MINING OUTLIERS IN TRADITIONAL AND WEB DATA SOURCES

Presented by

Name : Malik Agyemang

Index No.: 10048251

Project Group :A

Professor: Dr. C.I. Ezeife

Course : Data Warehousing & Mining

Abstract

For many KDD applications finding rare events can be more interesting and useful than finding common cases. The identification of outliers in a given dataset can lead to the discovery of unexpected knowledge in areas such as electronic commerce and credit card fraud.

In this survey, a critical look is taken at the various approaches proposed for mining outliers in a large dataset.

Acknowledgement

I am very much grateful to Dr. C. I. Ezeife for her support .

19/20

Table of Contents

1.0 INTRODUCTION

The amount of data collected and stored in databases increases tremendously everyday and there is the need for efficient and effective analysis methods to make use of such information contained implicitly in the data. The process of finding such useful information from data is a very complex task. Data mining and Knowledge Discovery (KDD) in databases has been described as the non-trivial process of identifying valid, novel, potentially useful and ultimately understandable patterns in the data [7]. Data mining is the analysis of data using software techniques to find interesting pattern in the data. Mining software are used to extract 'Gold' from large amounts of data which have not been previously identified by anybody. 'Gold' is thus a useful pattern or information that is conspicuously hidden in the database that need to be mined using specific algorithms and techniques.

It is generally recognised that Knowledge Discovery task can be classified into four main categories: **dependency detection**, **class identification**, **class description** and **exception /outlier detection**. The first three categories of the task correspond to patterns that apply to many objects and to large percentage of objects in the data set. In contrast the forth category (exception /outlier detection) focuses on very small minority of data objects, so small that these objects are often discarded as noise. But "one persons noise is another persons signal". For many applications, identifying exceptions can often lead to the discovery of really interesting and unexpected knowledge as far as data mining is concerned. Some existing algorithms in machine learning and data mining have considered outliers, but only to the extent of tolerating them in whatever the algorithms are supposed to do [6,7,14,15].

This survey mainly deals with finding such patterns that apply to a smaller percentage of the data deviating from the majority of data objects called outliers. In most application in data mining, finding such rare events are likely to be more interesting than finding common ones. Sample application include the detection of credit card fraud and monitoring of criminal and suspicious activities [18]

What is an outlier? "An outlier is an observation that deviates so much from other observations as to arouse suspicion that it was generated by a different mechanism" [8]. The concept of outliers though very complex is gradually gaining attention but the fact still remain that there are variety of definitions and approaches for mining them. Some of the algorithms proposed can be found in [4, 9,11, 12, 13, 16, 18, 21].

2.0 Mining Outliers

Mining outliers is the entire process of finding interestingly rare characteristics existing in large data set. This process is however not very simple because of the diversity of approaches to outlier mining. The mining approaches can be grouped into three main categories.

i. **Distribution-based** - This approach relies on fitting the data with a standard statistical model and outliers are declared based on how the data points appear in relation to the standard statistical model (mostly found in statistical arena)

ii. **Depth-based** - Data objects are organised in some K-D space. Depths are assigned to the data objects based on some algorithm of depth. Outliers are those points that tend to have shallow depths.

iii. **Distance-based** - Distances of points are computed from each other and those found to have higher distances are declared outliers.

2.1 Distribution-based approach

The distribution-based methods uses standard statistical distributions (eg Normal, Poisson, Student t etc) to fit data points and points are declared outliers depending on how they appear in relation to other points in the data set. A wide variety (over 100) of test exist for distribution based outliers called discordancy test for different scenarios:

i. whether the distribution of the data is known

ii. whether or not the distribution parameters are known (mean and variance)

iii. the number of expected outliers (single, pair, multiple).

iv. the type of expected outliers (lower or upper outlier) in an ordered sample [4,8].

For instance, for a normal distribution with known mean and variance, there will be a separate discordancy test for single upper outlier, upper outlier pair, k upper outliers, single lower outlier, lower outlier pair, k lower outliers etc.

In most cases visual plots such as the **box plot** or **scatter diagrams** are used to determine which of the data points are potential outliers before the appropriate test is conducted.

2.1.1 The box plot

This is a very simple visual plot that gives an idea of the data points likely to be outliers. The box is constructed using lower (q1), median (q2) and the upper (q3) quartiles. It is based on the principle that all data points that are outliers will lie outside (q1-(1.5*iqr) and q3+(1.5*iqr)) in general and (q1-(3*iqr) and q3+(3*iqr)) for problematic outliers in particular.

Given any point X in a data set, the following holds:

1. **q1-(1.5*iqr) ≤ X ≤ q3+(1.5*iqr)** , X is not an outlier
2. **X < q1-(1.5*iqr), X > q3+(1.5*iqr)** , X is an outlier
3. **X < q1-(3*iqr), X > q3+(3*iqr)** , X is a problematic outlier

Where

Iqr (interquartile range) = **q3-q1.**

q1 - which is the observation that lies exactly at the $1/4^{th}$ position of the entire data set.

q2- which is the observation that lies exactly at the $3/4^{th}$ position of the entire data set.

Example 1:

Consider the marks scored by 25 students in a examination given below:

{36,29,45,55,60,59,61,64,65,70,70,71,72,72,72,72,73,73,75,76,76,78,79,98,99}

q1= 61 , q3 = 75, iqr = 75-61 = 14 , lower = q1- (1.5*iqr)=40,

upper = q31+ (1.5*iqr)= 96 . The box plot is shown in Figure 1(overleaf).

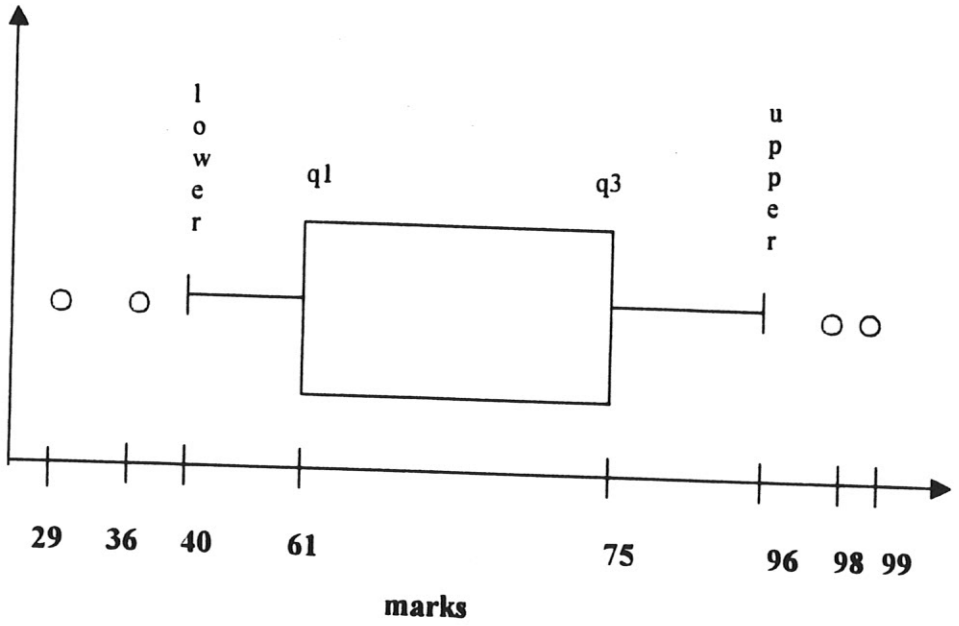

Figure 1: A box plot of the marks scored

The points {36,29,98,99} have to be tested using the appropriate discordancy test to verify if they are really outliers as shown by the box plot. It should be noted however, that conclusion is not drawn directly by just looking at the box plot.

2.1.2 The z-score test

The basic assumption here is that the underlying data is normally distributed. The test is usually conducted for data points that appear to be outliers from either the box plot or the scatter diagram. The test statistic is given below:

$$Z_i = (X_i - \overline{X})/s \qquad s = \sqrt{\frac{\sum (X_i - \overline{X})^2}{n-1}}$$

The test heuristics states that an observation with a z-score > 3 is labelled an outlier.

2.1.2.1 The modified z-score (Zmi)

The problem with the original z-score method is that the sample mean and variance are both affected by the value of the outlier, making it unreliable for labelling outliers.

A reliable method is obtained by modifying the z-score using a resistant estimator MAD.

The median of the absolute deviation about the mean (MAD) is computed and substituted for the standard deviation

$$MAD = Median\{|X_i - \bar{X}|\}$$

Hence the modified test Statistic: $$Z_{m_i} = (|X_i - \bar{X}|)/MAD$$

Any observation with a Zmi > 3.5 is declared as an outlier.

Results of our sample data using the modified Z-score

Xi	| Xi - mean |
70	2
70	2
65	3
71	3
64	4
72	4
72	4
72	4
72	4
73	5
73	5
61	7
75	7
75	7
60	8
76	8
59	9
78	10
79	11
55	13
45	23
98	30
99	31
36	32
29	39

```
Total = 1700 , Mean = 68,  MAD= 7

The Score to be tested are { 29,36,98,99)

For Xi =99

Zmi= |99-68| / 7
     = 4.42 > 3.5  ( 99 is an outlier)

For Xi =98
Zmi= |98-68| / 7
     = 4.25 > 3.5  ( 98 is an outlier)

For Xi =29
Zmi= |39-68| / 7
     = 5.57 > 3.5  ( 29 is an outlier)

For Xi =36
Zmi= |36-68| / 7
     = 4.57 > 3.5  ( 36 is an outlier)
Hence we conclude that all those four point
are outlies
```

Figure 2 : Results obtained using modified z-score

2.1.3 Outliers in multivariate data

They can be identified using visual methods such as the scatter diagram (2D-graph) or Spin-plot (3D-graph) and then tested using standard models (eg. regression, Mahalanobis). However for a truly multidimensional data with more than 3-dimensions, the above methods are not practicable. Asimov [1] proposed the grand tour method for identifying all multidimensional outliers irrespective of the number of dimensions.

2.1.3.1 The grand tour method

The grand tour algorithm for identifying outliers in a multidimensional data is based on geometrically rotating the space containing the data points. Data objects that are not outliers will fall within an ellipse always whilst those objects likely to be outliers will fall outside the ellipse at outstanding positions. The data containing the outliers are first standardised to obtain centered and reasonably scattered projections, the algorithm then proceeds in the following steps [2].

1. Generate the rotation plane <UOV> passing through the center O and the two randomly generated points U,V located on the unit hypersphere $\in R^p$, the p dimensional feature space.
2. Using U,O,V calculate the rotation matrix **A**.
3. Using the matrix **A** rotate the data matrix **X** obtaining rotated coordinates of the data points $\in R^p$(rotation is equivalent to a transformation).
4. Project the rotated points onto the plane $< \bar{x}1, \bar{x}2 >$ spanned by the horizontal ($\bar{x}1$) and vertical ($\bar{x}2$) axes of the screen.
5. Draw in the plane $< \bar{x}1, \bar{x}2 >$ a concentration ellipse calculated from coordinates $\{ \bar{x}i1, \bar{x}i2 \}$ of the visible (in the projection plane) point projections $\bar{P}i$, (i=1....n) The ellipse may be drawn as a robustified one.
6. Notify, which are found outside the concentration ellipse.

The steps 1-6 should be carried out continuously several hundred times. The points found outside the concentration ellipse are suspected to be atypical (outliers). The fact of being noticed beyond the border of the concentration ellipse is recorded in a neighbouring count plot. These suspected outliers are then verified using regression analysis.

2.1.3.2 Regression Analysis

In a multidimensional data analysis, using regression to identify outliers from scratch is a very complex issue when the dimensions are more than two. But testing the outliers once they are identified is an interesting task. Bartkowiak [3] proceeds in three steps for the analysis of the residual to confirm the outliers.

1. Regression analysis is performed using the entire data. The regression model is:

   $y= b_0 + b_1x_1 + b_2x_2 + \ldots\ldots\ldots + b_nx_n + e$

   Where y is the dependent variable, b_0 , b_1, b_2,——,b_n parameter to be estimated , x_1, x_2.....x_n the independent variables, e is the residual

2. The same regression model should be used but in estimating the parameters b_0 , b_1, b_2,——,b_n, the outliers should be drop from the actual data.

3. In the third step, artificial variables are introduced to account for the special effect of the outliers. That is : $y= b_0 + b_1x_1 + b_2x_2 + \ldots\ldots\ldots + b_nx_n + b_\gamma\gamma + e$

   where γ is an a artificial variable , $\gamma_i = 1$, for $i \in$ outliers, $\gamma_i = 0$, for $i \notin$ outliers,

The artificial variable is introduced for each outlier present in the data.

In all the three different cases, the regression parameters (squared multiple correlation coefficient (R^2), residual standard deviation (δ)) are noted and compared with standard values read from the F-distribution. Conclusion is drawn depending on how the computed values agree or disagree with the standard values to confirm or disprove the outliers.

Drawbacks:

The distribution-based approach has two major drawbacks:

1. Almost all the discordancy test available are for univariate variables and the few other test available for multivariate data are very complex to understand and use.
2. The underlying distribution of the data should necessarily be know before a test can be applied.

2.2 Depth-based approach

Depth-based approach for mining outliers was developed to solve the problems of distribution fitting and restriction to univariate distribution in the distribution-based approach. This approach is based on some definition of depth, data objects are organised in layers in a data space with the expectation that shallow layers are more likely to contain outlying data objects than are deep layers. A key property of depth–based approach is that the location depth is scaling –invariant.

A robust notion of depth called depth contours was introduced by Tukey [19,20] with the assumption that, a point p in a space is of depth k if k is the minimum number of data points that have to be removed to expose p. Minimum is defined across all half planes passing through p. The k-th depth contour makes boundary between all points with depth k and all those with depth k+1. An algorithm for computing 2-D depth contour was developed by Ruts et al [17] called ISODEPTH. ISODEPTH has been developed into a program using Fortran which accepts 2-dimensional input data, a list of x and their corresponding y and returns their depth. ISODEPTH has two major problems:

i. It relies on the computation of dividers (dividers defines next) for all n data points in the data set having at least quadratic complexity in n.
ii. It also relies on non-existence of collinear points meaning ISODEPTH has to remove all collinear points before computing depth, which is time consuming.

The major problems of ISODEPTH have been addressed by Johnson et al in [10], who proposed a fast algorithm for computing 2-dimensinal depth contours called FDC.
FDC is robust against collinear points and also restricts the computation of dividers to only a selected subset of points thereby making it more efficient.

Dividers:

Given a point cloud D consisting of n points, a line L is an e-divider of D if there are e points in D to the left of L, and (n-e) points in D to the right of L. This definition of e-divider is off by one point from the definition given in [17], that is an e-divider in [10] (e+1) divider in [17]. For the sake of outlier detection, whenever $e \leq (n-e)$, we say that the e points are to the '*outside*' of L and the remaining to the '*inside*' of L.

Just as e-divider L divides the data cloud of D into two disjoint subsets, it also divides the convex hull of D into two sub-regions called "*inside region* " and the "*outside region*" denoted by IR(L) and OR(L) respectively. The intersection of the all inside region of a collection of e-dividers (i.e. $\cap$ IR(L)) is called *e-intersected inside region*

A line segment between points P and Q is denoted by $\langle P,Q \rangle$. The above definition of e-divider can be extended to line segments. More precisely the depth of a line segment is the depth of the line that runs through the line segment. For instance IR($\langle P_1, P_2, P_3 \ldots P_n \rangle$) denote the inside region of the convex hull of D, i.e. the region of all points in the convex hull that are to the inside of the chain.

Every line segment in the chain is an e-divider. The chain is called *an expanded e-divider*.

Given any number of e-dividers but at least one expanded e-divider, we refer to the intersection of all their inside regions as the *expanded e-intersected inside region.*

The algorithm FDC is given in Figure 3.

Drawbacks

Theoretically the depth-based algorithms are supposed to work for high dimensions but in practice they don't work for more than two dimensions. This has been demonstrated both in the ISODEPTH and FDC [10,17].

Algorithm FDC

Input D ≡ the point cloud; K ≡ an integer
Output: Contours of depth from 0 to K

1. **G = convex hull of D;**
2. **For (d=0;d<k;) {/*new peel*/}**
   - **2.1 H=g;**
   - **2.2 Output H as the d-th depth contour;**
   - **2.3 If d == k, break; /* Done and Stop */**
   - **2.4 D=d+1; D= D-1; G= convex hull of (the updated) D;**
   - **2.5 If |H| ==3; continue; /* degenerate case of having a triangle as a convex hull*/**
   - **2.6 For (e=1; e<|H|/2){ /* Otherwise: the general case */}**
     - **2.6.1 For all points P∈ H {**
       - **2.6.1.1 Find the point $P_e \in H$ (if any) so that $\langle P, P_e \rangle$ is an e-divider of the remaining points in H**
       - **2.6.1.2 If every point G is contained in IR($\langle P, P_e \rangle$) {**
         - **2.6.1.2.1 IR[P] = IR($\langle P, P_e \rangle$; $\wedge$H[P] = 0;} /* end if */**
       - **2.6.1.3 Else { /* form an expanded e-divider */**
         - **2.6.1.3.1 Enumerate (in clockwise fashion) all the points in G that are outside $\langle P, P_e \rangle$ as $Q_1 \ldots\ldots Q_m$**
         - **2.6.1.3.2 Among them, find Q_i that maximises the angle between the segment $\langle Q_i, P \rangle$ and $\langle P, P_e \rangle$**
         - **2.6.1.3.3 Among them, find Q_j that maximises the angle between the segment $\langle Q_j, P_e \rangle$ and $\langle P_e, P \rangle$ /* without loss of generality, assume that i ≤ j */**
         - **2.6.1.3.4 IR[P] – IR[$\langle P, Q_i \ldots\ldots\ldots\ldots Q_j, P_e \rangle$); AH[P]={ $Q_i \ldots\ldots Q_j$}; }**

         } /* end for */
     - **2.6.2 Output the boundary of the region ($\cap_{P \in H}$ IR[P])as the d-th depth contour;**
     - **2.6.3 If d==k; break; /* Done and stop */**
     - **2.6.4 /* Otherwise */ d=d+1; e = e+1;**
     - **2.6.5 If ($\cup_{P \in H} \wedge$H[P] $\wedge \varnothing$ {/* G is not contained in the e-intersected inside region $\cap_{P \in H}$ IR($\langle P, P_e \rangle$) */**
       - **2.6.5.1 H –H $\cup$ ($\cup_{P \in H} \wedge$H[P]); D=D - ($\cup_{P \in H} \wedge$H[P]);**
       - **2.6.5.2 G — convex hull of (the updated) D;} /* end if */**

       } /* end for*/

} /* end for */

Figure 3: Pseudo Code of Algorithm FDC

2.3 Distance-based approach

The distance-based outlier concept was developed to remedy the problems of the distribution-based and depth-based methods. The distance-based outlier concept started with the unified notion of outliers proposed by Knorr et al in [11]. The motivation was that there were still a large number of outliers that went undetected either because there were no specific discordancy test designed for them or they could not fit into any of the existing statistical models. An oulier was defined as:

"An object O in a dataset T is a UO(p,D)- outlier if at least fraction of p of the objects in T are ≥ distance D from O".

The notion of outliers defined here is intuitive enough because it captures the general spirit of an outlier eloquently described by Hawkins [8], " ***An outlier is an observation that deviates so much from other observations to arouse the suspicion that it was generated by a different mechanism***". This notion is natural for situations where observed data do not fit any standard distribution or where no discordancy test has been developed. It has been proved that if an object O is an outlier according to a specific discordancy test, then O is also a UO(p, D)-outlier for some suitable defined p and D [11].

This preliminary work in [11] gave birth to the concept of the distance-based outliers in [12]. The distance based-outlier is not different from the unified outlier as evidenced from their definitions. ***"An object in a dataset T is a DB(p,D) outlier if at least fraction of p of the objects in T lies greater than distance D from O".***

The term DB(p,D)-outlier is used as a shorthand notation for a Distance-Based Outlier (detected using parameters p and D). Unlike depth-based methods which are restricted to small values of k (k<3}, the DB-outlier methods are not restricted computationally. The choice of p and D are left to the human expert to decide. The Index-Based (IB), Nested-Loop (NL) and the Cell-Based (CB) algorithms were proposed for mining all DB-outliers. The Index-Based and the Nested-Loop algorithms have a time complexity of $O(kN^2)$, where k is the number of dimensions and N is the dataset size. The Index-Based algorithm does not enjoy much popularity because of the initial high cost of building index for finding all DB(p, D)-outliers. The Cell-Based algorithm has a time complexity of O(N)

which is linear in N but exponential when k >3 making it unfeasible for most multidimensional applications. The Nested-Loop(NL) algorithm which enjoys high computational popularity has a quadratic time complexity in the data size ($O(kN^2)$).

Nested-Loop

The Nested-Loop (NL) algorithm uses block, nested-loop design.

Assuming a total buffer size of B% of dataset size, the algorithm divides the entire buffer into two halves called the first and second. It reads the dataset into arrays and directly computes the distance between pairs of object tuples. For each object t in the first array, a count of its D-neighbor is maintained. Counting stops for a particular tuple whenever the number of D-neighbors exceeds M.

Algorithm NL

1. Fill the first array (of size B/2% of the dataset) with a block of tuple from T
2. For each tuple t_i in the first array do:
   a. count ← 0
   b. For each tuple t_j in first array, if dist(ti,tj) ≤ D; Increment $count_i$ by 1
      If $count_i$ > M, mark t_i as a non outlier and proceed to next t_I
3. While block to be compared to the first array, do:
   a. Fill the second array with another block (but save a first block which has never served as the first array, for last)
   b. For each unmarked tuple t_j in the array, do:
      For each tuple t_j in the second array, if dist(ti,tj) ≤ D: Increment $count_i$ by 1
      If $count_I$ > M, mark t_i as a non outlier and proceed to next t_I
4. For each unmarked tuple t_I in the first array, report as an ootlier.
4. If the second array has served as he first array anytime before, stop ; otherwise, swap the names of first and second arrays and repeat the above from 2.

Figure 4: Pseudo Code for Algorithm NL

Example 2:

The following is an example of the number of passes made by the NL algorithm.

Consider 50% buffering of 4 blocks of data denotes as A, B, C, D i.e. each block contains ¼ of the data. The order of the filling the array and comparing is as follows:

1. A with A, then with B, C, D for a total of 4 blocks read
2. D, with D (no read required) , then with a (no read) ,B,C for a total of 2 blocks read
3. C with C, then with D, A, B for a total of 2 block read
4. B with B, then with C, A, D for a total of 2 blocks read

The table below shows the order, the blocks loaded, and the blocks staying in the first array

	B L	O C	K S	
Disk I/O order	A	B	C	C
1	L			
2	*	L		
3	*		L	
4	*			L
5		L		*
6			L	*
7	L		*	
8		L	*	
9		*		L
10	L	*		

L : Load
* : in buffer (first array)
T= total number of dist I/O

T= n+(n-2)(n-1)
= 4 +(4- 2)(4-1)
= 10 blocks

Total Number od data passes= 10/4
= 2.5

Drawback

1. The Nested-Loop (NL) algorithm has a time complexity $O(KN^2)$ which is linear in the number of dimensions but quadratic in N (dataset size) making it highly unfavorable when the data size is large. This means that NL does not scale well for warehouse systems where the size of data is emphatically large owing to the number of passes.
2. It requires he user to specify a distance d which could be very difficult
3. It does not provide ranking for outliers

Hung et al in [9] has addressed the scalability problem of the Nested-Loop (NL). The Enhanced NL (ENL) was proposed as a solution to the problems of NL. ENL reduces the number of (data passes) the disk I/O of NL by half. ENL is paralleled (PNL) to further reduce the running time in the shared-nothing system, which becomes more popular for its high performance and low cost. Actually when running in a processor, PNL is almost reduced to ENL.

Enhanced NL

In NL, there are redundant block reading and comparison. In ENL an order is proposed which results in reducing the computation time and disk I/O time to half that of NL. The arrangement is that, in each turn, the blocks read into a second array in the same order until the end of the series of ready block is reached, then the block in the first array is marked as done, the names of the two arrays are swapped and the order is reversed. The above is repeated until all blocks are done. The resultant is an Enhanced NL (ENL)

Algorithm ENL

1. label all blocks as " ready" (the block is either in "ready" or "done" state)
2. Fill the first array (of size B/2% of the dataset) with a block of tuple from T
3. For each tuple t_i in the first array do:
   a. count the number of tuples in first array which are close to t (distance $\leq$ D)
   If count > M, mark t as a non outlier
4. Set the block-reading order as "forward'
5. Repeat until 'ready" blocks (without marked as done) are compared to the first array in a specified block-reading order, do:
   a. fill the second array with next block
   b. for each unmarked tuple t_i in the first array , do:
      i. for each tuple t_j in the second array , if dist(ti,tj) $\leq$ D;
         A. increase $count_i$ and $count_j$ by 1; If $count_i$ > M, mark t_i as a non outlier proceed to next t_I; If $count_j$ > M, mark t_j as a non outlier
6. report unmarked tuples in the first array as outliers

7. If the second array is marked as done, stop; otherwise, mark the block in the first array as done, reverse the block–reading order, swap the names of the first and second arrays ad repeat the above from step 3.

Example 3:

The following a is an example of the number of passes made by the ENL algorithm. Consider 50% buffering of 4 blocks of data denotes as A, B, C, D i.e. each block contains ¼ of the data. The order of the filling the array and comparing is as follows:

1. A with A, then with B, C, D for a total of 4 blocks read
2. D, with D (no read required) , then with C, B for a total of 2 blocks read
3. B with B, then with C, for a total of 1 block read
4. C with C, for a total of 0 blocks read

The table below shows the order, the blocks loaded, and the blocks staying in the first array

	B L	O C	K S	
Disk I/O order	A	B	C	C
1	L			
2	*	L		
3	*		L	
4	*			L
5			L	*
6		L		*
7		*	L	

L : Load
* : in buffer (first array)
T= total number of dist I/O

T= 1+3+2+1
= 7 blocks

Total Number od data passes= 7/4
= 1.75

It is clear from Example 2 and Example3 that ENL reduces the number of passes (disk I/O) to half that of NL hence making it more efficient than NL. The time complexity of PNL is quadratic to the reciprocal of the number of processors. Hence ENL and PNL are more favourable for finding all DB(p,D) outliers in warehouse systems.

ENL and PNL reduce the number of disk I/O (data passes) but still rely on the definition of outlier in [12]. There is still no ranking for the outliers and the user will still have to specify the distance parameter d, which are the drawbacks of [12].

An all-inclusive solution to the problems in the approach by [12] has been given by Ramaswamy et al in [16]. A novel definition for a distance-based outlier as well as an efficient Partition-Based algorithm for mining all such outliers have been proposed based on the distance of a point from its k^{th} nearest neighbour. An outlier was defined as follow:

"***Given a k and n, a point p is an outlier if no more than n-1 other points in the data set have higher value of D^k than p. In other words, the top n points with a maximum D^k values are considered outliers. We refer to these outliers as the D^k_n (prnounce "dee-kay-en") outliers of the dataset***"

The above definition has intuitive appeal since in essence it rank each point based on its distance from its k^{th} nearest neighbour. It does not also require the user to specify the distance parameter d.

The Partition-Based algorithm

The Partition-Based algorithm divides (partitions) the entire points in a data set into disjoint subsets using clustering algorithms (*it should be noted that clustering algorithms are used here for the purpose of generating clusters and not for detecting outliers*). Points within each cluster are tested and pruned as soon as it is detected that the entire cluster can not contain outliers. The remaining partitions that contain outliers form the candidate partition. The candidate partition is used as an input to either the Nested-Loop or Index-Based algorithm to find the outliers. The Partition-Based algorithm has four major steps described below.

1. **Generate partitions**: In the first step, clustering algorithms are used to generate the clusters (partition) and each cluster is treated as a separate partition.
2. **Compute bounds on D^k for point in each partition:**
   Compute lower and upper bound on D^k and store them in P.lower and P.upper respectively. Thus for every point $p \in P$, $D^k(p) \geq$ P.lower, $D^k \leq$ P.upper
3. **Identify candidate partition containing outlies**: At this point we identify points that are candidate for outliers. These are points for which P.upper $\geq$ minDKDist. minDKDist is computed from P.lower (see [16] for details). The procedure for computing candidate partition is given in Figure 5.

```
Procedure computecadidatePartition(Pset, k, n)
begin
1.  for each partition P in Pset do
2.        insertIntoIndex(Tree,P)
3.  partheap : = ϕ
4.  minDKDist : = 0
5.  for each partition P in Pset do {
6.        computeLowerUpper(Tree, Root, P, k, minDKDist)
7.        if ( P.lower> minDKDist) {
8.            partHeap.insert(P)
9.            while partHeap.numPoints() – partHeaptop().numPoints() ≥ n do
10.               partHeap.deleteop()
11.           if (partHeap.numPoints() ≥ n)
12.           minDKDist  := partHeap.top().lower
13.       }
14. }
15. candset  : = ϕ
16.  for each partition P in Pset do
17.  if ( p.upper ≥ minDKDist) {
18.       candSet : = candSet ∪ {P}
19.     P.neighbour: = {Q :Q ∈ Pset and MINDIST(P,Q) ≤ P.upper}}
20. return CandSet
```

Figure 5: Computation of Candidate Partition

4. **Compute Outliers from points in candidate partition:** Once the candidate partition is obtained, it is used a an input to the Nested-Loop algorithm to generate the outliers

Example 4:

The exam score obtained by 25 students given below is used to go through the 4 basic steps of the algorithm. *{60, 70, 80, 90, 96, 84, 71, 65, 66, 72, 83, 97, 98, 86, 73, 81, 64, 75, 88, 92, 89, 87, 76, 78, 79}*

N=25, n=3 (3 outliers)

- ***Generate partitions:***

  Clustering algorithms are used to generate clusters based on the data.

Range	*Score*
90+	*90, 92, 96, 97, 98*
80+	*80, 81, 83, 84, 86, 87, 88, 89*
70+	*70, 71, 72, 73, 75, 76, 78, 79*
60+	*60, 64, 65, 66*

- ***Compute bounds on D^k to Obtain MinDkDist :***

  Compute lower and upper bounds for each partition and use the lower bound to determine the minimum distance that can be included in the outlier list (MinDkDist)

Partitions	*90+*	*80+*	*70+*	*60+*
***P.lower** (Min-60)*	*30*	*20*	*10*	*0*
***P.upper** (Max-60)*	*38*	*29*	*19*	*6*

*Obtain P** from P.lower = { 30, 32, 36, 37, 38)*

*MinDkDist = Min (p**) = min {30, 32, 36, 37, 38} = **30***

- ***Identify candidate partitions containing outliers:***

  *Candidate partitions (Candset) are partitions for which **P.upper** >= **MinDkDist** .*

  Hence Our candset = {90, 92, 96, 97, 98}

- ***Compute outliers from candidate partitions:***

  Outliers are computed by examining only the candidate set.

Points	*90*	*92*	***96***	***97***	***98***
Distance	*0*	*2*	*6*	*7*	*8*
Ranking	*5*	*4*	***3***	***2***	***1***

*The 3 outliers are **{98,97,96}***

Min – Minimum in each partition, Max- Maximum in each partition
P.upper – upper bound, P.lower – Lower bound
P** -- distance in the partition that has maximum P.lower (90+) and contains at least n points

In additions to the three main approaches described earlier, there are emerging approaches for mining outliers that are not yet popular. A typical example is FindOut [21], which uses signal processing technique for detecting outliers. FindOut is By-product of WaveCluster and the main idea is removing clusters from the original data to identify outliers. The algorithm removes identified clusters until there is no more cluster in the dataset. The remaining objects that do not belong to any cluster are labelled as outliers.

The last but not the least is LOF developed by Breunig et al in [5]. LOF is based on the principle that for many application being an outlier is not just a binary property (a point p is either an outlier or not) and as such it is more meaningful to assign each object a degree of being an outlier called a local outlier factor(LOF). The degree is called local because it depends on how isolated an object is, with respect to the surrounding neighbourhood. The LOF an object is based on the number of nearest neighbours used in finding the neighbourhood. of the object. **Outliers are objects that tend to have higher LOF values** (see [5] for LOF algorithm).

3.0 Conclusions

The mining approaches discussed so far can be used to mine outliers but in warehouse systems where the scalability of an algorithm is of prime importance because of the size of data, calls for more efficient and scalable algorithms for outlier mining. The integration of some the multidimensional statistical models with the distance-based methods might offer a good solution.

References:

[1] Asimov D.: "The grand tour, A Tool for Viewing Multidimensional Data" SIAM J.Sci. Stat. Compu, 6:128-143, 1985.

[2] Bartkowiak A., Szustalewicz A.: " Detecting Outliers by a grand tour" Machine Graphics and Vision . 6:487-505, 1997

[3] Bartkowiak A.: " Finding Outliers by Dynamic Projections" Unpublished., July 1998.

[4] Barnett V., Lewis T,: "Outliers in Statistcal data " John Willey, 1994

[5] Breunig M.M., Kriegel H-P., Ng R.T., Sander J.: "LOF: Identifying Outliers in Large dataset" Proc. ACM SIGMOND 2000 Int. Conf. On Management of Data, Dallas, TX 2000

[6] Ester M., Kriegel H-P., Sander J., Xu X.: " A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise" Proc. 2nd Int. Conf. On knowledge Discovery and Data Mining, Portland, OR 1996, pp 226-231

[7] Fayyad U., Piatesky-Shapiro G., Smyth P.: " Knowledge Discovery and Data Mining: Towards a Unifying Framework" Proc. 2nd Int. Conf. On knowledge Discovery and Data Mining, Portland, OR 1996, pp82-88

[8] Hawkins D.: "Identification of Outliers" Chapman and Hall, London 1980

[9] Hung E., Cheung D.W.: " Parallel Algorithms for Mining Outliers in Large Databases" In Proc of the 4th PAKDD 2000, Kyoto, Japan

[10] Johnson T., Kwok I., Ng R.: " Fast Computation of 2-D depth Contours" In Proc. KDD 1998 pp224-228.

[11] Knorr E.M, Ng R.T.: " A Unified Notion of Ourliers: Properties and Computaion" In Proc. of KDD 97, 19997 pp219-222

[12] Knorr E.M, Ng R.T.: " Algorithms for Mining Distance-Based Outliers in Large Dataset" In Proc. of the 24th VLDB Conf.. New York, USA 1998.

[13] Knorr E.M, Ng R.T.: "Finding Intentional Knowledge of Distance-Based Outliers" In Proc. of the 25th VLDB Conf.. Edinburg,Scottland, 19989, pp211-222

[14] Ng R., Han J.: " Efficient and Effective Clustering Methods for Spatial data miming" In Proc. of 20th VLDB pp 144-155

[15] Ramakrishnana R., Livny M.: " An Efficient data Clustering Methods for very Large databass " In Proc. of ACM SIGMOD, 1996 pp103-114

[16] Ramaswamy S., Rastogi R., Shim K.: " Efficeint Algorithms for Mining Outliers from Large Data sets" In Proc. of ACM SIGMOD 2000, USA pp127-138

[17] Ruts I., Poussecuw P. " Computing Depth Contours of Bivarate points Cloud" Computational Statistics and Data Analysis 1996, 23:153-16

[18] Tucakov V., Ng R." Identifying Unusual People Behaviour: A Case Study of of Mining Outliers In Spatio-Temporal Trajectory Databases" Proc. SIGMOD Workshop on Reserch Issues in KDD 1998.

[19] Tukey J.: "Mathematics and Picturing of Data " In Proc. Int. Congress. on Mathematics, 1975 pp523-531

[20] Tukey J.: " Exploratory Data Analysis" Adison Wesley, 1997

[21] Yu D., Sheikkholeslami G, Zhang A.: " FindOut: Finding Outliers In very Large Databases." A technical report supported by NSF CAREER grant, Paper ID 394