22p-9252-Tazmeen-Afroz-Lab-10

April 26, 2024

[ ]:
```
# This Python 3 environment comes with many helpful analytics libraries␣
↪installed
# It is defined by the kaggle/python Docker image: https://github.com/kaggle/
↪docker-python
# For example, here's several helpful packages to load

import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)

# Input data files are available in the read-only "../input/" directory
# For example, running this (by clicking run or pressing Shift+Enter) will list␣
↪all files under the input directory

import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))

# You can write up to 20GB to the current directory (/kaggle/working/) that␣
↪gets preserved as output when you create a version using "Save & Run All"
# You can also write temporary files to /kaggle/temp/, but they won't be saved␣
↪outside of the current session
```

```
/kaggle/input/titanic-dataset/titanic.csv
```

[ ]:
```
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.impute import SimpleImputer
from sklearn.model_selection import train_test_split
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import accuracy_score
from keras.models import Sequential
from keras.layers import Dense
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings('ignore')
```

```
'''1. Data Preprocessing:
• Load the Titanic dataset (titanic.csv).
• Preprocess the dataset by handling missing values and removing unnecessary␣
↪columns.
• Convert categorical data into numeric format using techniques like one-hot␣
↪encoding or
label encoding.'''


# Load dataset
df = pd.read_csv('/kaggle/input/titanic-dataset/titanic.csv')

# Preprocessing
df.drop(['PassengerId', 'Name', 'Ticket', 'Cabin'], axis=1, inplace=True)


# Encode categorical features
le = LabelEncoder()
df['Sex'] = le.fit_transform(df['Sex'])
df['Embarked'] = le.fit_transform(df['Embarked'])

# Impute missing values
imputer = SimpleImputer(strategy='mean')
df = pd.DataFrame(imputer.fit_transform(df), columns=df.columns)

df.head()
```

[ ]:

	Survived	Pclass	Sex	Age	SibSp	Parch	Fare	Embarked
0	0.0	3.0	1.0	22.0	1.0	0.0	7.2500	2.0
1	1.0	1.0	0.0	38.0	1.0	0.0	71.2833	0.0
2	1.0	3.0	0.0	26.0	0.0	0.0	7.9250	2.0
3	1.0	1.0	0.0	35.0	1.0	0.0	53.1000	2.0
4	0.0	3.0	1.0	35.0	0.0	0.0	8.0500	2.0

[ ]:

```
'''2. Normalization:
• If necessary, normalize the numeric features to ensure that they are on a␣
↪similar scale.'''
X = df.drop('Survived', axis=1)
y = df['Survived']

# Standardize features
scaler = StandardScaler()
X = pd.DataFrame(scaler.fit_transform(X), columns=X.columns)
```

[ ]:

```
'''3. Model Training:
• Train a predictive model using both scikit-learn and Keras with the following␣
↪parameters:
```

```
• Model: MLP for scikit-learn, and deep neural networks with different numbers␣
 ↪of
hidden layers and units for Keras.
• Use the same parameters for the sklearn and keras models and vary the␣
 ↪parameters
(number of hidden layers and units) for the neural networks.
• Split the dataset into training and testing sets (e.g., 80% training, 20%␣
 ↪testing).
• Train the models on the training data.'''



# Split dataset
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)

print(X_train.shape, X_test.shape, y_train.shape, y_test.shape)


# Define a list of MLPClassifier models with different architectures
models_MLP = [
    MLPClassifier(hidden_layer_sizes=(10,), max_iter=100),  # Single hidden␣
 ↪layer with 10 neurons
    MLPClassifier(hidden_layer_sizes=(10,20), max_iter=100),  # Two hidden␣
 ↪layers with 10 and 20 neurons respectively
    MLPClassifier(hidden_layer_sizes=(10, 20, 50), max_iter=100),  # Three␣
 ↪hidden layers with 10, 20, and 50 neurons respectively
    MLPClassifier(hidden_layer_sizes=(10, 20, 50, 100), max_iter=100)  # Four␣
 ↪hidden layers with 10, 20, 50, and 100 neurons respectively
]
```

```
(712, 7) (179, 7) (712,) (179,)
```

[ ]:

```
# Define a list of Keras Sequential models with different architectures
Models_Keras = [
    Sequential([
        Dense(10, input_dim=7, activation='relu'),  # First layer with 10␣
 ↪neurons and ReLU activation
        Dense(1, activation='sigmoid')  # Output layer with 1 neuron (binary␣
 ↪classification) and sigmoid activation
    ]),
    Sequential([
        Dense(10, input_dim=7, activation='relu'),  # First layer with 10␣
 ↪neurons and ReLU activation
        Dense(20, activation='relu'),  # Second layer with 20 neurons and ReLU␣
 ↪activation
```

```
        Dense(1, activation='sigmoid')  # Output layer with 1 neuron and␣
↪sigmoid activation
    ]),
    Sequential([
        Dense(10, input_dim=7, activation='relu'),  # First layer with 10␣
↪neurons and ReLU activation
        Dense(20, activation='relu'),  # Second layer with 20 neurons and ReLU␣
↪activation
        Dense(50, activation='relu'),  # Third layer with 50 neurons and ReLU␣
↪activation
        Dense(1, activation='sigmoid')  # Output layer with 1 neuron and␣
↪sigmoid activation
    ]),
    Sequential([
        Dense(10, input_dim=7, activation='relu'),  # First layer with 10␣
↪neurons and ReLU activation
        Dense(20, activation='relu'),  # Second layer with 20 neurons and ReLU␣
↪activation
        Dense(50, activation='relu'),  # Third layer with 50 neurons and ReLU␣
↪activation
        Dense(100, activation='relu'),  # Fourth layer with 100 neurons and␣
↪ReLU activation
        Dense(1, activation='sigmoid')  # Output layer with 1 neuron and␣
↪sigmoid activation
    ])
]
```

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

# Loop over each model in the list
for model in models_MLP:
    # Train the model on the training data
    model.fit(X_train, y_train)

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

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

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

```
0.776536312849162
0.8100558659217877
```

```
0.8156424581005587
0.8156424581005587
```

[ ]:
```python
# 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:
    # Compile the model with the Adam optimizer, binary cross-entropy loss␣
 ↪function, and accuracy as the metric
    model.compile(optimizer='adam', loss='binary_crossentropy',␣
 ↪metrics=['accuracy'])

    # Train the model on the training data for 20 epochs
    model.fit(X_train, y_train, epochs=20)

    # Evaluate the model on the test data and get the loss and accuracy
    _, accuracy = model.evaluate(X_test, y_test)

    # Append the accuracy to the accuracy list
    accuracy_keras.append(accuracy)

    # Print the accuracy of the current model
    print('Accuracy: %.2f' % (accuracy*100))
```

```
Epoch 1/20
23/23                 2s 3ms/step -
accuracy: 0.5226 - loss: 0.6606
Epoch 2/20
23/23                 0s 2ms/step -
accuracy: 0.6757 - loss: 0.6324
Epoch 3/20
23/23                 0s 3ms/step -
accuracy: 0.7715 - loss: 0.6067
Epoch 4/20
23/23                 0s 3ms/step -
accuracy: 0.7881 - loss: 0.5681
Epoch 5/20
23/23                 0s 3ms/step -
accuracy: 0.7932 - loss: 0.5521
Epoch 6/20
23/23                 0s 3ms/step -
accuracy: 0.8010 - loss: 0.5324
Epoch 7/20
23/23                 0s 3ms/step -
accuracy: 0.7858 - loss: 0.5242
Epoch 8/20
23/23                 0s 2ms/step -
```

```
accuracy: 0.8067 - loss: 0.5110
Epoch 9/20
23/23              0s 2ms/step -
accuracy: 0.8042 - loss: 0.4925
Epoch 10/20
23/23              0s 2ms/step -
accuracy: 0.7904 - loss: 0.5068
Epoch 11/20
23/23              0s 2ms/step -
accuracy: 0.8050 - loss: 0.4826
Epoch 12/20
23/23              0s 2ms/step -
accuracy: 0.8117 - loss: 0.4700
Epoch 13/20
23/23              0s 2ms/step -
accuracy: 0.8239 - loss: 0.4565
Epoch 14/20
23/23              0s 3ms/step -
accuracy: 0.8095 - loss: 0.4740
Epoch 15/20
23/23              0s 3ms/step -
accuracy: 0.8175 - loss: 0.4627
Epoch 16/20
23/23              0s 3ms/step -
accuracy: 0.8228 - loss: 0.4339
Epoch 17/20
23/23              0s 2ms/step -
accuracy: 0.8218 - loss: 0.4475
Epoch 18/20
23/23              0s 3ms/step -
accuracy: 0.8181 - loss: 0.4386
Epoch 19/20
23/23              0s 2ms/step -
accuracy: 0.8145 - loss: 0.4416
Epoch 20/20
23/23              0s 3ms/step -
accuracy: 0.7960 - loss: 0.4564
6/6              0s 3ms/step -
accuracy: 0.7997 - loss: 0.4247
Accuracy: 78.21
Epoch 1/20
23/23              2s 3ms/step -
accuracy: 0.5218 - loss: 0.6777
Epoch 2/20
23/23              0s 2ms/step -
accuracy: 0.7018 - loss: 0.6408
Epoch 3/20
23/23              0s 2ms/step -
```

```
accuracy: 0.7231 - loss: 0.6215
Epoch 4/20
23/23              0s 2ms/step -
accuracy: 0.7466 - loss: 0.5968
Epoch 5/20
23/23              0s 3ms/step -
accuracy: 0.7662 - loss: 0.5649
Epoch 6/20
23/23              0s 3ms/step -
accuracy: 0.7748 - loss: 0.5278
Epoch 7/20
23/23              0s 3ms/step -
accuracy: 0.7725 - loss: 0.5092
Epoch 8/20
23/23              0s 3ms/step -
accuracy: 0.7722 - loss: 0.5041
Epoch 9/20
23/23              0s 3ms/step -
accuracy: 0.8105 - loss: 0.4617
Epoch 10/20
23/23              0s 3ms/step -
accuracy: 0.7709 - loss: 0.4818
Epoch 11/20
23/23              0s 2ms/step -
accuracy: 0.8103 - loss: 0.4515
Epoch 12/20
23/23              0s 2ms/step -
accuracy: 0.7974 - loss: 0.4566
Epoch 13/20
23/23              0s 3ms/step -
accuracy: 0.8189 - loss: 0.4250
Epoch 14/20
23/23              0s 4ms/step -
accuracy: 0.8209 - loss: 0.4398
Epoch 15/20
23/23              0s 3ms/step -
accuracy: 0.8135 - loss: 0.4343
Epoch 16/20
23/23              0s 3ms/step -
accuracy: 0.8024 - loss: 0.4613
Epoch 17/20
23/23              0s 3ms/step -
accuracy: 0.8473 - loss: 0.3976
Epoch 18/20
23/23              0s 3ms/step -
accuracy: 0.7946 - loss: 0.4429
Epoch 19/20
23/23              0s 3ms/step -
```

```
accuracy: 0.8398 - loss: 0.3907
Epoch 20/20
23/23            0s 3ms/step -
accuracy: 0.8240 - loss: 0.4083
6/6            0s 2ms/step -
accuracy: 0.7918 - loss: 0.4379
Accuracy: 77.65
Epoch 1/20
23/23            2s 2ms/step -
accuracy: 0.6108 - loss: 0.6685
Epoch 2/20
23/23            0s 3ms/step -
accuracy: 0.6484 - loss: 0.6257
Epoch 3/20
23/23            0s 3ms/step -
accuracy: 0.6719 - loss: 0.6113
Epoch 4/20
23/23            0s 3ms/step -
accuracy: 0.7435 - loss: 0.5412
Epoch 5/20
23/23            0s 2ms/step -
accuracy: 0.7948 - loss: 0.4934
Epoch 6/20
23/23            0s 3ms/step -
accuracy: 0.7893 - loss: 0.4855
Epoch 7/20
23/23            0s 3ms/step -
accuracy: 0.8299 - loss: 0.4378
Epoch 8/20
23/23            0s 3ms/step -
accuracy: 0.8136 - loss: 0.4411
Epoch 9/20
23/23            0s 3ms/step -
accuracy: 0.8032 - loss: 0.4352
Epoch 10/20
23/23            0s 3ms/step -
accuracy: 0.8034 - loss: 0.4343
Epoch 11/20
23/23            0s 3ms/step -
accuracy: 0.8230 - loss: 0.4186
Epoch 12/20
23/23            0s 3ms/step -
accuracy: 0.8269 - loss: 0.3967
Epoch 13/20
23/23            0s 3ms/step -
accuracy: 0.8165 - loss: 0.4105
Epoch 14/20
23/23            0s 2ms/step -
```

```
accuracy: 0.8191 - loss: 0.4063
Epoch 15/20
23/23            0s 3ms/step -
accuracy: 0.8357 - loss: 0.3857
Epoch 16/20
23/23            0s 3ms/step -
accuracy: 0.8332 - loss: 0.3919
Epoch 17/20
23/23            0s 3ms/step -
accuracy: 0.8275 - loss: 0.3961
Epoch 18/20
23/23            0s 2ms/step -
accuracy: 0.8327 - loss: 0.3974
Epoch 19/20
23/23            0s 3ms/step -
accuracy: 0.8337 - loss: 0.3897
Epoch 20/20
23/23            0s 3ms/step -
accuracy: 0.8308 - loss: 0.3819
6/6            0s 3ms/step -
accuracy: 0.7888 - loss: 0.4429
Accuracy: 77.65
Epoch 1/20
23/23            2s 3ms/step -
accuracy: 0.6260 - loss: 0.6799
Epoch 2/20
23/23            0s 3ms/step -
accuracy: 0.7675 - loss: 0.6075
Epoch 3/20
23/23            0s 3ms/step -
accuracy: 0.8072 - loss: 0.5060
Epoch 4/20
23/23            0s 2ms/step -
accuracy: 0.8134 - loss: 0.4547
Epoch 5/20
23/23            0s 2ms/step -
accuracy: 0.8057 - loss: 0.4390
Epoch 6/20
23/23            0s 2ms/step -
accuracy: 0.7917 - loss: 0.4534
Epoch 7/20
23/23            0s 3ms/step -
accuracy: 0.8322 - loss: 0.3766
Epoch 8/20
23/23            0s 2ms/step -
accuracy: 0.8381 - loss: 0.3730
Epoch 9/20
23/23            0s 3ms/step -
```

```
accuracy: 0.8424 - loss: 0.3805
Epoch 10/20
23/23              0s 3ms/step -
accuracy: 0.8104 - loss: 0.4226
Epoch 11/20
23/23              0s 2ms/step -
accuracy: 0.8177 - loss: 0.3909
Epoch 12/20
23/23              0s 3ms/step -
accuracy: 0.8247 - loss: 0.3915
Epoch 13/20
23/23              0s 3ms/step -
accuracy: 0.8238 - loss: 0.4016
Epoch 14/20
23/23              0s 3ms/step -
accuracy: 0.8307 - loss: 0.3749
Epoch 15/20
23/23              0s 3ms/step -
accuracy: 0.8338 - loss: 0.3892
Epoch 16/20
23/23              0s 3ms/step -
accuracy: 0.8452 - loss: 0.3679
Epoch 17/20
23/23              0s 3ms/step -
accuracy: 0.8334 - loss: 0.3709
Epoch 18/20
23/23              0s 3ms/step -
accuracy: 0.8271 - loss: 0.4001
Epoch 19/20
23/23              0s 3ms/step -
accuracy: 0.8445 - loss: 0.3730
Epoch 20/20
23/23              0s 3ms/step -
accuracy: 0.8361 - loss: 0.3669
6/6              0s 3ms/step -
accuracy: 0.8235 - loss: 0.4323
Accuracy: 81.01
```

[ ]:
```
# Plotting

plt.figure(figsize=(12, 6))

plt.subplot(1, 2, 1)
plt.bar(range(4), accuracy_mlp, color= ['red', 'green', 'blue', 'orange'],␣
 ↪alpha=0.7, edgecolor='black', linewidth=2, tick_label=['(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= ['red', 'green', 'blue', 'orange'],␣
 ↪alpha=0.7, edgecolor='black', linewidth=2, tick_label=['(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.show()
```