SOFT COMPUTING ASSIGNMENT -6

Perumalla Dharan
AP21110010201

1. Write a Python program to implement a Multi-Layer Perceptron (MLP) for classifying whether a student passes or fails using a dataset of students' course marks.

```
import numpy as np
import pandas as pd

train_path = r"E:\SRM\Soft Computing\Lab 6\students_testing.csv"
train_df = pd.read_csv(train_path)
test_path = r"E:\SRM\Soft Computing\Lab 6\training_dataset_students(1000).csv"
test_df = pd.read_csv(test_path)

X_train = train_df[['c1', 'c2', 'c3', 'c4', 'c5', 'c6']].values
y_train = train_df[['result']].values

X_test = test_df[['c1', 'c2', 'c3', 'c4', 'c5', 'c6']].values
y_test = test_df[['result']].values

class MLP:
    def __init__(self, input_size, hidden_size, learning_rate=0.1, iterations=100):
        self.learning_rate = learning_rate
        self.iterations = iterations
        self.weights_input_hidden = np.random.uniform(-0.5, 0.5, (input_size, hidden_size))
        self.bias_hidden = np.zeros(hidden_size)
        self.weights_hidden_output = np.random.uniform(-0.5, 0.5, (hidden_size, 1))
```

```
        self.bias_output = np.zeros(1)

    def sigmoid(self, x):
        return 1 / (1 + np.exp(-x))

    def predict(self, X):
        hidden_input = np.dot(X, self.weights_input_hidden) + self.bias_hidden
        hidden_output = self.sigmoid(hidden_input)

        final_input = np.dot(hidden_output, self.weights_hidden_output) + self.bias_output
        final_output = self.sigmoid(final_input)
        return np.round(final_output)

    def train(self, X, y):
        for epoch in range(self.iterations):
            for i in range(len(X)):
                hidden_input = np.dot(X[i], self.weights_input_hidden) + self.bias_hidden
                hidden_output = self.sigmoid(hidden_input)

                final_input = np.dot(hidden_output, self.weights_hidden_output) + self.bias_output
                final_output = self.sigmoid(final_input)
                y_pred = np.round(final_output)

                output_error = y[i] - final_output
                hidden_error = output_error.dot(self.weights_hidden_output.T) * hidden_output * (1 - hidden_output)

                self.weights_hidden_output += self.learning_rate * np.outer(hidden_output, output_error)
```

```
                        self.bias_output += self.learning_rate *
 output_error

                self.weights_input_hidden += self.learning_rate *
 np.outer(X[i], hidden_error)
                        self.bias_hidden += self.learning_rate *
 hidden_error

    def print_weights(self):
        print("Input-Hidden Weights:\n", self.weights_input_hidden)
                                print("Hidden-Output    Weights:\n",
 self.weights_hidden_output)

input_size = X_train.shape[1]
hidden_size = 4

mlp    =    MLP(input_size=input_size,    hidden_size=hidden_size,
 learning_rate=0.1, iterations=100)

mlp.train(X_train, y_train)

mlp.print_weights()

predictions = mlp.predict(X_test)
accuracy = np.mean(predictions == y_test)
print(f"Test Accuracy: {accuracy * 100:.2f}%")
```

```
Input-Hidden Weights:
 [[-0.29566566 -0.06310135  0.41898353 -0.05605992]
 [-0.087182   -0.60413012  0.17030378 -0.5394112 ]
 [-0.32295963 -0.08301213 -0.37512255 -0.98291623]
 [ 0.17780732 -0.84257121  0.14854828 -0.84396401]
 [-0.15031219 -0.37544889  0.17065824 -1.16808103]
 [-0.38596397 -0.20979318  0.34222245 -0.69259762]]
Hidden-Output Weights:
 [[-0.24410555]
 [ 1.13980737]
 [-0.5982828 ]
 [-0.55066481]]
Test Accuracy: 50.00%
```