A New Distance Based Method to Detect Outliers- ***Centroid Factor Method***

Submitted By:-

Prayank Mathur
2012JE0719
IV Semester
Dept. Of Computer Science & Engg.

Acknowledgements

It is my great privilege to express my deepest and most sincere thanks to **Dr. Rajendra Pamula** , for his invaluable suggestions and guidance during the course of this project,without which the project would not have been successful.

I am also grateful to my our friends with whom we were able to discuss my doubts.

ABSTRACT

Detecting outliers which are grossly different from or inconsistent with the remaining dataset is a major challenge in real-world KDD applications. Existing outlier detection methods are ineffective on scattered real-world datasets due to implicit data patterns and parameter setting issues. We define a novel method Centroid Factor to measure the outlier-ness of objects in scattered datasets which addresses these issues.The method inculcates dividing the dataset into smaller blocks and then outliers from each of these is calculated by using the above factor. Centroid factor as the name suggests uses the centroid of the dataset to determine the degree to which the object deviates from its neighbourhood. In order to facilitate parameter settings in real-world applications, we employ a top-n technique in our outlier detection approach, where only the objects with the highest Centroid factor values are regarded as outliers. Compared to conventional approaches (such as top-n KNN and top-n LDOF), our method top-n Centroid Factor is more effective at detecting outliers in scattered data. It is also easier to set parameters, since its performance is relatively stable over a large range of parameter values, as illustrated by experimental results on both real-world and synthetic datasets.

INTRODUCTION

Of all the data mining techniques that are in vogue, outlier detection comes closest to the metaphor of mining for nuggets of information in real-world data. It is concerned with discovering the exceptional behavior of certain objects. Outlier detection techniques have widely been applied in medicine (e.g. adverse reactions analysis),finance (e.g. financial fraud detection), security (e.g. counter-terrorism), information security (e.g. intrusions detection) and so on. In the recent decades, many outlier detection approaches have been proposed, which can be broadly classified into several categories:

Distribution-based
Depth-based
Distance-based
Cluster-based
Density-based

However, these methods are often unsuitable in real-world applications due to a number of reasons. Firstly, real-world data usually have a scattered distribution, where objects are loosely distributed in the domain feature space. That is, from a ‘local’ point of view, these objects cannot represent explicit patterns (e.g. clusters) to indicate normal data ‘behavior’. However, from a ‘global’ point of view, scattered objects constitute several mini-clusters, which represent the pattern of a subset of objects. Only the objects which do not belong to any other object groups are genuine outliers. Unfortunately, existing outlier definitions depend on the assumption that most objects are crowded in a few main clusters. They are incapable of dealing with scattered datasets, because mini-clusters in the dataset evoke a high false-detection rate (or low precision).
Secondly, it is difficult in current outlier detection approaches to set accurate parameters for real-world datasets . Most outlier algorithms must be tuned through trial-and-error . This is impractical, because real-world data usually do not contain labels for anomalous objects. In addition, it is hard to evaluate detection performance without the confirmation of domain experts. Therefore, the detection result will be uncontrollable if parameters are not properly chosen.
To alleviate the parameter setting problem, researchers proposed top-n style outlier detection methods. Instead of a binary outlier indicator, top-n outlier methods provide a ranked list of objects to represent the degree of ‘outlier-ness’ for each object. The users (domain experts) can re-examine the selected top-n (where n is typically far smaller than the cardinality of dataset) anomalous objects to locate

real outliers. Since this detection procedure can provide a good interaction between data mining experts and users, top-n outlier detection methods become popular in real-world applications.

The centroid factor method consists of finding the top-n-centroid factor of all the elements in the dataset and therefore is advantageous than other distance based outlier detection approaches.

A density-based outlier detection technique, Local Outlier Factor (LOF), is a outlier factor which is assigned to each object w.r.t its surrounding neighbourhood. The outlier factor depends on how the data object is closely packed in its locally reachable neighbourhood . Since LOF uses a threshold to differentiate outliers from normal objects, the same problem of parameter setting arises. A lower outlier-ness threshold will produce high false-detection rate, while a high threshold value will result in missing genuine outliers. In recent real-world applications, researchers have found it more reliable to use LOF in a top-n manner, i.e. only objects with the highest LOF values will be considered outliers. Hereafter, we call it top-n LOF.

The existing top-n style outlier detection techniques alleviate the difficulty of parameter setting, the detection precision of these methods is low on scattered data. In Section 2, we will discuss further problems of top-n KNN and top-n LOF.

In this paper we propose a new outlier detection definition, named Centroid Factor which is sensitive to outliers in scattered datasets. Centroid factor uses the ratio of distances of an object from the common centroid to the centroid obtained by excluding that point to deduce how much objects deviate from their scattered neighbourhood. The higher the violation degree an object has, the more likely the object is an outlier. In addition, we theoretically analyse the properties of Centroid Factor, including its lower bound and false-detection probability, and provide guidelines for choosing a suitable neighbourhood size. In order to simplify parameter setting in real-world applications, the top-n technique is employed in our approach. To validate Centroid Factor Method, we perform various experiments on both synthetic and real-world datasets, and compare our outlier detection performance with top-n KNN and top-n LOF. The experimental results illustrate that our proposed top-n Centroid Factor Method represents a significant improvement on outlier detection capability for scattered datasets.

FORMAL DEFINITIONS OF CENTROID FACTOR METHOD

Definition 1 (Including centroid distance of x) *Let **N** be the set of all the elements that are present in the neighborhood of x, i.e. , they all are present in the block, then this distance, denoted by **dx** is the distance between the point x and the centroid $\boldsymbol{C_i}$ of all the elements present in the block.*

$$dx = dist(C_i , x)$$

Definition 2(Excluding centroid distance of x)) *Let **N** be the set of all the elements that are present in the neighborhood of x, i.e. , they all are present in the block, then this distance, denoted by **Dx** is the distance between the point x and the centroid $\boldsymbol{C_{ix}}$ which is obtained by excluding the element **x** from the block.*

$$Dx = dist(C_{ix} , x)$$

Where dist() function calculates the distance between the two input coordinates.

Definition 3(Centroid Factor of x) *Centroid distance of x is defined as the ratio dx/Dx.*

$$CF(x) = dx/Dx$$

CENTROID FACTOR ALGORITHM AND IT'S COMPLEXITY

The centroid factor algorithm is quite simple. However the choice of **n** **(**number of elements that are present in a particular block**)** plays a cruicial role in deciding the efficiency of the algorithm.

How to choose k: It is beneficial to use a large neighbourhood size **n**. However, too large **n** will lead to a global method with the same problems as top-n KNN outlier. For the best use of our algorithm, the lower bound of potentially suitable **n** is given as follows: If the effective dimension of the manifold on which **N** lies is m, then at least m points are needed to 'surround' another object. That is to say **n** > m is needed.However, when k increases to the dimension of the dataset, the detection performance of our method rises, and remains stable for a wide range of **n** values. Therefore, the parameter **n** in CFM is easier to choose than in other outlier detection approaches.
Algorithm Complexity : Let the total number of elements present in the complete dataset be |N| and the number of dimensions of each element be k.
Then the complexity of the algorithm is given as **O(|N|*k).**
The above complexity is easily obtained by looking at the various steps.

ALGORITHM FOR CFM

1. Enter the number of elements **n** that are to be present in a particular block.
2. For each block calculate the centroid of the block $\mathbf{C_i}$.
3. For every element calculate the centroid by excluding that element $\mathbf{C_{ix}}$.
4. Now calculate the distances Dx and dx.
5. Obtain the centroid factor = dx/Dx for every element.
6. Calculate this ratio's mean value for the block.
7. All the elements having centroid factor greater than this mean value are considered outliers.
8. Now after results for all the blocks are obtained collect the outlier elements from all the blocks and consdering them as one block obtain the centroid factor for all of them.
9. Again calculate the mean of those centroid factor and the elements having centroid factor greater than the mean value are outputted in top-n fashion.

A C++ Program to implement the above algorithm

```
#include <bits/stdc++.h>
using namespace std;
double abs(double n)
{
    if(n<0)
        n=-n;
    return n;
}
double x[400][30],cen[30]={0.0},cenx[400][30]={0.0},temp,dist[400],dx[400],Dx[400],sum;
int i,j,index[400],k,n,t=0,cnt=0,l,divide;
int main()
{
    string a;
    ifstream fin("data.txt");
    ofstream fout("Centroid_distance.txt");
    if(fin.is_open())
    {
        i=0;
        while(getline(fin,a))
        {
            istringstream iss(a);
            j=0;
            while(iss>>temp)
            {
                x[i][j++]=temp;
            }
            i++;
        }
    }
    else
    {
        cout<<"Cant open file";
        return 0;
    }

    cout<<"Enter the number of elements which you want in a particular block=";
    cin>>n;
    for(j=0;j<=(i/n);j++)
    {
        t=n*j;sum=0.0;
        if(i-t>=n)
            divide=n;
        else
            divide=i-t;
```

```
		for(k=0;k<30;k++)
		{
			cen[k]=0;
			for(l=t;l<(t+n)&&l<i;l++)
			{
				cen[k]+=x[l][k];
			}
		}
		for(k=t;k<(n+t)&&k<i;k++)
		{
			for(l=0;l<30;l++)
			{
				cenx[k][l]=cen[l]-x[k][l];
			}
			cenx[k][l]/=((double)divide-1.0);
		}
		for(k=t;k<(n+t)&&k<i;k++)
		{
			cen[k]/=(double)divide;
			dx[k]=0.0;
			Dx[k]=0.0;
			for(l=0;l<30;l++)
			{
				dx[k]+=(cen[l]-x[k][l])*(cen[l]-x[k][l]);
				Dx[k]+=(x[k][l]-cenx[k][l])*(x[k][l]-cenx[k][l]);
			}
			dx[k]=sqrt(dx[k]);
			Dx[k]=sqrt(Dx[k]);
			dist[k]=dx[k]/Dx[k];
			sum+=dist[k];
		}
		sum/=(double)divide;
		for(k=t;k<(n+t) && k<i;k++)
		{
			if(dist[k]>sum)
				index[cnt]=k,cnt++;
		}
	}
	for(k=0;k<30;k++)
	{
		cen[k]=0;
		for(j=0;j<cnt;j++)
		{
			cen[k]+=x[index[j]][k];
		}
	}
	for(j=0;j<cnt;j++)
```

```
    {
            for(k=0;k<30;k++)
            {
                    cenx[j][k]=cen[k]-x[index[j]][k];
            }
            cenx[j][k]/=(double)cnt-1.0;
    }
    for(k=0;k<30;k++)
            cen[k]/=(double)cnt;

    sum=0.0;
    for(i=0;i<cnt;i++)
    {
            dx[i]=0.0;
            Dx[i]=0.0;
            for(j=0;j<30;j++)
            {
                    dx[i]+=(cen[j]-x[index[i]][j])*(cen[j]-x[index[i]][j]);
                    Dx[i]+=(x[index[i]][j]-cenx[i][j])*(x[index[i]][j]-cenx[i][j]);
            }
            dx[i]=sqrt(dx[i]);
            Dx[i]=sqrt(Dx[i]);
            dist[i]=dx[i]/Dx[i];

    }
    for(i=0;i<cnt;i++)
            sum+=dist[i];
    sum/=(double)cnt;
    fout<<"Coordinate number"<<endl;
    for(i=0;i<cnt;i++)
    {
            if(dist[i]>sum)
                    fout<<index[i]<<"                    "<<dist[i]<<endl;
    }
    return 0;
}
```