UNIVERSITY OF NAIROBI

SCHOOL OF COMPUTING AND INFORMATICS

CSC324: MACHINE LEARNING

IMPLEMENTATION OF K MEANS AND PERCEPTRON ALGORITHMS

REPORT BY

ALLAN EMUSUGUT BARUA

P15/1719/2016

Table of Contents

Article I. K-MEANS ALGORITHM

Section 1.01 Introduction

K means clustering algorithm was developed by J. MacQueen (1967) and later improved by J. A. Hartigan and M. A. Wong around 1975

K-means clustering is an algorithm to group objects based on attributes OR features into K number of groups. K is positive integer.

Grouping is done by minimizing the sum of squares of distances between a given data and the corresponding cluster centroid.

The algorithm in this context has been implemented in the java programming language. Using data from medicine. The algorithm tries to determine which medicines belong to which cluster give a training set containing of four medicine each with two attributes (Weight index and pH)

Section 1.02 Pseudocode

1) Input the number of nearest neighbours as a positive integer k
2) Pick first k objects as the initial centroids
3) While convergence not reached
    1) Determine the centroid's coordinates for each cluster
    2) Determine the distance of each object to each centroid
    3) Compare the distances of each object to all the centroids
    4) Group the objects according to the minimum distance
    5) Test for convergence (no object has changed group)

Display the final groups

Section 1.03 Code

MainClass.java

```java
package MachineLearning.Clustering;
import java.util.Scanner;

public class MainClass {
    public static void main(String[] args){

        Scanner input = new Scanner(System.in);

        DataSet[] dataArray = new DataSet[4];
        dataArray[0] = new DataSet(1,1);
        dataArray[1] = new DataSet(2,1);
        dataArray[2] = new DataSet(4,3);
        dataArray[3] = new DataSet(5,4);

        System.out.println("How many clusters do you want to have");
        System.out.println("\t2 Clusters: Enter 2\n\t3 Clusters: Enter 3");
        System.out.print("Choice: ");
        int choice = input.nextInt();
        if(choice<1 || choice>4){
            System.err.println("Invalid input");
            System.exit(0);
        }

        do{
            DataSet.chooseCentroids(choice,dataArray);
            DataSet.computeDist(dataArray);
            DataSet.clusterObj(dataArray);
            DataSet.printState(dataArray);
        }
        while(!DataSet.convergenceCheck());
    }

}
```

DataSet.java

```
package MachineLearning.Clustering;
import javafx.geometry.Point2D;

public class DataSet {
    private Point2D coord;
    private int group;
    private static double[][] centroids;
    private static double[][] distances;
    private static int round = 1;
    private static boolean groupTest;

    public DataSet(int x, int y){
        coord = new Point2D(x,y);
    }

    public static void chooseCentroids(int choice, DataSet[] obj){
        groupTest = true;
        if(round==1){
            centroids = new double[choice][2];
            for(int i=0;i<choice;i++){
                centroids[i][0] = obj[i].coord.getX();
                centroids[i][1] = obj[i].coord.getY();
            }
        }
        else{
            for(int i=0;i<centroids.length;i++){
                double sumX=0;
                double sumY=0;
                double num =0;
                for(int j=0;j<obj.length;j++){

                    if(obj[j].group==i){
                        sumX += obj[j].coord.getX();
                        sumY += obj[j].coord.getY();
                        num++;
                    }
                }
                centroids[i][0] = sumX/num;
                centroids[i][1] = sumY/num;

            }
        }

    }
```

```java
    public static void computeDist(DataSet[] obj){
        distances = new double[obj.length][centroids.length];
        System.out.println("Round "+ round );
        System.out.println("--------------------------------------------------------------------------------");
        for(int i=0;i<obj.length;i++){
            for(int j=0;j<centroids.length;j++){
                distances[i][j] = obj[i].coord.distance(centroids[j][0], centroids[j][1]);
                System.out.printf("DataObject %d Distance with centroid %d: %4.2f\n",i+1,j+1,distances[i][j]);
            }
        }
    }

    public static void clusterObj(DataSet[] obj){
        for(int i=0;i<obj.length;i++){
            double min=distances[i][0];
            int m = 0;
            for(int j=1;j<distances[i].length;j++){
                if(min >= distances[i][j]){
                    min = distances[i][j];
                    m = j;
                }
            }
            //assign initial group
            if(round==1){
                obj[i].group = m;
                groupTest = false;
            }
            //check for group change first
            else{
                //compare first
                if(obj[i].group != m){
                    groupTest = false;
                    obj[i].group = m;
                }

            }
        }
        round++;
    }

    public static boolean convergenceCheck(){
        return groupTest;
    }

    public static void printState(DataSet[] obj){
        System.out.println("\nDataObject\t Coordinates\t Cluster\t Centroid");
        System.out.println("___________________________________________________________________");
        for(int i =0;i<obj.length;i++){
            System.out.printf("  %d\t\t(%4.2f,%4.2f)\t  %d\t\t(%4.2f,%4.2f)\n",
                        i+1,obj[i].coord.getX(),obj[i].coord.getY(),obj[i].group+1,
                        centroids[obj[i].group][0],centroids[obj[i].group][1]);
        }
        System.out.println("\n\n");
    }

}
```

Section 1.04 Results

Output - LearnJava (run) × DataSet.java × MainClass.java ×

```
run:
How many clusters do you want to have
        2 Clusters: Enter 2
        3 Clusters: Enter 3
Choice: 2
Round 1
--------------------------------------------------------------------------
DataObject 1 Distance with centroid 1: 0.00
DataObject 1 Distance with centroid 2: 1.00
DataObject 2 Distance with centroid 1: 1.00
DataObject 2 Distance with centroid 2: 0.00
DataObject 3 Distance with centroid 1: 3.61
DataObject 3 Distance with centroid 2: 2.83
DataObject 4 Distance with centroid 1: 5.00
DataObject 4 Distance with centroid 2: 4.24

DataObject       Coordinates     Cluster         Centroid
_________________________________________________________________
  1              (1.00,1.00)       1             (1.00,1.00)
  2              (2.00,1.00)       2             (2.00,1.00)
  3              (4.00,3.00)       2             (2.00,1.00)
  4              (5.00,4.00)       2             (2.00,1.00)



Round 2
--------------------------------------------------------------------------
DataObject 1 Distance with centroid 1: 0.00
DataObject 1 Distance with centroid 2: 3.14
DataObject 2 Distance with centroid 1: 1.00
DataObject 2 Distance with centroid 2: 2.36
DataObject 3 Distance with centroid 1: 3.61
DataObject 3 Distance with centroid 2: 0.47
DataObject 4 Distance with centroid 1: 5.00
DataObject 4 Distance with centroid 2: 1.89

DataObject       Coordinates     Cluster         Centroid
_________________________________________________________________
  1              (1.00,1.00)       1             (1.00,1.00)
  2              (2.00,1.00)       1             (1.00,1.00)
  3              (4.00,3.00)       2             (3.67,2.67)
  4              (5.00,4.00)       2             (3.67,2.67)
```

```
Round 3
--------------------------------------------------------------------------
DataObject 1 Distance with centroid 1: 0.50
DataObject 1 Distance with centroid 2: 4.30
DataObject 2 Distance with centroid 1: 0.50
DataObject 2 Distance with centroid 2: 3.54
DataObject 3 Distance with centroid 1: 3.20
DataObject 3 Distance with centroid 2: 0.71
DataObject 4 Distance with centroid 1: 4.61
DataObject 4 Distance with centroid 2: 0.71

DataObject       Coordinates     Cluster         Centroid
_________________________________________________________________
  1              (1.00,1.00)       1             (1.50,1.00)
  2              (2.00,1.00)       1             (1.50,1.00)
  3              (4.00,3.00)       2             (4.50,3.50)
  4              (5.00,4.00)       2             (4.50,3.50)



BUILD SUCCESSFUL (total time: 3 seconds)
```

Article II. PERCEPTRON

Section 2.01 Introduction

Perceptron is a classification algorithm that makes its predictions based on a linear predictor function combining a set of weights with the feature vector. The algorithm allows for online learning, in that it processes elements in the training set one at a time

The perceptron model is motivated by the biological neuron. The perceptron learning rule devices a procedure for modifying the weights and biases of a network.

My implementation of a perceptron takes a perceptron with a single hidden layer with unspecified number of neurons such that the user can decide the number of neurons in the hidden layer.

The training set used is a NAND gate.

The algorithm is implemented in the Java programming language.

Section 2.02 Pseudocode

1) Prompt the user for the number of nodes in the hidden layer
2) Set the number of nodes in the input layer equal to the number of inputs in one training instance
3) Set the number of node in the output layer to one
4) Set the learning rate to a value between 0 and 1
5) Initialize the weights of both the hidden and output layer with random values between 0 and 1

6) For each training example in an epoch
    1) Use the training example as an input to the input layer{Ii} of the perceptron
    2) Calculate the output of the input { Oi }
    3) Calculate the input of the hidden layer { Ih }
    4) Calculate the output of the hidden layer { Oh }
    5) Calculate input of the output layer { Io }
    6) Calculate The output of the perceptron {Oo}
    7) Calculate the error by subtracting the system output from the target output
       Error = Target - Oo

    8) Using the error. Calculate the change in weights of the system using ***nex***. Change in weight will include two sets of weights: Hidden layer weights and output layer weights
        a. The hidden layer weights are updated using the output of the input layer as X, n is learning rate and e the error
           Change (W) = Oi * n * Error

        b. The hidden layer weights are updated using the output of the hidden layer as X, n is learning rate and e the error
           Change (W) = Oh * n * Error

Repeat until convergence

Section 2.03 Code

Initializing weights

```java
//function to initialize the weights of the network
public void initializeWeights(int inputN, int hiddenN, int outputN){

    Random obj1 = new Random(new Date().getTime());

    //initialize weights for the hidden layer neurones
    H_weights = new double[inputN][hiddenN];
    for(int i=0;i<H_weights.length;i++){
        for(int j=0;j<H_weights[i].length;j++){
            H_weights[i][j] = obj1.nextDouble();
        }
    }

    //initialize weights for the output layer neurones
    O_weights = new double[hiddenN][outputN];
    for(int i=0;i<O_weights.length;i++){
        for(int j=0;j<O_weights[i].length;j++){
            O_weights[i][j] = obj1.nextDouble();
        }
    }

    //printing the weights
    System.out.println("\nInitialized weights of the hidden layer");
    vectorPrinter(H_weights);
    System.out.println("\n\nInitialized weights of the output layer");
    vectorPrinter(O_weights);
}
```

Changing weights

```java
/*--------------------------------------------Error Calculation--------------------------------------------------*/
//using nex;
e = target - (int)Oo[0][0];

//change in weights to the output layer
//nex
change_O = new double[O_weights.length][O_weights[0].length];
for(int i=0;i<O_weights.length;i++){
    for(int j=0;j<O_weights[i].length;j++){
        change_O[i][j] = Oh[i][j] * n * e;
    }
}


//change in weights of the hidden layer
change_H = new double[H_weights.length][H_weights[0].length];
for(int i=0;i<H_weights.length;i++){
    for(int j=0;j<H_weights[i].length;j++){
        change_H[i][j] = Oi[i][0] * n * e;
    }
}

//print the colum headings to the table
if(iter==0)
    printLabels();

//check if the epoch is complete
if(iter==data.length-1)
    epoch = true;
else
    iter++;

//print the values in a table
display(iter);



//new weights for O_weights
for(int i=0;i<O_weights.length;i++){
    O_weights[i][0] = O_weights[i][0] + change_O[i][0];
}

//new weights for H_weights
for(int i=0;i<change_H.length;i++){
    for(int j=0; j<change_H[i].length;j++){
        H_weights[i][j] = change_H[i][j] + H_weights[i][j];
    }
}
```

Section 2.04 Results

Please enter the following parameters:
Number of Nodes in the hidden layer: 2

Initialized weights of the hidden layer

0.4810	0.8261
0.8157	0.3804

Initialized weights of the output layer

0.5754
0.9520

EPOCH: 1

INPUT --> HIDDEN LAYER													HIDDEN --> OUTPUT LAYER					
inputs		weights							Change in weights				inputs		weights		Change in weights	
X1	X2	W1h1	W1h2	W2h1	W2h2	Target	Output	Error	CW1h1	CW1h2	CW2h1	CW2h2	X1	X2	W1h1	W2h1	CW1h1	CW2h1
0	0	0.48	0.83	0.82	0.38	1	0	1	0.00	0.00	0.00	0.00	0	0	0.58	0.95	0.00	0.00
0	1	0.48	0.83	0.82	0.38	1	1	0	0.00	0.00	0.00	0.00	1	1	0.58	0.95	0.00	0.00
1	0	0.48	0.83	0.82	0.38	1	1	0	0.00	0.00	0.00	0.00	1	1	0.58	0.95	0.00	0.00
1	1	0.48	0.83	0.82	0.38	0	1	-1	-0.32	-0.32	-0.32	-0.32	1	1	0.58	0.95	-0.32	-0.32

EPOCH: 2

INPUT --> HIDDEN LAYER													HIDDEN --> OUTPUT LAYER					
inputs		weights							Change in weights				inputs		weights		Change in weights	
X1	X2	W1h1	W1h2	W2h1	W2h2	Target	Output	Error	CW1h1	CW1h2	CW2h1	CW2h2	X1	X2	W1h1	W2h1	CW1h1	CW2h1
0	0	0.16	0.51	0.50	0.06	1	0	1	0.00	0.00	0.00	0.00	0	0	0.26	0.63	0.00	0.00
0	1	0.16	0.51	0.50	0.06	1	1	0	0.00	0.00	0.00	0.00	1	1	0.26	0.63	0.00	0.00
1	0	0.16	0.51	0.50	0.06	1	1	0	0.00	0.00	0.00	0.00	1	1	0.26	0.63	0.00	0.00
1	1	0.16	0.51	0.50	0.06	0	1	-1	-0.32	-0.32	-0.32	-0.32	1	1	0.26	0.63	-0.32	-0.32