2348441_Lab_10

April 19, 2024

: LAB Exercise – 10 – MLP Classifier

Created by : Nileem Kaveramma C C | 2348441 Created DATE:19-03-2024 Edited Date: 19-03-2024

AIM: To design and train an MLP Classifier capable of accurately categorizing input data into multiple classes, leveraging its ability to learn complex patterns in the data through multiple hidden layers of neurons. The aim is to achieve high classification performance metrics such as accuracy, precision, recall, and F1-score, thereby demonstrating the effectiveness of the MLP architecture in handling the given classification task.

IMPORTED LIBRARIES

- numpy - for numerical, array, matrices (Linear Algebra) processing
- Pandas - for loading and processing datasets
- matplotlib.pyplot - For visualisation
- Saeborn - for statistical graph
- scipy.stats use a variety of statistical functions
- %matplotlib inline: Enables inline plotting in Jupyter notebooks, displaying matplotlib plots directly below the code cell.

[ ]:

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib as mpl
import matplotlib.pyplot as plt
import seaborn as sns
```

[ ]:

```
df = pd.read_csv('/content/heart.csv')
df
```

[ ]:

```
     Age Sex ChestPainType  RestingBP  Cholesterol  FastingBS RestingECG  \
0     40   M           ATA        140          289          0     Normal
1     49   F           NAP        160          180          0     Normal
2     37   M           ATA        130          283          0         ST
3     48   F           ASY        138          214          0     Normal
4     54   M           NAP        150          195          0     Normal
..    …  ..           …          …            …          …        …
913   45   M            TA        110          264          0     Normal
914   68   M           ASY        144          193          1     Normal
915   57   M           ASY        130          131          0     Normal
```

```
916   57   F            ATA        130          236          0        LVH
917   38   M            NAP        138          175          0     Normal

     MaxHR ExerciseAngina  Oldpeak ST_Slope  HeartDisease
0      172              N      0.0       Up             0
1      156              N      1.0     Flat             1
2       98              N      0.0       Up             0
3      108              Y      1.5     Flat             1
4      122              N      0.0       Up             0
..     …             …      …      …           …
913    132              N      1.2     Flat             1
914    141              N      3.4     Flat             1
915    115              Y      1.2     Flat             1
916    174              N      0.0     Flat             1
917    173              N      0.0       Up             0

[918 rows x 12 columns]
```

Perform some basic EDA

df.shape - attribute is used to get the dimensions of the DataFrame

```
[ ]: df.shape
```

```
[ ]: (918, 12)
```

df.head() method is used to display the first few rows of a DataFrame

```
[ ]: df.head()
```

```
[ ]:    Age Sex ChestPainType  RestingBP  Cholesterol  FastingBS RestingECG  MaxHR  \
     0   40   M           ATA        140          289          0     Normal    172
     1   49   F           NAP        160          180          0     Normal    156
     2   37   M           ATA        130          283          0         ST     98
     3   48   F           ASY        138          214          0     Normal    108
     4   54   M           NAP        150          195          0     Normal    122

       ExerciseAngina  Oldpeak ST_Slope  HeartDisease
     0              N      0.0       Up             0
     1              N      1.0     Flat             1
     2              N      0.0       Up             0
     3              Y      1.5     Flat             1
     4              N      0.0       Up             0
```

df.tail() method is used to display the last few rows of a DataFrame.

```
[ ]: df.tail()
[ ]:      Age Sex ChestPainType  RestingBP  Cholesterol  FastingBS RestingECG  \
     913   45   M            TA        110          264          0     Normal
     914   68   M           ASY        144          193          1     Normal
     915   57   M           ASY        130          131          0     Normal
     916   57   F           ATA        130          236          0        LVH
     917   38   M           NAP        138          175          0     Normal

          MaxHR ExerciseAngina  Oldpeak ST_Slope  HeartDisease
     913    132              N      1.2     Flat             1
     914    141              N      3.4     Flat             1
     915    115              Y      1.2     Flat             1
     916    174              N      0.0     Flat             1
     917    173              N      0.0       Up             0
```

df.columns attribute is used to retrieve the column labels or names of the DataFrame.

```
[ ]: df.columns
```

```
[ ]: Index(['Age', 'Sex', 'ChestPainType', 'RestingBP', 'Cholesterol', 'FastingBS',
            'RestingECG', 'MaxHR', 'ExerciseAngina', 'Oldpeak', 'ST_Slope',
            'HeartDisease'],
           dtype='object')
```

df.dtypes attribute is used to retrieve the data types of each column in a DataFrame

```
[ ]: df.dtypes
```

```
[ ]: Age                 int64
     Sex                object
     ChestPainType      object
     RestingBP           int64
     Cholesterol         int64
     FastingBS           int64
     RestingECG         object
     MaxHR               int64
     ExerciseAngina     object
     Oldpeak           float64
     ST_Slope           object
     HeartDisease        int64
     dtype: object
```

the code df.isnull().count() in Pandas is used to count the total number of rows for each column in a DataFrame, including both missing (null or NaN) and non-missing values.

```
[ ]: df.isnull().count()
```

```
[ ]: Age               918
     Sex               918
```

```
ChestPainType     918
RestingBP         918
Cholesterol       918
FastingBS         918
RestingECG        918
MaxHR             918
ExerciseAngina    918
Oldpeak           918
ST_Slope          918
HeartDisease      918
dtype: int64
```

df.info() method in Pandas provides a concise summary of a DataFrame, including information about the data types, non-null values, and memory usage

```
[ ]: df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 918 entries, 0 to 917
Data columns (total 12 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   Age             918 non-null    int64
 1   Sex             918 non-null    object
 2   ChestPainType   918 non-null    object
 3   RestingBP       918 non-null    int64
 4   Cholesterol     918 non-null    int64
 5   FastingBS       918 non-null    int64
 6   RestingECG      918 non-null    object
 7   MaxHR           918 non-null    int64
 8   ExerciseAngina  918 non-null    object
 9   Oldpeak         918 non-null    float64
 10  ST_Slope        918 non-null    object
 11  HeartDisease    918 non-null    int64
dtypes: float64(1), int64(6), object(5)
memory usage: 86.2+ KB
```

The df.describe() method in Pandas is used to generate descriptive statistics that summarize the central tendency, dispersion, and shape of a dataset's distribution

```
[ ]: df.describe()
```

```
[ ]:              Age   RestingBP  Cholesterol   FastingBS       MaxHR  \
     count  918.000000  918.000000   918.000000  918.000000  918.000000
     mean    53.510893  132.396514   198.799564    0.233115  136.809368
     std      9.432617   18.514154   109.384145    0.423046   25.460334
     min     28.000000    0.000000     0.000000    0.000000   60.000000
     25%     47.000000  120.000000   173.250000    0.000000  120.000000
```

```
50%      54.000000  130.000000   223.000000    0.000000  138.000000
75%      60.000000  140.000000   267.000000    0.000000  156.000000
max      77.000000  200.000000   603.000000    1.000000  202.000000

          Oldpeak  HeartDisease
count  918.000000    918.000000
mean     0.887364      0.553377
std      1.066570      0.497414
min     -2.600000      0.000000
25%      0.000000      0.000000
50%      0.600000      1.000000
75%      1.500000      1.000000
max      6.200000      1.000000
```

```
[ ]: print(f"The shape of the dataset  is :{df.shape}")
     print(f"The size of the dataset  is :{df.size} \n")
     print(f"The columns in the dataset is:{df.columns}")
```

```
The shape of the dataset  is :(918, 12)
The size of the dataset  is :11016

The columns in the dataset is:Index(['Age', 'Sex', 'ChestPainType', 'RestingBP',
'Cholesterol', 'FastingBS',
       'RestingECG', 'MaxHR', 'ExerciseAngina', 'Oldpeak', 'ST_Slope',
       'HeartDisease'],
      dtype='object')
```

1 Modelling

```
[ ]: from sklearn.preprocessing import LabelEncoder

     for i in df.select_dtypes(include=['object']):
         le=LabelEncoder()
         df[i]=le.fit_transform(df[i])
```

```
[ ]: x=df.iloc[:,0:11]
     y=df.iloc[:,-1]
```

```
[ ]: # Splitting the dataset
     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=43)

     print("Shape of X_train:", x_train.shape)
     print("Shape of X_test:", x_test.shape)
```

```
Shape of X_train: (734, 11)
Shape of X_test: (184, 11)
```

[ ]:

```
from sklearn.preprocessing import StandardScaler
scaler=StandardScaler()
x_train_scaled=scaler.fit_transform(x_train)
x_test_scaled=scaler.transform(x_test)
```

[ ]:

```
from sklearn.svm import SVC
from sklearn.metrics import␣
 ↪accuracy_score,confusion_matrix,classification_report
```

2 Fitting Multilayer Perceptor

[ ]:

```
from sklearn.neural_network import MLPClassifier
clf=MLPClassifier(hidden_layer_sizes=(6,5),
                  random_state=43,
                  verbose=True,
                  learning_rate_init=0.01)
```

[ ]:

```
clf.fit(x_train_scaled,y_train)
```

```
Iteration 1, loss = 0.90635338
Iteration 2, loss = 0.81198219
Iteration 3, loss = 0.75202267
Iteration 4, loss = 0.70920091
Iteration 5, loss = 0.67754506
Iteration 6, loss = 0.65147800
Iteration 7, loss = 0.63115080
Iteration 8, loss = 0.61348910
Iteration 9, loss = 0.60155132
Iteration 10, loss = 0.58916489
Iteration 11, loss = 0.57824106
Iteration 12, loss = 0.56907214
Iteration 13, loss = 0.56027034
Iteration 14, loss = 0.55218334
Iteration 15, loss = 0.54382702
Iteration 16, loss = 0.53373733
Iteration 17, loss = 0.52066524
Iteration 18, loss = 0.50276439
Iteration 19, loss = 0.48149147
Iteration 20, loss = 0.45845143
Iteration 21, loss = 0.43181441
Iteration 22, loss = 0.40616311
Iteration 23, loss = 0.38130957
Iteration 24, loss = 0.35932406
Iteration 25, loss = 0.34172882
Iteration 26, loss = 0.32957671
Iteration 27, loss = 0.32125922
Iteration 28, loss = 0.31424140
Iteration 29, loss = 0.30872657
Iteration 30, loss = 0.30311797
Iteration 31, loss = 0.29895326
Iteration 32, loss = 0.29370663
Iteration 33, loss = 0.28970550
Iteration 34, loss = 0.28722889
Iteration 35, loss = 0.28294327
Iteration 36, loss = 0.27941819
Iteration 37, loss = 0.27672764
Iteration 38, loss = 0.27336982
Iteration 39, loss = 0.27099247
Iteration 40, loss = 0.26920402
Iteration 41, loss = 0.26765162
Iteration 42, loss = 0.26661422
Iteration 43, loss = 0.26478387
Iteration 44, loss = 0.26385748
Iteration 45, loss = 0.26280664
Iteration 46, loss = 0.26199230
Iteration 47, loss = 0.26119157
Iteration 48, loss = 0.26061996
Iteration 49, loss = 0.25939792
Iteration 50, loss = 0.25879732
Iteration 51, loss = 0.25832835
Iteration 52, loss = 0.25686246
Iteration 53, loss = 0.25589970
Iteration 54, loss = 0.25535611
Iteration 55, loss = 0.25460210
Iteration 56, loss = 0.25422248
Iteration 57, loss = 0.25391679
Iteration 58, loss = 0.25284002
Iteration 59, loss = 0.25272822
Iteration 60, loss = 0.25205886
Iteration 61, loss = 0.25173525
Iteration 62, loss = 0.25120909
Iteration 63, loss = 0.25085379
Iteration 64, loss = 0.25011899
Iteration 65, loss = 0.24997784
Iteration 66, loss = 0.24966490
Iteration 67, loss = 0.24890078
Iteration 68, loss = 0.24870430
Iteration 69, loss = 0.24784261
Iteration 70, loss = 0.24776582
Iteration 71, loss = 0.24751892
Iteration 72, loss = 0.24722684
Iteration 73, loss = 0.24692835
Iteration 74, loss = 0.24636673
Iteration 75, loss = 0.24687498
Iteration 76, loss = 0.24615292
Iteration 77, loss = 0.24566368
Iteration 78, loss = 0.24612453
Iteration 79, loss = 0.24488782
Iteration 80, loss = 0.24431993
Iteration 81, loss = 0.24371024
Iteration 82, loss = 0.24295307
Iteration 83, loss = 0.24296129
Iteration 84, loss = 0.24304775
Iteration 85, loss = 0.24181721
Iteration 86, loss = 0.24203990
Iteration 87, loss = 0.24157172
Iteration 88, loss = 0.24031406
Iteration 89, loss = 0.24083108
Iteration 90, loss = 0.24025773
Iteration 91, loss = 0.23940321
Iteration 92, loss = 0.23868377
Iteration 93, loss = 0.23889949
Iteration 94, loss = 0.23795085
Iteration 95, loss = 0.23837051
Iteration 96, loss = 0.23741598
Iteration 97, loss = 0.23696873
Iteration 98, loss = 0.23720886
Iteration 99, loss = 0.23633619
Iteration 100, loss = 0.23679113
Iteration 101, loss = 0.23683281
Iteration 102, loss = 0.23550858
Iteration 103, loss = 0.23577300
Iteration 104, loss = 0.23490211
Iteration 105, loss = 0.23420744
Iteration 106, loss = 0.23444992
Iteration 107, loss = 0.23410356
Iteration 108, loss = 0.23349192
Iteration 109, loss = 0.23345107
Iteration 110, loss = 0.23268858
Iteration 111, loss = 0.23199975
Iteration 112, loss = 0.23171422
Iteration 113, loss = 0.23134699
Iteration 114, loss = 0.23129010
Iteration 115, loss = 0.23168826
Iteration 116, loss = 0.23053918
Iteration 117, loss = 0.23137811
Iteration 118, loss = 0.22907549
Iteration 119, loss = 0.22966768
Iteration 120, loss = 0.22919649
Iteration 121, loss = 0.22814696
Iteration 122, loss = 0.22855556
Iteration 123, loss = 0.22706766
Iteration 124, loss = 0.22656450
Iteration 125, loss = 0.22646407
Iteration 126, loss = 0.22581692
Iteration 127, loss = 0.22525821
Iteration 128, loss = 0.22498725
Iteration 129, loss = 0.22466310
Iteration 130, loss = 0.22400742
Iteration 131, loss = 0.22348433
Iteration 132, loss = 0.22363440
Iteration 133, loss = 0.22277521
Iteration 134, loss = 0.22254239
Iteration 135, loss = 0.22302095
Iteration 136, loss = 0.22143186
Iteration 137, loss = 0.22054953
Iteration 138, loss = 0.22071109
Iteration 139, loss = 0.21975558
Iteration 140, loss = 0.21959101
Iteration 141, loss = 0.21924982
Iteration 142, loss = 0.21823091
Iteration 143, loss = 0.21788703
Iteration 144, loss = 0.21777729
Iteration 145, loss = 0.21663020
Iteration 146, loss = 0.21529927
Iteration 147, loss = 0.21550813
Iteration 148, loss = 0.21540175
Iteration 149, loss = 0.21409600
Iteration 150, loss = 0.21396547
Iteration 151, loss = 0.21338695
Iteration 152, loss = 0.21266226
Iteration 153, loss = 0.21347713
Iteration 154, loss = 0.21266648
Iteration 155, loss = 0.21133865
Iteration 156, loss = 0.21248459
Iteration 157, loss = 0.21129842
Iteration 158, loss = 0.21182495
Iteration 159, loss = 0.21012040
Iteration 160, loss = 0.20979122
Iteration 161, loss = 0.21018014
Iteration 162, loss = 0.20970456
Iteration 163, loss = 0.20886765
Iteration 164, loss = 0.20868096
Iteration 165, loss = 0.20806514
Iteration 166, loss = 0.20862856
Iteration 167, loss = 0.20676453
Iteration 168, loss = 0.20660975
Iteration 169, loss = 0.20541693
Iteration 170, loss = 0.20481526
Iteration 171, loss = 0.20351533
Iteration 172, loss = 0.20382466
Iteration 173, loss = 0.20251307
Iteration 174, loss = 0.20230421
Iteration 175, loss = 0.20365795
Iteration 176, loss = 0.20192041
Iteration 177, loss = 0.20248376
Iteration 178, loss = 0.20110416
Iteration 179, loss = 0.20098882
Iteration 180, loss = 0.20078737
Iteration 181, loss = 0.19916354
Iteration 182, loss = 0.19916515
Iteration 183, loss = 0.19875902
Iteration 184, loss = 0.20034710
Iteration 185, loss = 0.19828305
Iteration 186, loss = 0.19737687
Iteration 187, loss = 0.19669042
Iteration 188, loss = 0.19584373
Iteration 189, loss = 0.19572397
Iteration 190, loss = 0.19551192
Iteration 191, loss = 0.19463151
Iteration 192, loss = 0.19505691
Iteration 193, loss = 0.19430901
Iteration 194, loss = 0.19344999
Iteration 195, loss = 0.19298249
Iteration 196, loss = 0.19452006
Iteration 197, loss = 0.19293205
Iteration 198, loss = 0.19266043
Iteration 199, loss = 0.19293837
Iteration 200, loss = 0.19198076
```

```
/usr/local/lib/python3.10/dist-
packages/sklearn/neural_network/_multilayer_perceptron.py:686:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (200) reached and
the optimization hasn't converged yet.
  warnings.warn(
```

[ ]:

```
MLPClassifier(hidden_layer_sizes=(6, 5), learning_rate_init=0.01,
              random_state=43, verbose=True)
```

[ ]:

```python
# Make prediction on test dataset
ypred=clf.predict(x_test_scaled)

# Import accuracy score
from sklearn import metrics

# Calcuate accuracy
print("Accuracy:", '%.2f'%(metrics.accuracy_score(y_test,ypred)*100), "%")
```

```
# Model Precision
print("Precision:",'%.2f'%(metrics.precision_score(y_test, ypred)*100), "%")

# Model Recall
print("Recall:",'%.2f'%(metrics.recall_score(y_test, ypred)*100),"%")
```

```
Accuracy: 83.70 %
Precision: 86.87 %
Recall: 83.50 %
```

3 Fitting MLP after performing PCA

[ ]:

```
from sklearn.decomposition import PCA
pca=PCA().fit(x)
```

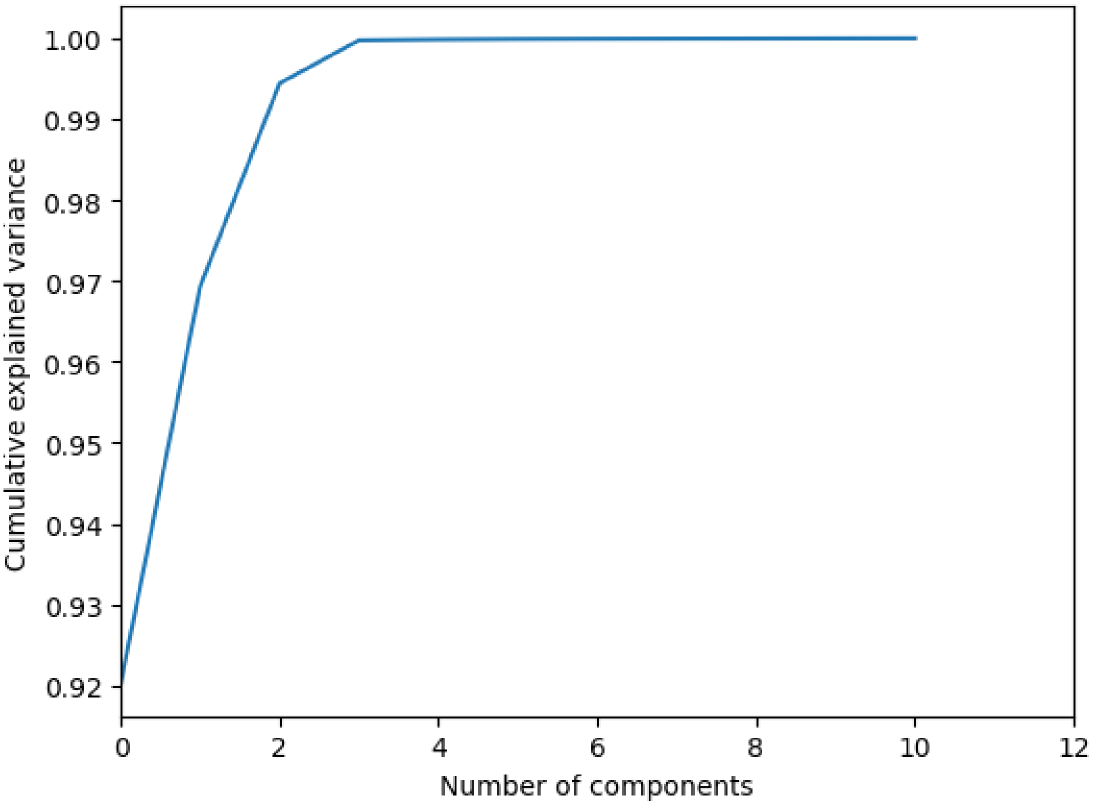
[ ]:

```
plt.plot(np.cumsum(pca.explained_variance_ratio_))
plt.xlim(0,12,1)
plt.xlabel('Number of components')
plt.ylabel('Cumulative explained variance')
```

```
<ipython-input-22-8dc5d8e80abc>:2: MatplotlibDeprecationWarning: Passing the
emit parameter of set_xlim() positionally is deprecated since Matplotlib 3.6;
the parameter will become keyword-only two minor releases later.
  plt.xlim(0,12,1)
```

[ ]:

```
Text(0, 0.5, 'Cumulative explained variance')
```



[ ]:

```
pca=PCA(n_components=3)
x_pca=pca.fit_transform(x_train)
x_test_pca=pca.transform(x_test_scaled)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:439: UserWarning: X does
not have valid feature names, but PCA was fitted with feature names
  warnings.warn(
```

[ ]:

```
from sklearn.neural_network import MLPClassifier
clf=MLPClassifier(hidden_layer_sizes=(6,5),
                  random_state=43,
                  verbose=True,
                  learning_rate_init=0.01)
```

[ ]:

```
clf.fit(x_pca,y_train)
```

```
Iteration 1, loss = 3.73574140
Iteration 2, loss = 1.36841122
Iteration 3, loss = 1.60206370
Iteration 4, loss = 1.60451039
Iteration 5, loss = 1.53495112
Iteration 6, loss = 1.32017945
Iteration 7, loss = 1.04212885
Iteration 8, loss = 0.76890746
Iteration 9, loss = 0.79448002
Iteration 10, loss = 0.75613625
Iteration 11, loss = 0.79978455
Iteration 12, loss = 0.71303939
Iteration 13, loss = 0.70280804
Iteration 14, loss = 0.64321830
Iteration 15, loss = 0.64105024
Iteration 16, loss = 0.60814275
Iteration 17, loss = 0.59333972
Iteration 18, loss = 0.60023476
Iteration 19, loss = 0.57628672
Iteration 20, loss = 0.59376484
Iteration 21, loss = 0.59333733
Iteration 22, loss = 0.58117239
Iteration 23, loss = 0.57260517
Iteration 24, loss = 0.57904596
Iteration 25, loss = 0.56908098
Iteration 26, loss = 0.57204055
Iteration 27, loss = 0.56906223
Iteration 28, loss = 0.56515462
Iteration 29, loss = 0.55663002
Iteration 30, loss = 0.56459773
Iteration 31, loss = 0.56117360
Iteration 32, loss = 0.55385776
Iteration 33, loss = 0.56461520
Iteration 34, loss = 0.56565371
Iteration 35, loss = 0.55503866
Iteration 36, loss = 0.56383764
Iteration 37, loss = 0.55499683
Iteration 38, loss = 0.55987750
Iteration 39, loss = 0.55152434
Iteration 40, loss = 0.55463132
Iteration 41, loss = 0.55441655
Iteration 42, loss = 0.55332479
Iteration 43, loss = 0.55009979
Iteration 44, loss = 0.54803054
Iteration 45, loss = 0.54861153
Iteration 46, loss = 0.55217781
Iteration 47, loss = 0.54938390
Iteration 48, loss = 0.54874628
Iteration 49, loss = 0.55074119
Iteration 50, loss = 0.55530357
Iteration 51, loss = 0.55545795
Iteration 52, loss = 0.55806500
Iteration 53, loss = 0.55750030
Iteration 54, loss = 0.56438188
Iteration 55, loss = 0.55286375
Training loss did not improve more than tol=0.000100 for 10 consecutive epochs.
Stopping.
```

```
[ ]: MLPClassifier(hidden_layer_sizes=(6, 5), learning_rate_init=0.01,
                   random_state=43, verbose=True)
```

[ ]:

```
# Make prediction on test dataset
ypred1=clf.predict(x_test_pca)

# Import accuracy score
from sklearn import metrics

# Calcuate accuracy
print("Accuracy:", '%.2f'%(metrics.accuracy_score(y_test,ypred1)*100), "%")

# Model Precision
print("Precision:",'%.2f'%(metrics.precision_score(y_test, ypred1)*100), "%")

# Model Recall
print("Recall:",'%.2f'%(metrics.recall_score(y_test, ypred1)*100),"%")
```

```
Accuracy: 55.98 %
Precision: 55.98 %
Recall: 100.00 %
```