22p-9295-amber-task10

April 26, 2024

[349]:
```python
# Import necessary libraries
import pandas as pd  # For data manipulation and analysis
import numpy as np  # For numerical computations
from sklearn.impute import SimpleImputer  # For handling missing values
from sklearn.preprocessing import StandardScaler, LabelEncoder  # For data␣
 ↪preprocessing
from sklearn.model_selection import train_test_split  # For splitting data into␣
 ↪training and testing sets
from sklearn.neural_network import MLPClassifier  # For scikit-learn's MLP␣
 ↪classifier
from keras.models import Sequential  # For Keras' sequential model
from keras.layers import Dense  # For Keras' dense layer
from sklearn.metrics import accuracy_score  # For calculating accuracy
import matplotlib.pyplot as plt  # For plotting

# Load the Titanic dataset from a CSV file
df = pd.read_csv('titanic.csv')

# Display the first few rows of the dataset
df
```

[349]:
```
     PassengerId  Survived  Pclass  \
0              1         0       3
1              2         1       1
2              3         1       3
3              4         1       1
4              5         0       3
..           …         …     …
886          887         0       2
887          888         1       1
888          889         0       3
889          890         1       1
890          891         0       3

                                                  Name     Sex   Age  SibSp  \
0                              Braund, Mr. Owen Harris    male  22.0      1
1    Cumings, Mrs. John Bradley (Florence Briggs Th…  female  38.0      1
```

```
2                                    Heikkinen, Miss. Laina  female  26.0      0
3          Futrelle, Mrs. Jacques Heath (Lily May Peel)  female  35.0      1
4                                  Allen, Mr. William Henry    male  35.0      0
..                                                      …       …   …     …
886                                   Montvila, Rev. Juozas    male  27.0      0
887                            Graham, Miss. Margaret Edith  female  19.0      0
888                Johnston, Miss. Catherine Helen "Carrie"  female   NaN      1
889                                   Behr, Mr. Karl Howell    male  26.0      0
890                                     Dooley, Mr. Patrick    male  32.0      0

     Parch            Ticket     Fare Cabin Embarked
0        0         A/5 21171   7.2500   NaN        S
1        0          PC 17599  71.2833   C85        C
2        0  STON/O2. 3101282   7.9250   NaN        S
3        0            113803  53.1000  C123        S
4        0            373450   8.0500   NaN        S
..     …               …      …   …      …
886      0            211536  13.0000   NaN        S
887      0            112053  30.0000   B42        S
888      2        W./C. 6607  23.4500   NaN        S
889      0            111369  30.0000  C148        C
890      0            370376   7.7500   NaN        Q

[891 rows x 12 columns]
```

[350]:
```python
# df.drop_duplicates(inplace=True)
```

[351]:
```python
# Drop unnecessary columns from the dataset
df.drop(['PassengerId', 'Name', 'Ticket', 'Cabin','Embarked'], axis=1,␣
 ↪inplace=True)
```

[352]:
```python
# Create a LabelEncoder object

encoder = LabelEncoder()

gender_encoded=encoder.fit_transform(df['Sex'])
df['Sex']=gender_encoded



df
```

[352]:
```
     Survived  Pclass  Sex   Age  SibSp  Parch     Fare
0           0       3    1  22.0      1      0   7.2500
1           1       1    0  38.0      1      0  71.2833
2           1       3    0  26.0      0      0   7.9250
3           1       1    0  35.0      1      0  53.1000
4           0       3    1  35.0      0      0   8.0500
```

```
..          …     …  …   …     …     …       …
886          0       2    1  27.0      0       0  13.0000
887          1       1    0  19.0      0       0  30.0000
888          0       3    0   NaN      1       2  23.4500
889          1       1    1  26.0      0       0  30.0000
890          0       3    1  32.0      0       0   7.7500

[891 rows x 7 columns]
```

[353]:
```python
# Remove rows with missing values from the dataframe
df.dropna
```

[353]:
```
<bound method DataFrame.dropna of      Survived  Pclass  Sex   Age  SibSp  Parch
Fare
0            0       3    1  22.0      1       0   7.2500
1            1       1    0  38.0      1       0  71.2833
2            1       3    0  26.0      0       0   7.9250
3            1       1    0  35.0      1       0  53.1000
4            0       3    1  35.0      0       0   8.0500
..          …     …  …   …     …     …       …
886          0       2    1  27.0      0       0  13.0000
887          1       1    0  19.0      0       0  30.0000
888          0       3    0   NaN      1       2  23.4500
889          1       1    1  26.0      0       0  30.0000
890          0       3    1  32.0      0       0   7.7500

[891 rows x 7 columns]>
```

[354]:
```python
# Replace all occurrences of '-' with NaN (Not a Number) in the dataframe
df.replace('-', np.nan, inplace=True)
```

[355]:
```python
# Impute missing values in the dataframe using the mean strategy
imputer = SimpleImputer(strategy='mean')
df = pd.DataFrame(imputer.fit_transform(df), columns=df.columns)
```

[356]:
```python
# Scale the dataframe to have zero mean and unit variance

scaler = StandardScaler()
scaler.fit(df)
normalized_data = scaler.transform(df)
```

[357]:
```python
# Split the data into training and testing sets
target=df['Survived']
train_data, test_data, train_target, test_target = train_test_split(df, target,␣
↪test_size=0.2, random_state=42)
print(train_data.shape, test_data.shape, train_target.shape, test_target.shape)
```

[359]:
```python
# Scale the training and testing data using the StandardScaler

X_train_scaled = scaler.fit_transform(train_data)
X_test_scaled = scaler.transform(test_data)
```

[360]:
```python
# Train scikit-learn MLP model
mlp = MLPClassifier(hidden_layer_sizes=(50, 50), activation='relu',␣
 ↪solver='adam', alpha=0.001, batch_size=100, max_iter=1000)
mlp.fit(X_train_scaled, train_target)
```

[360]:
```
MLPClassifier(alpha=0.001, batch_size=100, hidden_layer_sizes=(50, 50),
              max_iter=1000)
```

[361]:
```python
# Create a neural network model with two hidden layers
model = Sequential()
model.add(Dense(2, input_dim=3, activation='sigmoid'))  # Hidden layer with 2␣
 ↪units
model.add(Dense(2, activation='sigmoid'))  # Output layer with 2 units
```

[362]:
```python
# Compile the neural network model with mean squared error loss, stochastic␣
 ↪gradient descent optimizer, and accuracy metric
model.compile(loss='mean_squared_error', optimizer='sgd', metrics=['accuracy'])
```

[363]:
```python
# Define a list of Multi-Layer Perceptron (MLP) classifiers with varying hidden␣
 ↪layer sizes
models_MLP = [
MLPClassifier(hidden_layer_sizes=(10,), max_iter=100),
MLPClassifier(hidden_layer_sizes=(10,20), max_iter=100),
MLPClassifier(hidden_layer_sizes=(10, 20, 50), max_iter=100),
MLPClassifier(hidden_layer_sizes=(10, 20, 50, 100), max_iter=100) ]
```

[364]:
```python
# Define a list of Keras neural network models with varying complexity

Models_Keras = [
Sequential([
Dense(10, input_dim=7, activation='relu'),
Dense(1, activation='sigmoid')
]),
Sequential([
Dense(10, input_dim=7, activation='relu'),
Dense(20, activation='relu'),
Dense(1, activation='sigmoid')
]),
Sequential([
Dense(10, input_dim=7, activation='relu'),
Dense(20, activation='relu'),
Dense(50, activation='relu'),
```

```
Dense(1, activation='sigmoid')
]),
Sequential([
Dense(10, input_dim=7, activation='relu'),
Dense(20, activation='relu'),
Dense(50, activation='relu'),
Dense(100, activation='relu'),
Dense(1, activation='sigmoid')
])
]
```

[365]:
```
# Evaluate the accuracy of each MLP model in the list

accuracy_mlp = []

for model in models_MLP:
    # Train the model on the training data
    model.fit(train_data, train_target)

    # Use the trained model to predict the test data
    y_pred = model.predict(test_data)

    # Calculate the accuracy of the model and append it to the accuracy list
    accuracy_mlp.append(accuracy_score(test_target, y_pred))

    # Print the accuracy of the current model
    print(accuracy_score(test_target, y_pred))
```

```
0.7262569832402235
0.8044692737430168
0.9497206703910615
1.0
```

[366]:
```
# Initialize an empty list to store the accuracy of each model
accuracy_keras = []

# Loop over each model in the list
for model in Models_Keras:
    model.compile(optimizer='adam', loss='binary_crossentropy',␣
 ↪metrics=['accuracy'])

    model.fit(train_data, train_target, epochs=20)

    accuracy = model.evaluate(test_data, test_target)

    accuracy_keras.append(accuracy[1])
```

```
print('Accuracy: %.2f' % accuracy[1])
```

```
Epoch 1/20
23/23            1s 3ms/step -
accuracy: 0.6633 - loss: 11.6237
Epoch 2/20
23/23            0s 2ms/step -
accuracy: 0.5846 - loss: 10.6611
Epoch 3/20
23/23            0s 2ms/step -
accuracy: 0.6138 - loss: 7.6850
Epoch 4/20
23/23            0s 1ms/step -
accuracy: 0.5314 - loss: 5.0217
Epoch 5/20
23/23            0s 1ms/step -
accuracy: 0.3748 - loss: 3.7097
Epoch 6/20
23/23            0s 3ms/step -
accuracy: 0.3633 - loss: 1.9405
Epoch 7/20
23/23            0s 3ms/step -
accuracy: 0.4333 - loss: 0.9570
Epoch 8/20
23/23            0s 5ms/step -
accuracy: 0.6639 - loss: 0.6828
Epoch 9/20
23/23            0s 2ms/step -
accuracy: 0.6697 - loss: 0.6846
Epoch 10/20
23/23            0s 3ms/step -
accuracy: 0.7073 - loss: 0.6131
Epoch 11/20
23/23            0s 3ms/step -
accuracy: 0.7045 - loss: 0.6047
Epoch 12/20
23/23            0s 10ms/step -
accuracy: 0.6909 - loss: 0.5901
Epoch 13/20
23/23            0s 3ms/step -
accuracy: 0.7230 - loss: 0.5629
Epoch 14/20
23/23            0s 2ms/step -
accuracy: 0.7344 - loss: 0.5505
Epoch 15/20
23/23            0s 5ms/step -
accuracy: 0.7669 - loss: 0.5222
```

```
Epoch 16/20
23/23            0s 3ms/step -
accuracy: 0.7479 - loss: 0.5287
Epoch 17/20
23/23            0s 2ms/step -
accuracy: 0.7367 - loss: 0.5163
Epoch 18/20
23/23            0s 2ms/step -
accuracy: 0.7431 - loss: 0.5139
Epoch 19/20
23/23            0s 2ms/step -
accuracy: 0.7698 - loss: 0.4940
Epoch 20/20
23/23            0s 2ms/step -
accuracy: 0.7759 - loss: 0.4846
6/6            0s 2ms/step -
accuracy: 0.7712 - loss: 0.4508
Accuracy: 0.80
Epoch 1/20
23/23            2s 2ms/step -
accuracy: 0.6720 - loss: 1.3223
Epoch 2/20
23/23            0s 2ms/step -
accuracy: 0.6694 - loss: 0.9535
Epoch 3/20
23/23            0s 3ms/step -
accuracy: 0.6633 - loss: 0.7226
Epoch 4/20
23/23            0s 2ms/step -
accuracy: 0.6764 - loss: 0.6551
Epoch 5/20
23/23            0s 2ms/step -
accuracy: 0.7312 - loss: 0.5588
Epoch 6/20
23/23            0s 2ms/step -
accuracy: 0.7061 - loss: 0.5781
Epoch 7/20
23/23            0s 2ms/step -
accuracy: 0.7544 - loss: 0.5530
Epoch 8/20
23/23            0s 2ms/step -
accuracy: 0.7374 - loss: 0.5385
Epoch 9/20
23/23            0s 2ms/step -
accuracy: 0.7948 - loss: 0.5106
Epoch 10/20
23/23            0s 2ms/step -
accuracy: 0.7655 - loss: 0.5037
```

```
Epoch 11/20
23/23            0s 2ms/step -
accuracy: 0.8123 - loss: 0.4832
Epoch 12/20
23/23            0s 2ms/step -
accuracy: 0.8260 - loss: 0.4576
Epoch 13/20
23/23            0s 2ms/step -
accuracy: 0.8143 - loss: 0.4626
Epoch 14/20
23/23            0s 2ms/step -
accuracy: 0.8241 - loss: 0.4683
Epoch 15/20
23/23            0s 2ms/step -
accuracy: 0.7932 - loss: 0.4747
Epoch 16/20
23/23            0s 3ms/step -
accuracy: 0.8463 - loss: 0.4511
Epoch 17/20
23/23            0s 3ms/step -
accuracy: 0.8407 - loss: 0.4165
Epoch 18/20
23/23            0s 3ms/step -
accuracy: 0.8744 - loss: 0.3857
Epoch 19/20
23/23            0s 4ms/step -
accuracy: 0.8409 - loss: 0.3935
Epoch 20/20
23/23            0s 2ms/step -
accuracy: 0.8956 - loss: 0.3576
6/6            0s 2ms/step -
accuracy: 0.9101 - loss: 0.3155
Accuracy: 0.91
Epoch 1/20
23/23            2s 2ms/step -
accuracy: 0.6119 - loss: 2.6383
Epoch 2/20
23/23            0s 2ms/step -
accuracy: 0.5122 - loss: 0.7660
Epoch 3/20
23/23            0s 2ms/step -
accuracy: 0.6508 - loss: 0.6626
Epoch 4/20
23/23            0s 2ms/step -
accuracy: 0.7042 - loss: 0.6123
Epoch 5/20
23/23            0s 2ms/step -
accuracy: 0.6684 - loss: 0.6205
```

```
Epoch 6/20
23/23            0s 2ms/step -
accuracy: 0.6897 - loss: 0.6237
Epoch 7/20
23/23            0s 2ms/step -
accuracy: 0.6763 - loss: 0.6077
Epoch 8/20
23/23            0s 2ms/step -
accuracy: 0.6953 - loss: 0.5851
Epoch 9/20
23/23            0s 2ms/step -
accuracy: 0.6997 - loss: 0.5964
Epoch 10/20
23/23            0s 2ms/step -
accuracy: 0.7033 - loss: 0.6023
Epoch 11/20
23/23            0s 2ms/step -
accuracy: 0.6924 - loss: 0.5910
Epoch 12/20
23/23            0s 2ms/step -
accuracy: 0.6694 - loss: 0.6067
Epoch 13/20
23/23            0s 2ms/step -
accuracy: 0.6817 - loss: 0.5952
Epoch 14/20
23/23            0s 2ms/step -
accuracy: 0.6734 - loss: 0.6136
Epoch 15/20
23/23            0s 2ms/step -
accuracy: 0.6745 - loss: 0.5834
Epoch 16/20
23/23            0s 2ms/step -
accuracy: 0.6913 - loss: 0.5990
Epoch 17/20
23/23            0s 2ms/step -
accuracy: 0.6871 - loss: 0.6041
Epoch 18/20
23/23            0s 2ms/step -
accuracy: 0.6970 - loss: 0.5889
Epoch 19/20
23/23            0s 2ms/step -
accuracy: 0.6920 - loss: 0.5683
Epoch 20/20
23/23            0s 2ms/step -
accuracy: 0.6939 - loss: 0.5812
6/6            0s 2ms/step -
accuracy: 0.7135 - loss: 0.5709
Accuracy: 0.73
```

```
Epoch 1/20
23/23            6s 6ms/step -
accuracy: 0.5656 - loss: 0.7269
Epoch 2/20
23/23            0s 3ms/step -
accuracy: 0.6860 - loss: 0.6113
Epoch 3/20
23/23            0s 3ms/step -
accuracy: 0.7001 - loss: 0.6055
Epoch 4/20
23/23            0s 2ms/step -
accuracy: 0.7248 - loss: 0.5756
Epoch 5/20
23/23            0s 2ms/step -
accuracy: 0.7362 - loss: 0.5439
Epoch 6/20
23/23            0s 4ms/step -
accuracy: 0.7822 - loss: 0.4639
Epoch 7/20
23/23            0s 2ms/step -
accuracy: 0.8074 - loss: 0.4340
Epoch 8/20
23/23            0s 4ms/step -
accuracy: 0.8879 - loss: 0.3888
Epoch 9/20
23/23            0s 3ms/step -
accuracy: 0.9032 - loss: 0.2597
Epoch 10/20
23/23            0s 2ms/step -
accuracy: 0.9234 - loss: 0.2752
Epoch 11/20
23/23            0s 2ms/step -
accuracy: 0.9406 - loss: 0.1973
Epoch 12/20
23/23            0s 2ms/step -
accuracy: 0.9568 - loss: 0.1804
Epoch 13/20
23/23            0s 2ms/step -
accuracy: 0.9721 - loss: 0.1267
Epoch 14/20
23/23            0s 3ms/step -
accuracy: 0.9724 - loss: 0.1163
Epoch 15/20
23/23            0s 4ms/step -
accuracy: 0.9757 - loss: 0.1140
Epoch 16/20
23/23            0s 2ms/step -
accuracy: 0.9775 - loss: 0.0785
```

```
Epoch 17/20
23/23              0s 3ms/step -
accuracy: 0.9760 - loss: 0.0764
Epoch 18/20
23/23              0s 2ms/step -
accuracy: 0.9728 - loss: 0.0992
Epoch 19/20
23/23              0s 2ms/step -
accuracy: 0.9822 - loss: 0.0529
Epoch 20/20
23/23              0s 2ms/step -
accuracy: 0.9752 - loss: 0.0652
6/6              0s 2ms/step -
accuracy: 0.9877 - loss: 0.0432
Accuracy: 0.98
```

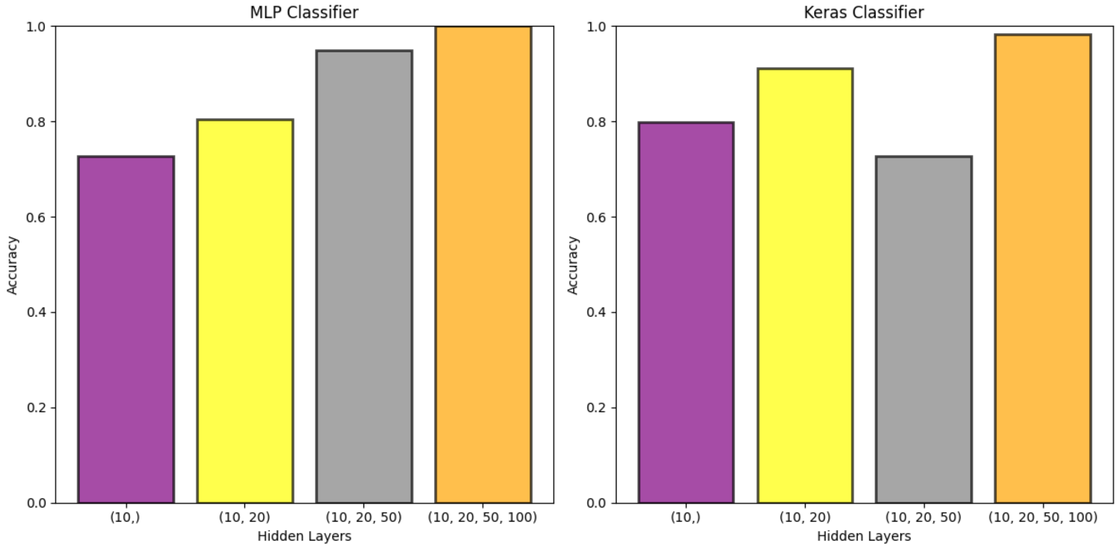
[367]:

```
# Visualize the accuracy of MLP and Keras classifiers with varying hidden␣
↪layers using bar charts
plt.figure(figsize=(12, 6))

plt.subplot(1, 2, 1)
plt.bar(range(4), accuracy_mlp, color=['purple', 'yellow', 'gray', 'orange'],␣
↪alpha=0.7, edgecolor='black', linewidth=2)
plt.xticks(range(4), ['(10,)', '(10, 20)', '(10, 20, 50)', '(10, 20, 50, 100)'])
plt.title('MLP Classifier')
plt.xlabel('Hidden Layers')
plt.ylabel('Accuracy')
plt.ylim([0, 1])

plt.subplot(1, 2, 2)
plt.bar(range(4), accuracy_keras, color=['purple', 'yellow', 'gray', 'orange'],␣
↪alpha=0.7, edgecolor='black', linewidth=2)
plt.xticks(range(4), ['(10,)', '(10, 20)', '(10, 20, 50)', '(10, 20, 50, 100)'])
plt.title('Keras Classifier')
plt.xlabel('Hidden Layers')
plt.ylabel('Accuracy')
plt.ylim([0, 1])

plt.tight_layout()
plt.show()
```


MLP Classifier
Keras Classifier
Accuracy
0.0
0.2
0.4
0.6
0.8
1.0
(10,)
(10, 20)
(10, 20, 50)
(10, 20, 50, 100)
Hidden Layers