BVG Trust

J. Sci. Trans. Environ. Technov. 2007, 1(1): 23-28

Scientific Transactions in Environment and Technovation

An efficient technique for outlier detection in data mining

10.20894/STET.116.001.001.004
www.stetjournals.com

E. Ramaraj and K. Subramanian*

Alagappa University, Computer Centre, Karaikudi - 630 003, Tamil Nadu, India

Abstract

The Outlier detection problem has important applications in the field of fraud detection, network robustness analysis and intrusion detection. Most of such uses are high dimensional domains in which the data can contain hundreds of dimensions. Many recent algorithms employed the concept of proximity in order to find the outliers based on their relationships to the rest of data. But in high dimensional space, the data is sparse and the notation of proximity fails to retain its meaning. In fact the sparsity of high dimensional data, the notation of finding meaningfull outliers becomes substantially more complex and non-obvious. This article is an attempt to enhance outlier detection and analysis, which is an interesting data mining task. This outlier mining has a plethora of applications in fraud detection and for finding abnormal responses in various fields. In computer based outlier mining there are many methods and techniques. This work is an attempt to highlight the merits and demerits on the application of these methods. This would enable to find a suitable technique for perfect outlier detection. The main objective of this present article is to find out the best approach by comparative analysis.

Keywords : data mining, dataset, fraud detection, outlier detection

INTRODUCTION

Fraud detection is an outstanding data mining task among the mining techniques, that has a lot of practical applications in may different domains. Fraud detection mining can be defined as follows: "Given a set of N data points or objects and the number n of expected outliers, find the top n objects that are considerably dissimilar, exceptional or inconsistent with respect to the remaining data". Many data mining algorithms consider outliers as noise that must be eliminated because it degrades their predictive accuracy (Aggarwal and Yu, 2001). For example, in classification algorithms (Aluru and Sevilgen, 1997) mislabeled instances are considered outliers and thus they are removed from the training set to improve the accuracy of the resulting classifier.

However, as pointed out "one person's noise could be another person's signal", outliers themselves can be of great interest. Fraud detection can be used in telecom or credit card frauds to detect the typical usage of telecom services or credit cards. Fraud detection actually consists of two sub problems-first is to define what data is deemed to be exceptional in a given data set and second is to find an efficient algorithm to obtain such data (Arning *et al.*,1996). Fraud detection methods can be categorized in several approaches, each assumes a specific concept of what exception is. Among them, the distance-based approach, introduced by Knorr and Ng (1998) adefinition of distance-based outlier relies on the Euclidean distance between two points. This kind of definition is relevant in a wide range of real-life application domains (Barnett and Lewis, 1994).

In this paper we propose a new definition of outlier (Yu *et al.*, 1999; Breunig *et al.*, 2000), distance-based and an efficient algorithm, called FDHD, designed to detect the top n outliers of a large and high-dimensional data set. Given an application dependent parameter k, the weight of a point is defined as the sum of the distances separating it from its k nearest-neighbors. Exceptions are thus the points scoring the largest values of weight (Brodley and Friedl, 1996). The computation of the weights, however, is an expensive task because it involves the calculation of the k nearest neighbors of each data point (Chan, 1997). To overcome this problem we present a definition of approximate set of outliers. Elements of this set have a weight greater than the weight of true outliers within a small factor. This set of points represents the points candidate to be the true outliers (Faloutsos, 1986; Faloutsos and Roseman, 1989). Thus we give an algorithm consisting of two phases. The first phase provides an approximate solution, within a factor O(kd1+1t),where d is the number of dimensions of the data set and t identifies the Lt metrics of interest, after executing at most d + 1 sorts and scans of the data set, with temporal cost O(d2Nk) and 2 spatial cost O(Nd), where N is the number of points in the data set. The algorithm avoids the distance computation of each pair of points because it makes use of the space-filling curves to linearize the data set. We fit the d-dimensional data set DB in the hypercube D = [0,1] d, then we map D into the interval I = [0,1] by using the Expert space filling curve and obtain the approximate k nearest neighbors of each point by examining its predecessors and successors on I. The mapping assures that if two points are close in I, they are close in D too,

*Corresponding author
subjjcit@rediffmail.com

although the reverse in not always true. To limit the loss of nearness, the data set is shifted d + 1 times along the main diagonal of the hypercube [0,2]d . During each scan the algorithm calculates a lower and an upper bound to the weight of each point and exploits such information to isolate points candidate to belong to the solution set. The number of points candidate to belong to the solution set is sensibly reduced at each scan. Hence, the first phase produces a set of approximate outliers that are candidate to be the true outliers. The second phase calculates the exact solution with a final scan of temporal cost O(NxNd), where Nx is the number of candidate outliers remained after the first phase. Experimental results show that the algorithm always stops, reporting the exact solution, during the first phase after d steps, with d much less than d + 1. We present both an in-memory and disk-based implementation of the FDHD algorithm and a throughout scaling analysis for real and synthetic data sets showing that the algorithm scales well in both cases.

DEFINING OUTLIERS

Definition 1

Let t be a positive number, then the Lt distance between two points p = (p1,...,pd) and q = (q1,...,qd) of Rd is defined as dt(p,q) = (di=1 l pi- qi l t) 1/t for $1 \le t < \infty$, and as max $1 \le i \le d$ l pi- qi l , t = ∞. Definition 1 (Weight) Let DB be a d-dimensional data set, k a parameter and p a point of DB. Then the weight of p in DB is defined as ω k(p) =ki=1dt(p,nni(p)), where nni(p) denotes the i-th nearest neighborhood of p in DB according to the Lt distance. That is, the weight of a point is the sum of the distances separating that point from its k nearest neighbors. Intuitively, the notion of weight captures the degree of isolation of a point with respect to its neighbors, higher is its weight, more distant are its neighbors (Han and Kamber, 2001).

Definition 2

Let DB be a data set, k and n two parameters, and let p be a point of DB. Then p is the nth outlier with respect to k in DB, denoted as outlier nk, if there are exactly n - 1 points q in DB such that ωk(q) ≥ ωk(p). We denote with Out nk the set of the top n outliers of DB with respect to k. Thus, given n, the expected number of outliers in the data set, and an application dependent parameter k, specifying the size of the neighborhood of interest, the outlier detection problem consists in finding the n points of the data set scoring the maximum ωk values. The computation of the weights is an expensive task because it involves the calculation of k nearest neighbors of each data point. While this problem is well solved in any fixed dimension, requiring O(log N) time to perform each search (with appropriate space and preprocessing time bounds) , when the dimension d is not fixed, the proposed solutions become impracticable since they have running time logarithmic in N but exponential in d. The lack of efficient algorithms when the dimension is high is known as "curse of dimensionality". In these cases a simple linear scan of the data set, requiring O(N2d) time, outperforms the proposed solutions. From what above stated, at the present, when large and high-dimensional data sets are considered, a good algorithm for the solution of the outlier detection problem is the naïve nested-loop algorithm which, in order to compute the weight of each point, it must consider the distance from all the points of the data set, thus requiring O(N2d) time. Data mining applications, however, require algorithms that scale near linearly with the size of the data set to be practically applicable. An approach to overcome this problem could be to first find an approximate, but fast, solution, and then obtain the exact solution from the approximate one. This motivate our definition of approximation of a set of outliers.

Definition 3

Approximation of Out nk Let DB be a data set, let Out x= {a1,...,an} be a set of n points of DB, with ωk(ai) ≥ ωk(ai+1), for i = 1,...,n - 1, and let be a positive real number greater than one. We say that Out x is an -approximation of Outnk, if ωk(ai) ≥ ωk (outlier ik), or each i = 1,...,n. In the following sections we give an algorithm that computes an approximate solution within a factor O(kd1+1t), where t is the Ltmetrics of interest, runs in O(d2Nk) time and has spatial cost O(Nd). The algorithm avoids the distance computation of each pair of points because it makes use of the space-filling curves to linearize the data set. To obtain the k approximate nearest neighbors of each point p it is sufficient to consider its successors and predecessors on the linearized data set (Jagadish, 1990). The algorithm produces a set of approximate (with respect to the above definition) outliers that are candidates to be the true outliers. The exact solution can then be obtained from this candidate set at a low cost.

ALGORITHM

In this section we give the description of the FDHD algorithm, which solves the Fraud Detection Problem. The method consists of two phases, the first does at most d + 1 sorts and scans of the input data set and guarantees a solution that is an kd -approximation of Outnk, where d= O(d1+1t), with a low time complexity cost (Jagadish, 1990). The second phase does a single scan of the data set and computes the set Outnk (Knorr and Ng, 1998). At each scan FDHD computes a lower bound and an upper bound to the weight of each point and it maintains the n greatest lower bound values of weight in a heap. The lowest value in this heap is a lower bound to the weight of the n-th exception and it is used to detect those points that can be considered candidate outliers. The upper and lower bound of the weight of each point are computed by exploring the neighborhood of the point according to the Expert order. The size of this neighborhood is initially set to 2k, then it is widened, proportionally to the number of remaining candidate outliers, to obtain a better estimate of the true k nearest neighbors. At each iteration, as experimental results show, the number of candidate outliers sensibly diminishes (Knorr *et al.,* 2000). This allows the algorithm to find the exact solution in few steps, in practice after d steps with d much less than d + 1. The algorithm FDHD, receives as input a data set DB of N points in the hypercube [0,1]d, the number n of top outliers to find and the number k of neighbors to consider (Lee *et al.,* 1998). The data structures employed by the algorithm are the two heaps OUT and WLB, the set TOP, and the list of point features PF: • OUT and WLB are two heaps of n point features. At the end of each iteration, the features stored in OUT are those with the n greatest values of the field ubound, while the features stored in WLB are those with the n greatest values of lbound • TOP is a set of at most 2n point features which is set stored in OUT and WLB at the end of the previous iteration • PF is a list of point features. In the following, with the notation Pfi we mean the i-th element of the list PF First, the algorithm builds the list PF associated to the input data set, i.e. for each point p of DB a point feature f with its own f.id value, f.point = p, f.ubound = ∞, f.level and f.lbound set to 0, and f.nn = x, is inserted in PF, and initializes the set TOP and the global variables ωx, Nx, and nx:• ωx is a lower bound to the weight of the outlier nk in DB. This value, initially set to 0, is then updated in the procedure Scan • Nx is the number of point features f of PF such that f.ubound ≥ ωx. The points whose point feature satisfies the above relation are called candidate outliers because the upper bound to their weight is greater than the current lower bound ωx. This value is updated in the procedure Expert• nx is the number of true outliers in the heap OUT. It is updated in the procedure True Outliers and it is equal to | {f x OUT : f.lbound = f.ubound ∍ f.ubound ≥ ωx} | The main cycle, consists of at most d + 1 steps. We explain the single operations performed during each step of this cycle.

The Expert procedure calculates the value H(PFi.point + v(j)) of each point feature PFi of PF, where j x {0,...,d} identifies the current main iteration, places this value in PFi .expert, and sorts the point features in the list PF using as order key the values PFi.expert. Thus it performs the Expert mapping of a shifted version of the input data set. It is straight forward to note that the shift operation does not alter the mutual distances between the points in PF. As v(0)is the zero vector, at the first step (j = 0) no shift is performed. Thus during this step we work on the original data set. After sorting, the procedure Expert up-dates the value of the field level of each point feature. In particular, the value Pfi. level is set to the order of the smallest r-region containing both Pfi .point and PFi+1 .point, i.e., to MinReg(Pfi .point, PFi+1.point), for each i = 1,...,N - 1.

```
Algorithm FDHD (DB, n, k)
{
Initialize(PF, DB);
TOP = x;
Nx = N;
 nx = 0;
ωx = 0;
j = 0;
while (j <= d) && (nx< n)
{
Initialize(OUT);
Initialize(WLB);
Expert(v(j));
Scan(v(j),kNNx);
TrueOutliers(OUT);
TOP = OUT  WLB;
j ++;
if (nx< n)
  Scan(v(d), N);
return OUT;
}
```

Scan.

This procedure performs a sequential scan of the list PF by considering only those features that have a weight upper bound not less than ωx, the lower bound to the weight of outlier nk of DB. These features are those candidate to be outliers, the others are simply skipped. If the value PFi.lbound is equal to Fi.ubound, then this is the true weight of PFi.point in DB. Otherwise PFi.ubound is an upper bound for the value ωk (PFi.point) and it could be improved. For this purpose the function FastUpperBound calculates a novel upper bound ω to the weight of PFi.point, given by k×MaxDist (PFi.point,2-level0), by examining k points among its successors and predecessors to find level 0 , the order of the smallest r-region containing both PFi.point and other k neighbors. If ω is less than ωx, no further elaboration is required. The procedure InnerScan returns a new lower bound newlb and a new upper bound newub for the weight of PFi.point (see the description of InnerScan below for details regarding the calculation of these bounds). If newlb is greater than PFi.lbound then a better lower bound for the weight of PFi.point is available, and the field lbound, is updated. Same considerations hold for the value PFi.ubound. Next, the heaps OUT and WLB process

PFi. That is, if PFi.ubound is greater than the smallest upper (lower resp.) bound .ubound (f.lbound resp.) stored in OUT (WLB resp.), then the point feature f stored in OUT (WLB resp.) is replaced with PFi. Finally, the lower bound ωx to the weight of the n-th outlier is updated if a greater lower bound has been computed.

Inner Scan

This procedure takes into account the set of points PF a.point,...,PFi-1.point,PFi+1.point,...,Fb.point i.e., the points whose Expert value lies in a one dimensional neighborhood of the integer value PFi.expert. The maximum size allowed for the above neighborhood is stored in the input parameter maxcount. In particular, if PFi belongs to TOP, i.e., the point is a candidate to be one of the n top outliers we are searching for, then the size b - a of the above neighborhood is at most N, the size of the entire data set, otherwise this size is at most 2k0.

We note that the parameter k0, that is the number of neighbors to consider on the above interval, of the procedure Scan is set to kN/Nx, i.e., it is inversely proportional to the number Nx of candidate outliers at the beginning of the current main iteration. This allows the algorithm to analyze further the remaining candidate outliers, maintaining at the same time the number of distance computations performed in each iteration constant.

```
procedure Scan(v, k0);
{
for (i = 1; i<N; i++)
 if (PFi.ubound >= ωx)
{
   if (PFi.lbound < PFi.ubound)
   {
    ω := FastUpperBound(i);
   if (ω < ωx)
      Fi.ubound := ω;
else
{
 maxcount := min(2k0,N);
 if (PFix TOP)
 maxcount := N;
InnerScan(i, maxcount, v, PFi.nn, newlb, newub);
if (newlb > PFi.lbound)
PFi.lbound := newlb;
if (newub < PFi.ubound)
PFi.ubound := newub;
}
}
Update(OUT, PFi);
Update(WLB, PFi);
ωx:= max(ωx,Min(WLB));
}
}
```

This procedure manages the set NN of at most k pairs (id,dist), where id is the identifier of a point feature f and dist is the distance between the current point PFi.point and the pointf.point. The variable levela (levelb respectively), initialized to the order h of the approximation of the space filling curve, represents the minimum among PFa-1.level, ..., PFi-1.level (Fi.level,..., Fb.level resp.) while level represents the maximum between levela and levelb. Thus level+1 is the order of the greatest entirely explored r-region (having side r = 2-(level+1)) containingPFi.point. The values a and b are initially set to i. Then, at each iteration of InnerScan, the former is decreased or the latter is increased, until a stop condition occurs or their difference exceeds the maximum size allowed. In particular, during each iteration, if PFa-1.level is greater than PFb.level then a is decreased, else b is increased. This enforces the algorithm to entirely explore the current r-region, having order level, before starting the exploration of the surrounding r-region, having order level - 1. The distances between the point PFi.point and the points of the above defined set are stored in NN by the procedure Insert. In particular nsert(NN,id,dist) works as follows: provided that the pair (id,dist) is not already present in NN, if NN contains less then k elements then procedure

```
InnerScan(i, maxcount, v, var NN, newlb, newub);
{
p = PFi.point;
 Initialize(NN);
a =  b = i;
level =levela = levelb =h;
count = 0;
stop = false;
while ((count < maxcount)&&(not stop))
{
count ++;
if (PFa-1.level > PFb.level)
{
a --;
levela = min(levela,PFa.level);
c = a;
else
{
levelb = min(levelb,PFb.level);
b ++;
c = b;
Insert(NN, PFc.id, dt(p,PFc.point));
if (Size(NN) = k)
{
if (Sum(NN) < ωx)
{
stop = true;
else
if (max(levela,levelb) < level)
{
level = max(levela,levelb);
δ = MinDist(p,2-(level+1));
if (δ ≥ Max(NN))
stop = true;
}
}
}
r = BoxRadius(p + v,PFa-1.point + v,PFb+1.point +
v);
newlb = SumLt(NN, r);
newub = Sum(NN);
}
```

The procedure InnerScan the pair (id,dist) is inserted in NN, otherwise if dist is less than the smallest distance stored in a pair of NN then this pair is replaced with the pair (id,dist). The procedure InnerScan stops in two cases. The first case occurs when the value Sum(NN) is less than ωx, where Sum(NN) denotes the sum of the distances stored in each pair of NN, i.e.,when the upper bound to the weight of PFi.point just determined is less than the lower bound to the weight of the outlier nk of DB. This means that PFi.point is not an outlier. The second case occurs when the value of level decreases and the distance between PFi.point and the nearest face of its 2 -(level+1)-region exceeds the value Max(NN), i.e. the distance between PFi.point and its k-th nearest neighbor in DB. This means that we already explored the r-region containing both PFi.point and its k nearest neighbors.

At the end of the procedure InnerScan, the function BoxRadius calculates the radius r of the greatest entirely explored neighborhood of PFi.point. This value can be obtained by using lemma 1, more simply by exploiting the values levela, levelb, PFa-1.level and PFb.level, i.e.,as 2-max(min(level a,PFa-1.level), min(level b,PFb.level) Finally, newlb is set to the sum of the distances stored in NN that are less or equal than r while newub is set to the sum of all the distances stored in NN. The main cycle of the algorithm FDHD stops when nx= n, i.e., when the heap OUT is equal to the set of top n outliers, or after d + 1 iterations. At the end of the first phase, the heap OUT contains a kd -approximation of Outnk. Finally, if nx< n, that is if the number of true outliers found by the algorithm is not n, then a final scan computes the exact solution. During this final scan the maximum size of the one dimensional neighborhood to consider for each remained candidate outlier is N, that is the entire data set. This terminates the description of the algorithm.To conclude, we distinguish between two versions of the above described algorithm: • nn-FDHD: this version of FDHD uses extended point features, i.e., the nearest neighbors of each point, determined in the procedure InnerScan, are stored in its associated point feature and then reused in the following iterations. • no-FDHD: this version uses point features with the field nn always set to x, i.e.,the nearest point determined during each iteration are discarded after their calculation. The former version of the algorithm has extra memory requirements over the latter version, but in general we expect that nn-FDHD presents an improved pruning ability.

MEMORY-BASED ALGORITHM

We described the algorithm FDHD assuming that it works with main memory resident data sets. Now we show how the in-memory algorithm can be adapted to manage efficiently disk-resident data sets (Moon *et al.*, 2000; Ramaswamy *et al.*, 2000). Basically, the disk-based implementation of FDHD has the same structure of its memory-based counterpart. The main difference is that the list PF is disk-resident, stored in a file of point features. In particular, the disk-based algorithm manages two files of point features, called F_{in} and F_{out}, and has an additional input parameter BUF, that is the size (in bytes) of the main memory buffer. First, the Procedure Initialize creates the file F_{in} with the appropriate values, and with the field f.expert of each record f set to H(f.point). The procedure Hilbert is substituted by the procedure Sort, performing an external sort of the file F_{in} and producing the file F_{out} ordered with respect to the field hilbert. We used the polyphase merge sort with replacement selection to establish initial runs to perform the external sort. This procedure requires the number FIL of auxiliary files allowed and the size BUF of the main memory buffer. After the sort, F_{in} is set to the empty file. The procedure Scan (and hence InnerScan) performs a sequential scan of the file F_{out} working on a circular buffer of size BUF containing a contiguous portion of the file. We have the following differences with the in-memory implementation: • After a record is updated (i.e.,at the end of each iteration of Scan), it is appended to the file F_{in} with the field f.expert set to H(f.point + v(j+1)), where j denotes the current main iteration of the algorithm •The maximum value allowed for the parameter k0 is limited by the number of records (point features) fitting in the buffer of size BUF •The records of the set TOP are maintained in main memory during the entire execution of Scan, compared with the entire data set, and flushed at the end of the overall scan in the appropriate position of the file F_{in}. As for the second phase of the algorithm, this is substituted by a semi-naive nested-loop algorithm. In practice, the records associated with the remaining candidate outliers are stored in the main memory buffer and compared with the entire data set until their upper bound is greater than ωx . The heaps OUT and WLB are updated at the end of the scan. If the remained candidate outliers do not fit into the buffer, then multiple scans of the feature file are needed. When the nn-FDHD version of the algorithm is considered, to save space and speed up the external sort step, the additional boolean field extended is added to every record. This field specifies the size of the record.

not contain the field nn, while f.extended set to 1 means that the field nn is present in f. Thus we have records of variable length. Only records associated with candidate outliers have their field extended set to 1. This field is managed as follows: • When a record f is appended to the file F_{in} at the end of each iteration of Scan, the field nn is added provided that f.ubound ≥ ωx • The procedure Sort must support records of variable length. Moreover, it is modified so that when it builds the file F_{out} by sorting the file F_{in} , it discharges the fields nn of the records f having f.ubound < ωx (we note that this condition could not be satisfied when the record f is appended to F_{in} , as ωx can decrease in the following iterations of Scan) when the disk-based implementation of the FDHD algorithm is considered, the extra time needed to no-FDHD to prune points from the data set, is partially balanced by the lower time required to perform the external sort of the feature file w.r.t. the nn-FDHD.

CONCLUSIONS

The new definition of outlier that is distance-based and an algorithm, called FDHD, are designed to efficiently detect the top n outliers of a large and high-dimensional data set. The algorithm consists of two phases. The first phase provides an approximate solution with temporal cost O(d2Nk) and spatial cost O(Nd). The second phase calculates the exact solution with a final scan. We presented both an in-memory and disk-based implementation of the FDHD algorithm to deal with data sets that cannot fit into main memory. Experimental results on real and synthetic data sets up to 500,000 points in the 128-dimensional space showed that the algorithm always stops, reporting the exact solution, during the first phase, and that it scales well with respect to both the dimensionality and the size of the data set.

REFERENCES

Aggarwal, C.C. and Yu, P.S. 2001. Outlier detection for high dimensional data. *In: Proc. ACM Int. Conference on Managment of Data* (SIGMOD'01) ACM Sigmoid Rec. 30. P. 37-46.

Aluru, S. and Sevilgen, F.E. 1997. Parallel domain decomposition and load balancing using space-filling curves. *In: Proc. Int. Conf. on High Performace Computing*. P. 230–235.

Arning, A., Aggarwal, C. and Raghavan, P. 1996. A linear method for deviation detection in large databases. *In: Proc. Int. Conf. on Knowledge Discovery and Data Mining* (KDD'96). P.164–169.

Barnett, V. and Lewis, T. 1994. *Outliers in Statistical Data*. John Wiley and Sons, New York.

Breunig, M.M., Kriegel, H. , Ng, R.T. and Sander, J. 2000. LOF: Identifying density-based local outliers. *In: Proc. ACM Int. Conf. on Managment of Data* (SIGMOD'00). ACM Sigmoid Rec. 29: 93-104.

Brodley, C. E. and Friedl, M. 1996. Identifying and eliminating mislabeled training instances. *In: Proc. National American Conf. on Artificial Intelligence* (AAAI/IAAI 96). P. 799–805.

Chan, T. 1997. Approximate nearest neighbor queries revisited. *In: Proc. 13th Annual ACM Symp. on Computational Geometry.* P. 352–358.

Faloutsos, C. 1986. Multiattribute hashing using gray codes. *In: Proc. ACM Int. Conference on Managment of Data* (SIGMOD'86). P. 227–238.

Faloutsos, C. and Roseman, S. 1989. Fractals for secondary key retrieval. *In: Proc. ACM Int. Conf. on Principles of Database Systems* (PODS'89). P. 247–252.

Han J. and Kamber, M. 2001. *Data Mining, Concepts and Technique*. Morgan Kaufmann, San Francisco.

Jagadish, H.V. 1990. Linear clustering of objects with multiple atributes. *In: Proc. ACM Int. Conf. on Managment of Data* (SIGMOD'90). P. 332–342.

Knorr E. and Ng. R. 1998. Algorithms for mining distance-based outliers in large datasets. *In: Proc. Int. Conf. on Very Large Databases* (VLDB98). P. 392–403.

Knorr, E. Ng, R. and Tucakov, V. 2000. Distance-based outlier: algorithms and applications. *VLDB J.* 8: 237–253.

Lee, W., Stolfo, S.J. and Mok. K.W. 1998. Mining audit data to build intrusion detection models. *In: Proc. Int. Conf on Knowledge Discovery and Data mining* (KDD-98). P. 66–72.

Moon, B., Jagadish, H.V., Faloutsos, C. and Saltz, J.H. 2001. Analysis of the clustering properties of hilbert space-filling curve. IEEE Trans. on Knowledge and Data Engineering (IEEE-TKDE). 13: 124-141.

Ramaswamy, S., Rastogi, R. and Shim, K. 2000. Efficient algorithms for mining outliers from large data sets. *In: Proc. ACM Int. Conf. on Managment of Data* (SIGMOD'00). P. 427–438.

Yu, D., Sheikholeslami S. and Zhang, A. 1999. Findout: Finding outliers in very large datasets. *In: Tech. Report*, 99-03, Univ. of New York, Buffalo. P. 1–19.