19MIS1018_LAB-8_ Implementing Multilayer Perceptron and compare with KNN

September 20, 2022

NAME: B DEVI PRASAD

REG.NO: 19MIS1018

SLOT: L13+L14

FACULTY:Dr. G. Bharadwaja Kumar

1 Multilayer Perceptron

[1]:
```python
import numpy as np
import pandas as pd

# Load data
data=pd.read_csv('HR_comma_sep.csv')

data.head()
```

[1]:
```
   satisfaction_level  last_evaluation  number_project  average_montly_hours  \
0                0.38             0.53               2                   157
1                0.80             0.86               5                   262
2                0.11             0.88               7                   272
3                0.72             0.87               5                   223
4                0.37             0.52               2                   159

   time_spend_company  Work_accident  left  promotion_last_5years  sales  \
0                   3              0     1                      0  sales
1                   6              0     1                      0  sales
2                   4              0     1                      0  sales
3                   5              0     1                      0  sales
4                   3              0     1                      0  sales

   salary
0     low
1  medium
2  medium
3     low
```

```
4     low
```

[4]:
```
# Import LabelEncoder
from sklearn import preprocessing

# Creating labelEncoder
le = preprocessing.LabelEncoder()

# Converting string labels into numbers.
data['salary']=le.fit_transform(data['salary'])
data['sales']=le.fit_transform(data['sales'])
```

[8]:
```
# Spliting data into Feature and
X=data[['satisfaction_level', 'last_evaluation', 'number_project',␣
 ↪'average_montly_hours', 'time_spend_company', 'Work_accident',␣
 ↪'promotion_last_5years', 'sales', 'salary']]
y=data['left']

# Import train_test_split function
from sklearn.model_selection import train_test_split

# Split dataset into training set and test set
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,␣
 ↪random_state=42)  # 70% training and 30% test
```

[9]:
```
from sklearn.neural_network import MLPClassifier

# Create model object
clf = MLPClassifier(hidden_layer_sizes=(6,5),
                    random_state=5,
                    verbose=True,
                    learning_rate_init=0.01)

# Fit data onto the model
clf.fit(X_train,y_train)
```

```
Iteration 1, loss = 0.61512605
Iteration 2, loss = 0.57545658
Iteration 3, loss = 0.55823146
Iteration 4, loss = 0.53011644
Iteration 5, loss = 0.50549749
Iteration 6, loss = 0.48004244
Iteration 7, loss = 0.47915513
Iteration 8, loss = 0.46239153
Iteration 9, loss = 0.47441120
Iteration 10, loss = 0.46241650
Iteration 11, loss = 0.45068143
Iteration 12, loss = 0.45071101
```

```
Iteration 13, loss = 0.45213613
Iteration 14, loss = 0.46049483
Iteration 15, loss = 0.45897398
Iteration 16, loss = 0.46170601
Iteration 17, loss = 0.45527116
Iteration 18, loss = 0.44996595
Iteration 19, loss = 0.44982305
Iteration 20, loss = 0.45384764
Iteration 21, loss = 0.46981282
Iteration 22, loss = 0.45010489
Iteration 23, loss = 0.46852413
Iteration 24, loss = 0.45242336
Iteration 25, loss = 0.45769894
Iteration 26, loss = 0.45074974
Iteration 27, loss = 0.44067556
Iteration 28, loss = 0.43205930
Iteration 29, loss = 0.41680331
Iteration 30, loss = 0.40752887
Iteration 31, loss = 0.39186392
Iteration 32, loss = 0.37054743
Iteration 33, loss = 0.35797517
Iteration 34, loss = 0.34539661
Iteration 35, loss = 0.33669847
Iteration 36, loss = 0.33052333
Iteration 37, loss = 0.31754074
Iteration 38, loss = 0.31470388
Iteration 39, loss = 0.31036267
Iteration 40, loss = 0.30717479
Iteration 41, loss = 0.31789566
Iteration 42, loss = 0.31782866
Iteration 43, loss = 0.30641517
Iteration 44, loss = 0.30123487
Iteration 45, loss = 0.30474254
Iteration 46, loss = 0.31051473
Iteration 47, loss = 0.28624198
Iteration 48, loss = 0.29316190
Iteration 49, loss = 0.28463864
Iteration 50, loss = 0.28582673
Iteration 51, loss = 0.28378548
Iteration 52, loss = 0.28098729
Iteration 53, loss = 0.28513299
Iteration 54, loss = 0.27992380
Iteration 55, loss = 0.28034603
Iteration 56, loss = 0.28447776
Iteration 57, loss = 0.27920526
Iteration 58, loss = 0.27545140
Iteration 59, loss = 0.27418263
Iteration 60, loss = 0.26921345
```

```
Iteration 61, loss = 0.27200225
Iteration 62, loss = 0.26757142
Iteration 63, loss = 0.26824884
Iteration 64, loss = 0.26115995
Iteration 65, loss = 0.25875549
Iteration 66, loss = 0.25919820
Iteration 67, loss = 0.26178076
Iteration 68, loss = 0.26228984
Iteration 69, loss = 0.26001505
Iteration 70, loss = 0.25199503
Iteration 71, loss = 0.27545142
Iteration 72, loss = 0.25545200
Iteration 73, loss = 0.25464872
Iteration 74, loss = 0.24550959
Iteration 75, loss = 0.24336325
Iteration 76, loss = 0.24215966
Iteration 77, loss = 0.23850104
Iteration 78, loss = 0.23889253
Iteration 79, loss = 0.23109586
Iteration 80, loss = 0.23399638
Iteration 81, loss = 0.23802679
Iteration 82, loss = 0.23385322
Iteration 83, loss = 0.24284518
Iteration 84, loss = 0.23134733
Iteration 85, loss = 0.24330075
Iteration 86, loss = 0.22532877
Iteration 87, loss = 0.22568393
Iteration 88, loss = 0.22081075
Iteration 89, loss = 0.22846698
Iteration 90, loss = 0.22502850
Iteration 91, loss = 0.21784989
Iteration 92, loss = 0.21292884
Iteration 93, loss = 0.18813462
Iteration 94, loss = 0.17769639
Iteration 95, loss = 0.18582654
Iteration 96, loss = 0.18127325
Iteration 97, loss = 0.16734828
Iteration 98, loss = 0.18286465
Iteration 99, loss = 0.16981846
Iteration 100, loss = 0.17330641
Iteration 101, loss = 0.15969499
Iteration 102, loss = 0.15829448
Iteration 103, loss = 0.15677615
Iteration 104, loss = 0.16861506
Iteration 105, loss = 0.15665635
Iteration 106, loss = 0.17290740
Iteration 107, loss = 0.16313185
Iteration 108, loss = 0.15127830
```

```
Iteration 109, loss = 0.15304460
Iteration 110, loss = 0.15963898
Iteration 111, loss = 0.16963601
Iteration 112, loss = 0.15063887
Iteration 113, loss = 0.15393529
Iteration 114, loss = 0.17528787
Iteration 115, loss = 0.17271381
Iteration 116, loss = 0.15426095
Iteration 117, loss = 0.14971746
Iteration 118, loss = 0.15078193
Iteration 119, loss = 0.16200046
Iteration 120, loss = 0.15242240
Iteration 121, loss = 0.15700943
Iteration 122, loss = 0.15836204
Iteration 123, loss = 0.15847074
Iteration 124, loss = 0.15069127
Iteration 125, loss = 0.14876175
Iteration 126, loss = 0.14993943
Iteration 127, loss = 0.15168619
Iteration 128, loss = 0.16245349
Iteration 129, loss = 0.15119375
Iteration 130, loss = 0.15576164
Iteration 131, loss = 0.15837115
Iteration 132, loss = 0.15242943
Iteration 133, loss = 0.15235532
Iteration 134, loss = 0.15423949
Iteration 135, loss = 0.15300715
Iteration 136, loss = 0.15038034
Training loss did not improve more than tol=0.000100 for 10 consecutive epochs.
Stopping.
```

[9]:

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

[10]:

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

# Import accuracy score
from sklearn.metrics import accuracy_score

# Calcuate accuracy
accuracy_score(y_test,ypred)
```

[10]:

```
0.9386666666666666
```

0.9386666666666 Got a classification rate of 93.8%, considered as good accuracy. IN Case of perceptron both training and testing score is 100% IN Case of KNN training score is 100% and testing accuracy is 0.9386666666666666 %

2 KNN

[21]:
```python
import pip
pip.main(['install','seaborn'])
```

```
WARNING: pip is being invoked by an old script wrapper. This will fail in a
future version of pip.
Please see https://github.com/pypa/pip/issues/5599 for advice on fixing the
underlying issue.
To avoid this problem you can invoke Python with '-m pip' instead of running pip
directly.
```

```
Collecting seaborn
```

```
  Downloading seaborn-0.12.0-py3-none-any.whl (285 kB)
```

```
Output()
```

```
Requirement already satisfied: pandas>=0.25 in c:
↪\users\admin\appdata\local\programs\python\python39\lib\site-packages (from␣
↪seaborn) (1.3.4)
```

```
Requirement already satisfied: matplotlib>=3.1 in c:
↪\users\admin\appdata\local\programs\python\python39\lib\site-packages (from␣
↪seaborn) (3.4.3)
```

```
Requirement already satisfied: numpy>=1.17 in c:
↪\users\admin\appdata\local\programs\python\python39\lib\site-packages (from␣
↪seaborn) (1.21.4)
```

```
Requirement already satisfied: pillow>=6.2.0 in c:
↪\users\admin\appdata\local\programs\python\python39\lib\site-packages (from␣
↪matplotlib>=3.1->seaborn) (8.4.0)
```

```
Requirement already satisfied: pyparsing>=2.2.1 in c:
↪\users\admin\appdata\local\programs\python\python39\lib\site-packages (from␣
↪matplotlib>=3.1->seaborn) (3.0.6)
```

```
Requirement already satisfied: kiwisolver>=1.0.1 in c:
↪\users\admin\appdata\local\programs\python\python39\lib\site-packages (from␣
↪matplotlib>=3.1->seaborn) (1.3.2)
```

```
Requirement already satisfied: python-dateutil>=2.7 in c:
↪\users\admin\appdata\local\programs\python\python39\lib\site-packages (from␣
↪matplotlib>=3.1->seaborn) (2.8.2)
```

```
Requirement already satisfied: cycler>=0.10 in c:
↪\users\admin\appdata\local\programs\python\python39\lib\site-packages (from␣
↪matplotlib>=3.1->seaborn) (0.11.0)
```

```
Requirement already satisfied: pytz>=2017.3 in c:
↪\users\admin\appdata\local\programs\python\python39\lib\site-packages (from␣
↪pandas>=0.25->seaborn) (2021.3)
```

```
Requirement already satisfied: six>=1.5 in c:
↪\users\admin\appdata\local\programs\python\python39\lib\site-packages (from␣
↪python-dateutil>=2.7->matplotlib>=3.1->seaborn) (1.16.0)
```

```
Installing collected packages: seaborn
```

```
Successfully installed seaborn-0.12.0
```

[21]: 0

[22]:

```
import numpy as np
import pandas as pd
from matplotlib import pyplot as plt
from sklearn.datasets import load_breast_cancer
from sklearn.metrics import confusion_matrix
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
import seaborn as sns
sns.set()
```

[23]:

```
breast_cancer = load_breast_cancer()
X = pd.DataFrame(breast_cancer.data, columns=breast_cancer.feature_names)
X = X[['mean area', 'mean compactness']]
y = pd.Categorical.from_codes(breast_cancer.target, breast_cancer.target_names)
y = pd.get_dummies(y, drop_first=True)
```

[24]:

```
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=1)
```

[25]:

```
knn = KNeighborsClassifier(n_neighbors=5, metric='euclidean')
knn.fit(X_train, y_train)
```

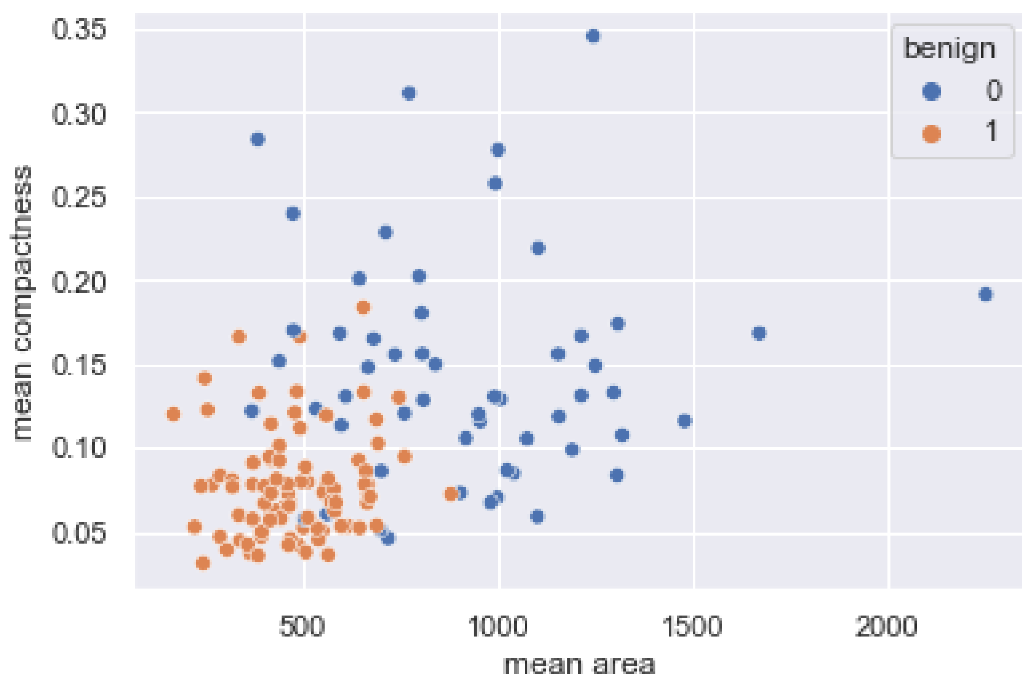
```
C:\Users\admin\AppData\Local\Programs\Python\Python39\lib\site-
packages\sklearn\neighbors\_classification.py:198: DataConversionWarning: A
column-vector y was passed when a 1d array was expected. Please change the shape
of y to (n_samples,), for example using ravel().
  return self._fit(X, y)
```

[25]:
```
KNeighborsClassifier(metric='euclidean')
```

[26]:
```python
y_pred = knn.predict(X_test)
```

[27]:
```python
sns.scatterplot(
    x='mean area',
    y='mean compactness',
    hue='benign',
    data=X_test.join(y_test, how='outer')
)
```

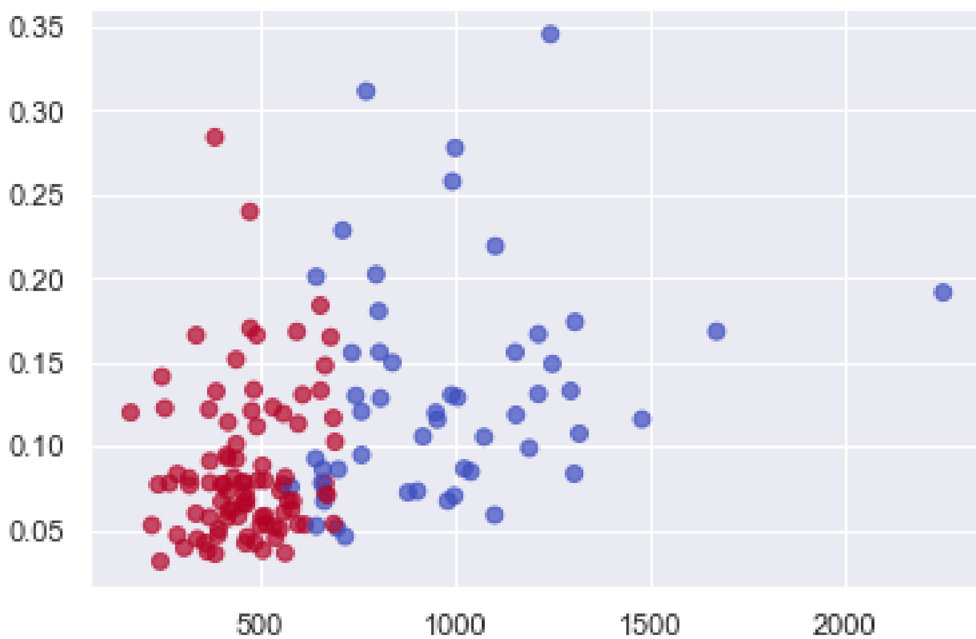
[27]:
```
<AxesSubplot:xlabel='mean area', ylabel='mean compactness'>
```



[28]:
```python
plt.scatter(
    X_test['mean area'],
    X_test['mean compactness'],
    c=y_pred,
    cmap='coolwarm',
    alpha=0.7
```

```
)
```

[28]: <matplotlib.collections.PathCollection at 0x1932c842ca0>



[29]:
```
confusion_matrix(y_test, y_pred)
```

[29]:
```
array([[42, 13],
       [ 9, 79]], dtype=int64)
```

Given our confusion matrix, our model has an accuracy of $121/143 = 84.6\%$. Conclusion The K Nearest Neighbors algorithm doesn't require any additional training when new data becomes available. Rather it determines the K closest points according to some distance metric (the samples must reside in memory)