KNearestNeighbor

KNN:

Knn is used in solving classification problems. With a dataset having various attributes and a label, knn help to find the label of an unclassified data item. Knn does this with distance calculation.

3-KNN: Example(1)

Customer	Age	Income	No. credit cards	Class	Distance from John
George	35	35K	3	No	sqrt [(35-37)^2+(35-50)^2 +(3-2)^2]=15.16
Rachel	22	50K	2	Yes	sqrt [(22-37)^2+(50-50)^2 +(2-2)^2]=15
Steve	63	200K	1	No	sqrt [(63-37)^2+(200-50)^2 +(1-2)^2]=152.23
Tom	59	170K	1	No	sqrt [(59-37)^2+(170-50)^2 +(1-2)^2]=122
Anne	25	40K	4	Yes	sqrt [(25-37)^2+(40-50)^2 +(4-2)^2]=15.74
John	37	50K	2	YES	

13

Sequential KNN:

For the unclassified data item, the Euclidean distance between this item and the other items in the dataset is calculated. And then the label of the K data item is taken into consideration and the maximum count label is taken as the label of the unclassified data item.

Implementation of Sequential KNN:

1. Define a function to calculate the distance between two points
2. Use the distance function to get the distance between a test point and all known data points
3. Sort distance measurements to find the points closest to the test point (find the k nearest neighbors)
4. Use majority class labels of those closest points to predict the label of the test point
5. Repeat steps 1 through 4 until all test data points are classified

Algorithm of Sequential KNN:

Procedure Sequential_KNN(testData,trainingData):

```
Step 1: for i in testData do
       distance[trainingData.length,2]
       for j in trainingData do
            distance[j][0]=sqrt(square(trainingData .class)- square(trainingData .class))
            distance[j][1]=trainingData.class
        endfor
        sort(Distance)
        class0=0,class1=0, prediction=0;
        for i=0 to k do:
            if(distance[i][1]==0)
                 count0++
            else
                  count1++
         result= if(count0>count1)? count0:count1
         if (testData.class==result)
               prediction++
endfor
Step 2:  accuracy=(prediction/testData.length)*100
```

Analysis of Sequential KNN:

Consider the length of test data is m, training data is n
Timecomplexity=(m*nlogn)+(m*n)+(m*k)
T(n)=O(mnlogn)
Computing the distance with all the n data item will take O(mnlogn)time complexity.
Moving to Parallel KNN will be effective only if cost C(n)<= O(mnlogn)

Implementation of Parallel KNN:

Consider there are n training dataset,m testing data,N processor

1. Divide the testdata set into m/N and give a set of test data to the respective processor
2. Define a function to calculate the distance between two points
3. Use the distance function to get the distance between a test point and all known data points
4, Sort distance measurements to find the points closest to the test point (find the k nearest neighbors)
5. Use majority class labels of those closest points to predict the label of the test point.
6. Repeat steps 1 through 4 until all test data points are classified
7. Calculate the accuracy

Algorithm for Parallel KNN:
procedure PARALLEL KNN (Data):

```
Step 1: split train(80%) and test data(20%).
            train[Data.length*0.8]=training data
            train[Data.length*0.2]=training data
Step 2:  for i=0 to N do in parallel
                        class0=0,class1=0, prediction=0;
                        for j=i(m/N) to (i+1)(m/N)-1  do
                              distance[trainingData.length,2]
                              for j in trainingData do
                                    distance[j][0]=sqrt(square(trainingData .class)- square(trainingData .class))
                                    distance[j][1]=trainingData.class
                              endfor
                              sort(Distance)
                              for i=0 to k do:
                                    if(distance[i][1]==0)
                                                count0++
                                    else
                                                count1++
                                    endif
                                  result= if(count0>count1)? count0:count1
                                  if (testData.class==result)
                                      prediction++
                                  end if
                              endfor
                        endfor
Step 3:  accuracy=(prediction/testData.length)*100
```

Analysis of Parallel KNN:

Time complexity: O(mnlogn)/N
Total time complexity= O(mnlogn)/N
Cost=C(n)=no. of processors*T(n)
= O(mnlogn)/N O(N)
= O(mnlogn)/N (Thus, cost optimal)

Output:

Sequential KNN:
1.Test case1   K=3

```
Sequential KNN Algorithm
No of neighbors : 3

Acurracy : 87.98%
Exec Time: 0.7648058810009388
```

2.Test case2 K=5

```
Sequential KNN Algorithm
No of neighbors : 5

Acurracy : 89.94%
Exec Time: 0.7515149110004131
```

3.Test case3   K=7

```
Sequential KNN Algorithm
No of neighbors : 7

Acurracy : 89.62%
Exec Time: 0.8213216010008182
```

Parallel KNN:

1. Test case1   K=3

```
Parallel KNN Algorithm
No of neighbors : 3

Acurracy : 87.31%
Exec Time:  0.5851900880006724
```

2.Test case2   K=5

```
Parallel KNN Algorithm
No of neighbors : 5

Acurracy : 92.74%
Exec Time:  0.6406828590006626
```

3.Test case3   K=7

```
Parallel KNN Algorithm
No of neighbors : 7

Acurracy : 94.35%
Exec Time:  0.5917390300000989
```