JawadAhmed_20P-0165_Lab06_Task

March 19, 2023

1 Lab 06 Task

2 Multilayer Perceptron

3 Name: Jawad Ahmed

4 Roll No: 20P-0165

5 Section: BCS-6A

6 1. Import the necessary Python libraries, such as pandas, numpy, and sklearn.

[1]:
```
import numpy as np
import pandas as pd
from sklearn.preprocessing import LabelEncoder, StandardScaler, OneHotEncoder
from sklearn.impute import SimpleImputer
from sklearn.neural_network import MLPClassifier
from sklearn.model_selection import train_test_split
import seaborn as sns
import matplotlib.pyplot as plt
```

7 2. Load the Titanic dataset into a pandas DataFrame using pandas.read_csv().

[2]:
```
titanic_data = pd.read_csv('titanic_train.csv')
```

[3]:
```
titanic_data
```

[3]:
```
   passenger_id  pclass                                               name  \
0          1216       3                                 Smyth, Miss. Julia
1           699       3                                    Cacic, Mr. Luka
2          1267       3  Van Impe, Mrs. Jean Baptiste (Rosalie Paula Go…
3           449       2              Hocking, Mrs. Elizabeth (Eliza Needs)
4           576       2                                    Veal, Mr. James
```

```
..           …      …                                                  …
845        158       1              Hipkins, Mr. William Edward
846        174       1                  Kent, Mr. Edward Austin
847        467       2      Kantor, Mrs. Sinai (Miriam Sternin)
848       1112       3             Peacock, Miss. Treasteall
849        425       2                   Greenberg, Mr. Samuel

        sex   age  sibsp  parch              ticket     fare cabin embarked  \
0    female   NaN      0      0              335432   7.7333   NaN        Q
1      male  38.0      0      0              315089   8.6625   NaN        S
2    female  30.0      1      1              345773  24.1500   NaN        S
3    female  54.0      1      3               29105  23.0000   NaN        S
4      male  40.0      0      0               28221  13.0000   NaN        S
..      …     …     …     …                  …       …    …       …
845    male  55.0      0      0                 680  50.0000   C39        S
846    male  58.0      0      0               11771  29.7000   B37        C
847  female  24.0      1      0              244367  26.0000   NaN        S
848  female   3.0      1      1  SOTON/O.Q. 3101315  13.7750   NaN        S
849    male  52.0      0      0              250647  13.0000   NaN        S

    boat   body                  home.dest  survived
0     13    NaN                        NaN         1
1    NaN    NaN                    Croatia         0
2    NaN    NaN                        NaN         0
3      4    NaN        Cornwall / Akron, OH         1
4    NaN    NaN  Barre, Co Washington, VT         0
..   …     …                      …         …
845  NaN    NaN        London / Birmingham         0
846  NaN  258.0                Buffalo, NY         0
847   12    NaN          Moscow / Bronx, NY         1
848  NaN    NaN                        NaN         0
849  NaN   19.0                  Bronx, NY         0

[850 rows x 15 columns]
```

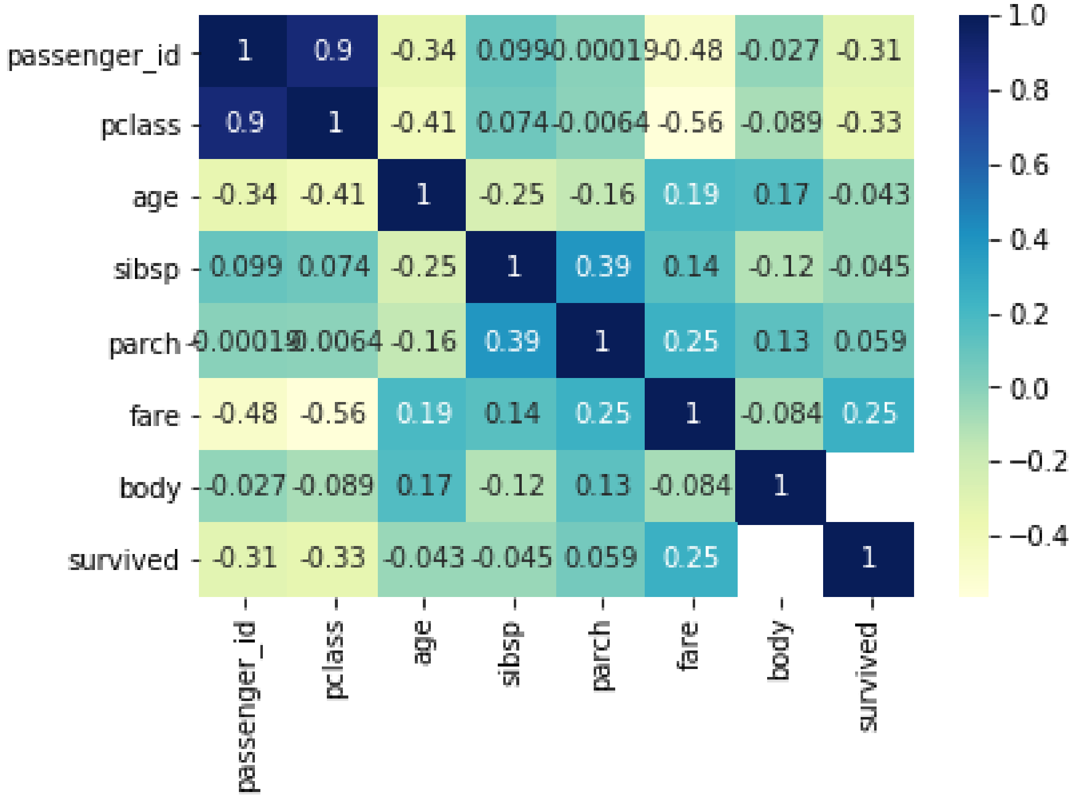
[4]:
```
titanic_data.describe()
# mean of sibsp and parch tell that most people travels alone
```

[4]:
```
       passenger_id     pclass         age       sibsp       parch  \
count    850.000000  850.00000  676.000000  850.000000  850.000000
mean     662.816471    2.32000   29.519847    0.522353    0.382353
std      380.751936    0.83853   14.562243    1.112132    0.879511
min        1.000000    1.00000    0.166700    0.000000    0.000000
25%      332.250000    2.00000   20.000000    0.000000    0.000000
50%      676.500000    3.00000   28.000000    0.000000    0.000000
75%      992.250000    3.00000   37.000000    1.000000    0.000000
max     1307.000000    3.00000   80.000000    8.000000    9.000000
```

```
             fare        body    survived
count  849.000000   73.000000  850.000000
mean    34.012701  165.821918    0.368235
std     53.705779   99.068487    0.482610
min      0.000000    4.000000    0.000000
25%      7.895800   75.000000    0.000000
50%     14.108300  166.000000    0.000000
75%     31.000000  260.000000    1.000000
max    512.329200  328.000000    1.000000
```

8 Correlation heat map indicating the correlation between certian features

[5]:
```python
import seaborn as sns

# .corr() method will get rid of the columns that are not suited for␣
↪correlation for you
sns.heatmap(titanic_data.corr(), annot=True, cmap="YlGnBu")
plt.show()
```



[6]:
```
sns.pairplot(titanic_data, height=1.5)
# Corelation is not the only thing but if any feature is positively or␣
↪negatively highly corelated with survival then
# you can say that is important feature
```

[6]: <seaborn.axisgrid.PairGrid at 0x7fddb95e6250>

[7]:
```
# Gender is not present in corelaton so we have to make gender to numeric 1 and␣
↪0
# You want to split the dataset to traning dataset and testing so not to get␣
↪biased So
# Drop unnecessary columns that donot influence the result
titanic_data.drop(['name', 'ticket', 'cabin'], axis=1, inplace=True)
```

9 3. Preprocess the data by converting categorical features into numerical ones, filling in missing values, and scaling the numerical features using sklearn.preprocessing.

[8]:
```
# Fill in missing values
titanic_data.fillna(titanic_data.mean(), inplace=True)
```

[9]:
```
titanic_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 850 entries, 0 to 849
Data columns (total 12 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   passenger_id  850 non-null    int64
 1   pclass        850 non-null    int64
 2   sex           850 non-null    object
 3   age           850 non-null    float64
 4   sibsp         850 non-null    int64
 5   parch         850 non-null    int64
 6   fare          850 non-null    float64
 7   embarked      849 non-null    object
 8   boat          308 non-null    object
 9   body          850 non-null    float64
 10  home.dest     464 non-null    object
 11  survived      850 non-null    int64
dtypes: float64(3), int64(5), object(4)
memory usage: 79.8+ KB
```

[10]:
```
# Convert categorical features into numerical ones
label_encoder = LabelEncoder()
titanic_data['sex'] = label_encoder.fit_transform(titanic_data['sex'])
titanic_data['age'].fillna(titanic_data['age'].mean(), inplace=True)
```

10 Fill in the missing values of embarked with "most-frequent strategy"

[11]:
```
# Fill in missing values
imputer = SimpleImputer(strategy='most_frequent')
titanic_data['embarked'] = imputer.fit_transform(titanic_data[['embarked']])
titanic_data.fillna(titanic_data.mean(), inplace=True)
```

[12]:
```
# Applying One-hot encode for the embarked feature
```

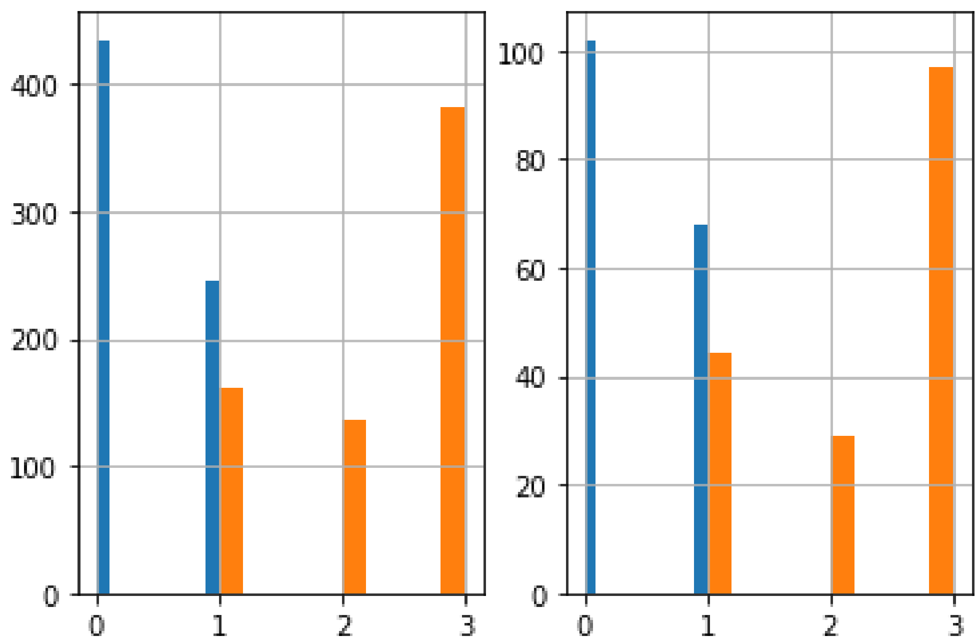
[13]:
```
titanic_data# One-hot encode the Embarked feature
onehot_encoder = OneHotEncoder(sparse=False)
embarked_onehot = onehot_encoder.fit_transform(titanic_data[['embarked']])
titanic_data = titanic_data.drop('embarked', axis=1)
titanic_data = pd.concat([titanic_data, pd.DataFrame(embarked_onehot,␣
↪columns=['C', 'Q', 'S'])], axis=1)
titanic_data['home.dest'] = label_encoder.fit_transform(titanic_data['home.
↪dest'].astype(str))
titanic_data['boat'] = titanic_data['boat'].fillna('0') # replace NaN values␣
↪with '0'
titanic_data['boat'] = label_encoder.fit_transform(titanic_data['boat'])
```

11 Split the dataset into training and test sets using sklearn.model_selection.train_test_split().

[14]:
```
# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(titanic_data.
↪drop(['survived'], axis=1), titanic_data['survived'], test_size=0.2,␣
↪random_state=42)
```

12 Graph show that both train and test datasets are splited uniformly

[15]:
```
plt.subplot(1,2,1)
y_train.hist()
X_train['pclass'].hist()

plt.subplot(1,2,2)
y_test.hist()
X_test['pclass'].hist()
plt.show()
# Both graphs look similar so both the graphs has quite similar distribution
```



13 Build an MLP classifier using sklearn.neural_network.MLPClassifier() and train it on the training data.

[16]:
```
print(X_train.shape)
print(y_train.shape)
```

```
(680, 13)
(680,)
```

[17]:
```
X_train.info
```

[17]:
```
<bound method DataFrame.info of      passenger_id  pclass  sex        age  sibsp
parch      fare  boat  \
332          1060       3    0  18.000000      0      0    7.7750    26
383           192       1    0  58.000000      0      0  146.5208     0
281          1100       3    0  29.000000      0      4   21.0750     0
2            1267       3    0  30.000000      1      1   24.1500     0
231            10       1    1  47.000000      1      0  227.5250     0
..            …       …   …         …      …      …        …     …
71            504       2    1  39.000000      0      0   13.0000     0
106           448       2    1  36.000000      0      0   13.0000     0
270           618       3    1  35.000000      0      0    8.0500     0
435           972       3    1  28.000000      0      0   56.4958     0
102          1214       3    1  29.519847      0      0    8.6625     0
```

```
           body  home.dest    C    Q    S
332  165.821918        272  0.0  0.0  1.0
383  165.821918        272  1.0  0.0  0.0
281  206.000000        272  0.0  0.0  1.0
2    165.821918        272  0.0  0.0  1.0
231  124.000000        171  1.0  0.0  0.0
..          …         …  …  …  …
71   165.821918        102  0.0  0.0  1.0
106  165.821918         61  0.0  0.0  1.0
270  165.821918        153  0.0  0.0  1.0
435  165.821918        272  0.0  0.0  1.0
102  165.821918        272  0.0  0.0  1.0

[680 rows x 13 columns]>
```

[18]:

```python
# Count the number of non-numeric values in each column
non_numeric_counts = titanic_data.isna().sum()

# Print the results
print(non_numeric_counts)
```

```
passenger_id    0
pclass          0
sex             0
age             0
sibsp           0
parch           0
fare            0
boat            0
body            0
home.dest       0
survived        0
C               0
Q               0
S               0
dtype: int64
```

[19]:

```python
# Build an MLP classifier and train it on the training data
mlp = MLPClassifier(hidden_layer_sizes=(100, 50), max_iter=1000, alpha=0.01,␣
 ↪solver='adam', random_state=42)
mlp.fit(X_train, y_train)
score = mlp.score(X_test, y_test)
print("Test accuracy:", score)
```

```
Test accuracy: 0.8941176470588236
```

[20]:
```
from sklearn.metrics import accuracy_score, precision_score, recall_score,␣
↪f1_score

# Make predictions on the test data
y_pred = mlp.predict(X_test)

# Calculate the accuracy
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)

# Calculate the precision
precision = precision_score(y_test, y_pred)
print("Precision:", precision)

# Calculate the recall
recall = recall_score(y_test, y_pred)
print("Recall:", recall)

# Calculate the F1-score
f1_score = f1_score(y_test, y_pred)
print("F1-score:", f1_score)
```

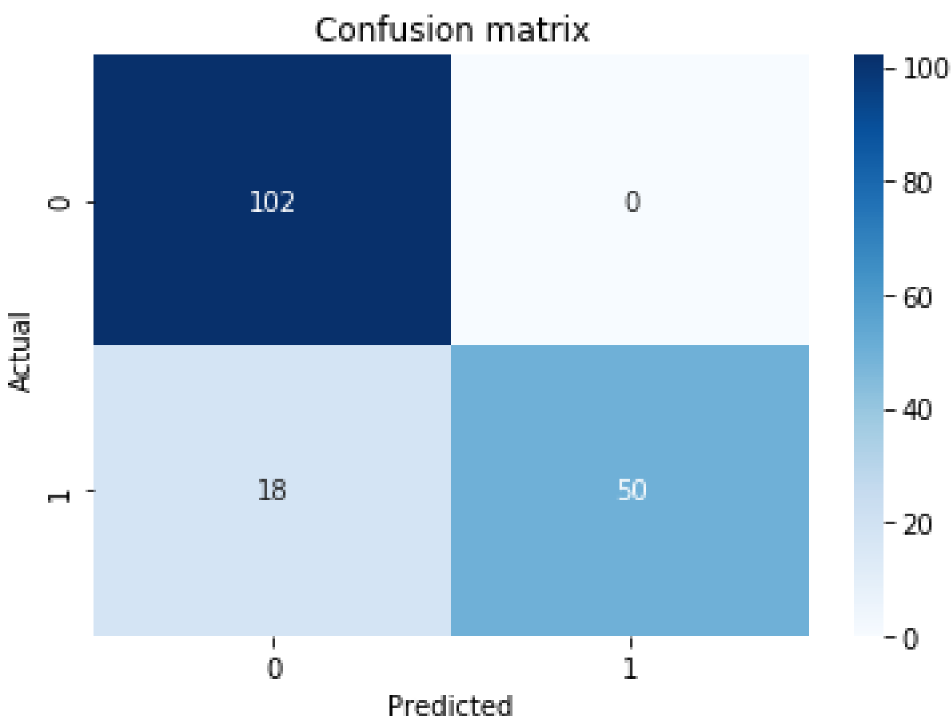
```
Accuracy: 0.8941176470588236
Precision: 1.0
Recall: 0.7352941176470589
F1-score: 0.8474576271186441
```

14 visualize the performance of an MLP classifier

[21]:
```
from sklearn.metrics import confusion_matrix, classification_report

# Get predicted labels for the test data
y_pred = mlp.predict(X_test)

# Compute confusion matrix
cm = confusion_matrix(y_test, y_pred)

# Plot confusion matrix
sns.heatmap(cm, annot=True, cmap='Blues', fmt='g')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion matrix')

# Compute classification report
cr = classification_report(y_test, y_pred)
```

```python
# Print classification report
print(cr)
```

```
              precision    recall  f1-score   support

           0       0.85      1.00      0.92       102
           1       1.00      0.74      0.85        68

    accuracy                           0.89       170
   macro avg       0.93      0.87      0.88       170
weighted avg       0.91      0.89      0.89       170
```



15 grid search => which exhaustively searches over specified hyperparameters and selects the best combination of hyperparameters that optimizes the performance metric of interest.

16 7. Fine-tune the MLP classifier by adjusting its hyperparameters, such as the number of hidden layers, and the number of neurons per layer.

[22]:
```python
from sklearn.model_selection import GridSearchCV

# Define the parameter grid to search over
param_grid = {
    'hidden_layer_sizes': [(50,), (100,), (50, 50), (100, 50)],
    'alpha': [0.0001, 0.001, 0.01, 0.1],
    'solver': ['adam', 'lbfgs'],
}

# Create an MLP classifier
mlp = MLPClassifier(max_iter=1000, random_state=42)

# Perform grid search to find the best hyperparameters
grid_search = GridSearchCV(mlp, param_grid, cv=5, scoring='accuracy')
grid_search.fit(X_train, y_train)

# Print the best hyperparameters and the corresponding score
print("Best hyperparameters:", grid_search.best_params_)
print("Best score:", grid_search.best_score_)
```

```
Best hyperparameters: {'alpha': 0.001, 'hidden_layer_sizes': (50,), 'solver':
'adam'}
Best score: 0.9308823529411765
```

17 Let's do 10000 iterations and then calculate hyperparameters

[23]:
```python
from sklearn.model_selection import GridSearchCV

# Define the parameter grid to search over
param_grid = {
    'hidden_layer_sizes': [(50,), (100,), (50, 50), (100, 50)],
    'alpha': [0.0001, 0.001, 0.01, 0.1],
    'solver': ['adam', 'lbfgs'],
}

# Create an MLP classifier
mlp = MLPClassifier(max_iter=10000, random_state=42)
```

```python
# Perform grid search to find the best hyperparameters
grid_search = GridSearchCV(mlp, param_grid, cv=5, scoring='accuracy')
grid_search.fit(X_train, y_train)

# Print the best hyperparameters and the corresponding score
print("Best hyperparameters:", grid_search.best_params_)
print("Best score:", grid_search.best_score_)
```

```
Best hyperparameters: {'alpha': 0.001, 'hidden_layer_sizes': (50,), 'solver':
'adam'}
Best score: 0.9308823529411765
```

18 Increasing the number of iterations may not always result in an improvement in model performanc

19 8. Evaluate the performance of the fine-tuned MLP classifier on the test data and compare it to the initial model.

[24]:
```python
mlp = MLPClassifier(hidden_layer_sizes=(50,), max_iter=1000, alpha=0.001,␣
 ↪solver='adam', random_state=42)
mlp.fit(X_train, y_train)
```

[24]:
```
MLPClassifier(activation='relu', alpha=0.001, batch_size='auto', beta_1=0.9,
              beta_2=0.999, early_stopping=False, epsilon=1e-08,
              hidden_layer_sizes=(50,), learning_rate='constant',
              learning_rate_init=0.001, max_fun=15000, max_iter=1000,
              momentum=0.9, n_iter_no_change=10, nesterovs_momentum=True,
              power_t=0.5, random_state=42, shuffle=True, solver='adam',
              tol=0.0001, validation_fraction=0.1, verbose=False,
              warm_start=False)
```

[25]:
```python
score = mlp.score(X_test, y_test)
print("Test accuracy:", score)
```

```
Test accuracy: 0.9588235294117647
```

20 Test accuracy: 0.8941176470588236 (Intial model we have selected hyperparameters on assumption in that case accuracy score is 0.894)

21 Test accuracy: 0.9588235294117647 (Hyperparameters selected based on grid search that increased the accuracy to 0.95)

22 9. Discuss the results and insights gained from the experiment, and identify potential areas for further improvement.

On selecting hyperparameters randomly and based on the data we have 89% accuracy. Once we used grid search for selecting best hyperparameters our accuracy increased to 95%. - We can increase number of iteration that may give different hyperparameters and provide us greater accuracy. - Also we can use any other method like random search may be that gives us different hyperparameters that increase accuracy. - There are alot of missing values in the data may adding that values also impact the results. - Computing missing values with any other method may also increase accuracy of our model.