Classifier Model

Problem Statement: Prediction of Heart Disease Using Multilayer Perceptron Neural Network

Task 1:

1. ***Problem that the paper solves:***

In the medical field the diagnosis of heart disease is the most difficult task. It depends on the careful analysis of different clinical and pathological data of the patient by medical experts, which is a complicated process. Due to advancement in machine learning and information technology, the researchers and medical practitioners in large extent are interested in the development of automated systems for the prediction of heart disease that is highly accurate, effective and helpful in early diagnosis.

There are a total of 463 instances in the dataset and the class distribution is as follows:
Absence of heart disease: 303
Presence of heart disease: 160

In this implementation, the classes are {0, 1}, representing the absence and presence of coronary heart disease respectively.

The features included in the dataset are the following:

1. sbp systolic blood pressure
2. tobacco cumulative tobacco (kg)
3. ldl low density lipoprotein cholesterol
4. adiposity
5. famhist family history of heart disease (Present, Absent)
6. typea type-A behavior
7. obesity
8. alcohol current alcohol consumption
9. age age at onset
10. chd response, coronary heart disease

Each feature has its (1) mean, (2) standard error, and (3) largest (mean of the three largest values) computed.

2. ***Description of the component involved in the MLP:***

The whole dataset in the experiment has been divided into two parts - training and testing. 80% of data has been taken for the training of the model and the remaining 20% used for
testing. The data is then re-scaled or normalized to get a better result. For re-scaling standard scalar is used.

- The activation function used for MLP is ReLU.
- There are two hidden layers that each consists of 500 nodes (500-500 architecture).
- The loss is computed using the cross entropy function
- The optimization algorithm used for this implementation of MLP is stochastic gradient descent (SGD).

3. *Dataset description:*

South Africa Heart Disease Dataset Source:
https://web.stanford.edu/~hastie/ElemStatLearn//data.html
https://www.openml.org/d/1498

A retrospective sample of males in a heart-disease high-risk region of the Western Cape, South Africa. There are roughly two controls per case of CHD. Many of the CHD positive men have undergone blood pressure reduction treatment and other programs to reduce their risk factors after their CHD event. In some cases the measurements were made after these treatments. These data are taken from a larger dataset, described in Rousseauw et al, 1983, South African Medical Journal.

4. *Results observed by the authors:*

The performance with varying number of neurons of the system is shown in table I. Out of 303 instances the network is trained with 212 instances and remaining 91 instances are used for testing. The system gives the highest accuracy of 98.58% for 20 neurons in a hidden layer with 1000 iterations. The different performance measures like TP, FP, TN and FN for different number of neurons is shown in table II.

TABLE I. PERFORMANCE OF THE SYSTEM WITH DIFFERENT NUMBER OF NEURONS

No. of Neurons	Acc	Sens.	Spec.	Error
5	92.92	92	93.75	7.07
10	95.75	95	96.42	4.24
15	96.69	96	98.21	3.30
20	98.58	98	98.21	1.41

TABLE II. DIFFERENT PERFORMANCE MEASURES OF THE SYSTEM

No. of Neurons	TP	FP	TN	FN
5	92	7	105	8
10	95	4	108	5
15	95	2	110	5
20	98	1	111	2

TABLE III. PERFORMANCE OF THE SYSTEM WITH DIFFERENT NUMBER OF EPOCHS

No. of Epochs	Acc	Sens.	Spec.	Error
1000	93.39	92	94.64	6.60
2000	94.33	93	95.53	5.66
3000	95.75	94	97.32	4.24
4000	97.64	98	97.32	2.35

In order to compute the accuracy and performance of the proposed prediction system with varying number of epochs, the number on neurons in hidden layer is set to 5 and 212 samples are used for training and 91 samples are used for testing. Table III shows the system performance with varying number of Epochs with highest accuracy of 97.64%.

5. *Observations and Conclusion:*

In this paper a useful and accurate technique for the classification and retrieval of image by using a self organizing map (SOM) is proposed and developed. In this technique the image texture is classified in two main phases, in the first phase the color features are extracted and classification is done based on color features using a self-organizing map. In the second phase the images in each class of the first phase are again classified using a self-organizing map based on texture features extracted using the GLCM matrix. The SOM is trained with different topology sizes and no. of iterations to improve the performance of the system. The experiments were performed on Wang's database of total 463 images including 10 categories. The experimental results show that the proposed method gives increased accuracy and improved retrieval performance for each

category of the image database.

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/tensorflow/keras/pytorch

Algorithm:

1. Load dataset: features and target.
2. Normalise features.
3. Perform train-test split.
4. Creation of MLP model via hyperparameter tuning using GridSearchCV.
5. Train the model with the training dataset.
6. Test the model with the testing dataset.
7. Find accuracy score.
8. Plot the graph of Training accuracy, test accuracy and loss function.

Code & Results:

```
import tensorflow as tf
from tensorflow import keras
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
data =
pd.read_csv('https://www.openml.org/data/get_csv/1592290/phpgNaXZe')
column =
['sbp','tobacco','ldl','adiposity','famhist','type','obesity','alcohol','age','chd']
data.columns=column
data.head()
```

	sbp	tobacco	ldl	adiposity	famhist	type	obesity	alcohol	age	chd
0	160	12.00	5.73	23.11	1	49	25.30	97.20	52	2
1	144	0.01	4.41	28.61	2	55	28.87	2.06	63	2
2	118	0.08	3.48	32.28	1	52	29.14	3.81	46	1
3	170	7.50	6.41	38.03	1	51	31.99	24.26	58	2
4	134	13.60	3.50	27.78	1	60	25.99	57.34	49	2

```
data.isnull().sum()
```

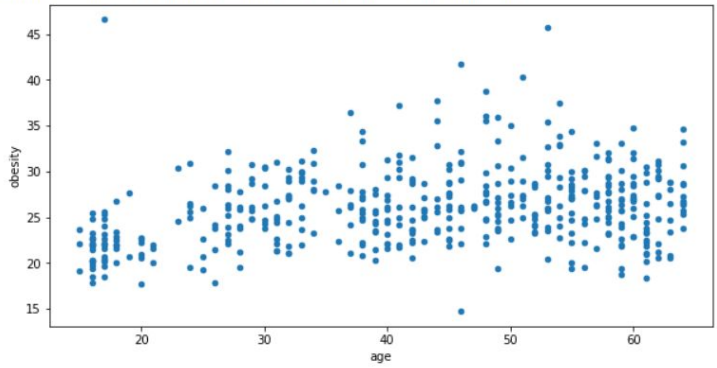
```
sbp          0
tobacco      0
ldl          0
adiposity    0
famhist      0
type         0
obesity      0
alcohol      0
age          0
chd          0
dtype: int64
```

```
from sklearn.preprocessing import LabelEncoder
encoder = LabelEncoder()
data['famhist']=encoder.fit_transform(data['famhist'])
data['chd']=encoder.fit_transform(data['chd'])
data.head()
```

	sbp	tobacco	ldl	adiposity	famhist	type	obesity	alcohol	age	chd
0	160	12.00	5.73	23.11	0	49	25.30	97.20	52	1
1	144	0.01	4.41	28.61	1	55	28.87	2.06	63	1
2	118	0.08	3.48	32.28	0	52	29.14	3.81	46	0
3	170	7.50	6.41	38.03	0	51	31.99	24.26	58	1
4	134	13.60	3.50	27.78	0	60	25.99	57.34	49	1

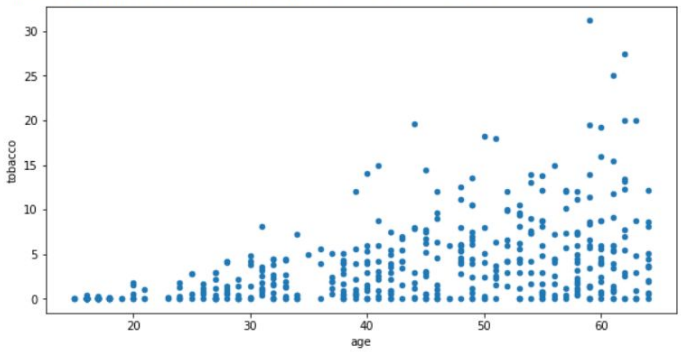
```
from sklearn.preprocessing import MinMaxScaler
scale = MinMaxScaler(feature_range =(0,100))

# setting scale of max min value for sbp in range of 0-100, normalise
data['sbp'] = scale.fit_transform(data['sbp'].values.reshape(-1,1))
data.plot(x='age',y='obesity',kind='scatter',figsize =(10,5))
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f6e5508b6d8>
```



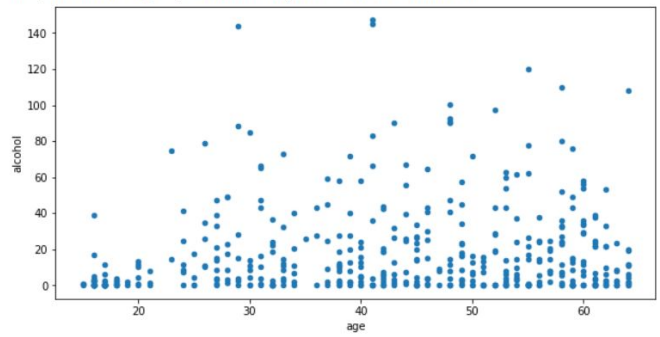

```
data.plot(x='age',y='tobacco',kind='scatter',figsize =(10,5))
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f6e54fea278>
```



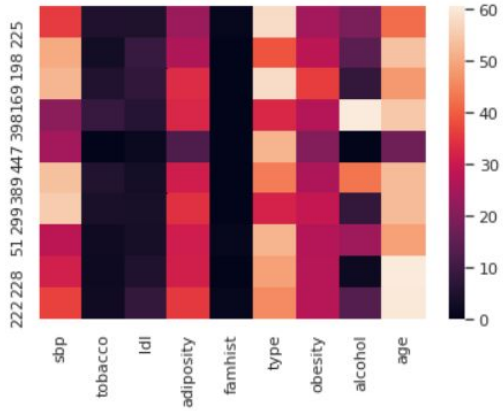

```
data.plot(x='age',y='alcohol',kind='scatter',figsize =(10,5))
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f6e54b16fd0>
```



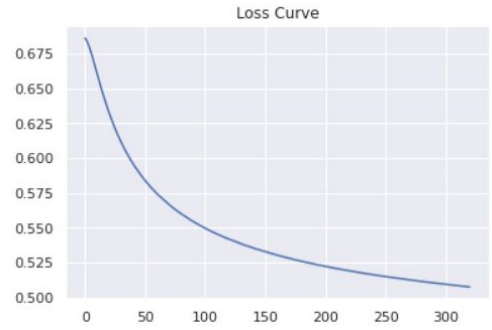

```
# splitting the data into test and train  having a test size of 20% and
80% train size
from sklearn.model_selection import train_test_split
col =
['sbp','tobacco','ldl','adiposity','famhist','type','obesity','alcohol','a
ge']
X_train, X_test, y_train, y_test = train_test_split(data[col],
data['chd'], test_size=0.2, random_state=1234)
sns.set()
sns.heatmap(X_train.head(10),robust = True)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f6e54a829b0>
```



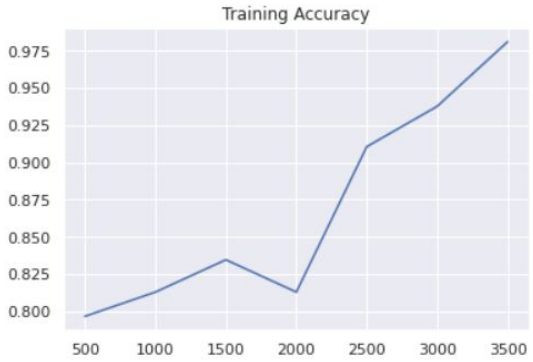
```
X_all = data[col]
y_all = data['chd']
from sklearn.metrics import make_scorer, accuracy_score
from sklearn.model_selection import GridSearchCV
from sklearn.neural_network import MLPClassifier
ann_clf = MLPClassifier(max_iter=150000)

#Parameters
parameters = {'solver': ['sgd','lbfgs','adam'],
              'alpha':[1e-2,1e-3],
              'hidden_layer_sizes':(9,500,500,2),   # 9 input, 500-500
neuron in 2 layers,1 output layer
              'random_state': [1]
              }
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)

# Type of scoring to compare parameter combos
acc_scorer = make_scorer(accuracy_score)

# Run grid search
grid_obj = GridSearchCV(ann_clf, parameters, scoring=acc_scorer,cv=10,
n_jobs =-1)
grid_obj = grid_obj.fit(X_train, y_train)

# Pick the best combination of parameters
ann_clf = grid_obj.best_estimator_
# Fit the best algorithm to the data
ann_clf.fit(X_train, y_train)
MLPClassifier(activation='relu', alpha=0.01, batch_size='auto',
beta_1=0.9,
              beta_2=0.999, early_stopping=False, epsilon=1e-08,
              hidden_layer_sizes=500, learning_rate='constant',
              learning_rate_init=0.001, max_fun=15000, max_iter=150000,
              momentum=0.9, n_iter_no_change=10, nesterovs_momentum=True,
              power_t=0.5, random_state=1, shuffle=True, solver='sgd',
              tol=0.0001, validation_fraction=0.1, verbose=False,
              warm_start=False)
y_pred_ann = ann_clf.predict(X_test)
# Making the Confusion Matrix
```

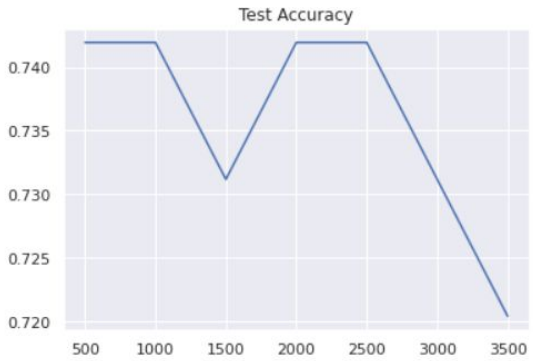
```
from sklearn.metrics import confusion_matrix
cm_ann = confusion_matrix(y_test, y_pred_ann)
cm_ann
array([[51,  9],
       [16, 17]])
y_pred_train = ann_clf.predict(X_train)
acc_train = accuracy_score(y_train, y_pred_train)
acc_train*100
75.88075880758808
ann_result = accuracy_score(y_test,y_pred_ann)
ann_result*100
73.11827956989248
recall_ann = cm_ann[0][0]/(cm_ann[0][0] + cm_ann[0][1])
precision_ann = cm_ann[0][0]/(cm_ann[0][0]+cm_ann[1][1])
recall_ann*100,precision_ann*100
(85.0, 75.0)
import matplotlib.pyplot as plt
loss_values = ann_clf.loss_curve_
plt.plot(loss_values)
plt.title('Loss Curve')
plt.show()
```




```
max_epochs = 3501
train_acc = []
test_acc = []
x = []
i = 500
while i < max_epochs:
  clf = MLPClassifier(hidden_layer_sizes=(500,500,), activation='relu',
solver='sgd',alpha=0.0001, batch_size=128, learning_rate='constant',
learning_rate_init=0.001,
                    max_iter=i,shuffle=True, random_state=None,
tol=0.0001, verbose=False, n_iter_no_change=10)
```

```
  clf.fit(X_train, y_train)
  y_pred = clf.predict(X_test)
  y_pred_train = clf.predict(X_train)
  acc_train = accuracy_score(y_train, y_pred_train)
  train_acc.append(acc_train)
  acc_test = accuracy_score(y_test, y_pred)
  test_acc.append(acc_test)
  x.append(i)
  i+=500

plt.plot(x,train_acc)
plt.title('Training Accuracy')
plt.show()
plt.plot(x,test_acc)
plt.title('Test Accuracy')
plt.show()
```





Conclusion:

1. Multi-layer Perceptron was implemented in the publicly available Heart Disease Prediction dataset which classifies the records into 2 classes – Absence and Presence of Coronary Heart Disease. The MLP classifier was implemented using the standard library function available in the sklearn library.
2. This can be attributed to the fact that the authors may have implemented the MLP

classifier from scratch while the implementation for this experiment was done using the standard library function.

3. The hyperparameters, viz. 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.