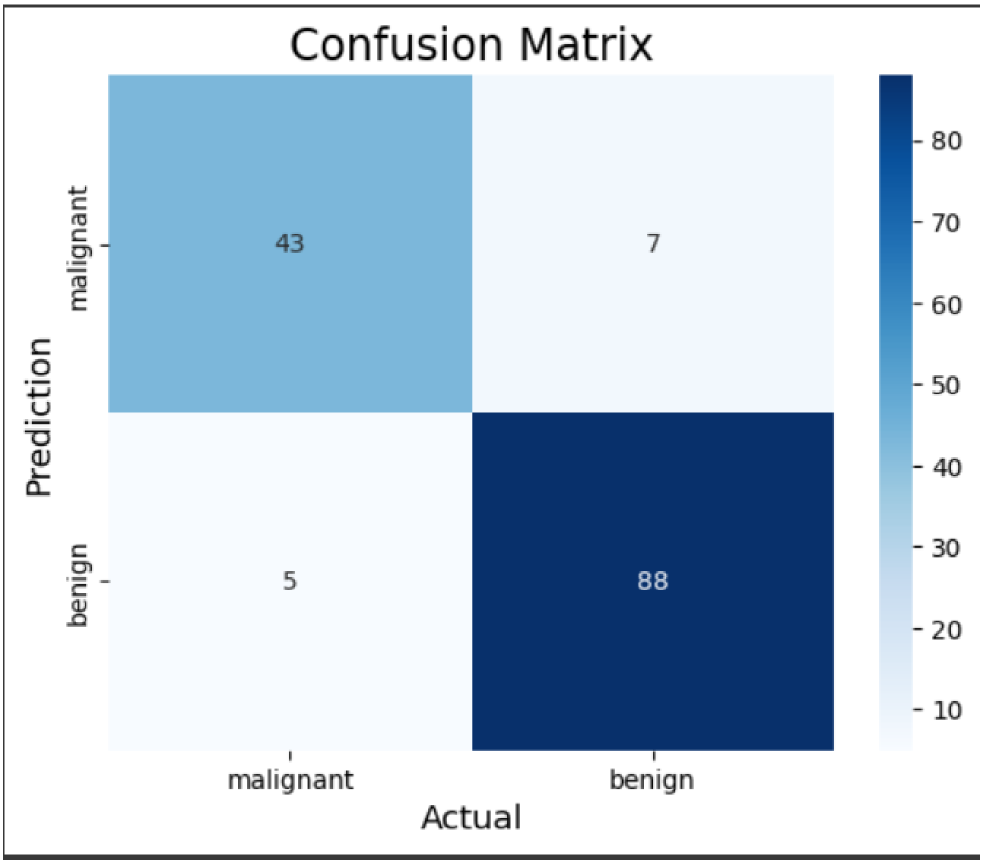
AIM: To demonstrate 2 class confusion matrix using python

PROGRAM:

```
#Import the necessary libraries
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.metrics import accuracy_score, precision_score,
recall_score, f1_score

# Load the breast cancer dataset
X, y= load_breast_cancer(return_X_y=True)
X_train, X_test, y_train, y_test = train_test_split(X,
y,test_size=0.25)

# Train the model
tree = DecisionTreeClassifier(random_state=23)
tree.fit(X_train, y_train)

# prediction
y_pred = tree.predict(X_test)

# compute the confusion matrix
cm = confusion_matrix(y_test,y_pred)

#Plot the confusion matrix.
sns.heatmap(cm,
            annot=True,
            fmt='g',
            xticklabels=['malignant', 'benign'],
            yticklabels=['malignant', 'benign'])
plt.ylabel('Prediction',fontsize=13)
plt.xlabel('Actual',fontsize=13)
plt.title('Confusion Matrix',fontsize=17)
plt.show()


# Finding precision and recall
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy   :", accuracy)
```

```
precision = precision_score(y_test, y_pred)
print("Precision :", precision)
recall = recall_score(y_test, y_pred)
print("Recall    :", recall)
F1_score = f1_score(y_test, y_pred)
print("F1-score  :", F1_score)
```

OUTPUT:




```
Accuracy   : 0.916083916083916
Precision : 0.9263157894736842
Recall    : 0.946236559139785
F1-score  : 0.9361702127659575
```

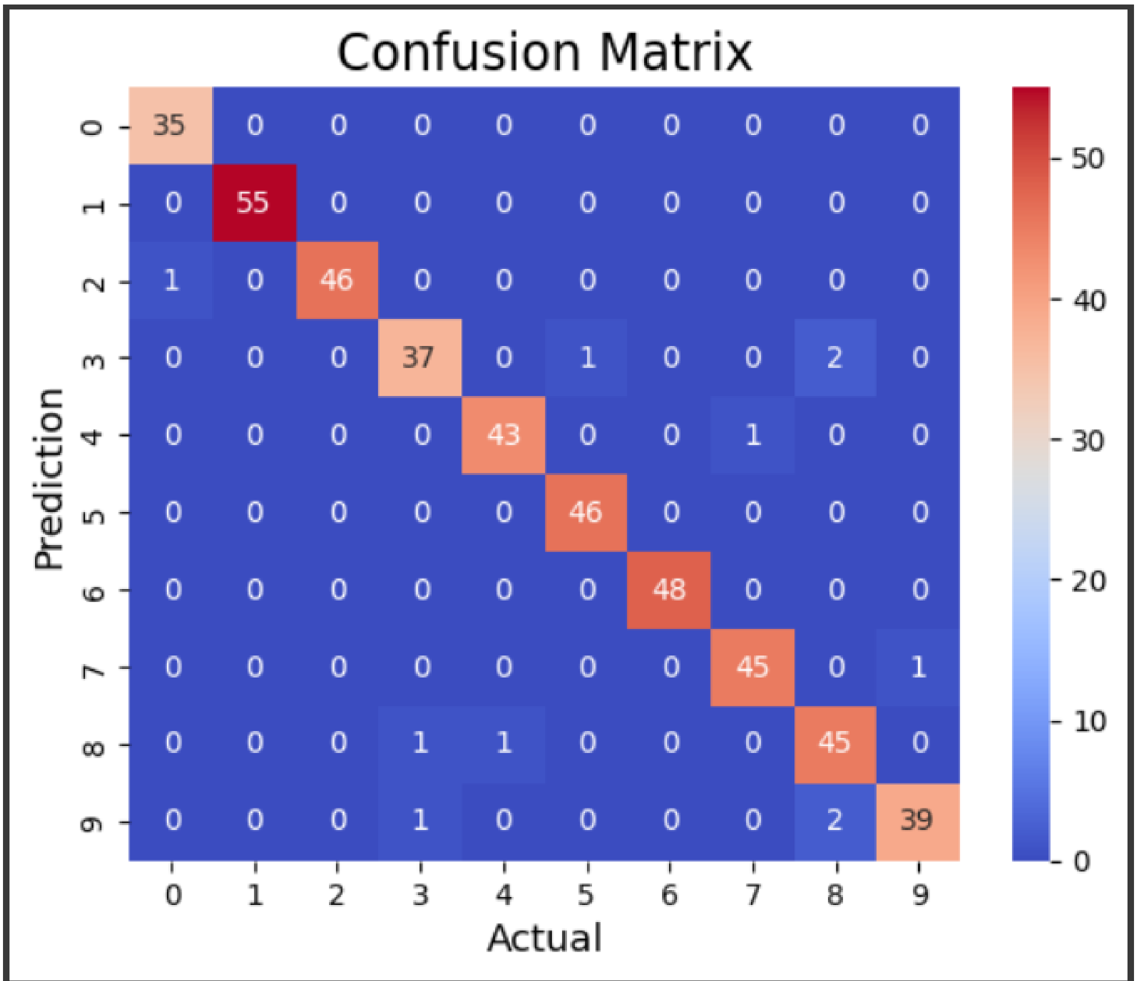
EXPERIMENT: 2

AIM: Verifying the performance of a multi class confusion matrix by using chosen database with python code

PROGRAM:

```
#Import the necessary libraries
from sklearn.datasets import load_digits
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.metrics import accuracy_score, precision_score,
recall_score, f1_score

# Load the breast cancer dataset
X, y= load_digits(return_X_y=True)
X_train, X_test, y_train, y_test = train_test_split(X,
y,test_size=0.25)

# Train the model
clf = RandomForestClassifier(random_state=23)
clf.fit(X_train, y_train)

# prediction
y_pred = clf.predict(X_test)

# compute the confusion matrix
cm = confusion_matrix(y_test,y_pred)

#Plot the confusion matrix.
sns.heatmap(cm,
            annot=True,
            fmt='g')
plt.ylabel('Prediction',fontsize=13)
plt.xlabel('Actual',fontsize=13)
plt.title('Confusion Matrix',fontsize=17)
plt.show()


# Finding precision and recall
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy   :", accuracy)
```

OUTPUT:




```
Accuracy    : 0.9755555555555555
```

EXPERIMENT:3

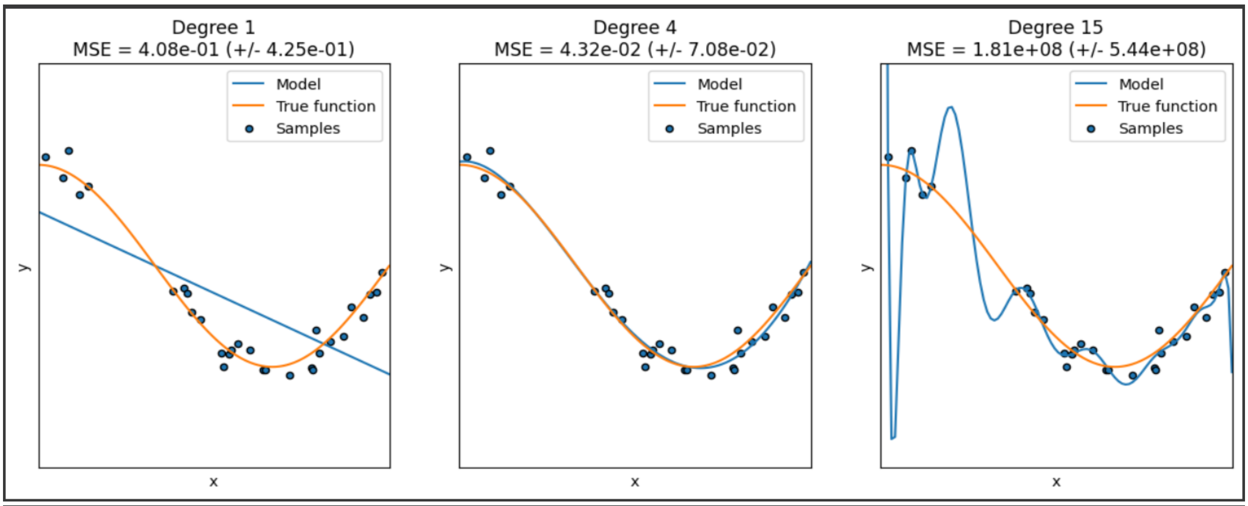
AIM: : Verifying the performance of an overfitting by using chosen database with python code

PROGRAM:

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import cross_val_score

def true_fun(X):
    return np.cos(1.5 * np.pi * X)

np.random.seed(0)

n_samples = 30
degrees = [1, 4, 15]

X = np.sort(np.random.rand(n_samples))
y = true_fun(X) + np.random.randn(n_samples) * 0.1

plt.figure(figsize=(14, 5))
for i in range(len(degrees)):
    ax = plt.subplot(1, len(degrees), i + 1)
    plt.setp(ax, xticks=(), yticks=())

    polynomial_features = PolynomialFeatures(degree=degrees[i],
include_bias=False)
    linear_regression = LinearRegression()
    pipeline = Pipeline(
        [
            ("polynomial_features", polynomial_features),
            ("linear_regression", linear_regression),
        ]
    )
    pipeline.fit(X[:, np.newaxis], y)

    # Evaluate the models using crossvalidation
    scores = cross_val_score(
        pipeline, X[:, np.newaxis], y,
scoring="neg_mean_squared_error", cv=10
    )
```

```
    X_test = np.linspace(0, 1, 100)
    plt.plot(X_test, pipeline.predict(X_test[:, np.newaxis]),
label="Model")
    plt.plot(X_test, true_fun(X_test), label="True function")
    plt.scatter(X, y, edgecolor="b", s=20, label="Samples")
    plt.xlabel("x")
    plt.ylabel("y")
    plt.xlim((0, 1))
    plt.ylim((-2, 2))
    plt.legend(loc="best")
    plt.title(
        "Degree {}\nMSE = {:.2e}(+/- {:.2e})".format(
            degrees[i], -scores.mean(), scores.std()) )
plt.show()
```

OUTPUT:



EXPERIMENT:4

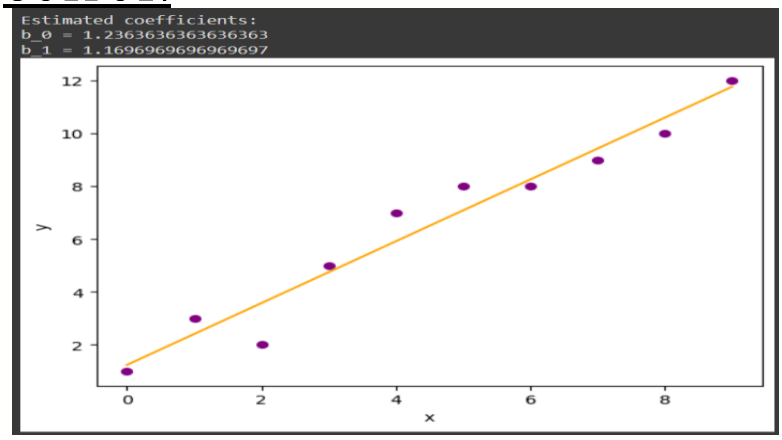
AIM: To demonstrate the performance of a linear regression by using chosen database with python code

PROGRAM: LINEAR REGRESSION

```
import numpy as np
import matplotlib.pyplot as plt
def estimate_coef(x, y):
    n = np.size(x)
    m_x = np.mean(x)
    m_y = np.mean(y)
    SS_xy = np.sum(y*x) - n*m_y*m_x
    SS_xx = np.sum(x*x) - n*m_x*m_x
```

```
    b_1 = SS_xy / SS_xx
    b_0 = m_y - b_1*m_x
    return (b_0, b_1)
def plot_regression_line(x, y, b):
    plt.scatter(x, y, color = "r",
               marker = "o", s = 30)
    y_pred = b[0] + b[1]*x
    plt.plot(x, y_pred, color = "b")
    plt.xlabel('x')
    plt.ylabel('y')
    plt.show()
def main():
    x = np.array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
    y = np.array([1, 3, 2, 5, 7, 8, 8, 9, 10, 12])
    b = estimate_coef(x, y)
    print("Estimated coefficients:\nb_0 = {}  \
          \nb_1 = {}".format(b[0], b[1]))
    plot_regression_line(x, y, b)
if __name__ == "__main__":
    main()
```

OUTPUT:



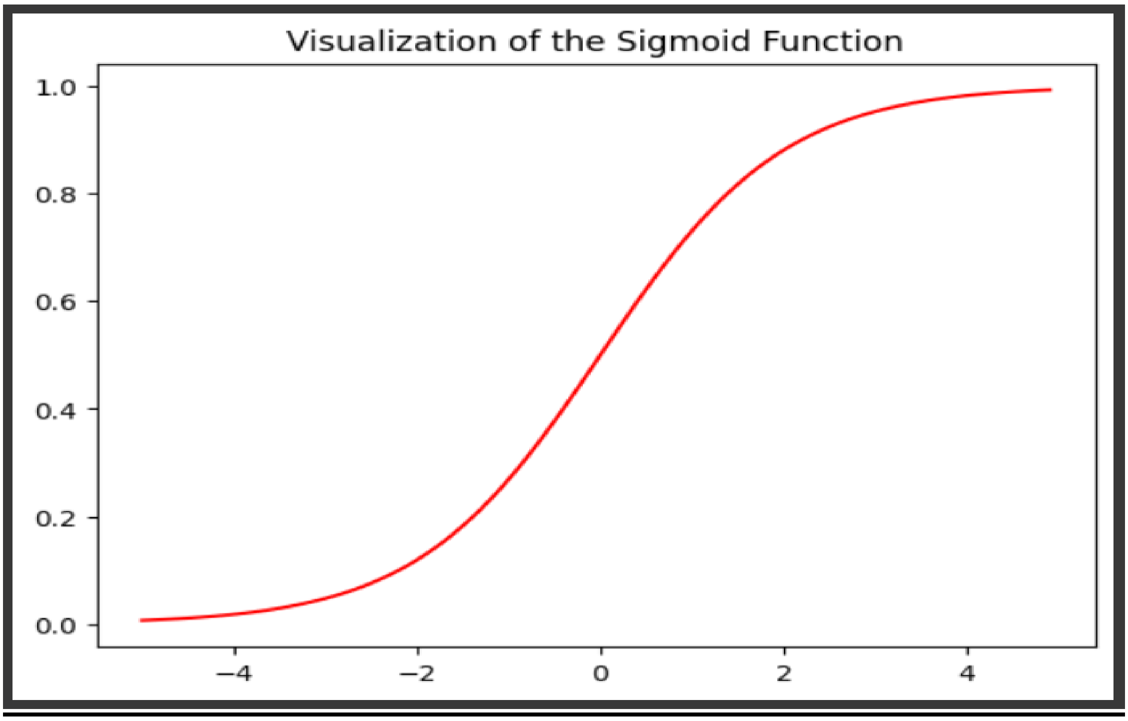
EXPERIMENT:5

AIM: To demonstrate the performance of a logistic regression by using chosen database with python code.

PROGRAM:

```
import numpy as np
import matplotlib.pyplot as plt
def sigmoid(z):
    return 1 / (1 + np.exp( - z))
plt.plot(np.arange(-5, 5, 0.1), sigmoid(np.arange(-5, 5, 0.1)))
plt.title('Visualization of the Sigmoid Function')
plt.show()
```

OUTPUT:



EXPERIMENT:6(a)KNN

AIM: Finding accuracy value of iris data set using KNN algorithm

PROGRAM:

```
import numpy as np
import pandas as pd
dataset = pd.read_csv("/Iris.csv")
X = dataset.iloc[:, :-1].values
y = dataset.iloc[:, -1].values
dataset.shape
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size =
0.20, random_state = 42)
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
from sklearn.neighbors import KNeighborsClassifier
classifier = KNeighborsClassifier(n_neighbors = 5, metric =
'minkowski', p = 2)
classifier.fit(X_train, y_train)
from sklearn.metrics import confusion_matrix, accuracy_score
y_pred = classifier.predict(X_test)
cm = confusion_matrix(y_test, y_pred)
print(cm)
accuracy_score(y_test, y_pred)
```

OUTPUT:

```
[[85  0]
 [ 2 50]]
0.9854014598540146
```

EXPERIMENT:6(B)NAVIE

AIM: : finding accuracy value of iris data set using NAVIE BAYES algorithm

PROGRAM:

```
import numpy as np
import pandas as pd
dataset = pd.read_csv("/Iris.csv")
X = dataset.iloc[:, :-1].values
y = dataset.iloc[:, -1].values
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size =
0.25, random_state = 0)
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
from sklearn.naive_bayes import GaussianNB
classifier = GaussianNB()
classifier.fit(X_train, y_train)
GaussianNB(priors=None, var_smoothing=1e-09)
from sklearn.metrics import confusion_matrix, accuracy_score
y_pred = classifier.predict(X_test)
cm = confusion_matrix(y_test, y_pred)
print(cm)
accuracy_score(y_test, y_pred)
```

OUTPUT:

```
[[114   2]
 [  2  53]]
0.9766081871345029
```

EXPERIMENT:6(C)LOGISTIC

AIM: : finding accuracy value of iris data set using LOGISTIC REGRESSION algorithm

PROGRAM:

```
import numpy as np
import pandas as pd
dataset = pd.read_csv("/Iris.csv")
X = dataset.iloc[:, :-1].values
y = dataset.iloc[:, -1].values
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size =
0.30, random_state = 2)
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
from sklearn.linear_model import LogisticRegression
classifier = LogisticRegression(random_state = 0)
classifier.fit(X_train, y_train)
LogisticRegression(C=1.0, class_weight=None, dual=False,
fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='warn', n_jobs=None, penalty='l2',
                   random_state=0, solver='warn', tol=0.0001,
verbose=0,
                   warm_start=False)
from sklearn.metrics import confusion_matrix, accuracy_score
y_pred = classifier.predict(X_test)
cm = confusion_matrix(y_test, y_pred)
print(cm)
accuracy_score(y_test, y_pred)
```

OUTPUT:

```
[[117   8]
 [  6  74]]
0.9317073170731708
```

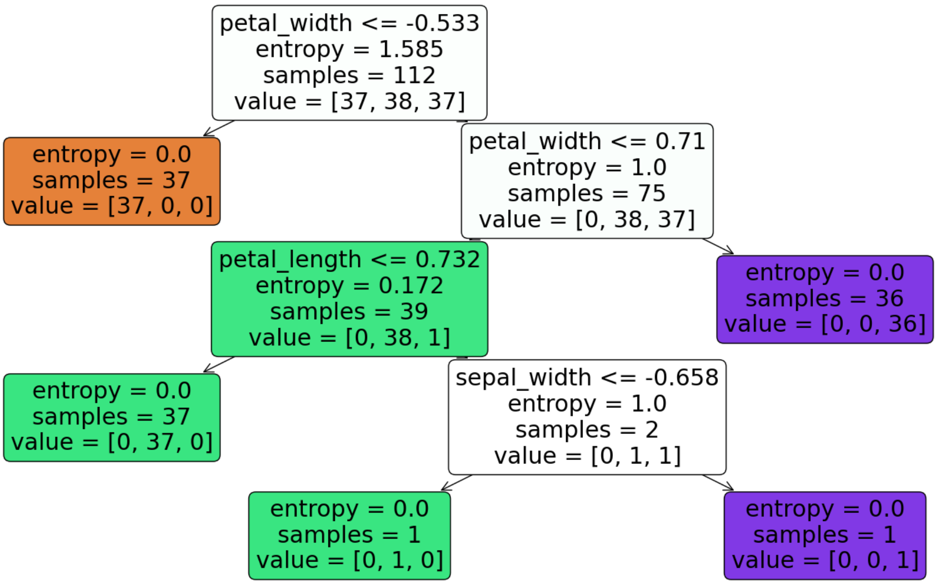
EXPERIMENT:6(D)DECISION

AIM: : finding accuracy value of iris data set using DECISION TREE algorithm

PROGRAM:

```
import numpy as np
import pandas as pd
dataset = pd.read_csv("/Iris.csv")
X = dataset.iloc[:, :-1].values
y = dataset.iloc[:, -1].values
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size =
0.25, random_state = 8)
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
from sklearn.tree import DecisionTreeClassifier
classifier = DecisionTreeClassifier(criterion = 'entropy', random_state
= 5)
classifier.fit(X_train, y_train)
from sklearn.tree import plot_tree
import matplotlib.pyplot as plt
plt.figure(figsize=(20,10))
plot_tree(classifier, filled=True, rounded=True,
feature_names=dataset.columns[:-1])
plt.show()
y_pred = classifier.predict(X_test)
from sklearn.metrics import confusion_matrix, accuracy_score
cm = confusion_matrix(y_test, y_pred)
print(cm)
accuracy_score(y_test, y_pred)
```

Output:



```
[[13  0  0]
 [ 0 12  0]
 [ 0  0 13]]
1.0
```

EXPERIMENT:6(E)SVM

AIM: : finding accuracy value of iris data set using SVM algorithm

PROGRAM:

```
import numpy as np
import pandas as pd
dataset = pd.read_csv("/Iris.csv")
X = dataset.iloc[:, :-1].values
y = dataset.iloc[:, -1].values
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.25, random_state=32)
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
from sklearn.svm import SVC
classifier = SVC(kernel='linear', random_state=0)
classifier.fit(X_train, y_train)
y_pred = classifier.predict(X_test)
from sklearn.metrics import confusion_matrix, accuracy_score
cm = confusion_matrix(y_test, y_pred)
print(cm)
print('Accuracy: {:.2f}%'.format(accuracy_score(y_test, y_pred) * 100))
```

OUTPUT:

```
[[108   1]
 [  5  57]]
Accuracy: 96.49%
```

EXPERIMENT:6(F)RANDOM

AIM: : finding accuracy value of iris data set using RANDOM FOREST algorithm

PROGRAM:

```
import numpy as np
import pandas as pd
dataset = pd.read_csv("/Iris.csv")
X = dataset.iloc[:, :-1].values
y = dataset.iloc[:, -1].values
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.25, random_state=39)
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
from sklearn.ensemble import RandomForestClassifier
classifier = RandomForestClassifier(n_estimators=100, random_state=42)
classifier.fit(X_train, y_train)
y_pred = classifier.predict(X_test)
from sklearn.metrics import confusion_matrix, accuracy_score
cm = confusion_matrix(y_test, y_pred)
print(cm)
print('Accuracy:', accuracy_score(y_test, y_pred))
```

OUTPUT:

```
[[111   1]
 [  2  57]]
Accuracy: 0.98245614
```

EXPERIMENT:7(A)

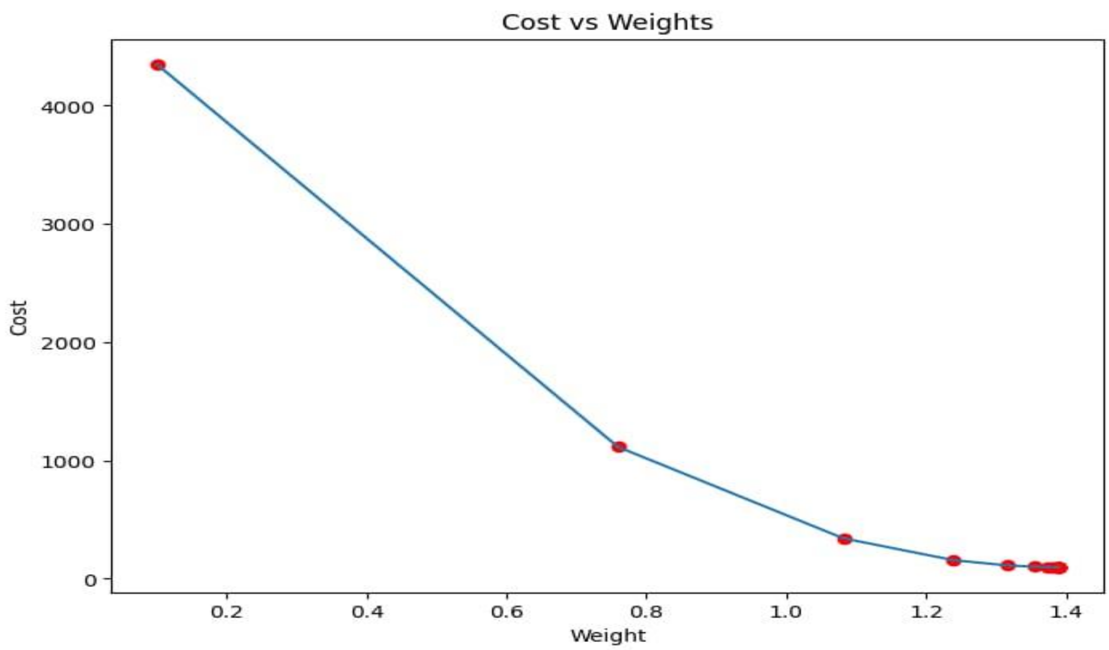
AIM: To demonstrate gradient descent using python(actual data)

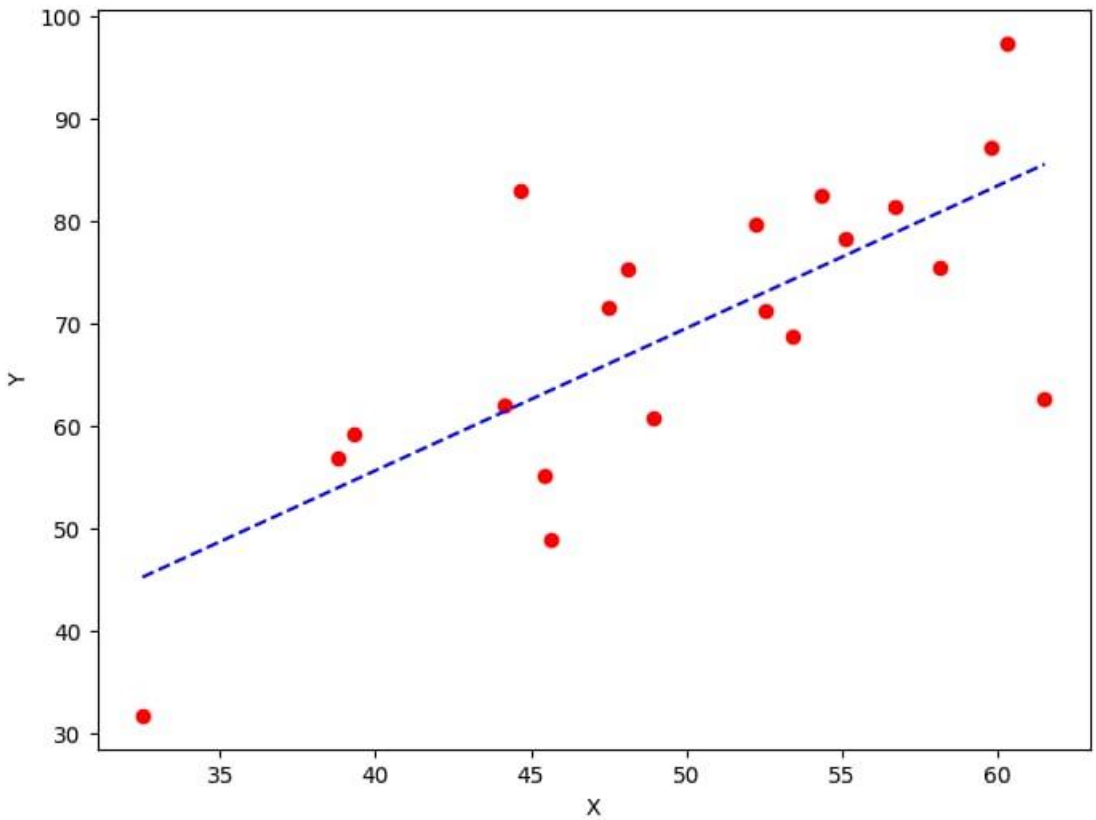
PROGRAM:

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

def mean_squared_error(y_true, y_predicted):
    cost = np.sum((y_true - y_predicted)**2) / len(y_true)
    return cost

def gradient_descent(x, y, iterations=1000, learning_rate=0.0001,
stopping_threshold=1e-6):
    current_weight = 0.1
    current_bias = 0.01
    n = float(len(x))
    costs = []
    weights = []
    previous_cost = None
    for i in range(iterations):
        y_predicted = (current_weight * x) + current_bias
        current_cost = mean_squared_error(y, y_predicted)
        if previous_cost and abs(previous_cost - current_cost) <=
stopping_threshold:
            break
        previous_cost = current_cost
        costs.append(current_cost)
        weights.append(current_weight)
        weight_derivative = -(2/n) * sum(x * (y - y_predicted))
        bias_derivative = -(2/n) * sum(y - y_predicted)
        current_weight -= (learning_rate * weight_derivative)
        current_bias -= (learning_rate * bias_derivative)
        if i % 100 == 0:
            print(f"Iteration {i+1}: Cost {current_cost}, Weight
{current_weight}, Bias {current_bias}", end=", ")
    plt.figure(figsize=(8, 6))
    plt.plot(weights, costs)
    plt.scatter(weights, costs, marker='o', color='red')
    plt.title("Cost vs Weights")
    plt.ylabel("Cost")
    plt.xlabel("Weight")
    plt.show()
    return current_weight, current_bias

def main():
```

```
    X = np.array([32.50234527, 53.42680403, 61.53035803, 47.47563963,
59.81320787, 55.14218841, 52.21179669, 39.29956669, 48.10504169,
52.55001444, 45.41973014, 54.35163488, 44.1640495, 58.16847072,
56.72720806, 48.95588857, 44.68719623, 60.29732685, 45.61864377,
38.81681754])
    Y = np.array([31.70700585, 68.77759598, 62.5623823, 71.54663223,
87.23092513, 78.21151827, 79.64197305, 59.17148932, 75.3312423,
71.30087989, 55.16567715, 82.47884676, 62.00892325, 75.39287043,
81.43619216, 60.72360244, 82.89250373, 97.37989686, 48.84715332,
56.87721319])
    estimated_weight, estimated_bias = gradient_descent(X, Y,
iterations=2000)
    print(f"Estimated Weight: {estimated_weight}, Estimated Bias:
{estimated_bias}")
    Y_pred = estimated_weight * X + estimated_bias
    plt.figure(figsize=(8, 6))
    plt.scatter(X, Y, marker='o', color='red')
    plt.plot([min(X), max(X)], [min(Y_pred), max(Y_pred)],
color='blue', markersize=10, linestyle='dashed')
    plt.xlabel("X")
    plt.ylabel("Y")
    plt.show()

if __name__ == "__main__":
    main()
```

output:




100
90
80
70
60
50
40
30
Y
35
40
45
50
55
60
X

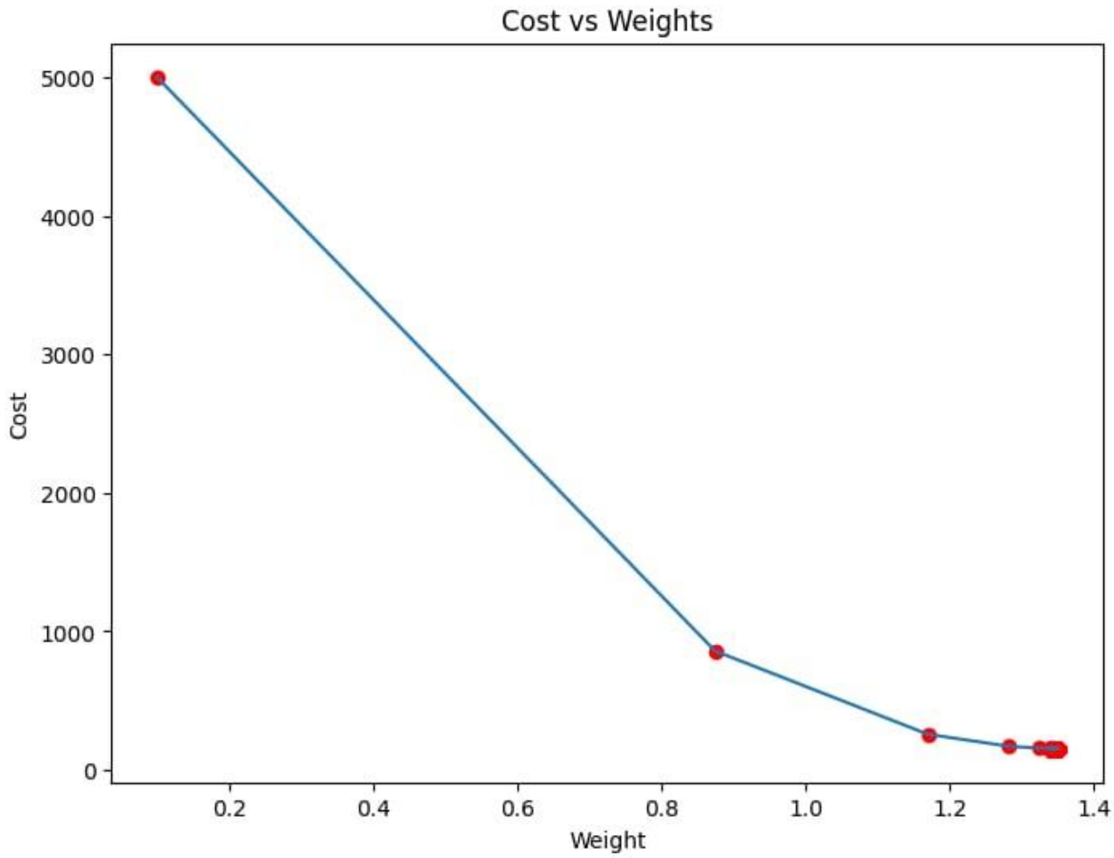
Experiment:7(b)

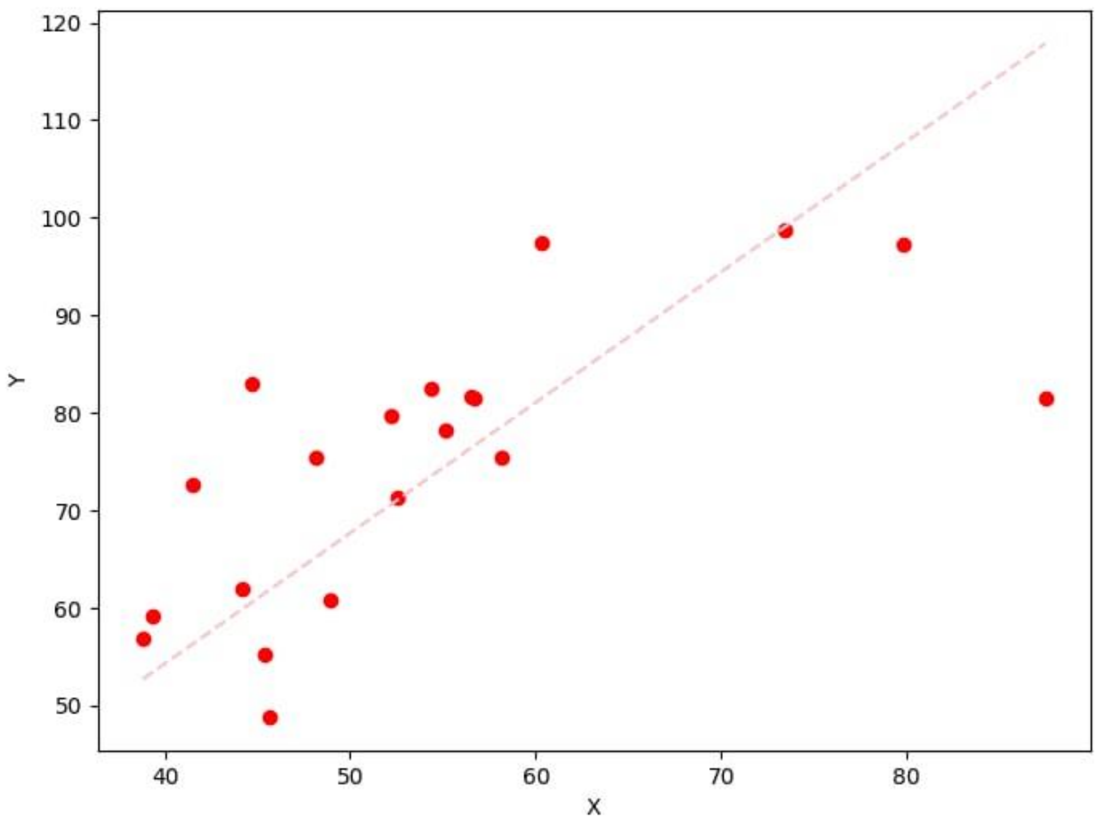
AIM: To demonstrate gradient descent using python (modified data)

PROGRAM:

```
import numpy as np
import matplotlib.pyplot as plt
def mean_squared_error(y_true, y_predicted):
    cost = np.sum((y_true - y_predicted)**2) / len(y_true)
    return cost
def gradient_descent(x, y, iterations=1000, learning_rate=0.0001,
stopping_threshold=1e-6):
    current_weight = 0.1
    current_bias = 0.01
    n = float(len(x))
    costs = []
    weights = []
    previous_cost = None
    for i in range(iterations):
        y_predicted = (current_weight * x) + current_bias
        current_cost = mean_squared_error(y, y_predicted)
        if previous_cost and abs(previous_cost - current_cost) <=
stopping_threshold:
            break
        previous_cost = current_cost
        costs.append(current_cost)
        weights.append(current_weight)
        weight_derivative = -(2/n) * sum(x * (y - y_predicted))
        bias_derivative = -(2/n) * sum(y - y_predicted)
        current_weight -= (learning_rate * weight_derivative)
        current_bias -= (learning_rate * bias_derivative)
        if i % 100 == 0:
            print(f"Iteration {i+1}: Cost={current_cost},
Weight={current_weight}, Bias={current_bias}")
    plt.figure(figsize=(8, 6))
    plt.plot(weights, costs)
    plt.scatter(weights, costs, marker='o', color='red')
    plt.title("Cost vs Weights")
    plt.ylabel("Cost")
    plt.xlabel("Weight")
    plt.show()
    return current_weight, current_bias
def main():
    X = np.array([52.50234527, 63.42680403, 81.53035803, 47.47563963,
89.81320787,
                  55.14218841, 52.21179669, 39.29956669, 48.10504169,
52.55001444,
```

```
                  45.41973014, 54.35163488, 44.1640495 , 58.16847072,
56.72720806,
                  48.95588857, 44.68719623, 60.29732685, 45.61864377,
38.81681754])

    Y = np.array([41.70700585, 78.77759598, 82.5623823 , 91.54663223,
77.23092513,
                  78.21151827, 79.64197305, 59.17148932, 75.3312423 ,
71.30087989,
                  55.16567715, 82.47884676, 62.00892325, 75.39287043,
81.43619216,
                  60.72360244, 82.89250373, 97.37989686, 48.84715332,
56.87721319])
    estimated_weight, estimated_bias = gradient_descent(X, Y,
iterations=2000)
    print(f"Estimated Weight: {estimated_weight}\nEstimated Bias:
{estimated_bias}")
    Y_pred = estimated_weight * X + estimated_bias
    plt.figure(figsize=(8, 6))
    plt.scatter(X, Y, color='orange')
    plt.plot([min(X), max(X)], [min(Y_pred), max(Y_pred)],
color='blue', linestyle='dashed', linewidth=2)
    plt.xlabel("X")
    plt.ylabel("Y")
    plt.title("Linear Regression Line")
    plt.show()
if __name__ == "__main__":
    main()
```


Cost vs Weights
5000
4000
3000
2000
1000
0
Cost
0.2
0.4
0.6
0.8
1.0
1.2
1.4
Weight


120
110
100
90
80
70
60
50
Y
40
50
60
70
80
X

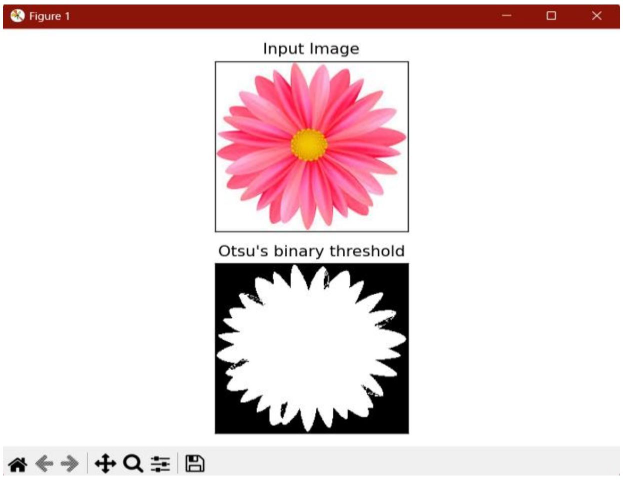
EXPERIMENT:8(A)SEGMENTATION

AIM: : Verifying the performance of a image processing by using choosen database with phython code

PROGRAM:

```
import numpy as np
import cv2
from matplotlib import pyplot as plt
img = cv2.imread(r'C:\Users\Saaniya\Pictures\Screenshots\.4)
flower.jpg')
b,g,r = cv2.split(img)
rgb_img = cv2.merge([r,g,b])
gray = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
ret, thresh =
cv2.threshold(gray,0,255,cv2.THRESH_BINARY_INV+cv2.THRESH_OTSU)
# noise removal
kernel = np.ones((2,2),np.uint8)
#opening = cv2.morphologyEx(thresh,cv2.MORPH_OPEN,kernel, iterations =
2)
closing = cv2.morphologyEx(thresh,cv2.MORPH_CLOSE,kernel, iterations =
2)
# sure background area
sure_bg = cv2.dilate(closing,kernel,iterations=3)
# Finding sure foreground area
dist_transform = cv2.distanceTransform(sure_bg,cv2.DIST_L2,3)
# Threshold
ret, sure_fg =
cv2.threshold(dist_transform,0.1*dist_transform.max(),255,0)
# Finding unknown region
sure_fg = np.uint8(sure_fg)
unknown = cv2.subtract(sure_bg,sure_fg)
# Marker labelling
ret, markers = cv2.connectedComponents(sure_fg)
# Add one to all labels so that sure background is not 0, but 1
markers = markers+1
markers[unknown==255] = 0
markers = cv2.watershed(img,markers)
img[markers == -1] = [255,0,0]
plt.subplot(211),plt.imshow(rgb_img)
plt.title('Input Image'), plt.xticks([]), plt.yticks([])
plt.subplot(212),plt.imshow(thresh, 'gray')
plt.imsave(r'thresh.png',thresh)
plt.title("Otsu's binary threshold"), plt.xticks([]), plt.yticks([])
plt.tight_layout()
plt.show()
```

OUTPUT:



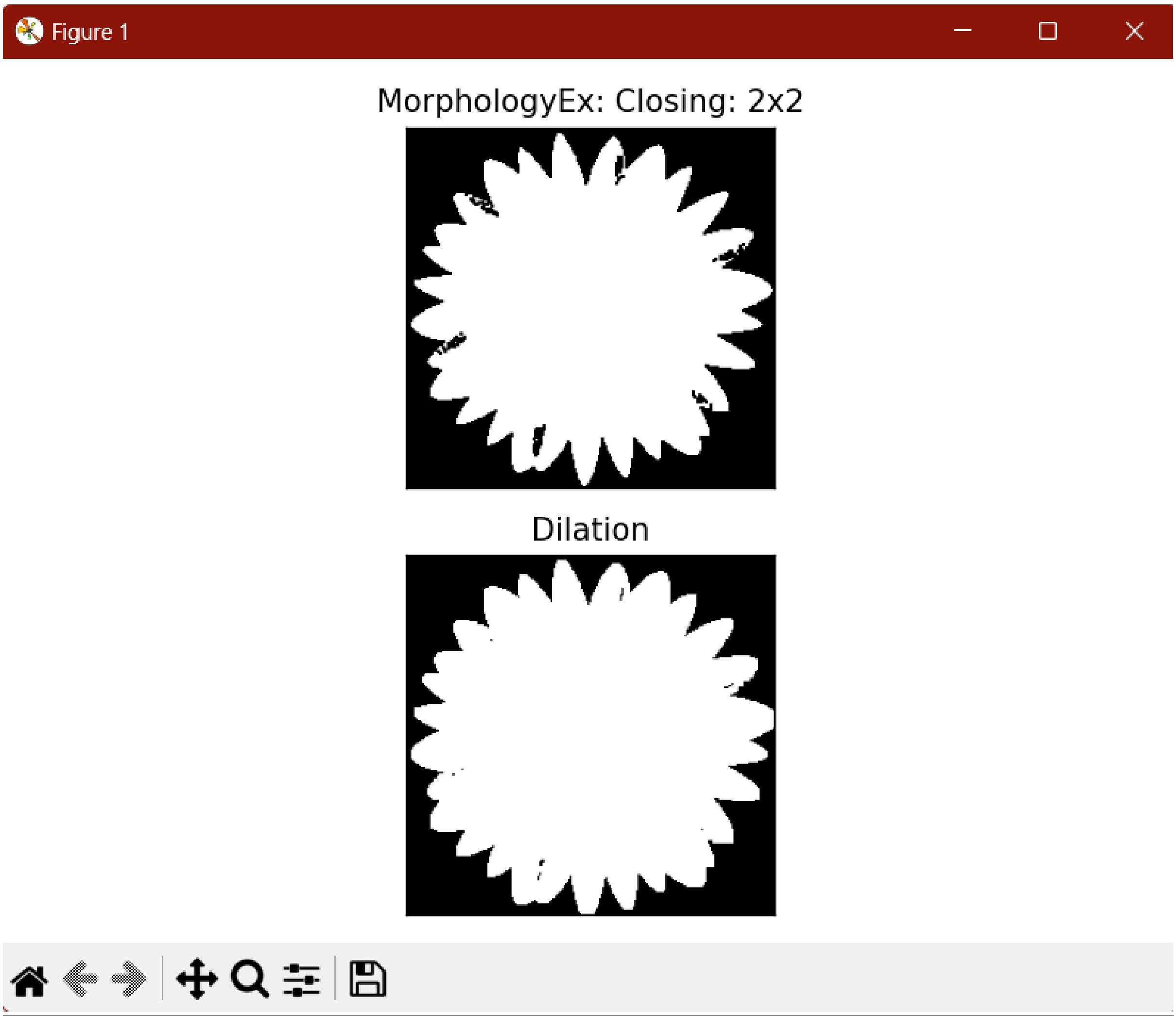
EXPERIMENT:8(B)

AIM: Verifying the performance of an image processing by using water shed database with python code

PROGRAM:

```
import numpy as np
import cv2
from matplotlib import pyplot as plt
img = cv2.imread(r'C:\Users\Saaniya\Pictures\Screenshots\.4)
flower.jpg')
b, g, r = cv2.split(img)
rgb_img = cv2.merge([r, g, b])
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
ret, thresh = cv2.threshold(gray, 0, 255, cv2.THRESH_BINARY_INV +
cv2.THRESH_OTSU)
kernel = np.ones((2, 2), np.uint8)
closing = cv2.morphologyEx(thresh, cv2.MORPH_CLOSE, kernel)
sure_bg = cv2.dilate(closing, kernel, iterations=3)
plt.subplot(211), plt.imshow(closing, 'gray')
plt.title("MorphologyEx: Closing: 2x2"), plt.xticks([]), plt.yticks([])
plt.subplot(212), plt.imshow(sure_bg, 'gray')
plt.title("Dilation"), plt.xticks([]), plt.yticks([])
plt.imsave(r'dilation.png', sure_bg)
plt.tight_layout()
plt.show()
```

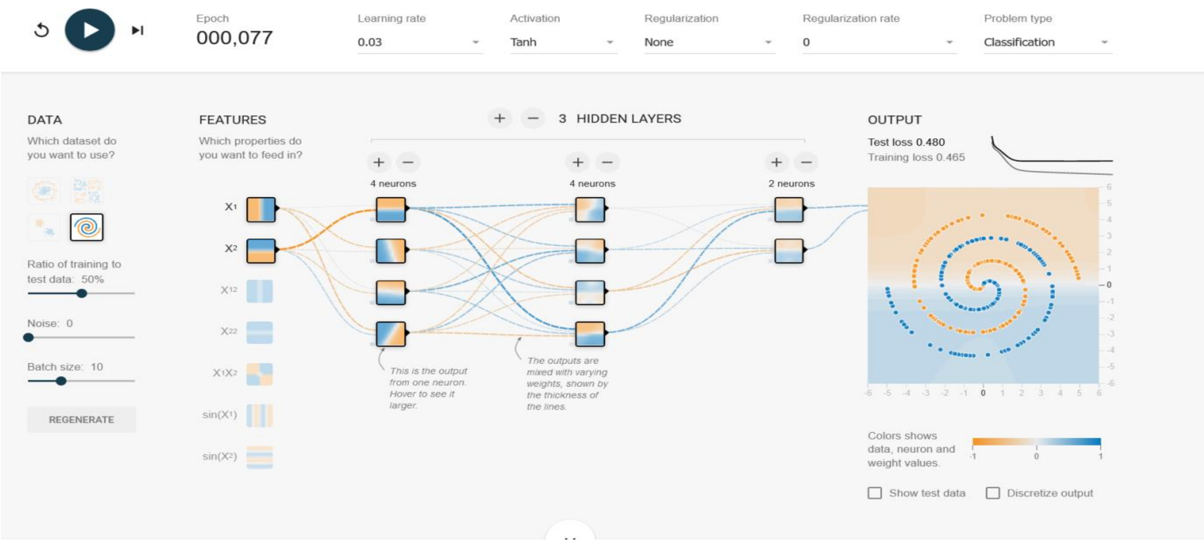
OUTPUT:



EXPERIMENT:9 (a) TANH

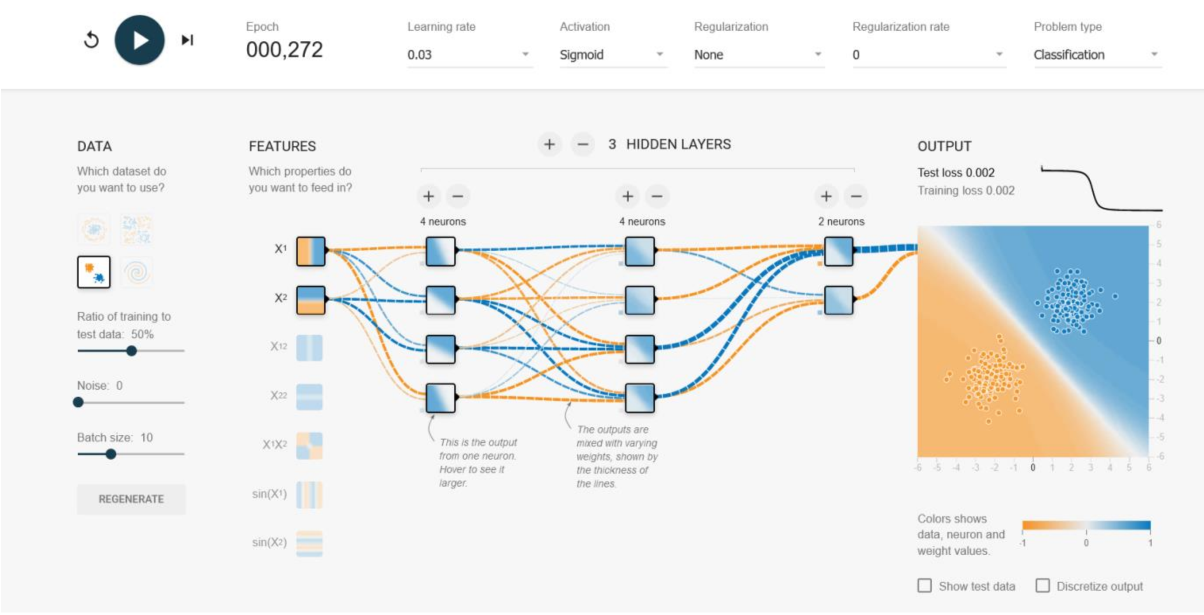
AIM: Neural network analysis using TANH activation

OUTPUT:



EXPERIMENT:9(B) SIGMIOD

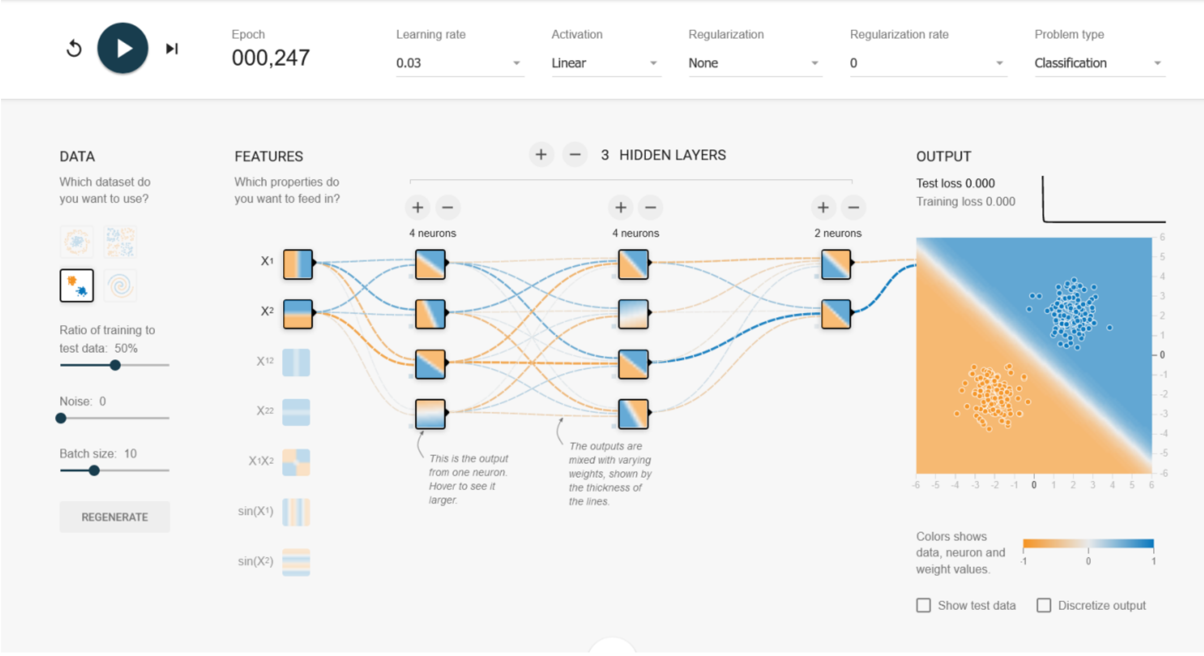
AIM: Neural network analysis using SIGMOID activation



EXPERIMENT:9(C) LINEAR

AIM: Neural network analysis using LINEAR activation

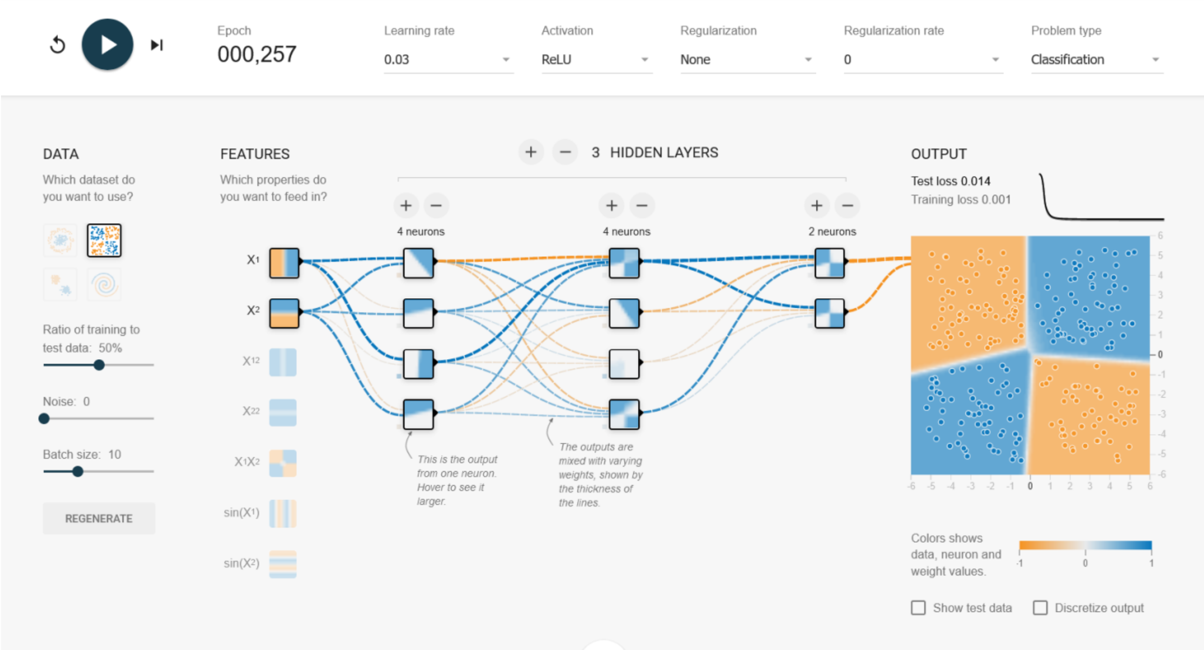
OUTPUT:



EXPERIMENT:9(D)RELU

AIM: Neural network analysis using ReLU activation

OUTPUT:



EXPERIMENT:10

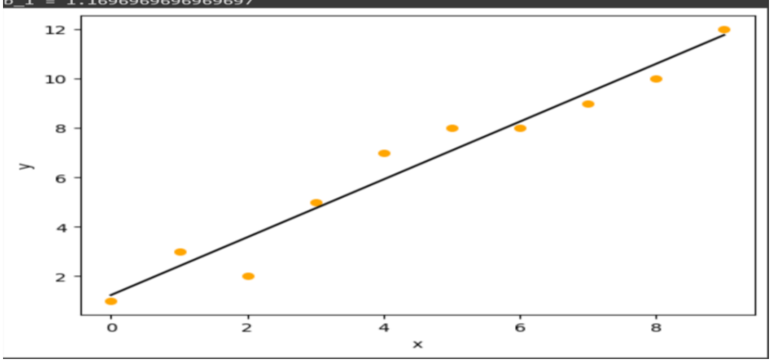
AIM: To demonstrate linear separability using python code

PROGRAM:

```
import numpy as np
import matplotlib.pyplot as plt
def estimate_coef(x, y):
    n = np.size(x)
    m_x = np.mean(x)
    m_y = np.mean(y)
    SS_xy = np.sum(y*x) - n*m_y*m_x
    SS_xx = np.sum(x*x) - n*m_x*m_x
    b_1 = SS_xy / SS_xx
    b_0 = m_y - b_1*m_x
    return (b_0, b_1)
def plot_regression_line(x, y, b):
    plt.scatter(x, y, color = "r",
                marker = "o", s = 30)
    y_pred = b[0] + b[1]*x
    plt.plot(x, y_pred, color = "b")
    plt.xlabel('x')
    plt.ylabel('y')
    plt.show()
def main():
    x = np.array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
    y = np.array([1, 3, 2, 5, 7, 8, 8, 9, 10, 12])
    b = estimate_coef(x, y)
    print("Estimated coefficients:\nb_0 = {}  \
          \nb_1 = {}".format(b[0], b[1]))
    plot_regression_line(x, y, b)
if __name__ == "__main__":
    main()
```

OUTPUT:

```
Estimated coefficients:
b_0 = 1.2363636363636363
b_1 = 1.1696969696969697
```



```python
import numpy as np

import pandas as pd

from sklearn.model_selection import train_test_split

from sklearn.preprocessing import StandardScaler

from sklearn.metrics import confusion_matrix, accuracy_score

# Load dataset

dataset = pd.read_csv("/Iris.csv")

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

y = dataset.iloc[:, -1].values

# Split dataset

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20, random_state=42)

# Standardize features

sc = StandardScaler()

X_train = sc.fit_transform(X_train)

X_test = sc.transform(X_test)

# Function to train models and evaluate performance

def evaluate_model(model_name, classifier):

    classifier.fit(X_train, y_train)

    y_pred = classifier.predict(X_test)

    cm = confusion_matrix(y_test, y_pred)

    accuracy = accuracy_score(y_test, y_pred)

    print(f"{model_name} Confusion Matrix:\n{cm}")

    print(f"{model_name} Accuracy: {accuracy:.2f}\n")

# K-Nearest Neighbors (KNN)

from sklearn.neighbors import KNeighborsClassifier

knn_classifier = KNeighborsClassifier(n_neighbors=5, metric='minkowski', p=2)

evaluate_model("KNN", knn_classifier)

# Naive Bayes

from sklearn.naive_bayes import GaussianNB

nb_classifier = GaussianNB()

evaluate_model("Naive Bayes", nb_classifier)
```

```python
# Logistic Regression

from sklearn.linear_model import LogisticRegression

lr_classifier = LogisticRegression(random_state=42)

evaluate_model("Logistic Regression", lr_classifier)

# Decision Tree

from sklearn.tree import DecisionTreeClassifier

dt_classifier = DecisionTreeClassifier(criterion='entropy', random_state=42)

evaluate_model("Decision Tree", dt_classifier)

# Support Vector Machine (SVM)

from sklearn.svm import SVC

svm_classifier = SVC(kernel='linear', random_state=42)

evaluate_model("SVM", svm_classifier)

# Random Forest

from sklearn.ensemble import RandomForestClassifier

rf_classifier = RandomForestClassifier(n_estimators=10, criterion='entropy', random_state=42)

evaluate_model("Random Forest", rf_classifier)
```