EXERCISE	5. MULTIPLE LAYER PRCEPTRON
DATE	30.08.2024

AIM:
To design and implement feed forward multiple layer perceptron (MLP) trained using backpropagation algorithm to classify the given dataset.

DESCRIPTION:
A MLP is a class of feedforward artificial neural network trained using backpropagation algorithms. The architecture uses, at least, three layers of nodes: an input layer, a hidden layer and an output layer.
Algorithm:
Step 1: Select the number of input, hidden and output nodes
Step 2: Initialize the network and set the required hyperparameters like initial weights, hidden nodes, learning rate, etc.
Step 3: Read a pattern, calculate the output of nodes in the successive layers
Step 4: Calculate the error E, at the output node.
Step 5: Propagate the error and adapt weights based on error derivatives
Step 6: Continue from step 3 for the required number of epochs.
Step 7: Evaluate the performance of the model using metrics and plot the hyperplane.
Calculation metrics:
Accuracy: Measures the proportion of correct predictions among the total predictions.

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

TP = True Positives
TN = True Negatives
FP = False Positives
FN = False Negatives

Precision: Measures the proportion of true positive predictions among all positive predictions.

$$\text{Precision} = \frac{TP}{TP + FP}$$

Recall (Sensitivity): Measures the proportion of true positives among all actual positives.

$$\text{Recall} = \frac{TP}{TP + FN}$$

F1 Score: The harmonic mean of precision and recall, providing a balance between the two.

$$\text{F1 Score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

SOURCE CODE:

1. Implement MLP to classify the given data set and analyse the performance of the classifier.

```
import numpy as np
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import classification_report, accuracy_score
data = load_iris()
X = data.data
y = data.target
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
mlp = MLPClassifier(hidden_layer_sizes=(10, 10), max_iter=1000, learning_rate_init=0.01)
mlp.fit(X_train, y_train)
y_pred = mlp.predict(X_test)
print("Accuracy:", accuracy_score(y_test, y_pred))
print(classification_report(y_test, y_pred))
```

Output:

```
Accuracy: 1.0
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        10
           1       1.00      1.00      1.00         9
           2       1.00      1.00      1.00        11

    accuracy                           1.00        30
   macro avg       1.00      1.00      1.00        30
weighted avg       1.00      1.00      1.00        30
```

2. Implement MLP for regression task.

```
import numpy as np
from sklearn.datasets import fetch_california_housing
from sklearn.model_selection import train_test_split
from sklearn.neural_network import MLPRegressor
from sklearn.metrics import mean_squared_error
data = fetch_california_housing()
X = data.data
y = data.target
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
mlp_regressor = MLPRegressor(hidden_layer_sizes=(10, 10), max_iter=1000, learning_rate_init=0.01)
mlp_regressor.fit(X_train, y_train)
y_pred = mlp_regressor.predict(X_test)
# Evaluate performance
mse = mean_squared_error(y_test, y_pred)
print("Mean Squared Error:", mse)
```

Output:

```
Mean Squared Error: 0.6832276841927444
```

3. Improve the performance of the model by adjusting the hyperparameters such as number of hidden nodes, learning rate parameter, momentum etc.

```
from sklearn.model_selection import GridSearchCV
param_grid = {
    'hidden_layer_sizes': [(10,), (20,), (10, 10)],
    'learning_rate_init': [0.001, 0.01, 0.1],
    'momentum': [0.9, 0.95, 0.99]
}
grid_search = GridSearchCV(MLPRegressor(max_iter=1000), param_grid, cv=5)
grid_search.fit(X_train, y_train)
print("Best parameters:", grid_search.best_params_)
best_model = grid_search.best_estimator_
y_pred_best = best_model.predict(X_test)
mse_best = mean_squared_error(y_test, y_pred_best)
print("Mean Squared Error with best parameters:", mse_best)
```

Output:

```
Best parameters: {'hidden_layer_sizes': (10,), 'learning_rate_init': 0.1, 'momentum': 0.99}
Mean Squared Error with best parameters: 0.8751337363673857
```

4. Implement SVM to classify the given data set and analyse the performance of the classifier.

```
from sklearn import svm
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, accuracy_score
data = load_iris()
X = data.data
y = data.target
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
svm_classifier = svm.SVC(kernel='rbf', C=1.0, gamma='scale')
svm_classifier.fit(X_train, y_train)
y_pred = svm_classifier.predict(X_test)
print("Accuracy:", accuracy_score(y_test, y_pred))
print(classification_report(y_test, y_pred))
```

Output:

```
Accuracy: 1.0
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        10
           1       1.00      1.00      1.00         9
           2       1.00      1.00      1.00        11

    accuracy                           1.00        30
   macro avg       1.00      1.00      1.00        30
weighted avg       1.00      1.00      1.00        30
```

5. Implement SVM for regression task.

```
from sklearn import svm
from sklearn.datasets import fetch_california_housing
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
data = fetch_california_housing()
X = data.data
y = data.target
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
svm_regressor = svm.SVR(kernel='rbf', C=1.0, gamma='scale')
svm_regressor.fit(X_train, y_train)
y_pred = svm_regressor.predict(X_test)
mse_svm = mean_squared_error(y_test, y_pred)
print("Mean Squared Error for SVM:", mse_svm)
```

Output:

```
Mean Squared Error for SVM: 1.3320115421348744
```

6. Improve the performance of the model by adjusting the hyperparameters such as number of hidden nodes, learning rate parameter, momentum etc.

```
from sklearn import svm
from sklearn.datasets import fetch_california_housing
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.metrics import mean_squared_error
data = fetch_california_housing()
X = data.data
y = data.target
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
param_grid = {
    'C': [0.1, 1, 10],
    'gamma': ['scale', 'auto'],
    'kernel': ['linear', 'rbf', 'poly']
```

```
}
grid_search = GridSearchCV(svm.SVR(), param_grid, cv=5)
grid_search.fit(X_train, y_train)
print("Best parameters:", grid_search.best_params_)
best_model = grid_search.best_estimator_
y_pred_best = best_model.predict(X_test)
mse_best = mean_squared_error(y_test, y_pred_best)
print("Mean Squared Error with best parameters:", mse_best)
```

Output:

```
Mean Squared Error: 0.012750063504026138
```

```
Mean Squared Error: 48378.798164496344
```

RESULT:

The above multiple layer perceptron is successfully executed.