**SOFT COMPUTING**

**ASSIGNMENT -6**

**Perumalla Dharan**

**AP21110010201**



*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}%")

