**CSA4705: DEEP LEARNING for DATA HANDLING**

**Students Reg No:192124148**

**Student’s Name: Subak Kumar S**

**Experiment 1: Linear Regression**

**Aim**: Verifying the performance of linear severability using chosen database with Python code.

**Work implemented** :

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 = "b",

marker = "o", s = 30)

y\_pred = b[0] + b[1]\*x

plt.plot(x, y\_pred, color = "y")

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 = {} \

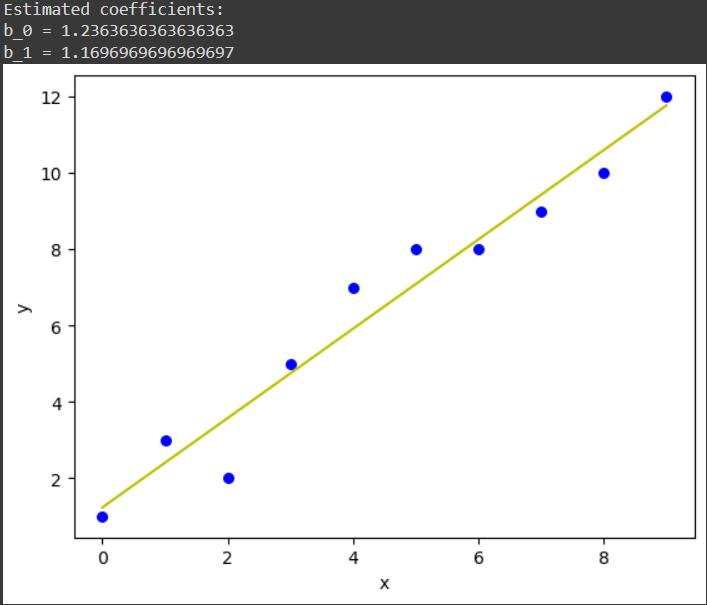
\nb\_1 = {}".format(b[0], b[1]))

plot\_regression\_line(x, y, b)

if \_\_name\_\_ == "\_\_main\_\_":

main()

**Result:**



**Conclusion:** Hence, using python coding we have successfully implemented Linear regression and verified.

**Experiment 2: Logistic regression**

**Aim:** Verifying the performance of logistic regression using chosen database with Python code.

**Work implemented:**

import numpy as np

import matplotlib.pyplot as plt

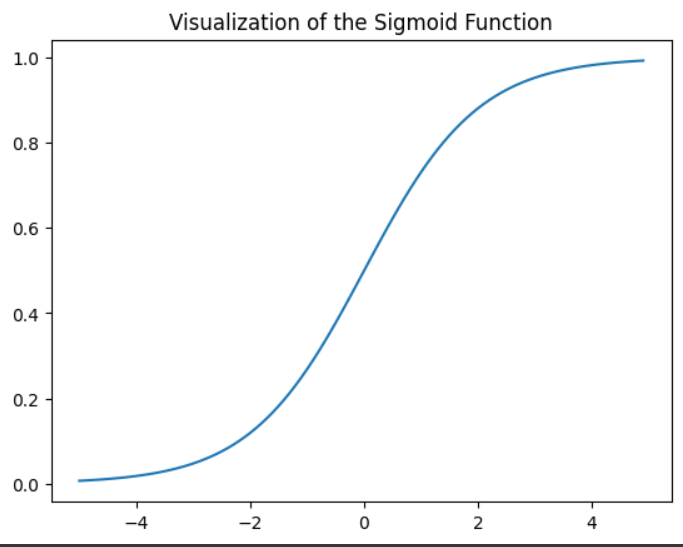
def sigmoid(z):

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()

**Result:** 

**Conclusion:** Hence, implementation of Logistic regression has been successfully implemented using python code and verified.

**Experiment 3: Confusion Matrix - 1**

**Aim:** Verifying the performance of confusion matrix eg.1 using chosen database with Python code.

**Work Implemented:**

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',

xticklabels=['Dog','Not Dog'],

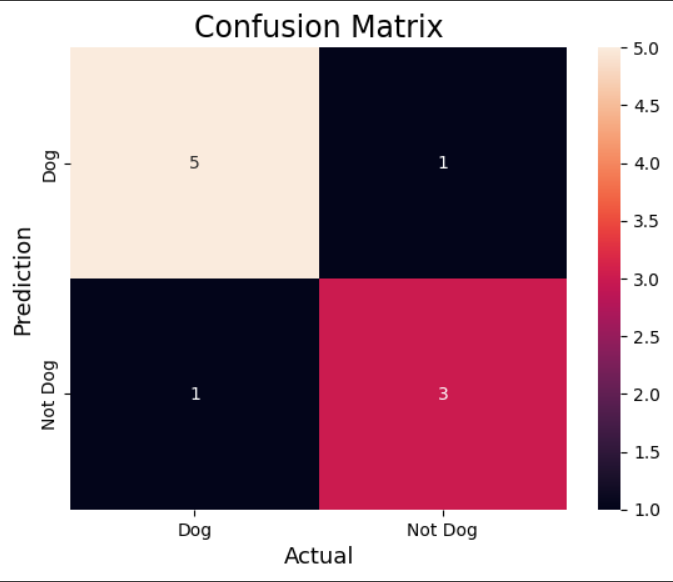
yticklabels=['Dog','Not Dog'])

plt.ylabel('Prediction',fontsize=13)

plt.xlabel('Actual',fontsize=13)

plt.title('Confusion Matrix',fontsize=17)

plt.show()

**Result:** 

**Conclusion:** Hence, implementation of Confusion matrix has been successfully implemented using python code and verified.

**Experiment 4: Confusion matrix – 2**

**Aim:** Verifying the performance of confusion matrix eg.2 using chosen database with Python code.

**Work Implemented :**

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

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)

tree = DecisionTreeClassifier(random\_state=23)

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

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

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

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()

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

print("Accuracy :", accuracy)

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

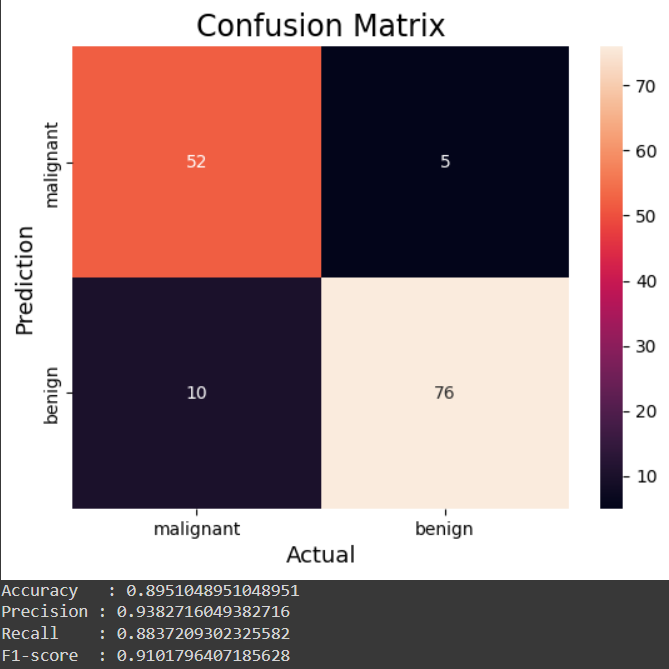
print("Precision :", precision)

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

print("Recall :", recall)

F1\_score = f1\_score(y\_test, y\_pred)

print("F1-score :", F1\_score)

**Result:** 

**Conclusion:** Hence, Confusion matrix has been successfully implemented using python code and verified.

**Experiment 5: Confusion matrix – 3**

**Aim:** Verifying the performance of confusion matrix eg.3 using chosen database with Python code.

**Work Implemented:**

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

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)

clf = RandomForestClassifier(random\_state=23)

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

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

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

sns.heatmap(cm,

annot=True,

fmt='g')

plt.ylabel('Prediction',fontsize=13)

plt.xlabel('Actual',fontsize=13)

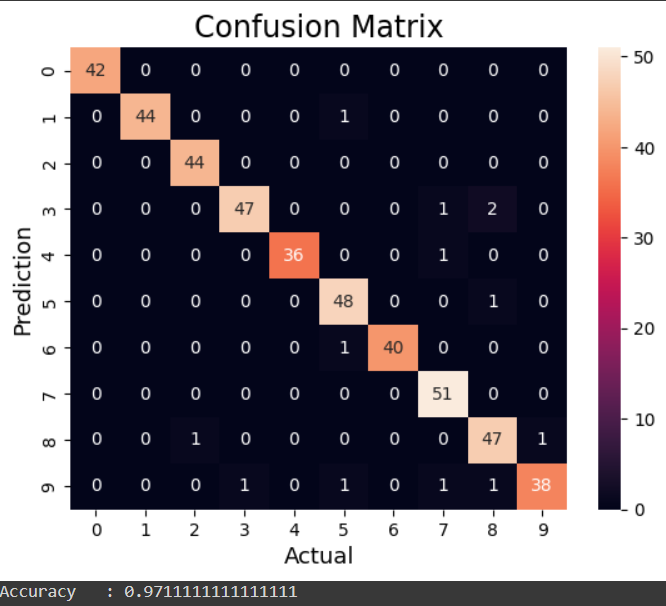
plt.title('Confusion Matrix',fontsize=17)

plt.show()

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

print("Accuracy :", accuracy)

**Result:**



**Conclusion:** Hence, Confusion matrix for the given database has been successfully implemented using python code and verified.

**Experiment 6: Build an Artificial Neural Network by implementing the Backpropagation algorithm and test the same using appropriate data sets.**

**Aim:** To build a ANN by Backpropagation Algorithm and test the same using datasets.

**Work Implemented:**

import numpy as np

X = np.array(([2, 9], [1, 5], [3, 6]), dtype=float)

y = np.array(([92], [86], [89]), dtype=float)

X = X/np.amax(X,axis=0)

y = y/100

def sigmoid (x):

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

def derivatives\_sigmoid(x):

return x \* (1 - x)

epoch=5

lr=0.1

inputlayer\_neurons = 2

hiddenlayer\_neurons = 3

output\_neurons = 1

wh=np.random.uniform(size=(inputlayer\_neurons,hiddenlayer\_neurons))

bh=np.random.uniform(size=(1,hiddenlayer\_neurons))

wout=np.random.uniform(size=(hiddenlayer\_neurons,output\_neurons))

bout=np.random.uniform(size=(1,output\_neurons))

for i in range(epoch):

hinp1=np.dot(X,wh)

hinp=hinp1 + bh

hlayer\_act = sigmoid(hinp)

outinp1=np.dot(hlayer\_act,wout)

outinp= outinp1+bout

output = sigmoid(outinp)

EO = y-output

outgrad = derivatives\_sigmoid(output)

d\_output = EO \* outgrad

EH = d\_output.dot(wout.T)

hiddengrad = derivatives\_sigmoid(hlayer\_act)

d\_hiddenlayer = EH \* hiddengrad

wout += hlayer\_act.T.dot(d\_output) \*lr

wh += X.T.dot(d\_hiddenlayer) \*lr

print ("-----------Epoch-", i+1, "Starts----------")

print("Input: \n" + str(X))

print("Actual Output: \n" + str(y))

print("Predicted Output: \n" ,output)

print ("-----------Epoch-", i+1, "Ends----------\n")

print("Input: \n" + str(X))

print("Actual Output: \n" + str(y))

print("Predicted Output: \n" ,output)

**Result:** -----------Epoch- 1 Starts----------

Input:

[[0.66666667 1. ]

[0.33333333 0.55555556]

[1. 0.66666667]]

Actual Output:

[[0.92]

[0.86]

[0.89]]

Predicted Output:

[[0.89883887]

[0.89030025]

[0.89760015]]

-----------Epoch- 1 Ends----------

-----------Epoch- 2 Starts----------

Input:

[[0.66666667 1. ]

[0.33333333 0.55555556]

[1. 0.66666667]]

Actual Output:

[[0.92]

[0.86]

[0.89]]

Predicted Output:

[[0.89881282]

[0.89027376]

[0.89757394]]

-----------Epoch- 2 Ends----------

-----------Epoch- 3 Starts----------

Input:

[[0.66666667 1. ]

[0.33333333 0.55555556]

[1. 0.66666667]]

Actual Output:

[[0.92]

[0.86]

[0.89]]

Predicted Output:

[[0.89878688]

[0.89024739]

[0.89754785]]

-----------Epoch- 3 Ends----------

-----------Epoch- 4 Starts----------

Input:

[[0.66666667 1. ]

[0.33333333 0.55555556]

[1. 0.66666667]]

Actual Output:

[[0.92]

[0.86]

[0.89]]

Predicted Output:

[[0.89876106]

[0.89022112]

[0.89752186]]

-----------Epoch- 4 Ends----------

-----------Epoch- 5 Starts----------

Input:

[[0.66666667 1. ]

[0.33333333 0.55555556]

[1. 0.66666667]]

Actual Output:

[[0.92]

[0.86]

[0.89]]

Predicted Output:

[[0.89873534]

[0.89019497]

[0.89749598]]

-----------Epoch- 5 Ends----------

Input:

[[0.66666667 1. ]

[0.33333333 0.55555556]

[1. 0.66666667]]

Actual Output:

[[0.92]

[0.86]

[0.89]]

Predicted Output:

[[0.89873534]

[0.89019497]

[0.89749598]]

**Conclusion:** Hence, using Backpropagation ANN has been successfully implemented using python code and verified.

**Experiment 7: Write a program for Implementation of K-Nearest Neighbors (K-NN) in Python**.

**Aim:** To implement K-Nearest Neighbours using Python code.

**Work Implemented:**

import numpy as np

import pandas as pd

dataset = pd.read\_csv("/content/drive/MyDrive/DATASET/breastcancer.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 = 0.25, random\_state = 51)

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)

**Result:**

[[105 1]

[ 2 63]]

0.9824561403508771

**Conclusion:** Hence, implementation of KNN has been successfully implemented using python code and verified.

**Experiment 8: Write a program to implement Naïve Bayes algorithm in python and to display the results using confusion matrix and accuracy. Java/Python ML library classes can be used for this problem.**

**Aim:** To write a program in python to implement Naïve Bayes Algorithm in Python and display result using confusion matrix and accuracy.

**Work Implemented:**

import numpy as np

import pandas as pd

dataset = pd.read\_csv("/content/drive/MyDrive/DATASET/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 = 55)

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)

**Result:**

[[13 0 0]

[ 0 11 1]

[ 0 0 13]]

0.9736842105263158

**Conclusion:** Hence, using Naive Bayes algorithm for finding confusion matrix and accuracy in python code is successfully implemented and verified.

**Experiment 9: Write a program to implement Linear Regression (LR) algorithm in python**

**Aim:** To implement Linear Regression using python code.

**Work Implemented:**

import numpy as np

import matplotlib.pyplot as plt

import pandas as pd

dataset = pd.read\_csv('/content/drive/MyDrive/DATASET/Position\_Salaries.csv')

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

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

from sklearn.linear\_model import LinearRegression

lin\_reg = LinearRegression()

lin\_reg.fit(X, y)

LinearRegression(copy\_X=True,fit\_intercept=True,n\_jobs=None)

from sklearn.preprocessing import PolynomialFeatures

poly\_reg = PolynomialFeatures(degree = 4)

X\_poly = poly\_reg.fit\_transform(X)

lin\_reg\_2 = LinearRegression()

lin\_reg\_2.fit(X\_poly, y)

LinearRegression(copy\_X=True,fit\_intercept=True,n\_jobs=None)

plt.scatter(X, y, color = 'cyan')

plt.plot(X, lin\_reg.predict(X), color = 'blue')

plt.title('Truth or Bluff (Linear Regression)')

plt.xlabel('Position Level')

plt.ylabel('Salary')

plt.show()

plt.scatter(X, y, color = 'yellow')

plt.plot(X, lin\_reg\_2.predict(poly\_reg.fit\_transform(X)), color = 'blue')

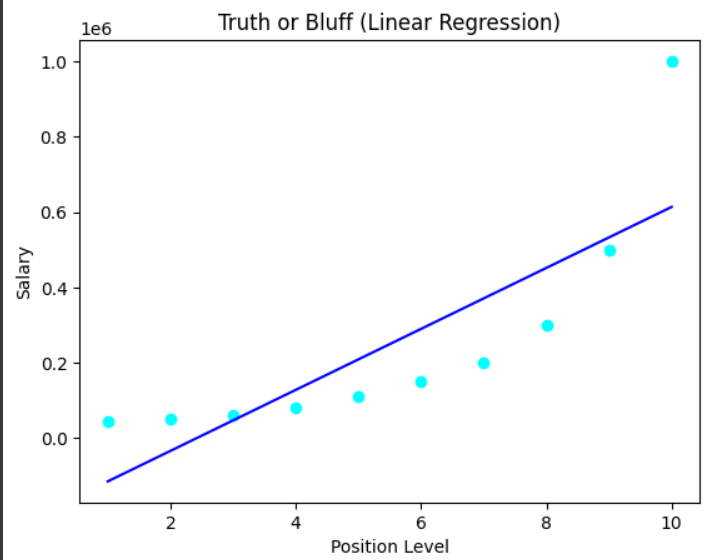
plt.title('Truth or Bluff (Polynomial Regression)')

plt.xlabel('Position level')

plt.ylabel('Salary')

plt.show()

**Result:**



A picture containing text, diagram, line, plot

Description automatically generated

**Conclusion:** Hence, implementation of linear regression(LR) algorithm in python is verified.

**Experiment 10: Write a program to implement Logistic Regression (LR) algorithm in python.**

**Aim:** To write a program to implement Logistic Regression algorithm in Python.

**Work Implemented :**

import numpy as np

import pandas as pd

dataset = pd.read\_csv("/content/drive/MyDrive/DATASET/breastcancer.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)

**Result:**

[[145 3]

[ 5 87]]

0.9666666666666667

**Conclusion:** Hence, Logistic Regression algorithm is successfully implemented and verified.

**Experiment 11: OVERFITTING/UNDERFITTING**

**Aim:** To implement and verify Overfitting and underfitting algorithms using python code.

**Work Implemented:**

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)

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()

**Result:**

A picture containing line, plot, diagram, screenshot

Description automatically generated

**Conclusion:** Hence, using python code we have successfully implemented Overfitting and Underfitting mode and verified.

**Experiment 12: Gradiant Descent.**

**Aim:** To implement and verify Gradiant Descent using python code.

**Work Implemented:**

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

iterations = iterations

learning\_rate = learning\_rate

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 = current\_weight - (learning\_rate \* weight\_derivative)

current\_bias = current\_bias - (learning\_rate \* bias\_derivative)

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([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}\nEstimated 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',markerfacecolor='red',

markersize=10,linestyle='dashed')

plt.xlabel("X")

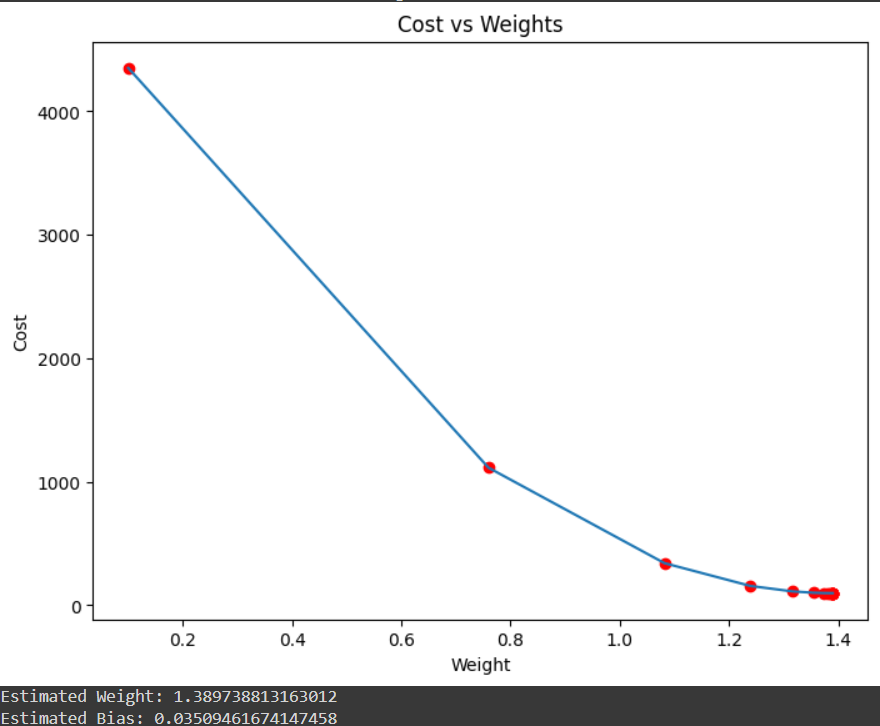
plt.ylabel("Y")

plt.show()

if \_\_name\_\_=="\_\_main\_\_":

main()

**Result:**



A picture containing screenshot, line, plot, diagram

Description automatically generated

**Conclusion:** Hence, Implementation of Gradiant Descent using python code is successfully implemented and verified.

**Experiment 13: Otsu’s Image segmentation (Threshold-based segmentation)**

**Aim:** To implement Otsu’s Image segmentation using Python code and get appropriate results for a medical image file.

**Work implemented:**

import numpy as np

import cv2

from matplotlib import pyplot as plt

img = cv2.imread(r'/content/drive/MyDrive/Medical Applicaion/A2.png')

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, iterations = 2)

sure\_bg = cv2.dilate(closing,kernel,iterations=3)

dist\_transform = cv2.distanceTransform(sure\_bg,cv2.DIST\_L2,3)

ret, sure\_fg = cv2.threshold(dist\_transform,0.1\*dist\_transform.max(),255,0)

sure\_fg = np.uint8(sure\_fg)

unknown = cv2.subtract(sure\_bg,sure\_fg)

ret, markers = cv2.connectedComponents(sure\_fg)

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, 'cool')

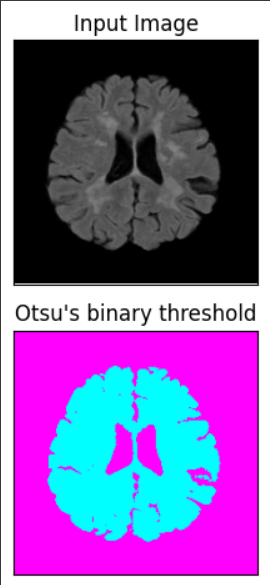
plt.imsave(r'thresh.png',thresh)

plt.title("Otsu's binary threshold"), plt.xticks([]), plt.yticks([])

plt.tight\_layout()

plt.show()

**Result:**



**Conclusion:** Hence, Otsu’s image segmentation has been successfully implemented using python code and verified.

**Experiment 14: Code implementation for Watershed segmentation.**

**Aim:** To implement Watershed segmentation using Python code and get results for a set of Medical Image file.

**Work Implemented:**

import numpy as np

import cv2

from matplotlib import pyplot as plt

img = cv2.imread(r'/content/drive/MyDrive/Medical Applicaion/B (12).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)

plt.subplot(211),plt.imshow(closing, 'winter')

plt.title("morphologyEx:Closing:2x2"), plt.xticks([]), plt.yticks([])

plt.subplot(212),plt.imshow(sure\_bg, 'cool')

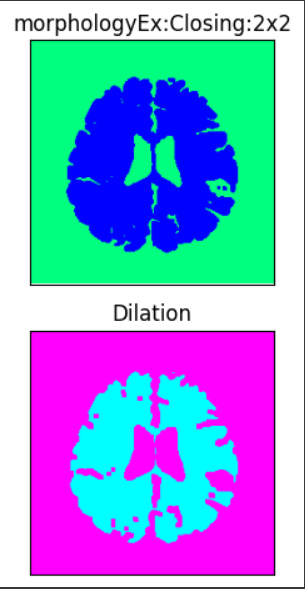
plt.imsave(r'dilation.png',sure\_bg)

plt.title("Dilation"), plt.xticks([]), plt.yticks([])

plt.tight\_layout()

plt.show()

**Result:**



**Conclusion:** Hence, implementation of watershed segmentation is verified successfully.

**Experiment 15: SVM**

**Aim:** To implement Support Vector Machine using python code and verify its output.

**Work Implemented:**

import numpy as np

import pandas as pd

dataset = pd.read\_csv('/content/drive/MyDrive/DATASET/breastcancer.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))

**Result:**

[[108 1]

[ 5 57]]

Accuracy: 96.49%

**Conclusion:** Hence, SVM has been successfully implemented and verified using python code.