Name - Yash Gandhi BE IT Batch A 2017140014

Aim:

To build a classifier model for a given scenario

Problem Statement: Nowadays increased spam e-mails are causing inconvenience to internet users and organizations and are considered as a serious wastage of resources, time, memory, space and efforts. Therefore, it is crucial to have an automatic email classification system for the identification of spam emails. Spam mails need to be classified and separated from ham (non-spam) mails as they are the source of financial loss and annoyance for the recipients. Hence this paper proposes a multilayer perceptron neural network using the linear sigmoid activation function

Task 1:

1. **Problem that the paper solves**
   The paper tries to address the issue of filtering out spam emails effectively using MLP Classifier.
2. **Description of the component involved in the MLP:**
   a. **Input Representation**
      Input consists of a number of features which are found in an email. The percentage of different types of words appearing in the email is given as the input.

   b. **No. of hidden layers**
      Neural network is a feed-forward network, with Single input layer (57 inputs), a hidden layer (1) and a single Output layer (1output).

   c. **Activation Function(s) used**
      The network is a multilayer perceptron neural network using the linear sigmoid activation function

3. **Dataset description**
   By looking deeply through the scientists and soliciting the experience of data science experts on emails classification, a number of factors that are considered to have an effect on the classification of emails. These factors were cautiously studied and synchronized into a convenient number appropriate for computer coding within the environment of the Just NN modeling. These factors were classified as input variables. The output variables embody some likely levels of performance of an emails classification for spam or not.

```
| 48 continuous real [0,100] attributes of type word_freq_WORD
| = percentage of words in the e-mail that match WORD,
| i.e. 100 * (number of times the WORD appears in the e-mail) /
| total number of words in e-mail.  A "word" in this case is any
| string of alphanumeric characters bounded by non-alphanumeric
| characters or end-of-string.
|
```

| 6 continuous real [0,100] attributes of type char_freq_CHAR
| = percentage of characters in the e-mail that match CHAR,
| i.e. 100 * (number of CHAR occurences) / total characters in e-mail
|
| 1 continuous real [1,...] attribute of type capital_run_length_average
| = average length of uninterrupted sequences of capital letters
|
| 1 continuous integer [1,...] attribute of type capital_run_length_longest
| = length of longest uninterrupted sequence of capital letters
|
| 1 continuous integer [1,...] attribute of type capital_run_length_total
| = sum of length of uninterrupted sequences of capital letters
| = total number of capital letters in the e-mail
|
| 1 nominal {0,1} class attribute of type spam
| = denotes whether the e-mail was considered spam (1) or not (0),
| i.e. unsolicited commercial e-mail.

4. **Results observed by the authors.**
   Test data evaluation shows that the MLP NN model is able to correctly classify more than (85.31% of data set) of prospective emails. Figures 2,3,4,5 illustrates the spam Emails classification.
   A total of 4601 email records were used in the analysis. About 70% of the total data (3233 email) were used as the training set 30% (1368) for validation and testing.

5. **Observations and Conclusion**
   An artificial MultiLayer Perceptron Neural Network model for classification of emails was presented. The model used a feed forward backpropagation algorithm for training. The factors for the model were obtained from Mark Hopkins, Erik Reeber, George Forman, Jaap Suermondt Hewlett-Packard Labs,
   1501 Page Mill Rd., Palo Alto, CA 94304 records. The model was tested and the overall result was 85.31%. This study showed the potential of the artificial neural network for classification of emails.

Task 2: Implement the chosen research paper by using the same components mentioned in the paper. Also vary the hyperparameters (hidden layer neurons, learning rate, activation function, optimizer) of the model built and obtain a comparative analysis.

Tool/Language:
Programming language: Python
Libraries: numpy/pandas/matplotlib/sklearn

Report Summary:

An artificial MultiLayer Perceptron Neural Network model for classification of emails was presented. The model used a feed forward backpropagation algorithm for training.

Algorithm:
-Initialize each wi to some small random value
-Until the termination condition is met, Do
-For each training example <(x1…xn),t> Do
-INPUT THE INSTANCE (X1,…,XN) TO THE NETWORK AND COMPUTE THE NETWORK OUTPUTS Ok
- For each output unit k: Δk=ok (1-ok)(tk-ok)
- For each hidden unit h: Δh=oh (1-oh) Δk wh,k Δk
- For each network weight wj Do
- wi,j=wi,j+Δwi,j,where Δwi,j= Δj xi,j and Δ is the learning rate.

1. Load dataset:
2. features :57 features and target -| 1 nominal {0,1} class attribute of type spam
   | = denotes whether the e-mail was considered spam (1) or not (0),
   | i.e. unsolicited commercial e-mail.
3. Perform train-test split
4. Creation of MLP model according to the hyperparameters delineated in the paper
5. Train the model with the training dataset
6. Test the model with the testing dataset
7. Find accuracy score

Code:

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import sklearn
from sklearn.neural_network import MLPClassifier
from sklearn.neural_network import MLPRegressor

# Import necessary modules
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
from math import sqrt
from sklearn.metrics import r2_score
```

In [2]:
```
import warnings
warnings.filterwarnings('ignore')
```

In [3]:
```
df = pd.read_csv('spambase_csv.csv')
print(df.shape)
```

```
(4601, 58)
```

In [4]:
```
df.head(1)
```

Out[4]:

	word_freq_make	word_freq_address	word_freq_all	word_freq_3d	word_freq_our	word_freq_over	word_freq_remove	word_freq_internet	word_freq_order	v
0	0.0	0.64	0.64	0.0	0.32	0.0	0.0	0.0	0.0	

1 rows × 58 columns

MLP Classifier [1] with hidden layer: 3 with each layer having 8 neurons and relu activation function

In [5]:
```
target_column = ['class']
predictors = list(set(list(df.columns))-set(target_column))
```

In [6]:
```
X = df[predictors].values
y = df[target_column].values

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30, random_state=40)
print(X_train.shape); print(X_test.shape)
```

```
(3220, 57)
(1381, 57)
```

In [8]:
```
train=[]
test=[]
epoch_arr = []
epoch=10
while epoch <= 200:
    mlp = MLPClassifier(hidden_layer_sizes=(8,8,8), activation='relu', solver='sgd',
                        alpha=0.0001, batch_size=64, learning_rate='constant', learning_rate_init=0.001, max_iter=epoch,
                        shuffle=True, random_state=None, tol=0.0001, verbose=False)
    mlp.fit(X_train,y_train)
    y_pred_train = mlp.predict(X_train)
    score_train = accuracy_score(y_train, y_pred_train)
    train.append(score_train)

    y_pred = mlp.predict(X_test)
    score_test = accuracy_score(y_test, y_pred)
    test.append(score_test)
    epoch_arr.append(epoch)
    epoch+=10
```

MLP Classifier [2] with hidden layer: 3 with each layer having 100 neurons and tanh

activation function

```
In [18]: train=[]
         test=[]
         epoch_arr = []
         epoch=10
         while epoch <= 200 :
             mlp = MLPClassifier(hidden_layer_sizes=(100,100,100), activation='tanh', solver='sgd',
                                 alpha=0.0001, batch_size=64, learning_rate='constant', learning_rate_init=0.001, max_iter=epoch,
                                 shuffle=True, random_state=None, tol=0.0001, verbose=False)
             mlp.fit(X_train,y_train)
             y_pred_train = mlp.predict(X_train)
             score_train = accuracy_score(y_train, y_pred_train)
             train.append(score_train)

             y_pred = mlp.predict(X_test)
             score_test = accuracy_score(y_test, y_pred)
             test.append(score_test)
             epoch_arr.append(epoch)
             epoch+=10
```

Results: for classifier [1] training and testing confusion matrix and accuracy

```
In [8]: print(confusion_matrix(y_train,y_pred_train))
        print(classification_report(y_train,y_pred_train))
```

```
[[1922    4]
 [1294    0]]
             precision    recall  f1-score   support

          0       0.60      1.00      0.75      1926
          1       0.00      0.00      0.00      1294

avg / total       0.36      0.60      0.45      3220
```

```
In [9]: accuracy_score(y_train,y_pred_train)
Out[9]: 0.5968944099378882
```

```
In [10]: print(confusion_matrix(y_test,y_pred))
         print(classification_report(y_test,y_pred))
```

```
[[862   0]
 [519   0]]
             precision    recall  f1-score   support

          0       0.62      1.00      0.77       862
          1       0.00      0.00      0.00       519

avg / total       0.39      0.62      0.48      1381
```

```
In [11]: accuracy_score(y_test,y_pred)
Out[11]: 0.6241853729181752
```

Model output and loss

```
In [15]: print(y_test[0:10])
```

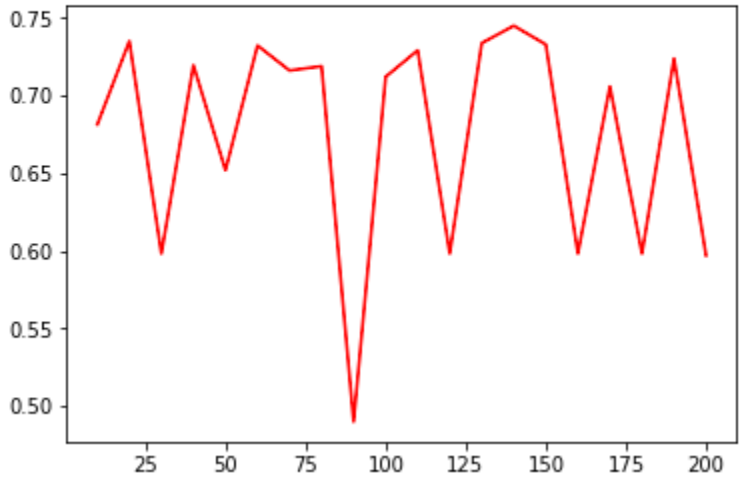
```
[[0]
 [0]
 [0]
 [0]
 [0]
 [1]
 [0]
 [1]
 [0]
 [1]]
```

```
In [16]: print(loss_values)
```

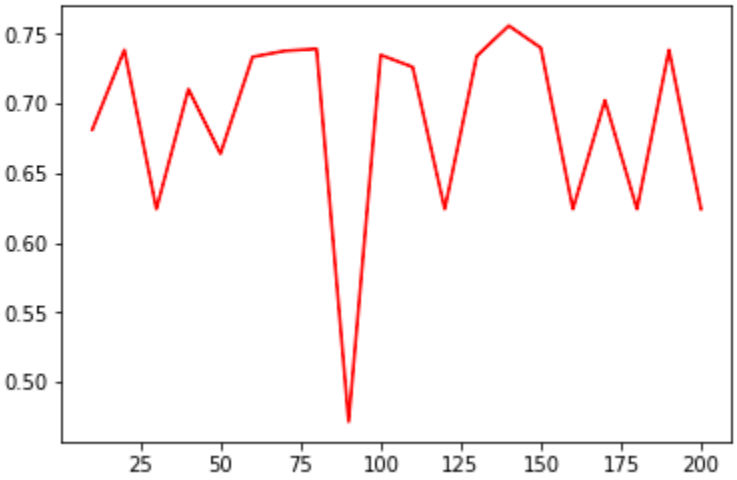
```
[1.5891352043970308, 0.7026433307513618, 0.6928747161788595, 0.6862662650846154, 0.6819858632641742, 0.6792426647244934, 0.677
4177691588736, 0.6762121827216908, 0.6754337099548614, 0.6749345131664146, 0.6745564403502968, 0.674315732321012, 0.6741553702
722864, 0.6740178424249432, 0.6739430336694435, 0.6738705391340238, 0.6738146749499829]
```

Accuracy score graph for training and testing and loss curve

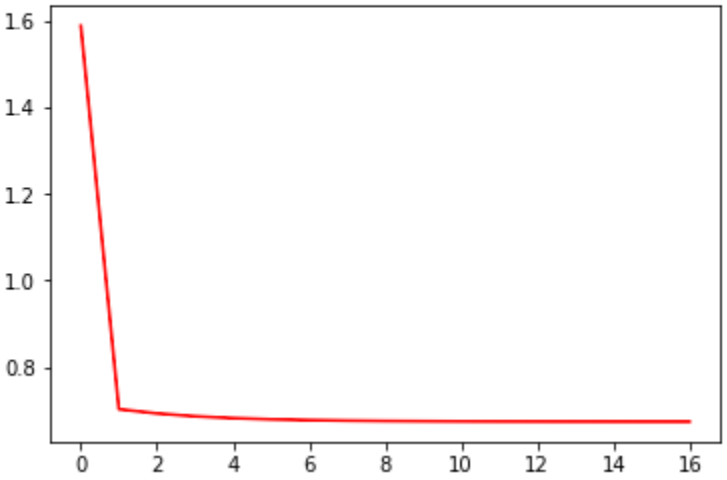
```
In [12]: plt.plot(epoch_arr,train,color="red")
         plt.show()
```



```
In [13]: plt.plot(epoch_arr,test,color="red")
         plt.show()
```




```
In [14]: loss_values = mlp.loss_curve_
         plt.plot(loss_values,color="red")
         plt.show()
```



Results: for classifier [2] training and testing confusion matrix and accuracy

```
In [18]: print(confusion_matrix(y_train,y_pred_train))
         print(classification_report(y_train,y_pred_train))
```

```
[[1539  387]
 [ 455  839]]
             precision    recall  f1-score   support

          0       0.77      0.80      0.79      1926
          1       0.68      0.65      0.67      1294

avg / total       0.74      0.74      0.74      3220
```

```
In [19]: accuracy_score(y_train,y_pred_train)
Out[19]: 0.7385093167701864
```

```
In [20]: print(confusion_matrix(y_test,y_pred))
         print(classification_report(y_test,y_pred))
```

```
[[688 174]
 [178 341]]
             precision    recall  f1-score   support

          0       0.79      0.80      0.80       862
          1       0.66      0.66      0.66       519

avg / total       0.74      0.75      0.74      1381
```

```
In [21]: accuracy_score(y_test,y_pred)
Out[21]: 0.7451122375090514
```

Model output and loss

```
In [25]: print(y_test[0:10])
```

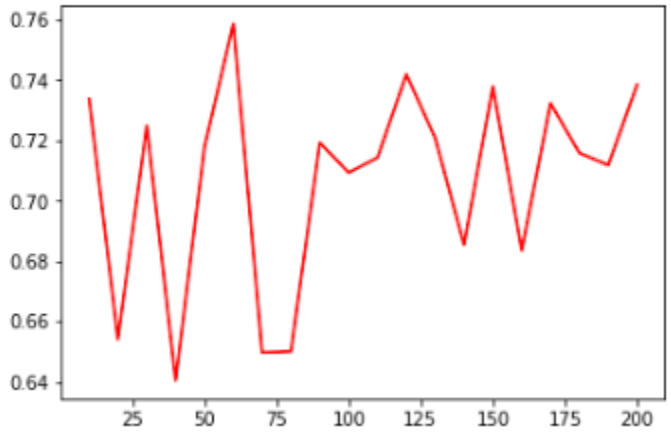
```
[[0]
 [0]
 [0]
 [0]
 [0]
 [1]
 [0]
 [1]
 [0]
 [1]]
```

```
In [26]: print(loss_values)
```

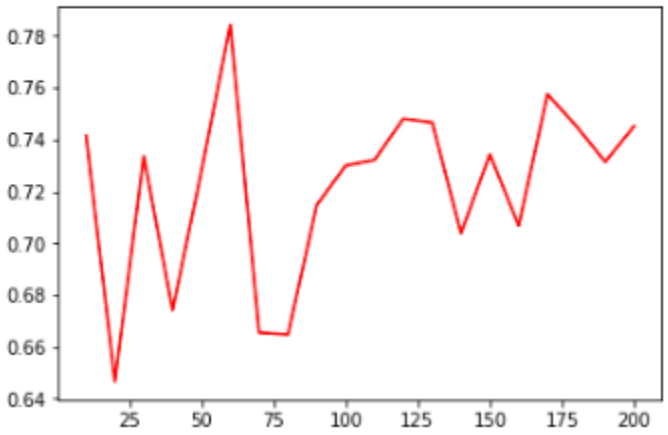
```
[0.6763489023418339, 0.5984557991767673, 0.5594979703999393, 0.5394973042327931, 0.554806840695968, 0.553064774903634, 0.5652666416850057]
```

Accuracy score graph for training and testing and loss curve

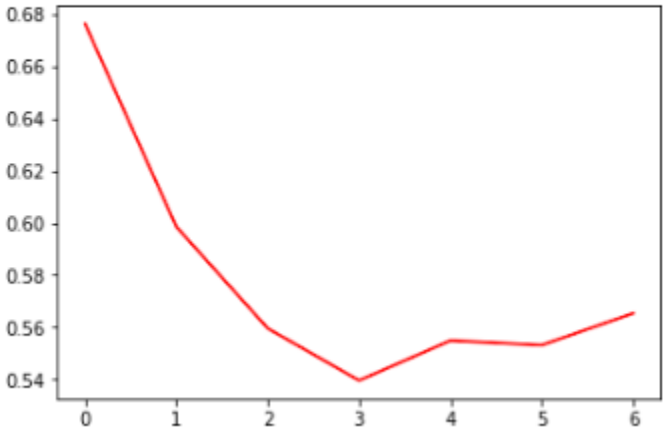
In [22]:
```
plt.plot(epoch_arr,train,color="red")
plt.show()
```



In [23]:
```
plt.plot(epoch_arr,test,color="red")
plt.show()
```



```
In [24]: loss_values = mlp.loss_curve_
         plt.plot(loss_values,color="red")
         plt.show()
```



Comparision:

Model	Training Accuracy	Testing Accuracy	Loss
8,8,8 Hidden layer,with relu activation function	0.59	0.62	0.67
100,100,100 Hidden layer with tanh activation function	0.73	0.74	0.66

Conclusion:

Multi-layer Perceptron Neural Network was implemented for detecting spam emails which classifies the records into 2 classes – spam and not spam.
The MLP classifier was implemented using the standard library function available in the sklearn library.The simulation was carried out using the hyperparameters mentioned in the paper as well as tuning them to achieve variable results. The model achieved 76% accuracy less than that proposed in the paper :85% The hyperparameters, number of epochs, activation function, the optimizer, learning rate were varied and the training accuracy, testing accuracy and loss were plotted and the variations were observed.

It was observed that more the number of hidden layer neurons more is the accuracy