Q3_current_4

January 7, 2023

[ ]:
```
import numpy as np
import pandas as pd
from matplotlib import pyplot as plt
import warnings
warnings.filterwarnings("ignore")
from sklearn.datasets import load_iris
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import RandomizedSearchCV
from scipy.stats import uniform
from sklearn.preprocessing import StandardScaler
from sklearn.datasets import load_digits
from sklearn.neural_network import MLPClassifier
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split, GridSearchCV,␣
↪StratifiedKFold
from sklearn.model_selection import ParameterGrid
```

3 MLP using sklearn.

In this problem, you will use the same dataset from Question 2 and implement a multi-layer perceptron using sklearn. Set aside 20% of the image for testing. Import the necessary classes and do a 5-fold cross-validation by defining a hyperparameter grid for the MLP classifier. Read about the hyperparameters supported and define a grid for them. Perform a random search on the grid that you have chosen. Report a single test accuracy with the best found hyperparameters.

[ ]:
```
X, y = load_digits(return_X_y=True)  # Load the digits dataset and get the␣
↪features (X) and labels (y)
rows, cols = X.shape  # Get the number of rows and columns in X
X = (X-np.mean(X))/np.std(X)  # Normalize the features by subtracting the mean␣
↪and dividing by the standard deviation
print('shapes X: {}, y: {}'.format(X.shape, y.shape))  # Print the shapes of X␣
↪and y
split = int(0.8 * rows)  # Calculate the split index for the training and test␣
↪sets (80% for training, 20% for testing)
p = np.random.permutation(rows)  # Generate a random permutation of the rows
x_train = X[p[:split]]  # Get the training features by selecting the first␣
↪"split" rows of the permuted X
```

```
y_train = y[p[:split]]  # Get the training labels by selecting the first␣
↪"split" rows of the permuted y
x_test = X[p[split:]]  # Get the test features by selecting the remaining rows␣
↪of the permuted X
y_test = y[p[split:]]  # Get the test labels by selecting the remaining rows of␣
↪the permuted y
```

```
shapes X: (1797, 64), y: (1797,)
```

[ ]:

```
clf = MLPClassifier(random_state=1, max_iter=300)  # Initialize a multi-layer␣
↪perceptron (MLP) classifier with a random state of 1 and max iterations of␣
↪300
param_grid = {
    'hidden_layer_sizes': [(5,), (10,), (20,)],  # Try hidden layer sizes of 5,␣
↪10, and 20 neurons
    'activation': ['relu', 'tanh'],  # Try ReLU and tanh activation functions
    'solver': ['adam', 'sgd'],  # Try Adam and SGD solvers
    'learning_rate': ['constant', 'invscaling', 'adaptive'],  # Try constant,␣
↪inverse scaling, and adaptive learning rates
    'max_iter': [10, 100, 150],  # Try max iteration values of 10, 100, and 150
    'alpha': [0.0001, 0.001, 0.01]  # Try alpha values of 0.0001, 0.001, and 0.
↪01
}
random_search = RandomizedSearchCV(clf, param_distributions=param_grid,␣
↪cv=StratifiedKFold(n_splits=5))  # Initialize a randomized search with␣
↪5-fold stratified cross-validation
random_search.fit(x_train, y_train)  # Fit the randomized search to the␣
↪training data
best_estimator = random_search.best_estimator_  # Get the best estimator from␣
↪the search
best_params = random_search.best_params_  # Get the best parameters from the␣
↪search
print('best estimator: ', best_estimator)  # Print the best estimator
print('best params: ', best_params)  # Print the best parameters
```

```
best estimator:  MLPClassifier(hidden_layer_sizes=(20,),
learning_rate='adaptive', max_iter=100,
              random_state=1)
best params:  {'solver': 'adam', 'max_iter': 100, 'learning_rate': 'adaptive',
'hidden_layer_sizes': (20,), 'alpha': 0.0001, 'activation': 'relu'}
```

[ ]:

```
test_accuracy = best_estimator.score(x_test, y_test) # Computing accuracy for␣
↪test set
print('Test Accuracy is: ', np.round(test_accuracy, 3)*100, '%')
```

```
Test Accuracy is:  95.3 %
```

```
[ ]:
```