Program 2a:

Apply:
Simple linear regression model for head Brain dataset and predict brain weight based on head size using the least square method.
Find out

i. R^2 score for the predicted model.
ii. Display all the data points along with the fitting the data points to the model.

```
#importing libraries

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

# Reading Data

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

print(data.shape)

data.head()
```

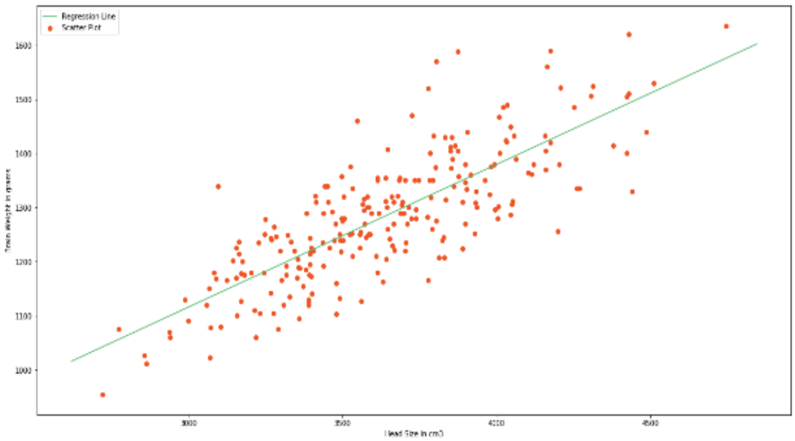
(237, 4)

	Gender	Age Range	Head Size(cm^3)	Brain Weight(grams)
0	1	1	4512	1530
1	1	1	3738	1297
2	1	1	4261	1335
3	1	1	3777	1282
4	1	1	4177	1590

```
# Collecting X and Y

X = data['Head Size(cm^3)'].values

Y = data['Brain Weight(grams)'].values

# Calculating coefficient

# Mean X and Y

mean_x = np.mean(X)

mean_y = np.mean(Y)
```

```
print(mean_x)

print(mean_y)

# Total number of values

n = len(X)

print(n)

3633.9915611814345
1282.873417721519
237
# Using the formula to calculate b1 and b0

numer = 0

denom = 0

for i in range(n):

numer += (X[i] - mean_x) * (Y[i] - mean_y)

denom += (X[i] - mean_x) ** 2

b1 = numer / denom

b0 = mean_y - (b1 * mean_x)

# Printing coefficients

print("Coefficients")

print(b1, b0)

Coefficients
b1:0.26342933948939945 b0:325.57342104944223

# Plotting Values and Regression Line

max_x = np.max(X) + 100

min_x = np.min(X) - 100

# Calculating line values x and y

x = np.linspace(min_x, max_x, 1000)

y = b0 + b1 * x

# Ploting Line

plt.plot(x, y, color='#58b970', label='Regression Line')
```

```
# Ploting Scatter Points
plt.scatter(X, Y, c='#ef5423', label='Scatter Plot')


plt.xlabel('Head Size in cm3')
plt.ylabel('Brain Weight in grams')
plt.legend()
plt.show()
```



Calculating R^2 Score

```
ss_tot = 0
ss_res = 0
for i in range(n):
y_pred = b0 + b1 * X[i]
ss_tot += (Y[i] - mean_y) ** 2
ss_res += (Y[i] - y_pred) ** 2
r2 = 1 - (ss_res/ss_tot)
print("R2 Score")
print(r2)
```

R^2 Score

0.6393117199570003

Conclusion: The simple linear regression model gives average accuracy depending on the R^2 score value.

2b. Simple linear regression model for housing_ prices_ SLR dataset and predict house price based on the area of the house using the library scikit_ learn.
Find out

i. Analyze the R^2score of predicted training and test models score.
ii. Display all the data points along with the fitting the data points to the model.

```
# Step1:importing all the libraries
import numpy as np

import pandas as pd

importmatplotlib.pyplot as plt

%matplotlib  inline

# Step2:load dataset

df=pd.read_csv("housing_prices_SLR.csv",delimiter=',')

df.head()
```

	AREA	PRICE
0	1000	5618
1	1030	5201
2	1060	4779
3	1090	5425
4	1120	5657

```
Step3: Feature matrix and Target vector

x=df[['AREA']].values#feature Matrix

y=df.PRICE.values#Target Matrix

x[:5] #slicing

y[:5]

Step4: Split the data into 80-20

#from packagename import function

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=100)

#80 20 split,random_state to reproduce the same split everytime

print(x_train.shape)

print(x_test.shape)

print(x_train.shape)
```

```
print(x_test.shape)
```

```
(40, 1)
(10, 1)
(40, 1)
(10, 1)
```

#step5: Fit the line:Train the SLR Model

```
From sklearn.linear_model import Linear Regression

lr_model= Linear Regression()

lr_model.fit(x_train,y_train)

print(lr_model.intercept_) # (PRICE=(-4481.80028058845)+8.65903854)*AREA

print(lr_model.coef_)#y=c+mx
```

```
b0:-3103.34066448488
b1:[7.75979089]
```

```
lr_model=Linear Regression(fit_intercept= False)

lr_model.fit(x_train,y_train)

print(lr_model.intercept_) # (PRICE=(-4481.80028058845)+8.65903854)*AREA

print(lr_model.coef_)#y=c+mx
```

```
b0:0.0
b1:6.03609138
```

#step6: predict using the model

```
From sklearn.metrics import r2_score

y_train

lr_model.predict(x_train)
```

step7: calculating R^2score using tain and test model

```
r2_score(y_train,lr_model.predict(x_train))
```

R^2_Train_Score:0.820250203127675

```
r2_score(y_test,lr_model.predict(x_test))
```

R^2_Test_Score:0.5059420550739799

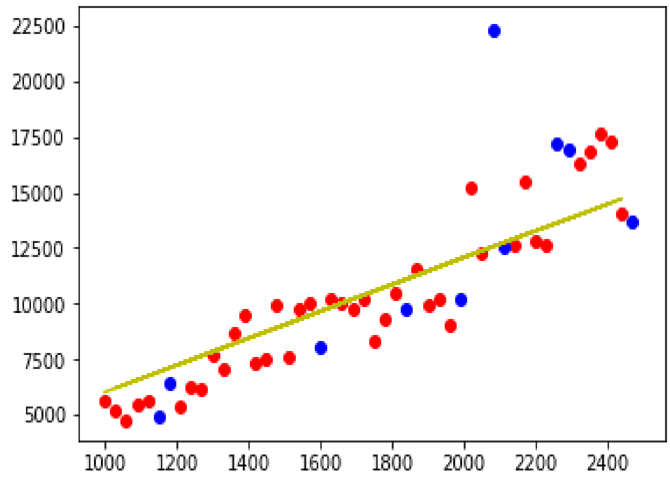
```
lr_model.score(x_test,y_test) #2.second way of calculating R2 score
```

R^2_Test_Score:0.5059420550739799

step8:Visualizing the model

```
plt.scatter(x_train[:,0],y_train,c='red')
```

```
plt.scatter(x_test[:,0],y_test,c='blue')

plt.plot(x_train[:,0],lr_model.predict(x_train),c='y')
```



Conclusion: Comparing the training and testing R^2 score values, the accuracy of the simple linear regression model with respect to this dataset is average.

Program 3

Apply:

a)Multiple linear regression model for student dataset and predict writing skill of student based on the math skill and reading skill of the student using the Gradient descent method. Find out R^2 score for the predicted model.

```
#importing Libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
```

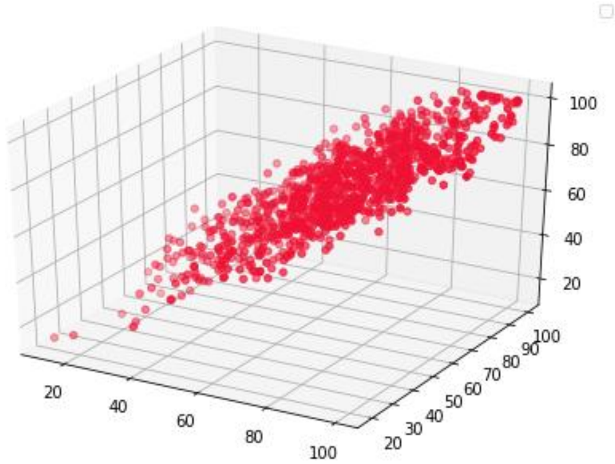
```
data = pd.read_csv('student.csv')
print(data.shape)
data.head()
```

(1000, 3)

	Math	Reading	Writing
0	48	68	63
1	62	81	72
2	79	80	78
3	76	83	79
4	59	64	62

```
math = data['Math'].values
read = data['Reading'].values
write = data['Writing'].values
```

```
# Ploting the scores as scatter plot
fig = plt.figure()
ax = Axes3D(fig)
ax.scatter(math, read, write, color='#ef1234')
plt.legend()
plt.show()
```




```
m = len(math)
x0 = np.ones(m)
X = np.array([x0, math, read]).T
```

```
# Initial Coefficients
B = np.array([0, 0, 0])
Y = np.array(write)
alpha = 0.0001
defcost_function(X, Y, B):
    m = len(Y)
    J = np.sum((X.dot(B) - Y) ** 2)/(2 * m)
    return J

inital_cost = cost_function(X, Y, B)
print("Initial Cost")
print(inital_cost)

defgradient_descent(X, Y, B, alpha, iterations):
cost_history = [0] * iterations
    m = len(Y)

    for iteration in range(iterations):
        # Hypothesis Values
        h = X.dot(B)
        # Difference b/w Hypothesis and Actual Y
        loss = h - Y
        # Gradient Calculation
```

```
        gradient = X.T.dot(loss) / m
        # Changing Values of B using Gradient
        B = B - alpha * gradient
        # New Cost Value
cost = cost_function(X, Y, B)
cost_history[iteration] = cost

    return B, cost_history

# 100000 Iterations
newB, cost_history = gradient_descent(X, Y, B, alpha, 100000)

# New Values of B
print("New Coefficients")
print(newB)

# Final Cost of new B
print("Final Cost")
print(cost_history[-1])
```

```
Initial Cost
2470.11
New Coefficients
[bo, b1,b2]:[-0.47889172  0.09137252  0.90144884]
Final Cost
10.475123473539167
```

```
# Model Evaluation - RMSE
defrmse(Y, Y_pred):
rmse = np.sqrt(sum((Y - Y_pred) ** 2) / len(Y))
    return rmse
```

```
# Model Evaluation - R2 Score
def r2_score(Y, Y_pred):
mean_y = np.mean(Y)
ss_tot = sum((Y - mean_y) ** 2)
ss_res = sum((Y - Y_pred) ** 2)
    r2 = 1 - (ss_res / ss_tot)
```

```
    return r2


Y_pred = X.dot(newB)


print("R2 Score")
print(r2_score(Y, Y_pred))
```

R^2 Score
0.9097223273061553

Conclusion:
The accuracy of the multiple linear regression model is good depending on the R^2score value.

b.) Multiple linear regression model for housing_prices dataset and predict house price based on the area, floor and room size of the house using the library scikit learn. Find out the accuracy of the model using R^2score statistics for the predicted model.

#importing libraries

```
import numpy as np
import pandas as pd
importmatplotlib.pyplot as plt
%matplotlib  inline
```

```
#Loading dataset
df=pd.read_csv("housing_prices.csv")
df.head()
```

	AREA	FLOOR	ROOM	PRICE
0	1000	7	2	5618
1	1030	7	1	5201
2	1060	1	1	4779
3	1090	6	1	5425
4	1120	0	2	5657

```
#setting Target and Feature Vectors
x=df.iloc[:,:3].values
y=df.iloc[:,3].values
```

```
#Splittiing the dataset
fromsklearn.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=100)
```

```
# Fitting the model
from sklearn.linear_model import LinearRegression
mlr_model= LinearRegression(fit_intercept=True)
mlr_model.fit(x_train,y_train)
print(mlr_model.intercept_) # (PRICE=(-4481.80028058845)+8.65903854)*AREA
print(mlr_model.coef_)
```

```
b0:-3106.4127920034116
[b1,b2,b3]:[   4.68576316   71.78274093 1894.45529322]
```

```
# Finding R2 score
print(mlr_model.score(x_train,y_train))
print(mlr_model.score(x_test,y_test))
```

```
R2_Train_Score:0.9220702400776505
R2_Test_Score:0.8090037959414931
```

Conclusion:**The multiple linear regression model accuracy is good with respect to this dataset by comparing R2 training and testing score values.**

<u>Program 4</u>

Apply:

a) Decision tree on breast cancer dataset.
Find out
i) No of benign and malignant cases in the testing phase.
ii) Predict the accuracy of the both classifier.

```
# ## Implementation of Decision Trees

# ### Step 1 : Load required packages

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


# ### Step 2 : Load the csv/excel file into pandas dataframe and clean the data

df = pd.read_csv("../data/breast_cancer.csv")
df = df.iloc[:, :-1]
df.head()


# ### Step 3 : Create the Feature Matrix and Target Vector and check the first 5 rows

x = df.iloc[:, 2:].values
y = df.diagnosis.values

print(x[:2])
print(y[:5])


# ### Step 4 : Split the data into 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)

# ### Step 5 : Instantiate a decision tree model and train the model

from sklearn.tree import DecisionTreeClassifier

dt_classifier = DecisionTreeClassifier()
dt_classifier.fit(x_train, y_train)


# ### Step 6 : Use the model to predict the class labels for new data
```

```
predictions = dt_classifier.predict(x_test)
prob_predictions = dt_classifier.predict_proba(x_test)

print(predictions)
print(prob_predictions)

# ### Step 7 : Calculate Accuracy score and confusion matrix for train and test data

from sklearn.metrics import accuracy_score, confusion_matrix

print("Training accuracy Score is : ", accuracy_score(y_train, dt_classifier.predict(x_train)))
print("Testing accuracy Score is : ", accuracy_score(y_test, dt_classifier.predict(x_test)))

print("Training Confusion Matrix is : \n", confusion_matrix(y_train, dt_classifier.predict(x_train)))

print("Testing Confusion Matrix is : \n", confusion_matrix(y_test, dt_classifier.predict(x_test)))
```

Output:

```
Training accuracy Score is :  1.0
Testing accuracy Score is :  0.9385964912280702

Training Confusion Matrix is:
 [[286   0]
 [0 169]]

Testing Confusion Matrix is:
 [[71  0]
 [ 7 36]]
```

Conclusion:

Comparing Training and testing accuracy scores the accuracy of Decision Tree model is good. The Correctly classified tuples for training set is (286+169) and the misclassified tuples are zero.The correctly classified for training set is (71+36)and misclassified tuples are(7+0).

4b. Apply Naïve Bayesian classifier on breast cancer dataset.

Find out

i) No of benign and malignant cases in the testing phase.

ii) Predict the accuracy of the classifier

```
# coding: utf-8

# ## Implementation of Naïve Bayes Algorithm

# ### Step 1 : Load required packages


import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import sklearn as sk

# ### Step 2 : Load the csv/excel file into pandas dataframeand clean the data

df = pd.read_csv("breast_cancer.csv")
df = df.iloc[:, :-1]
df.shape()
df.head()

# ### Step 3 : Create the Feature Matrix and Target Vector and check the first 5 rows

x = df.iloc[:, 2:].values
y = df.diagnosis.values

print(x[:2])
print(y[:5])


# ### Step 4 : Split the data into training set and test set

fromsklearn.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=500)

x_train.shape  #(455,30)

 x_test.shape#(114, 30)
y_train.shape

y_test.shape

(y_train == 'M').sum()

(y_train=='B').sum()
# Baseline model, accuracy, confusion_matrix,   classification_report
```

Step 5 : Instantiate a Guassian Naive Bayes model and train the model

278/len(y_train) # Baseline model of accuracy =(more number of occurrences)/total data elements

from sklearn.metrics import accuracy_score, confusion_matrix,classification_report

baseline_pred=["B"] *len(y_train) # baseline will have beningn for everything

Baseline model of accuracy :0.610989010989011

accuracy_score(y_train,baseline_pred) # takes actual and predicted as 2 arguments

confusion_matrix(y_train,baseline_pred)# takes actual and predicted as 2 arguments

```
from sklearn.naive_bayes import GaussianNB
nb_model=GaussianNB()
nb_model.fit(x_train,y_train)
print(x_train)
nb_model.score(x_train,y_train)
nb_model.score(x_test,y_test)
```

#confusion_matrix for training data

confusion_matrix(y_train,nb_model.predict(x_train))

Training Confusion Matrix:

```
  array([[269,   9],
         [ 22, 155]],
    dtype=int64)
```

#confusion_matrix for test data
confusion_matrix(y_test,nb_model.predict(x_test))

Testing Confusion Matrix:

```
array([[78,  1],
       [ 2, 33]],
dtype=int64)
```

print(classification_report(y_train,nb_model.predict(x_train)))

```
     precision    recall  f1-score   support

B       0.92      0.97      0.95       278
M       0.95      0.88      0.91       177

   avg / total       0.93      0.93      0.93       455
```

print(classification_report(y_test,nb_model.predict(x_test)))

```
             precision    recall  f1-score   support

          B       0.97      0.99      0.98        79
          M       0.97      0.94      0.96        35

avg / total       0.97      0.97      0.97       114
```

Conclusion: The naïve bayes model is good with respect to breast cancer dataset by comparing the precision recall and F1 score values of training and testing dataset (classification report)

Program 5:

Apply:

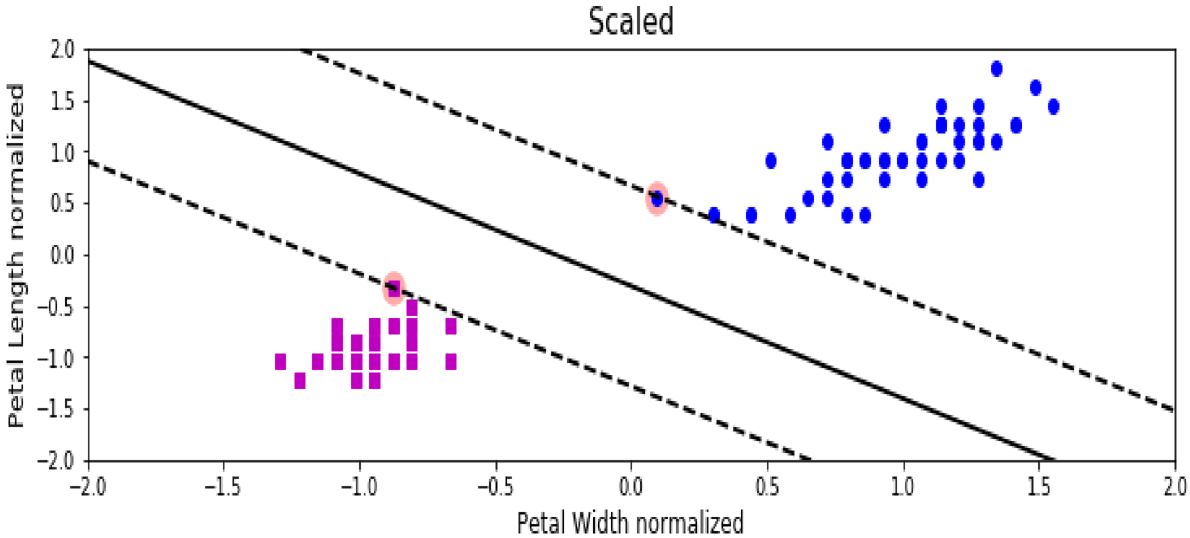
SVM classifier on:

a) Iris Dataset, Draw Linearly separable decision boundary for the generated dataset.

```
#Example of a Linear SVM Classifier (SVC) with hard margin decision boundaries

%matplotlib inline
import matplotlib
import matplotlib.pyplot as plt

defplot_svc_decision_boundary(svm_clf, xmin, xmax):
    w = svm_clf.coef_[0]
    b = svm_clf.intercept_[0]

    # At the decision boundary, w0*x0 + w1*x1 + b = 0
    # => x1 = -w0/w1 * x0 - b/w1
x0 = np.linspace(xmin, xmax, 200)
decision_boundary = -w[0]/w[1] * x0 - b/w[1]

margin = 1/w[1]
gutter_up = decision_boundary + margin
gutter_down = decision_boundary - margin

svs = svm_clf.support_vectors_
plt.scatter(svs[:, 0], svs[:, 1], s=180, facecolors='#FFAAAA')
plt.plot(x0, decision_boundary, "k-", linewidth=2)
plt.plot(x0, gutter_up, "k--", linewidth=2)
plt.plot(x0, gutter_down, "k--", linewidth=2)


#In [3]:
from sklearn.svm import SVC
from sklearn import datasets

iris = datasets.load_iris()
#print(iris)
X = iris["data"][:, (2, 3)]  # petal length, petal width
#print(X)

y = iris["target"]

setosa_or_versicolor = (y == 0) | (y == 1)
X = X[setosa_or_versicolor]
y = y[setosa_or_versicolor]

# SVM Classifier model
#the hyperparameter control the margin violations
#smaller C leads to more margin violations but wider street
#C can be inferred
svm_clf = SVC(kernel="linear", C=float("inf"))
svm_clf.fit(X, y)
```

```
svm_clf.predict([[2.4, 3.1]])

#SVM classifiers do not output a probability like logistic regression classifiers

#plot the decision boundaries
import numpy as np

plt.figure(figsize=(12,3.2))

fromsklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
svm_clf.fit(X_scaled, y)

plt.plot(X_scaled[:, 0][y==1], X_scaled[:, 1][y==1], "bo")
plt.plot(X_scaled[:, 0][y==0], X_scaled[:, 1][y==0], "ms")
plot_svc_decision_boundary(svm_clf, -2, 2)
plt.xlabel("Petal Width normalized", fontsize=12)
plt.ylabel("Petal Length normalized", fontsize=12)
plt.title("Scaled", fontsize=16)
plt.axis([-2, 2, -2, 2])
```

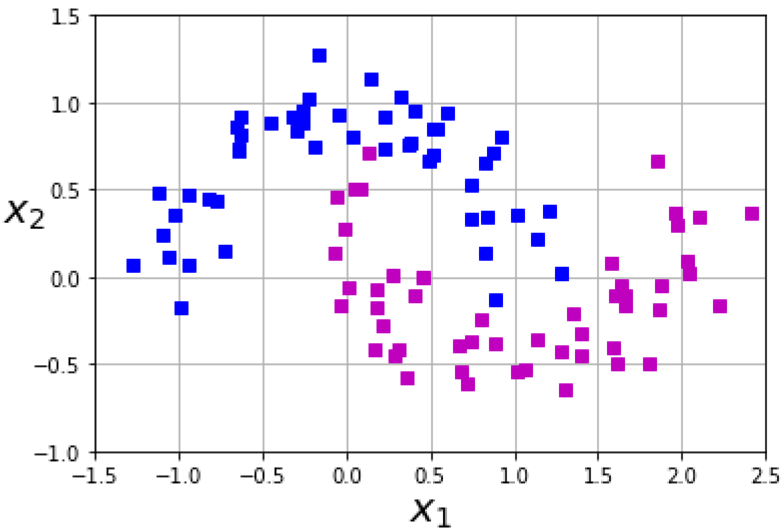
Output:



Conclusion: For Iris dataset SVM model is applied to linearly separate petal length and petal width with 2 support vectors.

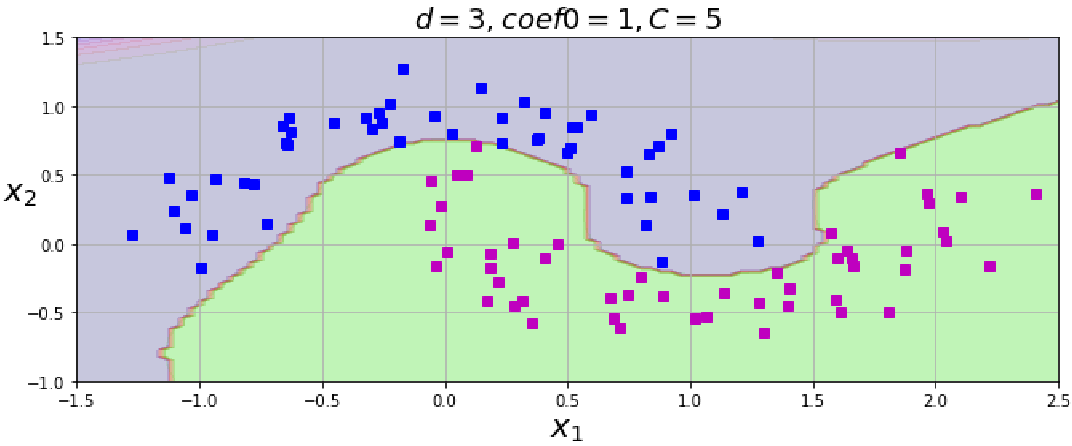
b)Randomly generated dataset using package library[MAKEMOON],Draw Non-linearly separable decision boundary for the generated dataset.

```
## Example of a Linear SVM Classifier (SVC) with hard margin decision boundaries

From sklearn.datasets import make_moons
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import Polynomial Features
from sklearn.preprocessing import Standard Scaler
from sklearn.svm import SVC

import numpy as np
%matplotlib inline
import matplotlib
import matplotlib.pyplot as plt

In[3]: ## Construct some test data
In[4]:
fromsklearn.datasets import make_moons
X, y = make_moons(n_samples=100, noise=0.15, random_state=42)

#define a function to plot the dataset
defplot_dataset(X, y, axes):
plt.plot(X[:, 0][y==0], X[:, 1][y==0], "bs")
plt.plot(X[:, 0][y==1], X[:, 1][y==1], "ms")
plt.axis(axes)
plt.grid(True, which='both')
plt.xlabel(r"$x_1$", fontsize=20)
plt.ylabel(r"$x_2$", fontsize=20, rotation=0)

#Let's have a look at the data we have generated
plot_dataset(X, y, [-1.5, 2.5, -1, 1.5])
plt.show()
```




```
#define a function plot the decision boundaries
defplot_predictions(clf, axes):
   #create data in continuous linear space
   x0s = np.linspace(axes[0], axes[1], 100)
```

```
    x1s = np.linspace(axes[2], axes[3], 100)
x0, x1 = np.meshgrid(x0s, x1s)
    X = np.c_[x0.ravel(), x1.ravel()]
y_pred = clf.predict(X).reshape(x0.shape)
y_decision = clf.decision_function(X).reshape(x0.shape)
plt.contourf(x0, x1, y_pred, cmap=plt.cm.brg, alpha=0.2)
plt.contourf(x0, x1, y_decision, cmap=plt.cm.brg, alpha=0.1)

## Build the model and set hyperparameters
#C controls the width of the street
#Degree of data

#create a pipeline to create features, scale data and fit the model
polynomial_svm_clf = Pipeline((
    ("poly_features", PolynomialFeatures(degree=3)),
    ("scalar", StandardScaler()),
    ("svm_clf", SVC(kernel="poly", degree=10, coef0=1, C=5)) ))

#call the pipeline
polynomial_svm_clf.fit(X,y)

## Plot the decision boundaries

#plot the decision boundaries
plt.figure(figsize=(11, 4))

#plot the decision boundaries
plot_predictions(polynomial_svm_clf, [-1.5, 2.5, -1, 1.5])

#plot the dataset
plot_dataset(X, y, [-1.5, 2.5, -1, 1.5])

plt.title(r"$d=3, coef0=1, C=5$", fontsize=18)
plt.show()
```



Conclusion: The moon dataset randomlygenerated. On this dataset SVM model is applied tonon-linearly separate X1 and X2 using polynomial kernel function.

<u>Program 6:</u>

Apply:

a)Partitioning k-means clustering technique on ch1ex1 dataset with different K (number of clusters) as input and record the output.

Step 1 and 2: Import the libraries and Load the dataset.

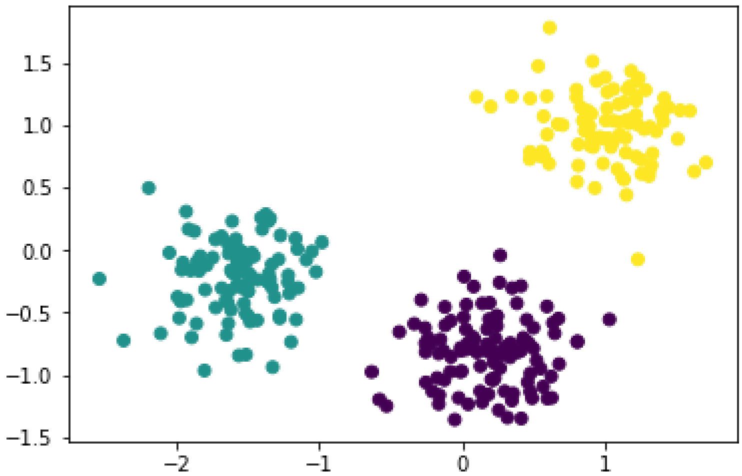
```
import pandas as pd

df = pd.read_csv('ch1ex1.csv')

points = df.values

from sklearn.cluster import KMeans

model = KMeans(n_clusters=3)

model.fit(points)

labels = model.predict(points)

importmatplotlib.pyplot as plt
```

Step 2: Assign column 0 of points to xs, and column 1 of points to ys

```
xs = points[:,0]

ys = points[:,1]
```

Step 3: Make a scatter plot of xs and ys, specifying the c=labels keyword arguments to color the points by their cluster label. You'll see that KMeans has done a good job of identifying the clusters!

```
plt.scatter(xs, ys, c=labels)

plt.show()
```



#This is great, but let's go one step further, and add the cluster centres (the "centroids") to the scatter plot.

Step 3: Obtain the coordinates of the centroids using the .cluster_centers_ attribute of model. Assign them to centroids.

centroids = model.cluster_centers_

Step 4: Assign column 0 of centroids to centroids_x, and column 1 of centroids to centroids_y.

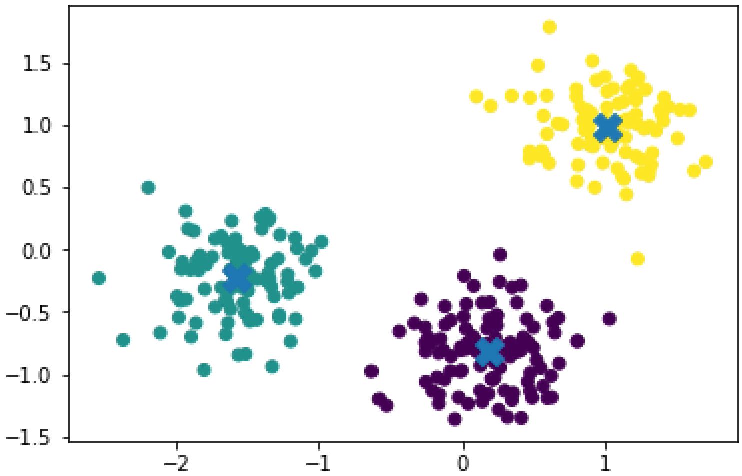
centroids_x = centroids[:,0]
centroids_y = centroids[:,1]

Step 5: In a single cell, create two scatter plots (this will show the two on top of one another). Call `plt.show()` just once, at the end.

Firstly, the make the scatter plot you made above. Secondly, make a scatter plot of `centroids_x` and `centroids_y`, using `'X'` (a cross) as a marker by specifying the `marker` parameter. Set the size of the markers to be `200` using `s=200`.

```
plt.scatter(xs, ys, c=labels)
plt.scatter(centroids_x, centroids_y, marker='X', s=200)
plt.show()
```

Output:



The centroids are important because they are what enables KMeans to assign new, previously unseen points to the existing clusters.

Conclusion: The k-means clustering technique is applied to ch1ex1 dataset to form clusters depending on the number of clusters as input. Then the centroid of the clustering is shown using the cross mark.

6b) Hierarchical Clustering Algorithm on seeds_less_rows dataset for extracting cluster labels of different varieties of seeds

#Extracting the cluster labels in heirarchial clustering

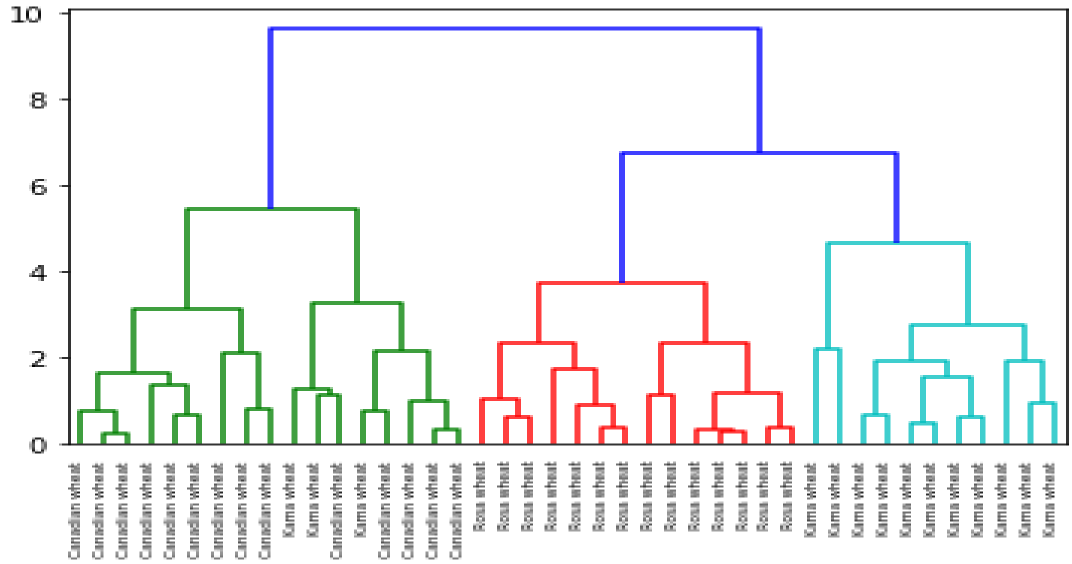
#we use the fcluster() function to extract the cluster labels for intermediate clustering, and #compare the labels with the grain varieties using a cross-tabulation.

Step 1 and 2: importing libraries and load the dataset:

```
import pandas as pd

seeds_df = pd.read_csv('seeds-less-rows.csv')

# remove the grain species from the DataFrame, save for later

varieties = list(seeds_df.pop('grain_variety'))

# extract the measurements as a NumPy array

samples = seeds_df.values
```

Step 3: Run the hierarchical clustering of the grain samples

```
fromscipy.cluster.hierarchy import linkage, dendrogram

importmatplotlib.pyplot as plt

mergings = linkage(samples, method='complete')

dendrogram(mergings,labels=varieties,leaf_rotation=90,leaf_font_size=6)

plt.show()
```



Step 4: Import fcluster from scipy.cluster.hierarchy

```
In[11]:  from scipy.cluster.hierarchy import fcluster
```

Step 5: Obtain a flat clustering by using the fcluster() function on mergings. Specify a maximum height of 6 and the keyword argument criterion='distance'. Assign the result to labels.

In[12]: labels = fcluster(mergings, 6, criterion='distance')

Step 6: Create a DataFramedf with two columns named 'labels' and 'varieties', using labels and varieties, respectively, for the column values.

In[13]: df = pd.DataFrame({'labels': labels, 'varieties': varieties})

Step 7: Create a cross-tabulation ct between df['labels'] and df['varieties'] to count the number of times each grain variety coincides with each cluster label.

In[14]: ct = pd.crosstab(df['labels'], df['varieties'])

Step 8: Display ct to see how your cluster labels correspond to the wheat varieties.

In[15]: ct

Output:-

Out[15]:

varieties	Canadian wheat	Kama wheat	Rosa wheat
labels			
1	14	3	0
2	0	0	14
3	0	11	0

Conclusion: Three varieties of labels extracted from 'seeds-less-rows' dataset by applying Hierarchical clustering technique as shown in the output table.

<u>Program 7</u>

Demonstrate:

a) Usage of Sigmoid activation function in artificial neural network

```
import numpy as np
from functools import reduce
def perceptron(weight, bias, x):
model = np.add(np.dot(x, weight), bias)
print('model: {}'.format(model))
logit = 1/(1+np.exp(-model))
print('Type: {}'.format(logit))
returnnp.round(logit)
def compute(logictype, weightdict, dataset):
weights = np.array([ weightdict[logictype][w] for w in weightdict[logictype].keys()])
output = np.array([ perceptron(weights, weightdict['bias'][logictype], val) for val in dataset])
    print(logictype)
    return logictype, output
def main():
    logic = {
        'logic_and' : {
            'w0': -0.1,
            'w1': 0.2,
            'w2': 0.2
        },
        'logic_nand': {
            'w0': 0.6,
            'w1': -0.8,
            'w2': -0.8
        },
        'bias': {
            'logic_and': -0.2,
            'logic_nand': 0.3,
```

```
        }
    }
dataset = np.array([
        [1,0,0],
        [1,0,1],
        [1,1,0],
        [1,1,1]   ])
logic_and = compute('logic_and', logic, dataset)
logic_nand = compute('logic_nand', logic, dataset)
def template(dataset, name, data):
   # act = name[6:]
print("Logic Function: {}".format(name[6:].upper()))
        print("X0\tX1\tX2\tY")
to Print = ["{1}\t{2}\t{3}\t{0}".format(output, *datas) for datas, output in zip(dataset, data)]
for i in to Print:
 print(i)
 gates = [logic_and, logic_nand]
for i in gates:
template(dataset, *i)
if __name__ == '__main__':
main()
```

output:

model: -0.30000000000000004

Type: 0.425557483188341

model: -0.1

Type: 0.47502081252106

model: -0.1

Type: 0.47502081252106

model: 0.10000000000000003

Type: 0.52497918747894

logic_and

model: 0.8999999999999999

Type: 0.7109495026250039

model: 0.09999999999999992

Type: 0.5249791874789399

model: 0.09999999999999992

Type: 0.5249791874789399

model: -0.7

Type: 0.3318122278318339

logic_nand

Logic Function: AND

X0	X1	X2	Y
1	0	0	0.0
1	0	1	0.0
1	1	0	0.0
1	1	1	1.0

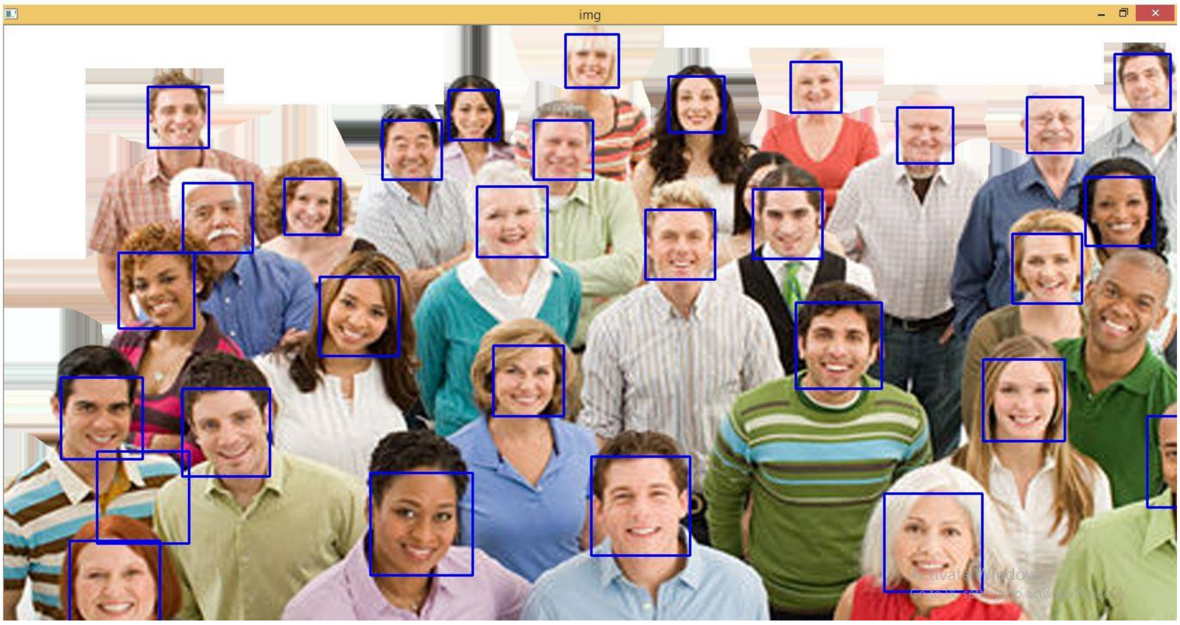
Logic Function: NAND

X0	X1	X2	Y
1	0	0	1.0
1	0	1	1.0
1	1	0	1.0
1	1	1	0.0

Conclusion: Sigmoid or logistic function used to display the working of AND and NAND logic functions.

7b)Identification of face using opencv library

```
#using opencv

    #install -c menpoopencv

    import numpy as np

    import cv2

    face_cascade = cv2.CascadeClassifier('haarcascade_frontalface_default.xml')

    img = cv2.imread('people.jpg')

    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

    faces = face_cascade.detectMultiScale(gray, 1.1, 5)

    for (x,y,w,h) in faces:

    cv2.rectangle(img,(x,y),(x+w,y+h),(255,0,0),2)

    roi_gray = gray[y:y+h, x:x+w]

    roi_color = img[y:y+h, x:x+w]

    cv2.imshow('img',img)

    cv2.waitKey(0)

    cv2.destroyAllWindows()
```



Conclusion: Using open cv library of Neural Networks, faces are detected.

Program 8

Using Keras and Tensor flow framework

i) Load the Pima_indians_diabetes dataset
ii) Design a two-layer neural network   with one hidden layer and one output layer
  a. Use Relu activation function for the hidden layer
  b. Use sigmoid activation function for the output layer
iii) Train the designed network for Pima_indians_diabetes
iv)Evaluate the network
v) Generate Predictions for 10 samples

Seven key steps in using Keras to create a neural network or deep learning model, step-by-step including:

1) Importing necessary Libraries 2) How to load data. 3) How to define a neural network in Keras. 4) How to compile a Keras model using the efficient numerical backend. 5) How to train a model on data. 6) How to evaluate a model on data. 7) How to make predictions with the model.

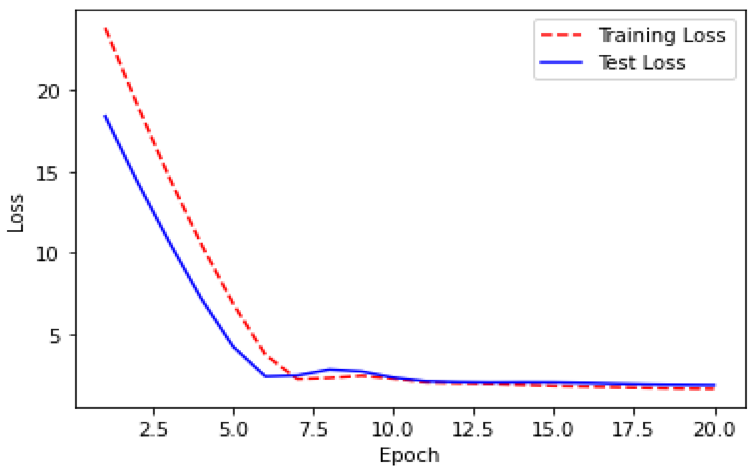
```
# first neural network with keras tutorial
from numpy import loadtxt
import numpy as np
import pandas as pd
from keras import models
from keras.models import Sequential
from keras.layers import Dense
from keras import layers
from sklearn.model_selection import train_test_split
from sklearn import preprocessing
import matplotlib.pyplot as plt
dataframe=pd.read_csv('pima-indians-diabetes.csv',delimiter=',')
dataframe.head()
```

	6	148	72	35	0	33.6	0.627	50	1
0	1	85	66	29	0	26.6	0.351	31	0
1	8	183	64	0	0	23.3	0.672	32	1
2	1	89	66	23	94	28.1	0.167	21	0
3	0	137	40	35	168	43.1	2.288	33	1
4	5	116	74	0	0	25.6	0.201	30	0

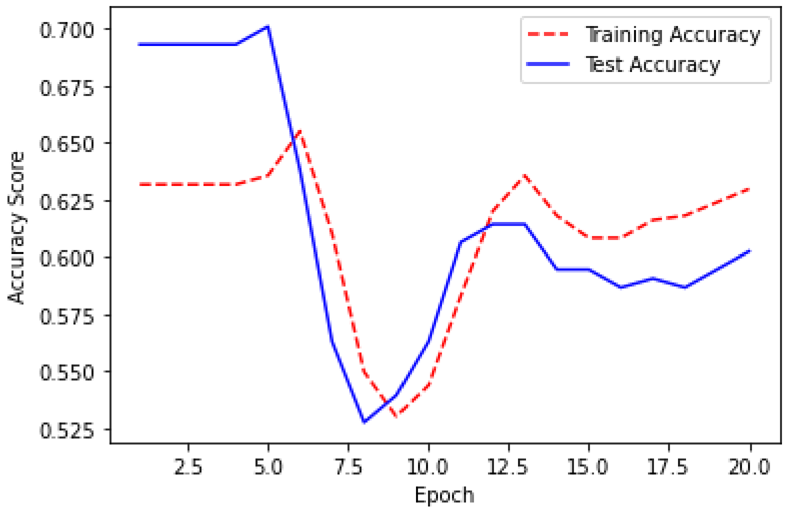
```
# split into input (X) and output (y) variables
X=dataframe.iloc[:,:8]
y=dataframe.iloc[:,8]

dataframe.shape
(767, 9)
```

```
features_train,features_test,target_train,target_test=train_test_split(X,y,
test_size=0.33,random_state=0)

# define the keras model
network=models.Sequential()
network.add(Dense(units=8,activation="relu",input_shape=(features_train.sha
pe[1],)))
network.add(Dense(units=8,activation="relu"))
#network.add(Dense(units=16,activation="relu"))
network.add(Dense(units=1,activation="sigmoid"))
# compile the keras model
network.compile(loss='binary_crossentropy',optimizer='adam',metrics=['accur
acy'])
#network.compile(loss='mse', optimizer='RMSprop', metrics=['accuracy'])
# fit the keras model on the dataset
#network.fit(features_train,features_test, epochs=10, batch_size=100,verbos
e=2)
history=network.fit(features_train,target_train,epochs=20,verbose=1,batch_s
ize=100,validation_data=(features_test,target_test))
Train on 513 samples, validate on 254 samples
Epoch 1/20
513/513 [==============================] - 0s 327us/step - loss: 23.8525 -
accuracy: 0.6316 - val_loss: 18.4057 - val_accuracy: 0.6929
Epoch 2/20
513/513 [==============================] - 0s 29us/step - loss: 19.1240 - a
ccuracy: 0.6316 - val_loss: 14.3790 - val_accuracy: 0.6929
Epoch 3/20
513/513 [==============================] - 0s 39us/step - loss: 14.6355 - a
ccuracy: 0.6316 - val_loss: 10.6533 - val_accuracy: 0.6929
Epoch 4/20
513/513 [==============================] - 0s 47us/step - loss: 10.5196 - a
ccuracy: 0.6316 - val_loss: 7.1659 - val_accuracy: 0.6929
Epoch 5/20
513/513 [==============================] - 0s 45us/step - loss: 6.8415 - ac
curacy: 0.6355 - val_loss: 4.1935 - val_accuracy: 0.7008
Epoch 6/20
513/513 [==============================] - 0s 43us/step - loss: 3.7177 - ac
curacy: 0.6550 - val_loss: 2.3824 - val_accuracy: 0.6378
Epoch 7/20
513/513 [==============================] - 0s 33us/step - loss: 2.2131 - ac
curacy: 0.6101 - val_loss: 2.4434 - val_accuracy: 0.5630
Epoch 8/20
513/513 [==============================] - 0s 37us/step - loss: 2.2830 - ac
curacy: 0.5497 - val_loss: 2.8009 - val_accuracy: 0.5276
Epoch 9/20
513/513 [==============================] - 0s 37us/step - loss: 2.4204 - ac
curacy: 0.5302 - val_loss: 2.6900 - val_accuracy: 0.5394
Epoch 10/20
513/513 [==============================] - 0s 39us/step - loss: 2.2307 - ac
curacy: 0.5439 - val_loss: 2.3109 - val_accuracy: 0.5630
Epoch 11/20
```

```
513/513 [==============================] - 0s 49us/step - loss: 2.0121 - ac
curacy: 0.5828 - val_loss: 2.0812 - val_accuracy: 0.6063
Epoch 12/20
513/513 [==============================] - 0s 45us/step - loss: 1.9620 - ac
curacy: 0.6199 - val_loss: 2.0272 - val_accuracy: 0.6142
Epoch 13/20
513/513 [==============================] - 0s 37us/step - loss: 1.9209 - ac
curacy: 0.6355 - val_loss: 2.0020 - val_accuracy: 0.6142
Epoch 14/20
513/513 [==============================] - 0s 49us/step - loss: 1.8549 - ac
curacy: 0.6179 - val_loss: 2.0124 - val_accuracy: 0.5945
Epoch 15/20
513/513 [==============================] - 0s 55us/step - loss: 1.7957 - ac
curacy: 0.6082 - val_loss: 2.0066 - val_accuracy: 0.5945
Epoch 16/20
513/513 [==============================] - 0s 45us/step - loss: 1.7566 - ac
curacy: 0.6082 - val_loss: 1.9706 - val_accuracy: 0.5866
Epoch 17/20
513/513 [==============================] - 0s 51us/step - loss: 1.7174 - ac
curacy: 0.6160 - val_loss: 1.9221 - val_accuracy: 0.5906
Epoch 18/20
513/513 [==============================] - 0s 39us/step - loss: 1.6742 - ac
curacy: 0.6179 - val_loss: 1.8809 - val_accuracy: 0.5866
Epoch 19/20
513/513 [==============================] - 0s 47us/step - loss: 1.6343 - ac
curacy: 0.6238 - val_loss: 1.8540 - val_accuracy: 0.5945
Epoch 20/20
513/513 [==============================] - 0s 49us/step - loss: 1.6173 - ac
curacy: 0.6296 - val_loss: 1.8372 - val_accuracy: 0.6024
training_loss=history.history["loss"]
test_loss=history.history["val_loss"]
epoch_count=range(1,len(training_loss)+1)
plt.plot(epoch_count,training_loss,"r--")
plt.plot(epoch_count,test_loss,"b-")
plt.legend(["Training Loss","Test Loss"])
plt.xlabel("Epoch")
plt.ylabel("Loss")
plt.show()
```



```
_,accuracy=network.evaluate(features_train,target_train)
print('Accuracy: %.2f'%(accuracy*100))
513/513 [==============================] - 0s 215us/step
Accuracy: 63.16
# predict using the keras model
predicted_target=network.predict(features_test)
_,accuracy=network.evaluate(features_test,target_test)
print('Accuracy: %.2f'%(accuracy*100))
254/254 [==============================] - 0s 35us/step
Accuracy: 60.24
#Y=target_train
foriinrange(10):
print(predicted_target[i])
[0.44970706]
[0.4993118]
[0.9906837]
[0.44786653]
[0.02075692]
[0.03176354]
[0.999443]
[0.5751261]
[0.04377431]
[0.8482277]
training_accuracy=history.history["accuracy"]
test_accuracy=history.history["val_accuracy"]
plt.plot(epoch_count,training_accuracy,"r--")
plt.plot(epoch_count,test_accuracy,"b-")
plt.legend(["Training Accuracy","Test Accuracy"])
plt.xlabel("Epoch")
plt.ylabel("Accuracy Score")
plt.show()
```



Conclusion :Using Keras and Tensor flow framework loaded the Pima_indians_diabetes dataset and designed a two-layer neural network with one hidden layer and one output layer and generated predictions for 10 samples.

<u>Program 9:</u>

Using Keras and tensor flow network

i) Load the mnist image dataset
ii) Design a two-layer neural network with one hidden layer and one output layer
a. Use CNN with Leaky Relu activation function for the hidden layer
b. Use sigmoid activation function for the output layer
iii)Train the designed network for mnist dataset
iv)Visualize the results of
a) Training vs validation accuracy
b) Training vs Validation loss

```
import numpy as np
from keras.datasets import mnist
from keras.utils import to_categorical
import matplotlib.pyplot as plt
%matplotlib inline
```

```
Using TensorFlow backend.
import keras
from keras.models import Sequential,Input,Model
from keras.layers import Dense, Dropout, Flatten
from keras.layers import Conv2D, MaxPooling2D
from keras.layers.normalization import BatchNormalization
from keras.layers.advanced_activationsimport LeakyReLU
```

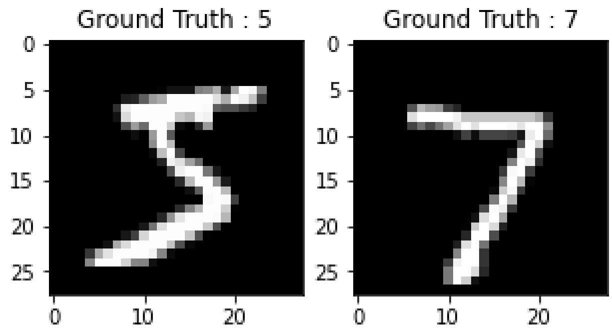
```
#from keras.datasets import mnist
(train_X,train_Y), (test_X,test_Y) = mnist.load_data()
```

```
print('Training data shape : ', train_X.shape, train_Y.shape)

print('Testing data shape : ', test_X.shape, test_Y.shape)
```

```
Training data shape :  (60000, 28, 28) (60000,)
Testing data shape :  (10000, 28, 28) (10000,)
# Find the unique numbers from the train labels
classes = np.unique(train_Y)
nClasses =len(classes)
print('Total number of outputs : ', nClasses)
print('Output classes : ', classes)
```

```
Total number of outputs :  10
Output classes :  [0 1 2 3 4 5 6 7 8 9]
plt.figure(figsize=[5,5])

# Display the first image in training data
plt.subplot(121)
plt.imshow(train_X[0,:,:], cmap='gray')
plt.title("Ground Truth : {}".format(train_Y[0]))

# Display the first image in testing data
plt.subplot(122)
plt.imshow(test_X[0,:,:], cmap='gray')
plt.title("Ground Truth : {}".format(test_Y[0]))
```

```
Text(0.5, 1.0, 'Ground Truth : 7')
```




```
train_X = train_X.reshape(-1, 28,28, 1)
test_X = test_X.reshape(-1, 28,28, 1)
train_X.shape, test_X.shape
```

```
((60000, 28, 28, 1), (10000, 28, 28, 1))
```

```
train_X = train_X.astype('float32')
test_X = test_X.astype('float32')
train_X = train_X /255
test_X = test_X /255

# Change the labels from categorical to one-hot encoding
train_Y_one_hot = to_categorical(train_Y)
test_Y_one_hot = to_categorical(test_Y)

# Display the change for category label using one-hot encoding
print('Original label:', train_Y[0])
print('After conversion to one-hot:', train_Y_one_hot[0])
```

```
Original label: 5
After conversion to one-hot: [0. 0. 0. 0. 0. 1. 0. 0. 0. 0.]
```

```
From sklearn.model_selection import train_test_split
train_X,valid_X,train_label,valid_label = train_test_split(train_X, train_Y
_one_hot, test_size=0.2, random_state=13)

train_X.shape,valid_X.shape,train_label.shape,valid_label.shape
```

```
((48000, 28, 28, 1), (12000, 28, 28, 1), (48000, 10), (12000, 10))
```

```
batch_size =64
epochs =3
num_classes =10

m_model = Sequential()
m_model.add(Conv2D(32, kernel_size=(3, 3),activation='linear',input_shape=(
28,28,1),padding='same'))
m_model.add(LeakyReLU(alpha=0.1))
m_model.add(MaxPooling2D((2, 2),padding='same'))
#fashion_model.add(Conv2D(64, (3, 3), activation='linear',padding='same'))
#fashion_model.add(LeakyReLU(alpha=0.1))
```

```
#fashion_model.add(MaxPooling2D(pool_size=(2, 2),padding='same'))
#fashion_model.add(Conv2D(128, (3, 3), activation='linear',padding='same'))
#fashion_model.add(LeakyReLU(alpha=0.1))
#fashion_model.add(MaxPooling2D(pool_size=(2, 2),padding='same'))
m_model.add(Flatten())
m_model.add(Dense(128, activation='linear'))
m_model.add(LeakyReLU(alpha=0.1))
m_model.add(Dense(num_classes, activation='softmax'))
```

```
m_model.compile(loss=keras.losses.categorical_crossentropy, optimizer=keras
.optimizers.Adam(),metrics=['accuracy'])
```

```
m_model.summary()
```

```
Model: "sequential_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_3 (Conv2D)            (None, 28, 28, 32)        320
_________________________________________________________________
leaky_re_lu_5 (LeakyReLU)    (None, 28, 28, 32)        0
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 14, 14, 32)        0
_________________________________________________________________
flatten_3 (Flatten)          (None, 6272)              0
_________________________________________________________________
dense_5 (Dense)              (None, 128)               802944
_________________________________________________________________
leaky_re_lu_6 (LeakyReLU)    (None, 128)               0
_________________________________________________________________
dense_6 (Dense)              (None, 10)                1290
=================================================================
Total params: 804,554
Trainable params: 804,554
Non-trainable params: 0
_________________________________________________________________
```

```
m_train = m_model.fit(train_X, train_label, batch_size=batch_size,epochs=ep
ochs,verbose=1,validation_data=(valid_X, valid_label))
```

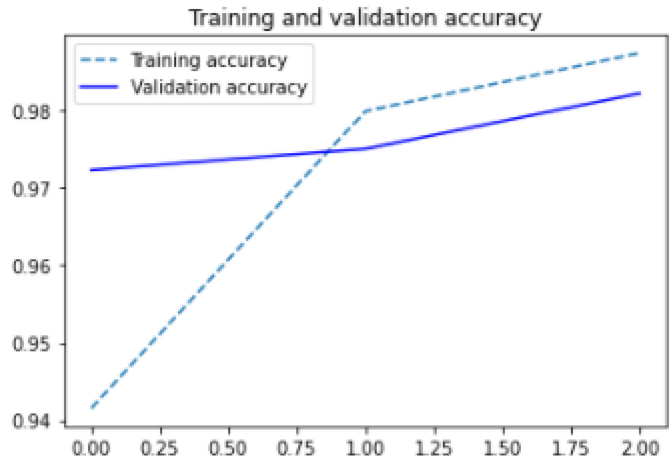
```
Train on 48000 samples, validate on 12000 samples
Epoch 1/3
48000/48000 [==============================] - 45s 928us/step - loss: 0.194
6 - accuracy: 0.9427 - val_loss: 0.0938 - val_accuracy: 0.9713
Epoch 2/3
48000/48000 [==============================] - 46s 948us/step - loss: 0.063
0 - accuracy: 0.9811 - val_loss: 0.0733 - val_accuracy: 0.9762
Epoch 3/3
48000/48000 [==============================] - 43s 897us/step - loss: 0.043
3 - accuracy: 0.9871 - val_loss: 0.0570 - val_accuracy: 0.9819
```

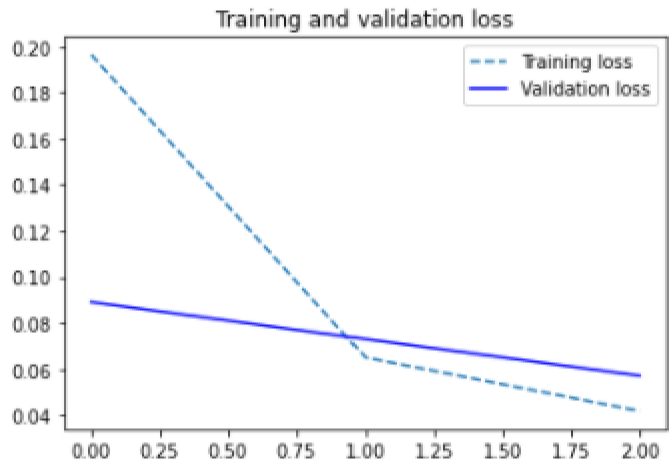
```
test_eval = m_model.evaluate(test_X, test_Y_one_hot, verbose=0)
```

```
print('Test loss:', test_eval[0])
print('Test accuracy:', test_eval[1])
```

```
Test loss: 0.05222021067142486
Test accuracy: 0.9824000000953674
```

```
accuracy = m_train.history['accuracy']
val_accuracy = m_train.history['val_accuracy']
loss = m_train.history['loss']
val_loss = m_train.history['val_loss']
epochs =range(len(accuracy))
plt.plot(epochs, accuracy, '--', label='Training accuracy')
plt.plot(epochs, val_accuracy, 'b', label='Validation accuracy')
plt.title('Training and validation accuracy')
plt.legend()
plt.figure()
plt.plot(epochs, loss, '--', label='Training loss')
plt.plot(epochs, val_loss, 'b', label='Validation loss')
plt.title('Training and validation loss')
plt.legend()
plt.show()
```






```
epochs=1
# ADDING DROPOUT
m_model = Sequential()
m_model.add(Conv2D(32, kernel_size=(3, 3),activation='linear',padding='same
',input_shape=(28,28,1)))
```

```
m_model.add(LeakyReLU(alpha=0.1))
m_model.add(MaxPooling2D((2, 2),padding='same'))
m_model.add(Dropout(0.25))
#fashion_model.add(Conv2D(64, (3, 3), activation='linear',padding='same'))
#fashion_model.add(LeakyReLU(alpha=0.1))
#fashion_model.add(MaxPooling2D(pool_size=(2, 2),padding='same'))
#fashion_model.add(Dropout(0.25))
#fashion_model.add(Conv2D(128, (3, 3), activation='linear',padding='same'))
#fashion_model.add(LeakyReLU(alpha=0.1))
#fashion_model.add(MaxPooling2D(pool_size=(2, 2),padding='same'))
#fashion_model.add(Dropout(0.4))
m_model.add(Flatten())
m_model.add(Dense(128, activation='linear'))
m_model.add(LeakyReLU(alpha=0.1))
m_model.add(Dropout(0.3))
m_model.add(Dense(num_classes, activation='softmax'))
```

```
m_model.summary()
```

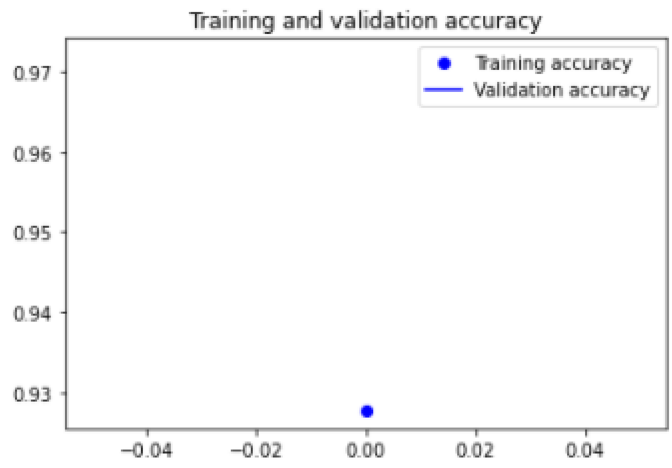
```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_2 (Conv2D)            (None, 28, 28, 32)        320
_________________________________________________________________
leaky_re_lu_3 (LeakyReLU)    (None, 28, 28, 32)        0
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 14, 14, 32)        0
_________________________________________________________________
dropout_1 (Dropout)          (None, 14, 14, 32)        0
_________________________________________________________________
flatten_2 (Flatten)          (None, 6272)              0
_________________________________________________________________
dense_3 (Dense)              (None, 128)               802944
_________________________________________________________________
leaky_re_lu_4 (LeakyReLU)    (None, 128)               0
_________________________________________________________________
dropout_2 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_4 (Dense)              (None, 10)                1290
=================================================================
Total params: 804,554
Trainable params: 804,554
Non-trainable params: 0
_________________________________________________________________
```

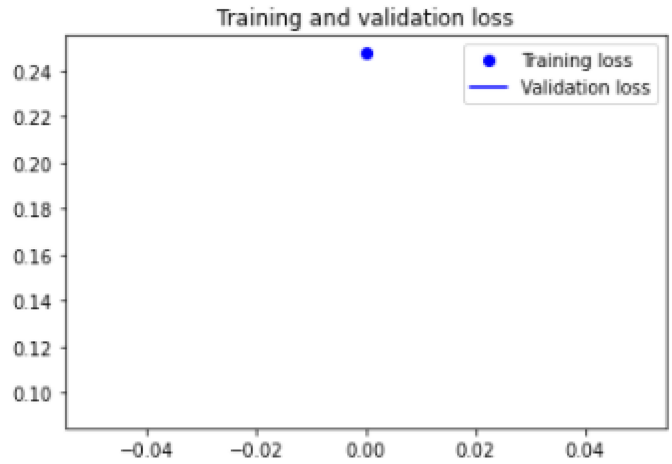
```
m_model.compile(loss=keras.losses.categorical_crossentropy, optimizer=keras
.optimizers.Adam(),metrics=['accuracy'])
```

```
m_train_dropout = m_model.fit(train_X, train_label, batch_size=batch_size,e
pochs=epochs,verbose=1,validation_data=(valid_X, valid_label))
```

```
Train on 48000 samples, validate on 12000 samples
Epoch 1/1
```

```
48000/48000 [==============================] - 49s 1ms/step - loss: 0.2479
- accuracy: 0.9265 - val_loss: 0.1026 - val_accuracy: 0.9700
m_model.save("fashion_model_dropout.h5py")
test_eval = m_model.evaluate(test_X, test_Y_one_hot, verbose=1)

10000/10000 [==============================] - 3s 263us/step

print('Test loss:', test_eval[0])
print('Test accuracy:', test_eval[1])

Test loss: 0.08918832793608308
Test accuracy: 0.9713000059127808

accuracy = m_train_dropout.history['accuracy']
val_accuracy = m_train_dropout.history['val_accuracy']
loss = m_train_dropout.history['loss']
val_loss = m_train_dropout.history['val_loss']
epochs =range(len(accuracy))
plt.plot(epochs, accuracy, 'bo', label='Training accuracy')
plt.plot(epochs, val_accuracy, 'b', label='Validation accuracy')
plt.title('Training and validation accuracy')
plt.legend()
plt.figure()
plt.plot(epochs, loss, 'bo', label='Training loss')
plt.plot(epochs, val_loss, 'b', label='Validation loss')
plt.title('Training and validation loss')
plt.legend()
plt.show()
```



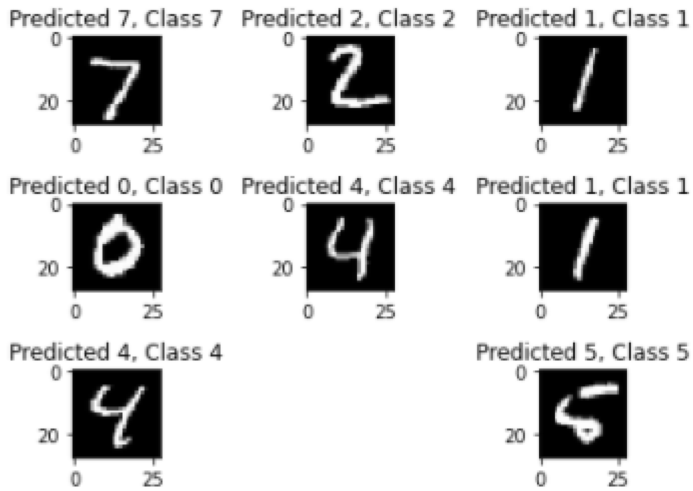



```
predicted_classes = m_model.predict(test_X)

predicted_classes = np.argmax(np.round(predicted_classes),axis=1)

predicted_classes.shape, test_Y.shape
```

```
((10000,), (10000,))
```

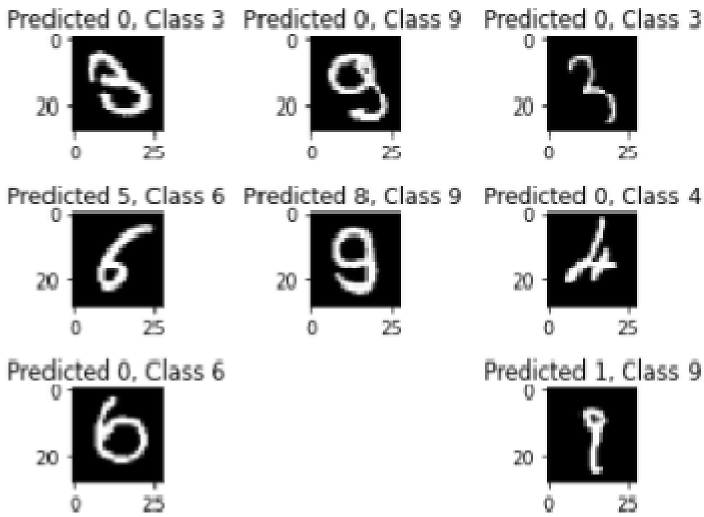
```
correct = np.where(predicted_classes==test_Y)[0]
print ("Found %d correct labels"%len(correct))
for i, correct inenumerate(correct[:9]):
    plt.subplot(3,3,i+1)
    plt.imshow(test_X[correct].reshape(28,28), cmap='gray', interpolation='
none')
    plt.title("Predicted {}, Class {}".format(predicted_classes[correct],
test_Y[correct]))
    plt.tight_layout()
```

```
Found 9680 correct labels
```




```
incorrect = np.where(predicted_classes!=test_Y)[0]
print ("Found %d incorrect labels"%len(incorrect))
for i, incorrect in enumerate(incorrect[:9]):
    plt.subplot(3,3,i+1)
    plt.imshow(test_X[incorrect].reshape(28,28), cmap='gray', interpolation
='none')
    plt.title("Predicted {}, Class {}".format(predicted_classes[incorrect],
test_Y[incorrect]))
    plt.tight_layout()
```

```
Found 320 incorrect labels
```




```
from sklearn.metrics import classification_report
target_names = ["Class {}".format(i) for i inrange(num_classes)]
print(classification_report(test_Y, predicted_classes, target_names=target_
names))
```

	precision	recall	f1-score	support
Class 0	0.90	0.99	0.94	980
Class 1	0.98	0.99	0.99	1135
Class 2	0.99	0.94	0.96	1032
Class 3	0.97	0.99	0.98	1010
Class 4	0.98	0.98	0.98	982

```
     Class 5       1.00      0.93      0.96       892
     Class 6       0.97      0.98      0.98       958
     Class 7       0.95      0.98      0.97      1028
     Class 8       0.97      0.95      0.96       974
     Class 9       0.99      0.94      0.96      1009

    accuracy                           0.97     10000
   macro avg       0.97      0.97      0.97     10000
weighted avg       0.97      0.97      0.97     10000
```

Conclusion: Using Keras and tensor flow network loaded the mnist image dataset and designed a two-layer neural network with one hidden layer and one output layer using CNN with Leaky Relu activation function for the hidden layer.

Program 10:

Using Keras and tensor flow network

i) Load the imdb text dataset
ii) Design a two-layer neural network with one hidden layer and one output layer
a. Use simple RNN in the hidden layer
b. Use sigmoid activation function for the output layer
iii) Train the designed network for imdb dataset
iv) Visualize the results of
a) Training vs validation accuracy
b) Training vs Validation loss

```
from keras.models import Sequential
from keras.layers import Embedding, SimpleRNN
from keras.datasets import imdb
from keras.preprocessing import sequence
from keras.layers import Dense
max_features =10000
maxlen =500
batch_size =32
print('Loading data...')
(input_train, y_train), (input_test, y_test) = imdb.load_data( num_words=ma
x_features)
#(input_train, y_train), (input_test, y_test) = imdb.load_data()
print(len(input_train), 'train sequences')
print(len(input_test), 'test sequences')
print('Pad sequences (samples x time)')
input_train = sequence.pad_sequences(input_train, maxlen=maxlen)
input_test = sequence.pad_sequences(input_test, maxlen=maxlen)
print('input_train shape:', input_train.shape)
print('input_test shape:', input_test.shape)
```

```
25000 train sequences
25000 test sequences
Pad sequences (samples x time)
input_train shape: (25000, 500)
input_test shape: (25000, 500)
```

```
model = Sequential()
model.add(Embedding(max_features, 32)) #max_feature=10,000 so, 320,000
model.add(SimpleRNN(32))                #(32+32+1)*32=2080
model.add(Dense(1, activation='sigmoid'))#(32+1)*1=33
model.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_2 (Embedding)      (None, None, 32)          320000
_________________________________________________________________
simple_rnn_2 (SimpleRNN)     (None, 32)                2080
```

```
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 33
=================================================================
Total params: 322,113
Trainable params: 322,113
Non-trainable params: 0
_________________________________________________________________
```

```
model.compile(optimizer='rmsprop', loss='binary_crossentropy',metrics=['acc
'])
history = model.fit(input_train, y_train,epochs=10, batch_size=128, validat
ion_split=0.2)
```

```
Train on 20000 samples, validate on 5000 samples
Epoch 1/10
20000/20000 [==============================] - 33s 2ms/step - loss: 0.5955
- acc: 0.6679 - val_loss: 0.5106 - val_acc: 0.7566
Epoch 2/10
20000/20000 [==============================] - 36s 2ms/step - loss: 0.3544
- acc: 0.8530 - val_loss: 0.4272 - val_acc: 0.8158
Epoch 3/10
20000/20000 [==============================] - 37s 2ms/step - loss: 0.2823
- acc: 0.8870 - val_loss: 0.3698 - val_acc: 0.8652
Epoch 4/10
20000/20000 [==============================] - 41s 2ms/step - loss: 0.2192
- acc: 0.9174 - val_loss: 0.4816 - val_acc: 0.7870
Epoch 5/10
20000/20000 [==============================] - 36s 2ms/step - loss: 0.1675
- acc: 0.9376 - val_loss: 0.4021 - val_acc: 0.8440
Epoch 6/10
20000/20000 [==============================] - 32s 2ms/step - loss: 0.1261
- acc: 0.9570 - val_loss: 0.4502 - val_acc: 0.8312
Epoch 7/10
20000/20000 [==============================] - 32s 2ms/step - loss: 0.0758
- acc: 0.9740 - val_loss: 0.4815 - val_acc: 0.8328
Epoch 8/10
20000/20000 [==============================] - 35s 2ms/step - loss: 0.0552
- acc: 0.9829 - val_loss: 0.5122 - val_acc: 0.8474
Epoch 9/10
20000/20000 [==============================] - 33s 2ms/step - loss: 0.0313
- acc: 0.9908 - val_loss: 0.5852 - val_acc: 0.8282
Epoch 10/10
20000/20000 [==============================] - 32s 2ms/step - loss: 0.0239
- acc: 0.9933 - val_loss: 0.6137 - val_acc: 0.8376
```

```
predicted_classes = model.predict(input_test)
```

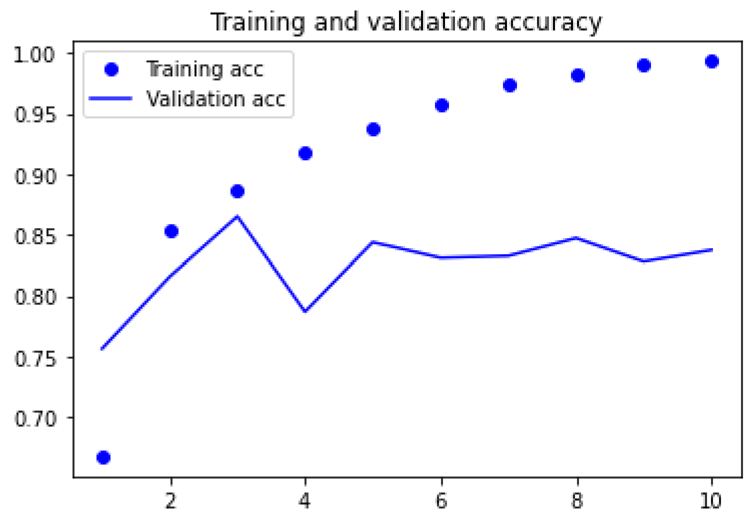
```
import numpy as np
predicted_classes = np.argmax(np.round(predicted_classes),axis=1)
```

```
predicted_classes.shape, y_test.shape
```

```
 ((25000,), (25000,))
```

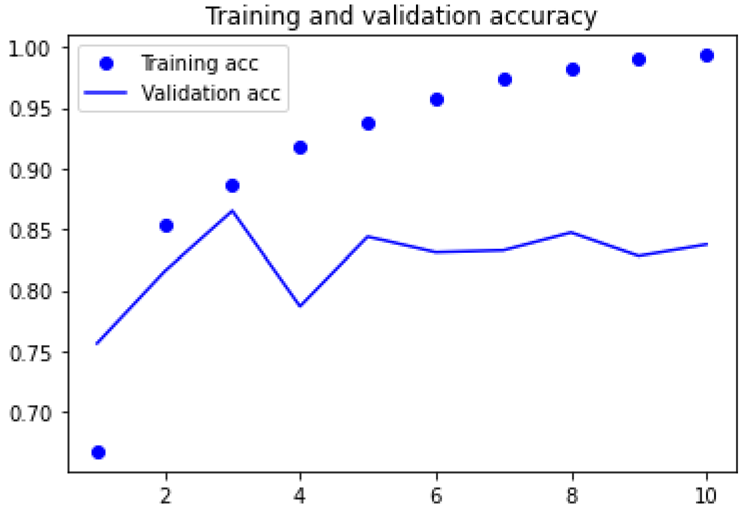
```
correct = np.where(predicted_classes==y_test)[0]
print ("Found %d correct labels"%len(correct))
```

```
Found 12500 correct labels
incorrect = np.where(predicted_classes!=y_test)[0]
print ("Found %d incorrect labels"%len(incorrect))
Found 12500 incorrect labels

from sklearn.metrics import classification_report
num_classes=2
target_names = ["Class {}".format(i) for i inrange(num_classes)]
print(classification_report(y_test, predicted_classes, target_names=target_
names))

              precision    recall  f1-score   support

     Class 0       0.50      1.00      0.67     12500
     Class 1       0.00      0.00      0.00     12500

    accuracy                           0.50     25000
   macro avg       0.25      0.50      0.33     25000
weighted avg       0.25      0.50      0.33     25000
_warn_prf(average, modifier, msg_start, len(result))

import matplotlib.pyplot as plt
acc = history.history['acc']
val_acc = history.history['val_acc']
epochs =range(1, len(acc) +1)
plt.plot(epochs, acc, 'bo', label='Training acc')
plt.plot(epochs, val_acc, 'b', label='Validation acc')
plt.title('Training and validation accuracy')
plt.legend()
<matplotlib.legend.Legend at 0x22133e2fd08>
```




```
plt.figure()
loss = history.history['loss']
val_loss = history.history['val_loss']
epochs =range(1, len(acc) +1)
plt.plot(epochs, loss, 'bo', label='Training loss')
```

```
plt.plot(epochs, val_loss, 'b', label='Validation loss')
plt.title('Training and validation loss')
plt.legend()
plt.show().
```



Conclusion: Using Keras and tensor flow network loaded the imdb text dataset and designed a two-layer neural network with one hidden layer and one output layer using simple RNN in the hidden layer.