



NEURAL NETWORK



Name: Abdelrahman Salah Amin Elazab.

ID: 802320667

1-Hebbian algorithm.

```
1 import numpy as np
2 from sklearn.metrics import accuracy_score, confusion_matrix
3 from prettytable import PrettyTable
4 import matplotlib.pyplot as plt
5
6 class MyHebbian:
7     def __init__(self, n_iterations=1):
8         self.epochs = n_iterations
9         self.weights = None
10        self.bias = None
11
12    def fit(self, X, y):
13        self.weights = np.zeros(X.shape[1])
14        self.bias = 0
15
16        for epoch in range(self.epochs):
17            for i in range(X.shape[0]):
18                self.weights = self.weights + X[i] * y[i]
19                self.bias = self.bias + y[i]
20
21        pt = PrettyTable()
22        pt.field_names = ['Final W1', 'Final W2', 'Final Bias']
23
24        pt.add_row([self.weights[0], self.weights[1], self.bias])
25        print(pt)
26
27    def activation_function(self, activation):
28        if activation >= 0:
29            return 1
30        else:
31            return 0
32
33    def predict(self, X):
34        y_pred = []
35        for i in range(X.shape[0]):
36            y_pred.append(self.activation_function(np.dot(self.weights, X[i]) + self.bias))
37        return np.array(y_pred)
38
39 X = np.array([[1, 1], [1, 0], [0, 1], [0, 0]])
40 y = np.array([1, 0, 0, 0])
41
42 clf = MyHebbian()
43
44 clf.fit(X, y)
45 y_pred = clf.predict(X)
46 print("Accuracy:", accuracy_score(y, y_pred))
47
48 conf_matrix = confusion_matrix(y, np.sign(y_pred))
49 print("Confusion Matrix:")
50 print(conf_matrix)
51
52 plt.figure()
53 plt.scatter(X[:, 0], X[:, 1], c=y)
54 plt.plot([0, -clf.bias / clf.weights[0]], [-clf.bias / clf.weights[1], 0], 'r--')
55 plt.title("Decision Boundary")
56 plt.xlabel("X1")
57 plt.ylabel("X2")
58 plt.show()
```

2-Perceptron Algorithm.

```
1 import numpy as np
2 from sklearn.metrics import accuracy_score, confusion_matrix
3 from prettytable import PrettyTable
4
5 class MyPerceptron:
6     def __init__(self, learning_rate=1, n_iterations=1):
7         self.lr = learning_rate
8         self.epochs = n_iterations
9         self.weights = None
10        self.bias = None
11
12    def fit(self, X, y):
13        self.weights = np.zeros(X.shape[1])
14        self.bias = 0
15
16        for epoch in range(self.epochs):
17            for i in range(X.shape[0]):
18                y_pred = self.activation_function(np.dot(self.weights, X[i]) + self.bias)
19                self.weights = self.weights + self.lr * (y[i] - y_pred) * X[i]
20                self.bias = self.bias + self.lr * (y[i] - y_pred)
21
22            pt = PrettyTable()
23            pt.field_names = ['Final W1', 'Final W2', 'Final Bias']
24
25            pt.add_row([self.weights[0], self.weights[1], self.bias])
26            print(pt)
27
28    def activation_function(self, activation):
29        if activation > 0:
30            return 1
31        elif activation == 0:
32            return 0
33        else:
34            return -1
35
36    def predict(self, X):
37        y_pred=[]
38        for i in range(X.shape[0]):
39            y_pred.append(self.activation_function(np.dot(self.weights, X[i]) + self.bias))
40        return np.array(y_pred)
41
42 X = np.array([[1, 1], [1, 0], [0, 1], [0, 0]])
43 y = np.array([1, -1, -1, -1])
44
45 clf = MyPerceptron()
46
47 clf.fit(X, y)
48 y_pred = clf.predict(X)
49 print("Accuracy:", accuracy_score(y, y))
50
51 conf_matrix = confusion_matrix(y, np.sign(y_pred))
52 print("Confusion Matrix:")
53 print(conf_matrix)
```

3-Adaline Algorithm.

```
1 import numpy as np
2 from prettytable import PrettyTable
3 from sklearn.metrics import accuracy_score, confusion_matrix
4
5 class MyAdaline:
6     def __init__(self, learning_rate=0.1, n_iterations=5):
7         self.lr = learning_rate
8         self.epochs = n_iterations
9         self.weights = None
10        self.bias = None
11
12    def fit(self, X, y):
13        self.weights = np.array([0.1] * X.shape[1])
14        self.bias = 0.1
15
16        pt = PrettyTable()
17        pt.field_names = ['Epoch', 'Final W1', 'Final W2', 'Final Bias', 'Total Error']
18
19        for epoch in range(self.epochs):
20            total_error = 0
21            for i in range(X.shape[0]):
22                y_in = np.dot(self.weights, X[i]) + self.bias
23                error = y[i] - y_in
24                total_error += (y[i] - y_in) ** 2
25                self.weights = self.weights + self.lr * error * X[i]
26                self.bias = self.bias + self.lr * error
27
28            pt.add_row([epoch+1, self.weights[0], self.weights[1], self.bias, total_error])
29            print(pt)
30
31    def activation_function(self, activation):
32        if activation >= 0:
33            return 1
34        else:
35            return -1
36
37    def predict(self, X):
38        y_pred = []
39        for i in range(X.shape[0]):
40            y_pred.append(self.activation_function(np.dot(self.weights, X[i]) + self.bias))
41        return np.array(y_pred)
42
43 X = np.array([[1, 1], [1, -1], [-1, 1], [-1, -1]])
44 y = np.array([1, 1, 1, -1])
45
46 clf = MyAdaline(n_iterations=5)
47 clf.fit(X, y)
48
49 y_pred = clf.predict(X)
50 print("Accuracy:", accuracy_score(y, np.sign(y_pred)))
51
52 conf_matrix = confusion_matrix(y, np.sign(y_pred))
53 print("Confusion Matrix:")
54 print(conf_matrix)
```