**PRACTICAL 4**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Name:** | Harsh Shah | **Semester:** | VI | **Division:** | 6 |
| **Roll No.:** | 21BCP359 | **Date:** |  | **Batch:** | G11 |
| **Aim:** | Implement the Fixed Increment Perceptron Learning algorithm as presented in the attachment. The training set for a 2- classification problem is also attached.  Iterate the perceptron through the training set and obtain the weights. | | | | |

**Program**

#include <bits/stdc++.h>

using namespace std;

pair<int, int> input\_dimensions = {8, 2};

vector<pair<double, double>> X = {{0.32, 0.41}, {0.27, 0.54}, {0.57, 0.42}, {0.59, 0.71}, {0.78, 0.82}, {0.79, 0.95}, {1.0, 0.85}, {1.0, 1.1}};

vector<double> coefficients, results, hidden\_values, bias;

vector<double> targets = {0, 1, 0, 1, 0, 1, 0, 1};

void initialize\_coefficients()

{

    uniform\_real\_distribution<double> unif(-0.00002, 0.00002);

    default\_random\_engine re;

    for (int i = 0; i < input\_dimensions.second; i++)

    {

        coefficients.push\_back(unif(re));

    }

    bias.push\_back(unif(re));

}

void calculate\_hidden\_values()

{

    double res1, sum;

    sum = 0;

    hidden\_values.clear();

    for (int i = 0; i < input\_dimensions.first; i++)

    {

        res1 = (X[i].first) \* (coefficients[0]) + (X[i].second) \* (coefficients[1]);

        hidden\_values.push\_back((res1 + bias[0]));

    }

}

void make\_predictions()

{

    calculate\_hidden\_values();

    results.clear();

    for (int i = 0; i < input\_dimensions.first; i++)

    {

        if (hidden\_values[i] > 0)

        {

            results.push\_back(1);

        }

        else

        {

            results.push\_back(0);

        }

    }

}

double evaluate\_accuracy()

{

    double acc = 0;

    for (int i = 0; i < input\_dimensions.first; i++)

    {

        if (results[i] == targets[i])

        {

            acc++;

        }

    }

    return acc / input\_dimensions.first;

}

void adjust\_coefficients(double *learning\_rate*)

{

    double error1, error2;

    error1 = 0;

    error2 = 0;

    for (int i = 0; i < input\_dimensions.first; i++)

    {

        error1 += (targets[i] - results[i]) \* X[i].first;

        error2 += (targets[i] - results[i]) \* X[i].second;

        bias[0] += learning\_rate \* (targets[i] - results[i]);

    }

    coefficients[0] += learning\_rate \* error1;

    coefficients[1] += learning\_rate \* error2;

}

int main()

{

    double learning\_rate, accuracy;

    learning\_rate = 0.000000001;

    initialize\_coefficients();

    make\_predictions();

    accuracy = evaluate\_accuracy();

    long long ep = 1;

    while (accuracy < 1)

    {

        adjust\_coefficients(learning\_rate);

        make\_predictions();

        accuracy = evaluate\_accuracy();

*// cout << "\tepoch: " << ep << "\taccuracy: " << accuracy << "\n";*

        ep++;

    }

    cout << "Weights: " << coefficients[0] << ", " << coefficients[1] << endl;

    return 0;

}

**Output**

