INSTITUTE OF TECHNOLOGY
& MANAGEMENT
GWALIOR • MP • INDIA

Laboratory Manual

Machine Learning
(CS-601)

For

Third Year Students CSE
Dept: Computer Science & Engineering

INSTITUTE OF TECHNOLOGY & MANAGEMENT
GWALIOR • MP • INDIA

Department of Computer Science and Engineering

Vision of CSE Department:

The department envisions to nurture students to become technologically proficient, research competent and socially accountable for the welfare of the society.

Mission of the CSE Department:

I. To provide high quality education through effective teaching-learning process emphasizing active participation of students.

II. To build scientifically strong engineers to cater to the needs of industry, higher studies, research and startups.

III. To awaken young minds ingrained with ethical values and professional behaviors for the betterment of the society.

Program Educational Objectives:

Graduates will be able to

I. Our engineers will demonstrate application of comprehensive technical knowledge forinnovation and entrepreneurship.

II. Our graduates will employ capabilities of solving complex engineering problemsto succeed in research and/or higher studies.

III. Our graduates will exhibit team-work and leadership qualities to meet stakeholder business objectives in their careers.

IV. Our graduates will evolve in ethical and professional practices and enhance socioeconomic contributions to the society.

INSTITUTE OF TECHNOLOGY & MANAGEMENT
GWALIOR • MP • INDIA

Program Outcomes (POs):

Engineering Graduates will be able to:

1. **Engineering knowledge**: Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.
2. **Problem analysis**: Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.
3. **Design/development of solutions**: Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.
4. **Conduct investigations of complex problems**: Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.
5. **Modern tool usage**: Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.
6. **The engineer and society**: Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.
7. **Environment and sustainability**: Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.
8. **Ethics**: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.
9. **Individual and team work**: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.
10. **Communication**: Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.
11. **Project management and finance**: Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.
12. **Life-long learning**: Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

INSTITUTE OF TECHNOLOGY & MANAGEMENT
GWALIOR • MP • INDIA

Course Outcomes

Machine Learning(CS-601)

CO1:	Understand and apply knowledge of computing and mathematics to machine learning problems, models and algorithms
CO2:	Understand the concepts of machine learning by applying different algorithms to create various models
CO3:	Analyze machine learning algorithms to design and develop programs using python
CO4:	Develop experiments and implement image recognition algorithms on various datasets using python
CO5:	Understand and apply knowledge of neural network concepts for implementing speech recognition algorithms using python.

Course	Course Outcomes	CO Attainment	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12	PSO1	PSO2	PSO3
CO1	Understand and apply knowledge of computing and mathematics to machine learning problems, models and algorithms		3	1		2	1								3		
CO2	Understand the concepts of machine learning by applying different algorithms to create various models		3	3	2	2									3	2	
CO3	Analyze machine learning algorithms to design and develop programs using python		3			2	2				2	2		2	3		
CO4	Develop experiments and implement image recognition algorithms on various datasets using python		3	3	3	3	3				2	2		2	3	3	3
CO5	Understand and apply knowledge of neural network concepts for implementing speech recognition algorithms using python.		3	3	2	2					1	1		1	3		

INSTITUTE OF TECHNOLOGY & MANAGEMENT
GWALIOR • MP • INDIA

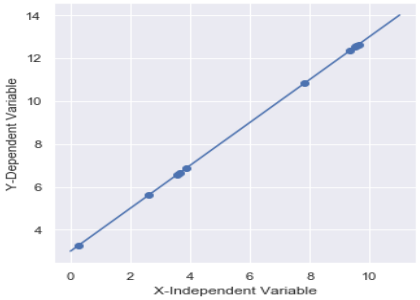
List of Program

PROGRAM -1

Program to implement Linear Regression

```
# importing libraries
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression

x = 11 * np.random.random((10, 1))

# y = a * x + b
y = 1.0 * x + 3.0

# create a linear regression model
model = LinearRegression()
model.fit(x, y)

# predict y from the data where the x is predicted from the x
x_pred = np.linspace(0, 11, 100)
y_pred = model.predict(x_pred[:, np.newaxis])

# plot the results
plt.figure(figsize =(5, 5))
ax = plt.axes()
ax.scatter(x, y)

ax.plot(x_pred, y_pred)
ax.set_xlabel('X-Independent Variable')
ax.set_ylabel('Y-Dependent Variable')
ax.axis('tight')

plt.show()
```

Output:


14
12
10
8
6
4
Y-Dependent Variable
0
2
4
6
8
10
X-Independent Variable

PROGRAM-2

Implement multiple Linear Regression

```
import numpy as np
import matplotlib as mpl
from mpl_toolkits.mplot3d import Axes3D
import matplotlib.pyplot as plt

def generate_dataset(n):
x = []
y = []
random_x1 = np.random.rand()
random_x2 = np.random.rand()
for i in range(n):
x1 = i
x2 = i/2 + np.random.rand()*n
x.append([1, x1, x2])
y.append(random_x1 * x1 + random_x2 * x2 + 1)
return np.array(x), np.array(y)

x, y = generate_dataset(200)

mpl.rcParams['legend.fontsize'] = 12

fig = plt.figure()
ax = fig.gca(projection ='3d')

ax.scatter(x[:, 1], x[:, 2], y, label ='y', s = 5)
ax.legend()
ax.view_init(45, 0)

plt.show()
```

Result:

PROGRAM -3

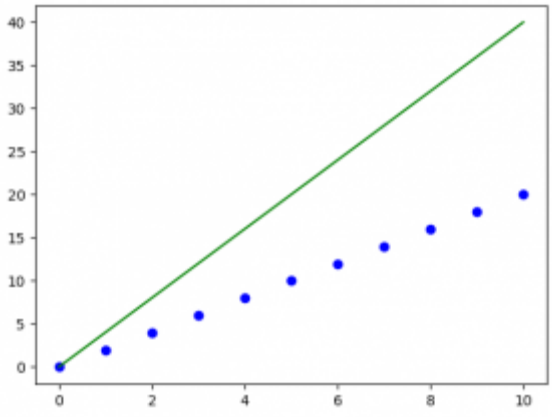
Implementation of gradient descent in linear regression

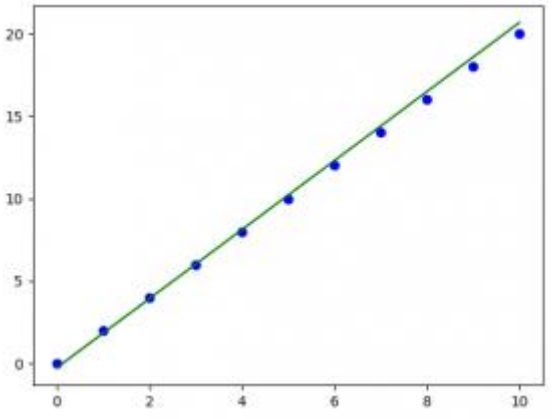
```
import numpy as np
import matplotlib.pyplot as plt

class Linear_Regression:
def __init__(self, X, Y):
self.X = X
self.Y = Y
self.b = [0, 0]

def update_coeffs(self, learning_rate):
Y_pred = self.predict()
Y = self.Y
m = len(Y)
self.b[0] = self.b[0] - (learning_rate * ((1/m) *
np.sum(Y_pred - Y)))

self.b[1] = self.b[1] - (learning_rate * ((1/m) *
np.sum((Y_pred - Y) * self.X)))

def predict(self, X=[]):
Y_pred = np.array([])
if not X: X = self.X
b = self.b
for x in X:
Y_pred = np.append(Y_pred, b[0] + (b[1] * x))

return Y_pred

def get_current_accuracy(self, Y_pred):
p, e = Y_pred, self.Y
n = len(Y_pred)
return 1-sum(
[
abs(p[i]-e[i])/e[i]
for i in range(n)
if e[i] != 0]
)/n
#def predict(self, b, yi):
```

```
def compute_cost(self, Y_pred):

m = len(self.Y)
J = (1 / 2*m) * (np.sum(Y_pred - self.Y)**2)
return J

def plot_best_fit(self, Y_pred, fig):
f = plt.figure(fig)
plt.scatter(self.X, self.Y, color='b')
plt.plot(self.X, Y_pred, color='g')
f.show()


def main():
X = np.array([i for i in range(11)])
Y = np.array([2*i for i in range(11)])

regressor = Linear_Regression(X, Y)

iterations = 0
steps = 100
learning_rate = 0.01
costs = []

#original best-fit line
Y_pred = regressor.predict()
regressor.plot_best_fit(Y_pred, 'Initial Best Fit Line')


while 1:
Y_pred = regressor.predict()
cost = regressor.compute_cost(Y_pred)
costs.append(cost)
regressor.update_coeffs(learning_rate)

iterations += 1
if iterations % steps == 0:
print(iterations, "epochs elapsed")
print("Current accuracy is :",
regressor.get_current_accuracy(Y_pred))

stop = input("Do you want to stop (y/*)??")
if stop == "y":
```

```
break

#final best-fit line
regressor.plot_best_fit(Y_pred, 'Final Best Fit Line')

#plot to verify cost fuction decreases
h = plt.figure('Verification')
plt.plot(range(iterations), costs, color='b')
h.show()

# if user wants to predict using the regressor:
regressor.predict([i for i in range(10)])

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

Output:

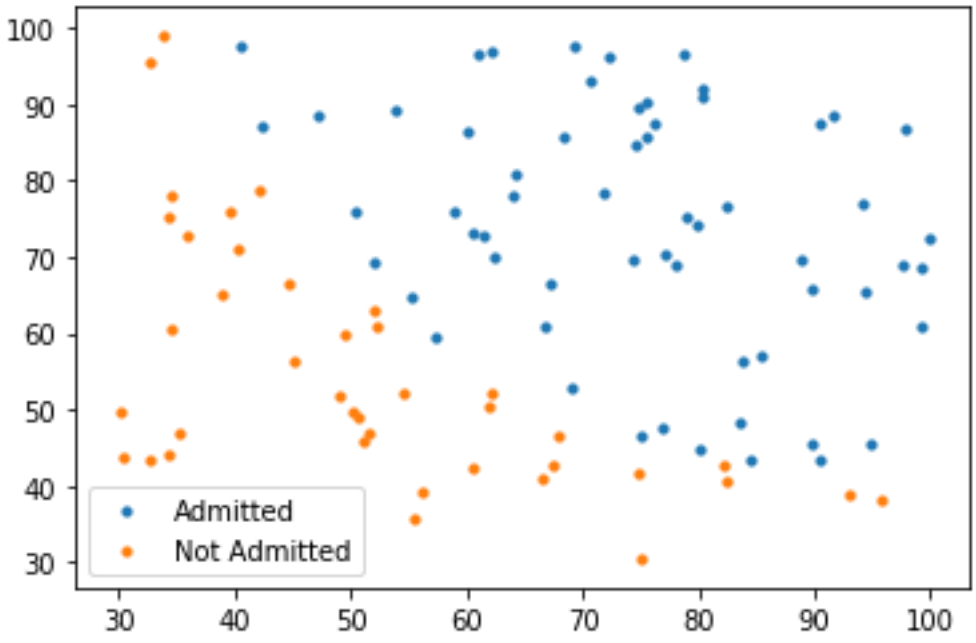




PROGRAM-4

Implementation of Logistic Regression

```
# imports
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

data = pd.read_csv("marks.csv")

# X = feature values, all the columns except the last column
X = data.iloc[:, :-1]

# y = target values, last column of the data frame
y = data.iloc[:, -1]

# filter out the applicants that got admitted
admitted = data.loc[y == 1]

# filter out the applicants that din't get admission
not_admitted = data.loc[y == 0]

# plots
plt.scatter(admitted.iloc[:, 0], admitted.iloc[:, 1], s=10, label='Admitted')
plt.scatter(not_admitted.iloc[:, 0], not_admitted.iloc[:, 1], s=10, label='Not Admitted')
plt.legend()
plt.show()
```


100
90
80
70
60
50
40
30
30
40
50
60
70
80
90
100
Admitted
Not Admitted

PROGRAM-5

Neural Networks Activation Functions.The code shows briefly activation functions used in neural networks' layers and compares their properties.

Currently supported:
- sigmoid
- tanh (hyperbolic tangent)
- relu (rectified linear unit)
"""

```
e = 2.7182818284590451

def frange(start, stop, step):
    f = start
    while f <= stop:
        yield f
        f += step

def sigmoid(x):
    return (e**x)/((e**x)+1)

def tanh(x):
    return ((e**x)-(e**(-x)))/((e**x)+(e**(-x)))

def relu(x):
    return max(0, x)

start = -4.0
stop  = 4.1
step  = 0.1

print('Neural Networks Activation Functions')
print(' ================================')
print(' |  x   | sigmoid |  tanh  | relu |')
print(' ================================')
for i in frange(start, stop, step):
    print(' | {0:+.1f} | {1:+.4f} | {2:+.3f} | {3:+.1f} |'.format(i, sigmoid(i), tanh(i), relu(i)))
print(' ================================')
print('\nLegend:\n')
print('  sigmoid: e**x / e**x + 1')
print('  tanh   : e**x - e**-x / e**x + e**-x')
print('  relu   : max(0, x)')
```

Neural Networks Activation Functions

x	sigmoid	tanh	relu
-4.0	+0.0180	-0.999	+0.0
-3.9	+0.0198	-0.999	+0.0
-3.8	+0.0219	-0.999	+0.0
-3.7	+0.0241	-0.999	+0.0
-3.6	+0.0266	-0.999	+0.0
-3.5	+0.0293	-0.998	+0.0
-3.4	+0.0323	-0.998	+0.0
-3.3	+0.0356	-0.997	+0.0
-3.2	+0.0392	-0.997	+0.0
-3.1	+0.0431	-0.996	+0.0
-3.0	+0.0474	-0.995	+0.0
-2.9	+0.0522	-0.994	+0.0
-2.8	+0.0573	-0.993	+0.0
-2.7	+0.0630	-0.991	+0.0
-2.6	+0.0691	-0.989	+0.0
-2.5	+0.0759	-0.987	+0.0
-2.4	+0.0832	-0.984	+0.0
-2.3	+0.0911	-0.980	+0.0
-2.2	+0.0998	-0.976	+0.0
-2.1	+0.1091	-0.970	+0.0
-2.0	+0.1192	-0.964	+0.0
-1.9	+0.1301	-0.956	+0.0
-1.8	+0.1419	-0.947	+0.0
-1.7	+0.1545	-0.935	+0.0
-1.6	+0.1680	-0.922	+0.0
-1.5	+0.1824	-0.905	+0.0
-1.4	+0.1978	-0.885	+0.0
-1.3	+0.2142	-0.862	+0.0
-1.2	+0.2315	-0.834	+0.0
-1.1	+0.2497	-0.800	+0.0
-1.0	+0.2689	-0.762	+0.0
-0.9	+0.2891	-0.716	+0.0
-0.8	+0.3100	-0.664	+0.0
-0.7	+0.3318	-0.604	+0.0
-0.6	+0.3543	-0.537	+0.0
-0.5	+0.3775	-0.462	+0.0
-0.4	+0.4013	-0.380	+0.0
-0.3	+0.4256	-0.291	+0.0
-0.2	+0.4502	-0.197	+0.0

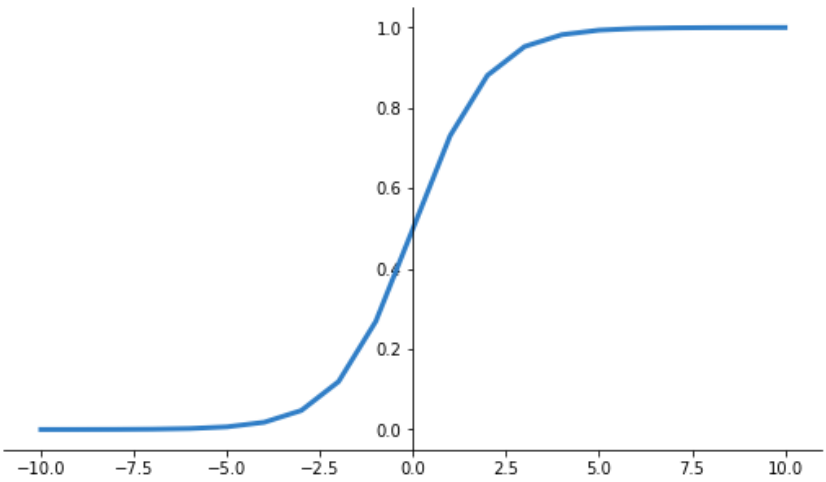
-0.1	+0.4750	-0.100	+0.0
+0.0	+0.5000	+0.000	+0.0
+0.1	+0.5250	+0.100	+0.1
+0.2	+0.5498	+0.197	+0.2
+0.3	+0.5744	+0.291	+0.3
+0.4	+0.5987	+0.380	+0.4
+0.5	+0.6225	+0.462	+0.5
+0.6	+0.6457	+0.537	+0.6
+0.7	+0.6682	+0.604	+0.7
+0.8	+0.6900	+0.664	+0.8
+0.9	+0.7109	+0.716	+0.9
+1.0	+0.7311	+0.762	+1.0
+1.1	+0.7503	+0.800	+1.1
+1.2	+0.7685	+0.834	+1.2
+1.3	+0.7858	+0.862	+1.3
+1.4	+0.8022	+0.885	+1.4
+1.5	+0.8176	+0.905	+1.5
+1.6	+0.8320	+0.922	+1.6
+1.7	+0.8455	+0.935	+1.7
+1.8	+0.8581	+0.947	+1.8
+1.9	+0.8699	+0.956	+1.9
+2.0	+0.8808	+0.964	+2.0
+2.1	+0.8909	+0.970	+2.1
+2.2	+0.9002	+0.976	+2.2
+2.3	+0.9089	+0.980	+2.3
+2.4	+0.9168	+0.984	+2.4
+2.5	+0.9241	+0.987	+2.5
+2.6	+0.9309	+0.989	+2.6
+2.7	+0.9370	+0.991	+2.7
+2.8	+0.9427	+0.993	+2.8
+2.9	+0.9478	+0.994	+2.9
+3.0	+0.9526	+0.995	+3.0
+3.1	+0.9569	+0.996	+3.1
+3.2	+0.9608	+0.997	+3.2
+3.3	+0.9644	+0.997	+3.3
+3.4	+0.9677	+0.998	+3.4
+3.5	+0.9707	+0.998	+3.5
+3.6	+0.9734	+0.999	+3.6
+3.7	+0.9759	+0.999	+3.7
+3.8	+0.9781	+0.999	+3.8
+3.9	+0.9802	+0.999	+3.9
+4.0	+0.9820	+0.999	+4.0

==========

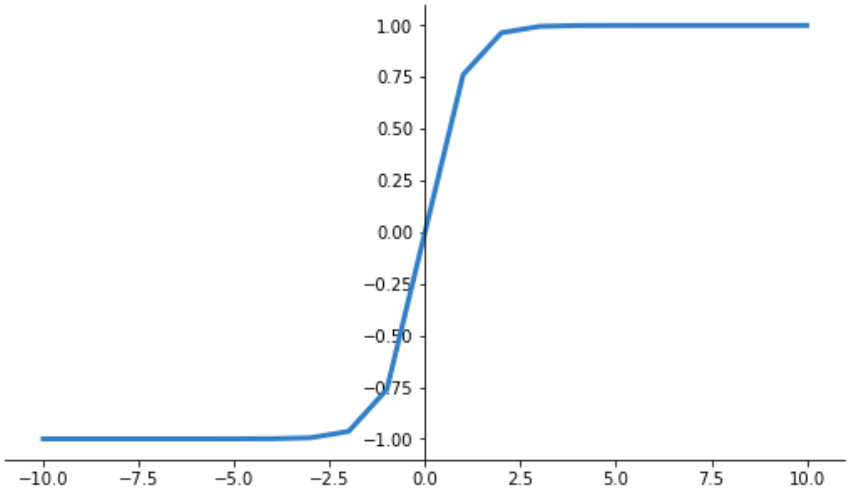
Legend:

sigmoid: e**x / e**x + 1
tanh : e**x - e**-x / e**x + e**-x
relu : max(0, x)

```
# plot inputs and outputs
from matplotlib import pyplot as plt

# define a series of inputs
series_in = [x for x in range(-10, 11)]
# calculate outputs for our inputs
series_out = [sigmoid(x) for x in series_in]
# Setup centered axes
fig, ax = plt.subplots(figsize=(9, 5))
ax.spines['left'].set_position('center')
ax.spines['right'].set_color('none')
ax.spines['top'].set_color('none')
ax.xaxis.set_ticks_position('bottom')
ax.yaxis.set_ticks_position('left')
# Create and show plot
# line plot of raw inputs to rectified outputs
plt.plot(series_in, series_out, color="#307EC7", linewidth=3, label="sigmoid")
plt.show()
```

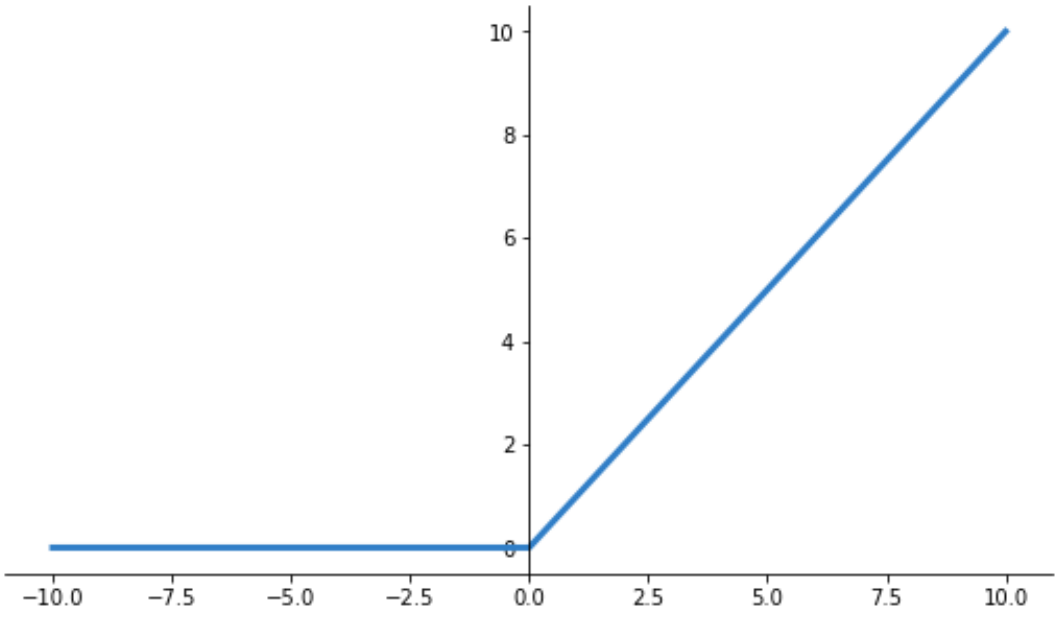


```
# plot inputs and outputs
from matplotlib import pyplot as plt

# define a series of inputs
series_in = [x for x in range(-10, 11)]
# calculate outputs for our inputs
series_out = [tanh(x) for x in series_in]
# Setup centered axes
fig, ax = plt.subplots(figsize=(9, 5))
ax.spines['left'].set_position('center')
ax.spines['right'].set_color('none')
ax.spines['top'].set_color('none')
ax.xaxis.set_ticks_position('bottom')
ax.yaxis.set_ticks_position('left')
# Create and show plot
# line plot of raw inputs to rectified outputs
plt.plot(series_in, series_out, color="#307EC7", linewidth=3, label="sigmoid")
plt.show()
```




```
# plot inputs and outputs
from matplotlib import pyplot as plt

# define a series of inputs
series_in = [x for x in range(-10, 11)]
# calculate outputs for our inputs
series_out = [relu(x) for x in series_in]
# Setup centered axes
fig, ax = plt.subplots(figsize=(9, 5))
ax.spines['left'].set_position('center')
```

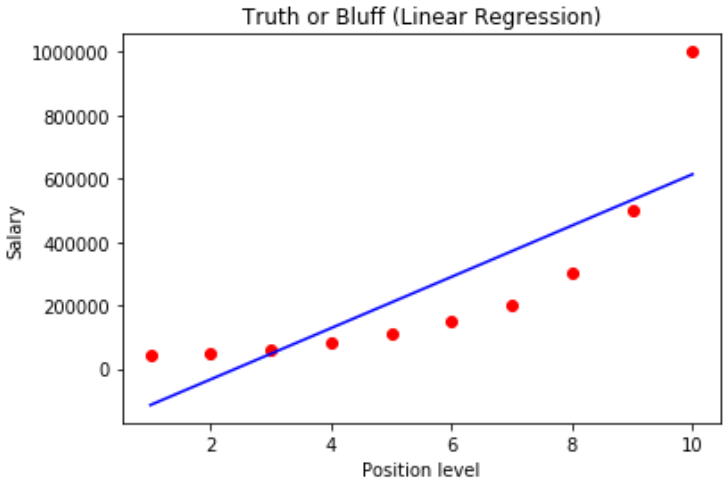
```
ax.spines['right'].set_color('none')
ax.spines['top'].set_color('none')
ax.xaxis.set_ticks_position('bottom')
ax.yaxis.set_ticks_position('left')
# Create and show plot
# line plot of raw inputs to rectified outputs
plt.plot(series_in, series_out, color="#307EC7", linewidth=3, label="sigmoid")
plt.show()
```



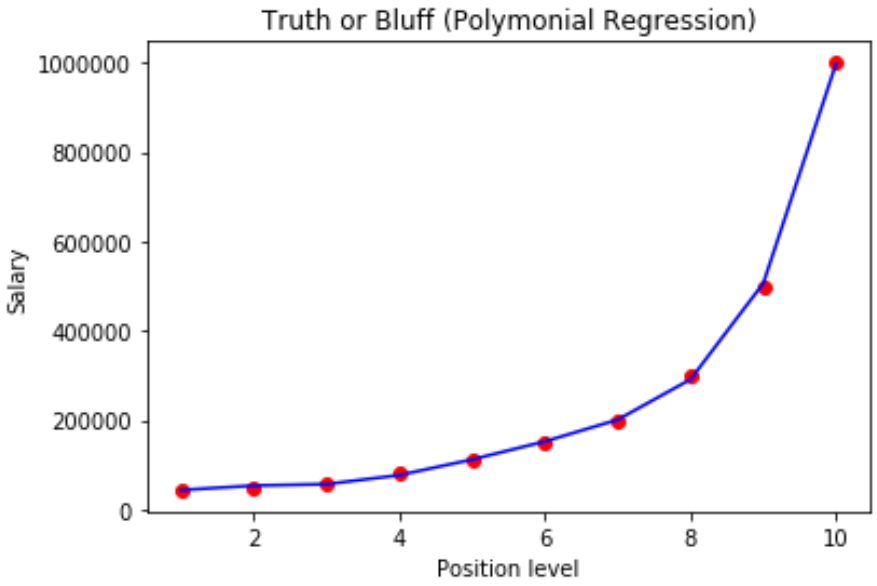
PROGRAM -6

Implemetation of Polynomial Regression

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

# Importing the dataset
# dataset = pd.read_csv('https://s3.us-west-
2.amazonaws.com/public.gamelab.fun/dataset/position_salaries.csv')
dataset = pd.read_csv ('position_salaries.csv')
X = dataset.iloc[:, 1:2].values
y = dataset.iloc[:, 2].values

# Splitting the dataset into the Training set and Test set
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)

# Fitting Linear Regression to the dataset
from sklearn.linear_model import LinearRegression
lin_reg = LinearRegression()
lin_reg.fit(X, y)

# Visualizing the Linear Regression results
def viz_linear():
    plt.scatter(X, y, color='red')
    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()
    return
viz_linear()
```




```
# Fitting Polynomial Regression to the dataset
from sklearn.preprocessing import PolynomialFeatures
poly_reg = PolynomialFeatures(degree=5)
X_poly = poly_reg.fit_transform(X)
pol_reg = LinearRegression()
pol_reg.fit(X_poly, y)

# Visualizing the Polymonial Regression results
def viz_polymonial():
    plt.scatter(X, y, color='red')
    plt.plot(X, pol_reg.predict(poly_reg.fit_transform(X)), color='blue')
    plt.title('Truth or Bluff (Polymonial Regression)')
    plt.xlabel('Position level')
    plt.ylabel('Salary')
    plt.show()
    return
viz_polymonial()
```


Truth or Bluff (Polymonial Regression)
Salary
Position level
1000000
800000
600000
400000
200000
0
2
4
6
8
10

PROGRAM-7

CNN model: we will develop a one-dimensional convolutional neural network model (1D CNN) for the human activity recognition dataset. The data is provided as a single zip file that is about 58 megabytes in size. The direct link for this download is below:

- UCI HAR Dataset.zip

```
from numpy import mean
from numpy import std
from numpy import dstack
from pandas import read_csv
from matplotlib import pyplot
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import Flatten
from keras.layers import Dropout
from keras.layers.convolutional import Conv1D
from keras.layers.convolutional import MaxPooling1D
from keras.utils import to_categorical

# load a single file as a numpy array
def load_file(filepath):
dataframe = read_csv(filepath, header=None, delim_whitespace=True)
return dataframe.values

# load a list of files and return as a 3d numpy array
def load_group(filenames, prefix=''):
loaded = list()
for name in filenames:
data = load_file(prefix + name)
loaded.append(data)
# stack group so that features are the 3rd dimension
loaded = dstack(loaded)
return loaded

# load a dataset group, such as train or test
def load_dataset_group(group, prefix=''):
filepath = prefix + group + '/Inertial Signals/'
# load all 9 files as a single array
filenames = list()
# total acceleration
filenames += ['total_acc_x_'+group+'.txt', 'total_acc_y_'+group+'.txt',
```

```
'total_acc_z_'+group+'.txt']
# body acceleration
filenames += ['body_acc_x_'+group+'.txt', 'body_acc_y_'+group+'.txt',
'body_acc_z_'+group+'.txt']
# body gyroscope
filenames += ['body_gyro_x_'+group+'.txt', 'body_gyro_y_'+group+'.txt',
'body_gyro_z_'+group+'.txt']
# load input data
X = load_group(filenames, filepath)
# load class output
y = load_file(prefix + group + '/y_'+group+'.txt')
return X, y

# load the dataset, returns train and test X and y elements
def load_dataset(prefix=''):
# load all train
trainX, trainy = load_dataset_group('train', prefix + 'HARDataset/')
print(trainX.shape, trainy.shape)
# load all test
testX, testy = load_dataset_group('test', prefix + 'HARDataset/')
print(testX.shape, testy.shape)
# zero-offset class values
trainy = trainy - 1
testy = testy - 1
# one hot encode y
trainy = to_categorical(trainy)
testy = to_categorical(testy)
print(trainX.shape, trainy.shape, testX.shape, testy.shape)
return trainX, trainy, testX, testy

# fit and evaluate a model
def evaluate_model(trainX, trainy, testX, testy):
verbose, epochs, batch_size = 0, 10, 32
n_timesteps, n_features, n_outputs = trainX.shape[1], trainX.shape[2], trainy.shape[1]
model = Sequential()
model.add(Conv1D(filters=64, kernel_size=3, activation='relu',
input_shape=(n_timesteps,n_features)))
model.add(Conv1D(filters=64, kernel_size=3, activation='relu'))
model.add(Dropout(0.5))
model.add(MaxPooling1D(pool_size=2))
model.add(Flatten())

model.add(Dense(100, activation='relu'))
```

```
                                                    model.add(Dense(n_outputs,
activation='softmax'))


model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
# fit network
model.fit(trainX, trainy, epochs=epochs, batch_size=batch_size, verbose=verbose)
# evaluate model
_, accuracy = model.evaluate(testX, testy, batch_size=batch_size, verbose=0)
return accuracy

# summarize scores
def summarize_results(scores):
print(scores)
m, s = mean(scores), std(scores)
print('Accuracy: %.3f%% (+/-%.3f)' % (m, s))

# run an PROGRAM
def run_PROGRAM(repeats=10):
# load data
trainX, trainy, testX, testy = load_dataset()
# repeat PROGRAM
scores = list()
for r in range(repeats):
score = evaluate_model(trainX, trainy, testX, testy)
score = score * 100.0
print('>#%d: %.3f' % (r+1, score))
scores.append(score)
# summarize results
summarize_results(scores)

# run the PROGRAM
run_PROGRAM()
```

RESULT:

```
(7352, 128, 9) (7352, 1)
(2947, 128, 9) (2947, 1)
(7352, 128, 9) (7352, 6) (2947, 128, 9) (2947, 6)

>p=False #1: 91.483
>p=False #2: 91.245
>p=False #3: 90.838
```

>p=False #4: 89.243
>p=False #5: 90.193
>p=False #6: 90.465

>p=False #7: 90.397
>p=False #8: 90.567
>p=False #9: 88.938
>p=False #10: 91.144
>p=True #1: 92.908
>p=True #2: 90.940
>p=True #3: 92.297
>p=True #4: 91.822
>p=True #5: 92.094
>p=True #6: 91.313
>p=True #7: 91.653
>p=True #8: 89.141
>p=True #9: 91.110
>p=True #10: 91.890

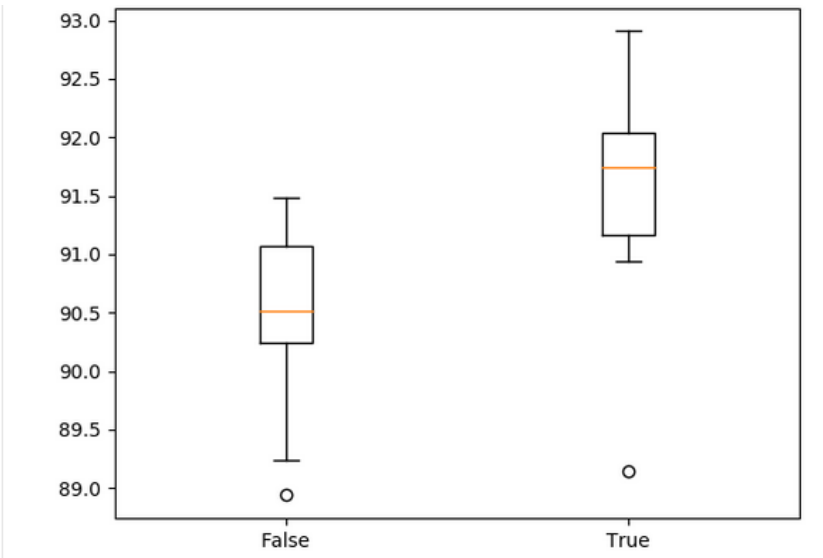
[[91.48286392941975, 91.24533423820834, 90.83814048184594, 89.24329826942655, 90.19341703427214, 90.46487953851374, 90.39701391245333, 90.56667797760434, 88.93790295215473, 91.14353579911774], [92.90804207668816, 90.93993892093654, 92.29725144214456, 91.82219205972176, 92.09365456396336, 91.31319986426874, 91.65252799457076, 89.14149983033593, 91.10960298608755, 91.89005768578215]]
[False, True]

Param=False: 90.451% (+/-0.785)
Param=True: 91.517% (+/-0.965)

A box and whisker plot of the results is also created.

This allows the two samples of results to be compared in a nonparametric way, showing the median and the middle 50% of each sample.

We can see that the distribution of results with standardization is quite different from the distribution of results without standardization. This is likely a real effect.



Box and whisker plot of 1D CNN with and without standardization

PROGRAM-8

Develop a Long Short-Term Memory network model (LSTM) for the human activity recognition dataset.

```
# lstm model
from numpy import mean
from numpy import std
from numpy import dstack
from pandas import read_csv
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import Flatten
from keras.layers import Dropout
from keras.layers import LSTM
from keras.utils import to_categorical
from matplotlib import pyplot

# load a single file as a numpy array
def load_file(filepath):
dataframe = read_csv(filepath, header=None, delim_whitespace=True)
return dataframe.values

# load a list of files and return as a 3d numpy array
def load_group(filenames, prefix=''):
loaded = list()
for name in filenames:
data = load_file(prefix + name)
loaded.append(data)
# stack group so that features are the 3rd dimension
loaded = dstack(loaded)
return loaded

# load a dataset group, such as train or test
def load_dataset_group(group, prefix=''):
filepath = prefix + group + '/Inertial Signals/'
# load all 9 files as a single array
filenames = list()
# total acceleration
filenames += ['total_acc_x_'+group+'.txt', 'total_acc_y_'+group+'.txt',
'total_acc_z_'+group+'.txt']
# body acceleration
```

```
filenames += ['body_acc_x_'+group+'.txt', 'body_acc_y_'+group+'.txt',
'body_acc_z_'+group+'.txt']

# body gyroscope
filenames += ['body_gyro_x_'+group+'.txt', 'body_gyro_y_'+group+'.txt',
'body_gyro_z_'+group+'.txt']
# load input data
X = load_group(filenames, filepath)
# load class output
y = load_file(prefix + group + '/y_'+group+'.txt')
return X, y

# load the dataset, returns train and test X and y elements
def load_dataset(prefix=''):
# load all train
trainX, trainy = load_dataset_group('train', prefix + 'HARDataset/')
print(trainX.shape, trainy.shape)
# load all test
testX, testy = load_dataset_group('test', prefix + 'HARDataset/')
print(testX.shape, testy.shape)
# zero-offset class values
trainy = trainy - 1
testy = testy - 1
# one hot encode y
trainy = to_categorical(trainy)
testy = to_categorical(testy)
print(trainX.shape, trainy.shape, testX.shape, testy.shape)
return trainX, trainy, testX, testy

# fit and evaluate a model
def evaluate_model(trainX, trainy, testX, testy):
verbose, epochs, batch_size = 0, 15, 64
n_timesteps, n_features, n_outputs = trainX.shape[1], trainX.shape[2], trainy.shape[1]
model = Sequential()
model.add(LSTM(100, input_shape=(n_timesteps,n_features)))
model.add(Dropout(0.5))
model.add(Dense(100, activation='relu'))
model.add(Dense(n_outputs, activation='softmax'))
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
# fit network
model.fit(trainX, trainy, epochs=epochs, batch_size=batch_size, verbose=verbose)
# evaluate model
_, accuracy = model.evaluate(testX, testy, batch_size=batch_size, verbose=0)
```

```
return accuracy

# summarize scores

def summarize_results(scores):
print(scores)
m, s = mean(scores), std(scores)
print('Accuracy: %.3f%% (+/-%.3f)' % (m, s))

# run an PROGRAM
def run_PROGRAM(repeats=10):
# load data
trainX, trainy, testX, testy = load_dataset()
# repeat PROGRAM
scores = list()
for r in range(repeats):
score = evaluate_model(trainX, trainy, testX, testy)
score = score * 100.0
print('>#%d: %.3f' % (r+1, score))
scores.append(score)
# summarize results
summarize_results(scores)

# run the PROGRAM
run_PROGRAM()
```

Result:

```
(7352, 128, 9) (7352, 1)
(2947, 128, 9) (2947, 1)
(7352, 128, 9) (7352, 6) (2947, 128, 9) (2947, 6)

>#1: 90.058
>#2: 85.918
>#3: 90.974
>#4: 89.515
>#5: 90.159
>#6: 91.110
>#7: 89.718
>#8: 90.295
>#9: 89.447
>#10: 90.024
```

[90.05768578215134, 85.91788259246692, 90.97387173396675, 89.51476077366813, 90.15948422124194, 91.10960298608755, 89.71835765184933, 90.29521547336275, 89.44689514760775, 90.02375296912113]

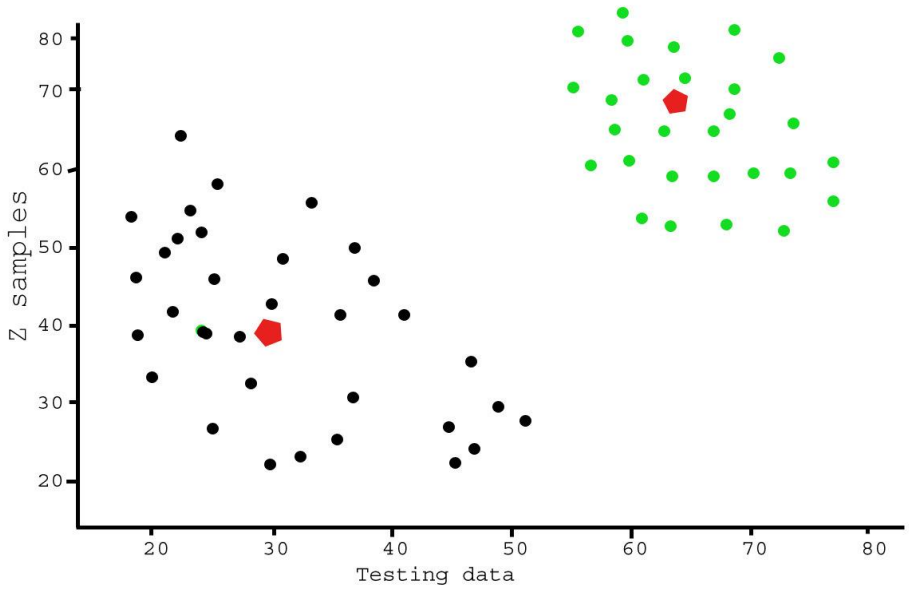
Accuracy: 89.722% (+/-1.371)

PROGRAM-9

K-means clustering on a sample random data using open-cv library.

```
# importing required tools
import numpy as np
from matplotlib import pyplot as plt

# creating two test data
X = np.random.randint(10,35,(25,2))
Y = np.random.randint(55,70,(25,2))
Z = np.vstack((X,Y))
Z = Z.reshape((50,2))
# convert to np.float32
Z = np.float32(Z)

plt.xlabel('Test Data')
plt.ylabel('Z samples')

plt.hist(Z,256,[0,256])

plt.show()
```

RESULT



PROGRAM-10

Implementation of decision tree algorithm.

```
import numpy as np
import pandas as pd
from sklearn.metrics import confusion_matrix
from sklearn.cross_validation import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report

# Function importing Dataset
def importdata():
balance_data = pd.read_csv(
'https://archive.ics.uci.edu/ml/machine-learning-'+
'databases/balance-scale/balance-scale.data',
sep= ',', header = None)

# Printing the dataswet shape
print ("Dataset Length: ", len(balance_data))
print ("Dataset Shape: ", balance_data.shape)

# Printing the dataset obseravtions
print ("Dataset: ",balance_data.head())
return balance_data

# Function to split the dataset
def splitdataset(balance_data):

# Separating the target variable
X = balance_data.values[:, 1:5]
Y = balance_data.values[:, 0]

# Splitting the dataset into train and test
X_train, X_test, y_train, y_test = train_test_split(
X, Y, test_size = 0.3, random_state = 100)

return X, Y, X_train, X_test, y_train, y_test

# Function to perform training with giniIndex.
def train_using_gini(X_train, X_test, y_train):
```

```
# Creating the classifier object
clf_gini = DecisionTreeClassifier(criterion = "gini",
random_state = 100,max_depth=3, min_samples_leaf=5)

# Performing training
clf_gini.fit(X_train, y_train)
return clf_gini

# Function to perform training with entropy.
def tarin_using_entropy(X_train, X_test, y_train):

# Decision tree with entropy
clf_entropy = DecisionTreeClassifier(
criterion = "entropy", random_state = 100,
max_depth = 3, min_samples_leaf = 5)

# Performing training
clf_entropy.fit(X_train, y_train)
return clf_entropy


# Function to make predictions
def prediction(X_test, clf_object):

# Predicton on test with giniIndex
y_pred = clf_object.predict(X_test)
print("Predicted values:")
print(y_pred)
return y_pred

# Function to calculate accuracy
def cal_accuracy(y_test, y_pred):

print("Confusion Matrix: ",
confusion_matrix(y_test, y_pred))

print ("Accuracy : ",
accuracy_score(y_test,y_pred)*100)

print("Report : ",
classification_report(y_test, y_pred))

# Driver code
```

```
def main():

# Building Phase
data = importdata()
X, Y, X_train, X_test, y_train, y_test = splitdataset(data)
clf_gini = train_using_gini(X_train, X_test, y_train)
clf_entropy = tarin_using_entropy(X_train, X_test, y_train)

# Operational Phase
print("Results Using Gini Index:")

# Prediction using gini
y_pred_gini = prediction(X_test, clf_gini)
cal_accuracy(y_test, y_pred_gini)

print("Results Using Entropy:")
# Prediction using entropy
y_pred_entropy = prediction(X_test, clf_entropy)
cal_accuracy(y_test, y_pred_entropy)


# Calling main function
if __name__=="__main__":
main()
```

RESULT:

Data Infomation:

```
Dataset Length:  625
Dataset Shape:  (625, 5)
Dataset:     0  1  2  3  4
0  B  1  1  1  1
1  R  1  1  1  2
2  R  1  1  1  3
3  R  1  1  1  4
4  R  1  1  1  5
```

Results Using Gini Index:

Predicted values:
['R' 'L' 'R' 'R' 'R' 'L' 'R' 'L' 'L' 'L' 'R' 'L' 'L' 'L' 'R' 'L' 'R' 'L'
'L' 'R' 'L' 'R' 'L' 'L' 'R' 'L' 'L' 'L' 'R' 'L' 'L' 'L' 'R' 'L' 'L' 'L'
'L' 'R' 'L' 'L' 'R' 'L' 'R' 'L' 'R' 'R' 'L' 'L' 'R' 'L' 'R' 'R' 'L' 'R'
'R' 'L' 'R' 'R' 'L' 'L' 'R' 'R' 'L' 'L' 'L' 'L' 'L' 'R' 'R' 'L' 'L' 'R'
'R' 'L' 'R' 'L' 'R' 'R' 'R' 'L' 'R' 'L' 'L' 'L' 'L' 'R' 'R' 'L' 'R' 'L'
'R' 'R' 'L' 'L' 'L' 'R' 'R' 'L' 'L' 'L' 'R' 'L' 'R' 'R' 'R' 'R' 'R' 'R'
'R' 'L' 'R' 'L' 'R' 'R' 'L' 'R' 'R' 'R' 'R' 'R' 'L' 'R' 'L' 'L' 'L' 'L'
'L' 'L' 'L' 'R' 'R' 'R' 'R' 'L' 'R' 'R' 'R' 'L' 'L' 'R' 'L' 'R' 'L' 'R'
'L' 'L' 'R' 'L' 'L' 'R' 'L' 'R' 'L' 'R' 'R' 'R' 'L' 'R' 'R' 'R' 'R' 'R'
'L' 'L' 'R' 'R' 'R' 'R' 'L' 'R' 'R' 'R' 'L' 'R' 'L' 'L' 'L' 'L' 'R' 'R'
'L' 'R' 'R' 'L' 'L' 'R' 'R' 'R']

Confusion Matrix: [[0 6 7]
[0 67 18]
[0 19 71]]
Accuracy : 73.4042553191
Report :

precision	recall	f1-score	support
B 0.00	0.00	0.00	13
L 0.73	0.79	0.76	85
R 0.74	0.79	0.76	90
avg/total 0.68	0.73	0.71	188

Results Using Entropy:

Predicted values:
['R' 'L' 'R' 'L' 'R' 'L' 'R' 'L' 'R' 'R' 'R' 'R' 'L' 'L' 'R' 'L' 'R' 'L'
'L' 'R' 'L' 'R' 'L' 'L' 'R' 'L' 'R' 'L' 'R' 'L' 'R' 'L' 'R' 'L' 'L' 'L'
'L' 'L' 'R' 'L' 'R' 'L' 'R' 'L' 'R' 'R' 'L' 'L' 'R' 'L' 'L' 'R' 'L' 'L'
'R' 'L' 'R' 'R' 'L' 'R' 'R' 'R' 'L' 'L' 'R' 'L' 'L' 'R' 'L' 'L' 'L' 'R'
'R' 'L' 'R' 'L' 'R' 'R' 'R' 'L' 'R' 'L' 'L' 'L' 'L' 'R' 'R' 'L' 'R' 'L'
'R' 'R' 'L' 'L' 'L' 'R' 'R' 'L' 'L' 'L' 'R' 'L' 'L' 'R' 'R' 'R' 'R' 'R'
'R' 'L' 'R' 'L' 'R' 'R' 'L' 'R' 'R' 'L' 'R' 'R' 'L' 'R' 'R' 'R' 'L' 'L'
'L' 'L' 'L' 'R' 'R' 'R' 'R' 'L' 'R' 'R' 'R' 'L' 'L' 'R' 'L' 'R' 'L' 'R'
'L' 'R' 'R' 'L' 'L' 'R' 'L' 'R' 'R' 'R' 'R' 'R' 'L' 'R' 'R' 'R' 'R' 'R'
'R' 'L' 'R' 'L' 'R' 'R' 'L' 'R' 'L' 'R' 'L' 'R' 'L' 'L' 'L' 'L' 'L' 'R'
'R' 'R' 'L' 'L' 'L' 'R' 'R' 'R']

Confusion Matrix: [[0 6 7]
[0 63 22]
[0 20 70]]
Accuracy : 70.7446808511
Report :

	precision	recall	f1-score	support
B	0.00	0.00	0.00	13
L	0.71	0.74	0.72	85
R	0.71	0.78	0.74	90
avg / total	0.66	0.71	0.68	188

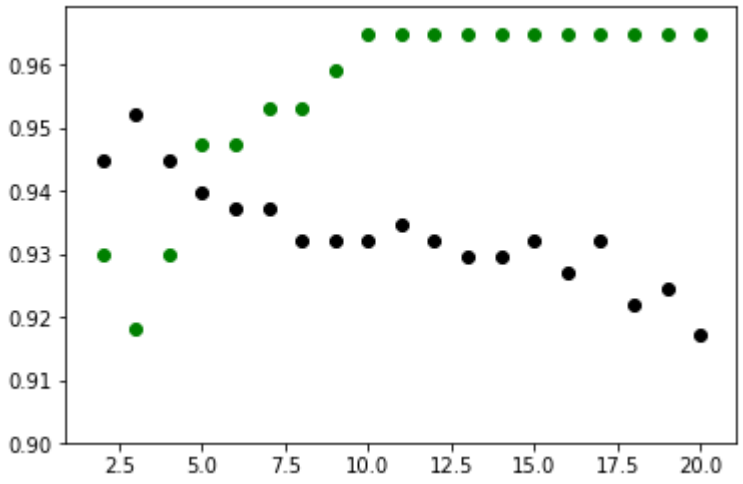
PROGRAM-11

K-Nearest neighbors classifier algorithm

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
import matplotlib.pyplot as plt
import seaborn as sns

d C:\Users\Dev\Desktop\Kaggle\Breast_Cancer
# Changing the read file location to the location of the file

df = pd.read_csv('data.csv')

y = df['diagnosis']
X = df.drop('diagnosis', axis = 1)
X = X.drop('Unnamed: 32', axis = 1)
X = X.drop('id', axis = 1)
# Separating the dependent and independent variable

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

K = []
training = []
test = []
scores = {}

for k in range(2, 21):
   clf = KNeighborsClassifier(n_neighbors = k)
   clf.fit(X_train, y_train)

   training_score = clf.score(X_train, y_train)
   test_score = clf.score(X_test, y_test)
   K.append(k)

   training.append(training_score)
   test.append(test_score)
   scores[k] = [training_score, test_score]
```

```
for keys, values in scores.items():
        print(keys, ':', values)
ax = sns.stripplot(K, training);
ax.set(xlabel ='values of k', ylabel ='Training Score')

plt.show()
# function to show plot

ax = sns.stripplot(K, test);
ax.set(xlabel ='values of k', ylabel ='Test Score')
plt.show()
plt.scatter(K, training, color ='k')
plt.scatter(K, test, color ='g')
plt.show()
```

RESULT:



PROGRAM-12

Implementation of Multinomial Logisitc Regression using scikit-learn to make predictions on digit dataset.

```
from sklearn import datasets, linear_model, metrics

# load the digit dataset
digits = datasets.load_digits()

# defining feature matrix(X) and response vector(y)
X = digits.data
y = digits.target

# splitting X and y into training and testing sets
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.4,
random_state=1)

# create logistic regression object
reg = linear_model.LogisticRegression()

# train the model using the training sets
reg.fit(X_train, y_train)

# making predictions on the testing set
y_pred = reg.predict(X_test)

# comparing actual response values (y_test) with predicted response values (y_pred)
print("Logistic Regression model accuracy(in %):",
metrics.accuracy_score(y_test, y_pred)*100)
```

RESULT: Logistic Regression model accuracy(in %): 95.6884561892