The VLDB Journal (2005) 14: 211–221 / Digital Object Identifier (DOI) 10.1007/s00778-004-0125-5

An effective and efficient algorithm for high-dimensional outlier detection

Charu C. Aggarwal, Philip S. Yu

IBM T.J. Watson Research Center, 19 Skyline Drive, Hawthorne, NY 10532, USA

Edited by R. Ng. Received: November 19, 2002 / Accepted: February 6, 2004
Published online: August 19, 2004 –

Abstract. The outlier detection problem has important applications in the field of fraud detection, network robustness analysis, and intrusion detection. Most such applications are most important for high-dimensional domains in which the data can contain hundreds of dimensions. Many recent algorithms have been proposed for outlier detection that use several concepts of proximity in order to find the outliers based on their relationship to the other points in the data. However, in high-dimensional space, the data are sparse and concepts using the notion of proximity fail to retain their effectiveness. In fact, the sparsity of high-dimensional data can be understood in a different way so as to imply that every point is an equally good outlier from the perspective of distance-based definitions. Consequently, for high-dimensional data, the notion of finding meaningful outliers becomes substantially more complex and nonobvious. In this paper, we discuss new techniques for outlier detection that find the outliers by studying the behavior of projections from the data set.

Keywords: Data mining – High-dimensional spaces – Outlier detection

1 Introduction

An outlier is defined as a data point that is very different from the rest of the data based on some measure. Such a data point often contains useful information on abnormal behavior in the system that is characterized by the data. The outlier detection technique finds applications in credit card fraud, network intrusion detection, financial applications, and marketing. This problem typically arises in the context of very high-dimensional data sets. Much of the recent work on finding outliers use methods that make implicit assumptions of relatively low dimensionality of the data. These methods work quite poorly when the dimensionality is high and the data become sparse,

Many data-mining algorithms in the literature find outliers as a by-product of clustering algorithms [2,3,6,16,24]. However, these techniques define outliers as points that do not lie in clusters. Thus, the techniques implicitly define outliers as the background noise in which the clusters are embedded. Starting with the work in [8], recent literature [10,20,21,23] defines outliers as points that are neither a part of a cluster nor a part of the background noise; rather they are specifically points that behave very differently from the norm. Outliers are more useful based on their value for determining behavior that deviates significantly from average behavior. In many applications (e.g., network intrusion detection), such records may provide guidance in discovering important anamolies in the data. Such points are also referred to as *strong outliers* in the work discussed in [21]. In this paper, we will develop algorithms that generate only outliers that are based on their deviation value.

Many algorithms have been proposed in recent years for outlier detection [10,20,21,23], but they are mostly either distance based or density based; these are generally not methods specifically designed to deal with the curse of high dimensionality. Two interesting distance-based algorithms are discussed in [20,23], which define outliers by using the distribution of (full-dimensional) distances of the other points to a given point. This kind of measure is naturally susceptible to the dimensionality curse. For example, consider the definition by Knorr and Ng [20]: *A point p in a data set is an outlier with respect to the parameters k and λ, if no more than k points in the data set are at a distance λ or less from p.*

As pointed out in [23], this method is sensitive to the use of the parameter λ that is hard to figure out a priori. In addition, when the dimensionality increases, it becomes increasingly difficult to pick λ since most of the points are likely to lie in a thin shell about any point [9]. Thus, if we pick too small a λ, then all points are outliers; whereas if we pick too large a λ, then no point is an outlier. This means that a user would need to pick λ to a very high degree of accuracy in order to find a modest number of points that can then be defined as outliers. Aside from this, the data in real applications are very noisy, and the abnormal deviations may be embedded in some lower-dimensional subspace that cannot be determined by the spreading out behavior [9] of the data in full dimensionality. The algorithm also does not scale well for high dimensions. Consequently, the work in [23] discusses the following defi-

Correspondence to: C.C. Aggarwal (e-mail: charu@us.ibm.com)

nition for an outlier: *Given a k and n, a point p is an outlier if the distance to its k-th nearest neighbor of the point is smaller than the corresponding value for no more than $n-1$ other points.*

Although the definition in [23] has some advantages over that provided in [20], it is again not specifically designed to work for high-dimensional problems. In fact, it has been indicated in [23] that by using fewer features for a given run, more interesting outliers on the NBA98 basketball statistics database were obtained. This was again because the data often got spread out uniformly with increasing dimensionality. Another interesting recent technique finds outliers based on their local density [10], particularly with respect to the densities of local neighborhoods. This technique has some advantages in accounting for local levels of skews and abnormalities in data collections. To compute the outlier factor of a point, the method in [10] computes the local reachability density of a point o by using the average smoothed distances to a certain number of points in the locality of o. Unfortunately, this is again a problem in high dimensionality, where the concept of locality becomes difficult to define because of the sparsity of the data. In order to use the concept of local density, we need a meaningful concept of distance for sparse high-dimensional data; if this does not exist, then the outliers found are unlikely to be very useful.

Thus the techniques proposed in [10,20,23] try to define outliers based on the distances in full-dimensional space in one way or another. The sparsity of the data in high dimensionality [9] can be interpreted slightly differently to infer that each point is as good as an outlier in high-dimensional space. This is because if all pairs of points are almost equidistant [9], then meaningful clusters cannot be found in the data [2,3,6,11]; similarly, it is difficult to detect abnormal deviations.

For problems such as clustering and similarity search, it has been shown [1–4,6,11,17] that by examining the behavior of the data in subspaces, it is possible to design more meaningful clusters that are specific to the particular subspace in question. This is because different localities of the data are dense with respect to different subsets of attributes. By defining clusters that are specific to particular projections of the data, it is possible to design more effective techniques for finding clusters. The same insight is true for outliers, because in typical applications such as credit card fraud, only the subset of the attributes actually affected by the abnormality of the activity is likely to be useful in detecting the behavior.

In order to explain our point a little bit better, let us consider the example illustrated in Fig. 1. In the above example, we have shown several two-dimensional cross sections of a very high-dimensional data set. It is quite likely that for high-dimensional data, many of the cross sections may be structured, whereas others may be more noisy. For example, the points A and B show abnormal behavior in views 1 and 4 of the data. In other views, the points show average behavior. In the context of a credit card fraud application, both the points A and B may correspond to different kinds of fraudulent behavior yet may show average behavior when distances are measured in all the dimensions. Thus, by using full-dimensional distance measures, it would be more difficult to detect points that are outliers because of the averaging behavior of the noisy and irrelevant dimensions. Furthermore, it is impossible to prune off specific features a priori since different points (such as A and B) may show different kinds of abnormal patterns, each of which uses different features or views.

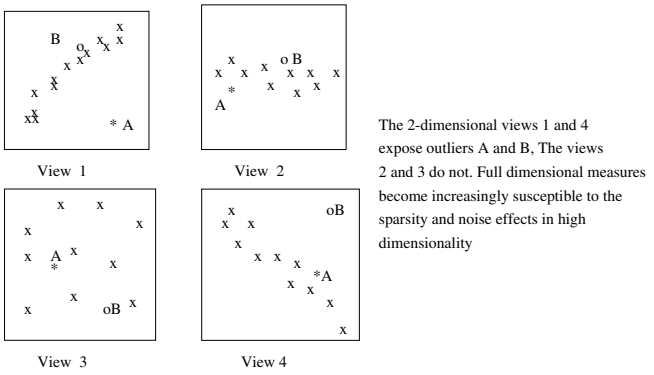

Fig. 1. Illustrations of outliers in various views of the data

Thus the problem of outlier detection becomes increasingly difficult for very high-dimensional data sets, just as any of the other problems in the literature such as clustering, indexing, classification, or similarity search. Previous work on outlier detection has not focused on the high-dimensionality aspect of outlier detection and has used methods that are more applicable for low-dimensional problems by using relatively straightforward proximity measures [10,20,23]. On the other hand, we note that most practical data-mining applications are likely to arise in the context of a very large number of features. In this paper, we focus on the effects of high dimensionality on the problem of outlier detection. Recent work has discussed some of the concepts of defining the intensional knowledge that characterizes distance-based outliers in terms of subsets of attributes. Unfortunately, this technique was not intended for high-dimensional data, and the complexity increases exponentially with dimensionality. As the empirical results in [21] show, even for the relatively small dimensionality of 4, the technique is highly computation intensive. For even slightly higher dimensionalities, the technique is infeasible from a computational standpoint.

In this paper, we discuss a new technique for outlier detection that finds outliers by observing the density distributions of projections from the data. Intuitively speaking, this new definition considers a point to be an outlier if in some lower-dimensional projection it is present in a local region of abnormally low density.

1.1 Defining outliers in lower-dimensional projections

The idea is to define outliers for data by looking at those projections of the data that have abnormally low density. Thus our first step is to identify and mine those patterns that have abnormally low presence that cannot be justified by randomness. This is important since we value outlier patterns not for their noise value but for their deviation value. Once such patterns have been identified, then the outliers are defined as those records that have such patterns present in them. An interesting observation is that such lower-dimensional projections can be mined even in data sets that have missing attribute values [22]. This is quite useful for many real applications in which feature extraction is a difficult process and full feature descriptions often do not exist. In such cases, the only difference is that for

a given projection, only those points need to be used that are fully specified in that projection.

1.2 Defining abnormal lower-dimensional projections

In order to find such abnormal lower-dimensional projections, we need to define and characterize what we mean by an abnormal lower-dimensional projection. An abnormal lower-dimensional projection is one in which the density of the data is exceptionally lower than average.

In order to find such projections, we first perform a grid discretization of the data. Each attribute of the data is divided into ϕ ranges. These ranges are created on an equidepth basis; thus each range contains a fraction $f = 1/\phi$ of the records. The reason for using equidepth ranges as opposed to equiwidth ranges is that different localities of the data have different densities; we would like to find outliers while normalizing for this factor. These ranges form the units of locality that we will use to define low-dimensional projections that have unreasonably sparse regions.

Let us consider a k-dimensional cube that is created by picking grid ranges from k different dimensions. The expected fraction of the records in that region if the attributes were statistically independent would be equal to f^k. Of course, the data are far from statistically independent; therefore, the actual distribution of points in a cube would differ significantly from average behavior; it is precisely those deviations that are abnormally below the average that are useful for the purpose of outlier detection.

Let us assume that there are a total of N points in the database, and the dimensionality is d. If the data were uniformly distributed, then the presence or absence of any point in a k-dimensional cube is a bernoulli random variable with probability f^k. Then the expected fraction and standard deviation of the points in a a k-dimensional cube is given by $N \cdot f^k$ and $\sqrt{N \cdot f^k \cdot (1 - f^k)}$. Also, under the assumption of uniformly distributed data, the number of points in a cube can be approximated by a normal distribution. Let $n(\mathcal{D})$ be the number of points in a k-dimensional cube $\mathcal{D}$. Then we calculate the sparsity coefficient $S(\mathcal{D})$ of the cube $\mathcal{D}$ as follows:

$$S(\mathcal{D}) = \frac{n(D) - N \cdot f^k}{\sqrt{N \cdot f^k \cdot (1 - f^k)}} . \quad (1)$$

Only sparsity coefficients that are negative indicate cubes in which the presence of the points is significantly lower than expected. Note that if $n(\mathcal{D})$ is assumed to fit a normal distribution, then the normal distribution tables can be used to quantify the probabilistic level of significance for a point to deviate significantly from average behavior for an a priori assumption of uniformly distributed data. In general, the uniformly distributed assumption is not true; however, the sparsity coefficient provides a reasonable approximation to the level of significance for a given projection. We also note that we are only searching for cubes that are nonempty in order to find outliers. Therefore, cubes that are empty are considered infeasible. From an implementation perspective, this is achieved by setting such sparsity coefficients to the very high value of 10^6.

1.3 A note on the nature of the problem

At this stage we would like to make a comment on the nature of the problem of finding the most sparse k-dimensional cubes in the data. The nature of this problem is such that there are no upward- or downward-closed properties in the set of dimensions (along with associated ranges) that are unusually sparse.[1] This is not unexpected: unlike problems such as large itemset detection [7] where one is looking for large aggregate patterns, the problem of finding subsets of dimensions that are sparsely populated is akin to finding a needle in a haystack since one is looking for patterns that rarely exist. Furthermore, it may often be the case that even though particular regions may be well populated on certain sets of dimensions, they may be very sparsely populated when such dimensions are combined together. (For example, there may be a large number of people below the age of 20, and a large number of people with diabetes, but very few with both.) From the perspective of an outlier detection technique a person below the age of 20 with diabetes is a very interesting record; however, it is very difficult to find such a pattern using structured search methods. Therefore, the best projections are often created by an a priori unknown combination of dimensions, which cannot be determined by looking at any lower-dimensional projection. One solution is to change the measure in order to force better closure or pruning properties; however, this can worsen the quality of the solution substantially by forcing the choice of the measure to be driven by algorithmic considerations. In general, it is not possible to predict the behavior of the data when two sets of dimensions are combined; therefore, the best qualitative option is to develop search methods that can identify such hidden combinations of dimensions. In order to search the exponentially increasing space of possible projections, we borrow ideas from a class of evolutionary search methods in order to create an efficient and effective algorithm for the outlier detection problem.

2 Evolutionary algorithms for outlier detection

In this section, we will discuss the algorithms that are useful for outlier detection in high-dimensional problems. A natural class of methods for outlier detection are the naive brute-force techniques in which all subsets of dimensions are examined for possible patterns that are sparse. These patterns are then used to determine the points that are possibly outliers. We discuss two algorithms for outlier detection: a naive brute-force algorithm that is very slow at finding the best patterns because of its exhaustive search of the entire space and a much faster evolutionary algorithm that is able to quickly find hidden combinations of dimensions in which the data are sparse. We assume that one of the inputs to the algorithm is the dimensionality k of the projections that are used to determine the outliers. Aside from this, the algorithm uses the number m of projections to be determined as an input parameter.

The brute-force algorithm is illustrated in Fig. 2. The algorithm works by examining all possible sets of k-

[1] An upward-closed pattern is one in which all supersets of the pattern are also valid patterns. A downward-closed set of patterns is one in which all subsets of the pattern are also members of the set.

Algorithm BruteForce(Number: m, Dimensionality: k)
begin
  $R_1 = Q_1 =$ Set of all $d \cdot \phi$ ranges;
  for $i = 2$ **to** $k + 1$ **do**
    begin
    $R_i = R_{i-1} \oplus Q_1$;
    end;
  Determine sparsity coefficients of all
    elements in R_k;
  $R_k =$ Set of m elements in Q_k
     with most negative sparsity coefficients;
  $\mathcal{O} =$ Set of points covered by R_k;
  return(R_k, $\mathcal{O}$);
end

Fig. 2. The brute-force technique

dimensional candidate projections (together with corresponding grid ranges) and retaining the m projections that have the most negative sparsity coefficients. In order to actually determine the candidate projections, the method uses a bottom-up recursive algorithm in which $(k + 1)$ candidate cubes are determined by concatenating the candidate k projections with all $d \cdot \phi$ possible sets of one-dimensional projections and their grid ranges (denoted by Q_1). The concatenation operation is illustrated in Fig. 2 by $\oplus$. Note that, for a given cube, it only makes sense to concatenate with grid ranges from dimensions not included in the current projection in order to create a higher-dimensional projection. The candidate set of dimensionality i is denoted by R_i. At termination, the set of projections with most negative sparsity coefficients in R_k are retained. The set of points in the data that contain the corresponding ranges for the projections is reported as the final set of outliers.

As we shall see in later sections, the algorithm discussed in Fig. 2 is computationally untenable for problems of even modest complexity. This is because of the exponentially increasing search space of the outlier detection problem. To overcome this, we will illustrate an innovative use of evolutionary search techniques for the outlier detection problem.

2.1 An overview of evolutionary search techniques

Evolutionary algorithms [18] are methods that imitate the process of organic evolution [12] in order to solve parameter optimization problems. The fundamental idea underlying the Darwinian view of evolution is that, in nature resources are scarce, and this leads to a competition among the various species. As a result, all the species undergo a selection mechanism in which only the fittest survive. Consequently, the fitter individuals tend to mate each other more often, resulting in still better individuals. At the same time, once in a while, nature also throws in a variant by the process of mutation, so as to ensure a sufficient amount of diversity among the species, and hence also a greater scope for improvement. The basic idea behind an evolutionary search technique is similar; every solution to an optimization problem can be "disguised" as an individual in an evolutionary system. The measure of fitness of this "individual" is equal to the objective function value of the corresponding solution, and the other species that this individual has to compete with are a group of other solutions to the problems; thus, unlike other optimization methods such as hill climbing or simulated annealing [19], they work with an entire population of current solutions rather than a single solution. This is one of the reasons why evolutionary algorithms are more effective as search methods than either hill-climbing, random search, or simulated annealing techniques; they use the essence of the techniques of all these methods in conjunction with recombination of multiple solutions in a population. Appropriate operations are defined in order to imitate the recombination and mutation processes as well, and the simulation is complete.

Each feasible solution to the problem being solved by an evolutionary technique is defined as an **individual**. This feasible solution is in the form of a string and is the genetic representation of the individual. The process of conversion of feasible solutions of the problem into strings that the algorithm can use is called **coding**. For example, a possible coding for a feasible solution to the traveling salesman problem could be a string containing a sequence of numbers representing the order in which he visits the cities. The genetic material at each locus on the string is referred to as a **gene** and the possible values that it could possibly take on are the **alleles**. The measure of fitness of an individual is evaluated by the **fitness function**, which has as its argument the string representation of the individual and returns a value indicating its fitness. The fitness value of an individual is analogous to the objective function value of the solution; the better the objective function value, the better the fitness value.

As the process of evolution progresses, all the individuals in the population tend to become more and more similar to each other genetically. This phenomenon is referred to as **convergence**. Dejong [13] defined convergence of a $gene$ as the stage at which 95% of the population has the same value for that gene. The population is said to have converged when all genes have converged.

The application of such evolutionary search procedures should be based on a good understanding of the problem at hand. Typically black-box GA software on straightforward string encodings does not work very well [5], and it is often a nontrivial task to design the proper search methods such as recombinations, selections, and mutations that work well for a given problem. In the next section, we will discuss the details of the evolutionary search procedures that work effectively for the outlier detection problem.

2.2 A description of the outlier detection technique

In this section, we will discuss the application of the search technique to the outlier detection problem. Let us assume that the grid range for the i-th dimension is denoted by m_i. Then the value of m_i can take on any of the values 1 through ϕ, or it can take on the value $*$, which denotes a "don't care". Thus, there are a total of $\phi + 1$ values that the dimension m_i can take on. Thus, consider a four-dimensional problem with $\phi = 10$. Then, one possible example of a solution to the problem is given by *3*9. In this case, the ranges for the second and fourth dimensions are identified, whereas the first and third are left as "don't cares". The fitness for the corresponding solution may be computed using the sparsity coefficient discussed

Algorithm EvolutionaryOutlierSearch(Number: m, Dimensionality: k)
begin
$S =$ Initial seed population of p strings;
$BestSet = null$;
while not(*termination_criterion*) **do begin**
  $S =$Selection(S);
  $S =$CrossOver(S);
  $S =$Mutation(S, p_1, p_2);
  Update $BestSet$ to be the m solutions in
    $BestSet \cup S$ with most negative sparsity coefficients;
  end;
  $\mathcal{O} =$ Set of data points covered by $BestSet$;
  return($BestSet$, $\mathcal{O}$);
end

Fig. 3. Evolutionary framework for outlier detection

Algorithm Selection(S)
begin
  Compute the sparsity coefficient of each solution
    in the population S;
  Let $r(i)$ be the rank of solution i in order
    of sparsity coefficient (most negative occurs first);
  $S' = null$;
  for $i = 1$ to p **do**
    begin
    Roll a die with the i-th side proportional to $p - r(i)$;
    Add the solution corresponding to the i-th side to S';
    end;
  Replace S by S';
  return(S);
end

Fig. 4. Selection criterion for genetic algorithm

earlier. The evolutionary search technique starts with a population of p random solutions and iteratively used the processes of selection, crossover, and mutation to perform a combination of hill climbing, solution recombination, and random search over the space of possible projections. The process was continued until the population converged to a global optimum. We used the De Jong [13] convergence criterion to determine the termination condition. At each stage of the algorithm, the m best projection solutions (most negative sparsity coefficients) were kept track of. At the end of the algorithm, these solutions were reported as the best projections in the data. The overall procedure for the genetic algorithm is illustrated in Fig. 3. The population of solutions in any given iteration is denoted by S. This set S is refined in subsequent iterations of the algorithm, and the best set of projections found so far is always maintained by the evolutionary algorithm.

- **Selection:** Several alternatives are possible [15] for selection in an evolutionary algorithm; the most popularly known ones are rank selection and fitness proportional selection. The idea is to replicate copies of a solution by ordering them by rank and biasing the population in favor of the higher-ranked solutions. This is called rank selection and is often more stable than straightforward fitness proportional methods that sample the set of solutions in proportion to the actual value of the objective function. This strategy of biasing the population in favor of fitter strings in conjunction with effective solution recombination creates newer sets of children strings that are more likely to be fit. This results in a global hill climbing of an entire population of solutions. For the particular case of our implementation, we used a roulette wheel mechanism, where the probability of sampling a string from the population was proportional to $p - r(i)$, where p is the total number of strings and $r(i)$ is the rank of the i-th string. Note that the strings are ordered in such a way that the strings with the most negative sparsity coefficients occur first. Thus, the selection mechanism ensures that the new population is biased in such a way that the the most abnormally sparse solutions are likely to have a greater number of copies. The overall selection algorithm is illustrated in Fig. 4.
- **Crossover:** Since the crossover technique is a key method in evolutionary algorithms for finding optimum combinations of solutions, it is important to implement this operation effectively for making the overall method work effectively. We will first discuss the natural two-point crossover mechanism used in evolutionary algorithms and show how to suitably modify it for the outlier detection problem.

  Unbiased two-point crossover: The standard procedure in evolutionary algorithms is to use uniform two-point crossover in order to create the recombinant children strings. The two-point crossover mechanism works by determining a point in the string at random called the crossover point and exchanging the segments to the right of this point. For example, consider the strings 3*2*1 and 1*33*. If the crossover is performed after the third position, then the two resulting strings are 3*23* and 1*3*1. Note that in this case, both the parent and children strings correspond to three-dimensional projections in five-dimensional data. However, if the crossover occurred after the fourth position, then the two resulting children strings would be 3*231 and 1*3**. These correspond to two-dimensional and two-dimensional projections. In general, since the evolutionary algorithm only finds projections of a given dimensionality in a run, this kind of crossover mechanism often creates infeasible solutions after the crossover process. Such solutions are discarded in subsequent iterations since they are assigned very low fitness values. In general, evolutionary algorithms work very poorly when the recombination process cannot create sets of solutions of high quality or that are viable in terms of feasibility. To address this, we create an optimized crossover process that takes both these factors into account.

  Since it is clear that the dimensionality of the projection needs to be kept in mind while performing a crossover operation, it is desirable that the two children obtained after solution recombination also correspond to a k-dimensional projection. In order to achieve this goal, we need to classify the different positions in the string into three types. This classification is specific to a given pair of strings s_1 and s_2.

  Type I: Both strings have a "don't care".

  Type II: Neither has a "don't care". Let us assume that there are $k' \leq k$ positions of this type.

Type III: One has a "don't care", since each string has exactly $k - k'$ such positions in each string, and these positions are disjoint. Thus, there are a total of $2 \cdot (k - k')$ such positions.

For example, consider the strings 2*3* and 4**5. Then, the second position is a Type I position, the third and fourth positions are Type II positions, and the first is a Type III position. The crossover is designed differently for each segment of the string. The technique is obvious for the Type I segment, where both strings have a "don't care". In this case, both offspring strings have a * in a position.

To perform the crossover of the Type II and Type III positions on the children strings, we apply the optimized crossover mechanism:

Optimized crossover: The optimized crossover [5] technique is a useful method for finding the best combinations of the features present in the two solutions. The idea is to create at least one child string from the two parent strings that is a fitter solution recombination than either parent. The nature of the children strings is biased in such a way that at least one of the two strings is likely to be an effective solution recombination of the parent strings. An ideal goal would be to find the best possible recombination from the two parents; however, this is difficult to achieve since there are a total of $2^{k'} \cdot \binom{(2k-2k')}{(k-k')}$ possibilities for the children. To implement the crossover operation effectively, we make the observation that k' is typically quite small when we are looking for low-dimensional projections of high-dimensional data. Therefore, we first search the space of the $2^{k'}$ possibilities for the Type II positions for the best possible combination. After having found the optimal combination for the Type II positions, we use a greedy algorithm to find a solution recombinant for the $(k - k')$ Type III positions. To find the remaining positions, we always extend the string with the position that results in the string with the most negative sparsity coefficient. We keep extending the string for an extra $(k - k')$ positions until all k positions have been set. This string s is a recombinant of the parent strings. The idea of using such a recombination procedure is to create a new solution that combines the good aspects of both parent strings. The crossover technique is a key method in evolutionary algorithms that distinguishes it from hill climbing methods; by devising new combinations of solutions it is possible to create a new string in the search space that combines aspects from both parents. It now remains to create the second child s' in order to replace both parent strings. The second child is created by always picking the positions from a different parent than the one from which the string s derives its positions. The overall crossover algorithm is illustrated in Fig. 5.

- **Mutation:** We perform mutations of two types:

  Type I: Let Q be the set of positions in the string that are $*$. Then we pick a position in the string that is not in Q and change it to $*$. At the same time, we change a randomly picked position in Q to a number between 1 and ϕ. Thus, the total dimensionality of the projection represented by a string remains unchanged by the process of mutation.

  Type II: This kind of mutation only affects a position that is not $*$. The value of such a position is changed from a value between 1 and ϕ to another value between 1 and ϕ. For this purpose, we have two sets of mutation probabilities p_1 and p_2. With a mutation probability of p_1, we perform an interchange of Type I. The corresponding probability to perform an interchange of Type II is p_2. For the purpose of our implementation, we used an equal number of Type I and Type II mutations; therefore we have $p_1 = p_2$. The overall mutation algorithm is illustrated in Fig. 6.

Algorithm Crossover(S)
begin
  Match the solutions in the population pairwise;
  for each pair of solutions s_1, s_2 thus matched **do**
    begin
    $(s, s') =$ Recombine(s_1, s_2);
    Replace s_1 and s_2 in the population by s and s';
    end;
  return(S);
end

Algorithm Recombine(s_1, s_2)
begin
  $Q =$ Set of positions in which either s_1 or s_2 is *;
  $R =$ Set of positions in which neither s_1 nor s_2 is *;
  Enumerate the $2^{|R|}$ possibilities for recombining the
    positions in R and pick the string s with most negative
      sparsity coefficient;
  Extend string s greedily from the positions in Q by always
    picking the position with the most negative sparsity coefficient;
  Let s' be the complementary string to s;
  { A complementary string is defined as one in which a
    given position in s' is always derived from a different
    parent than the one s derives it from; }
  return(s, s');
end

Fig. 5. The crossover algorithm

2.3 Postprocessing phase

We note that the genetic algorithm finds the regions and projections with the most negative sparsity coefficients as opposed to the data points themselves. Therefore, a postprocessing phase is required to map these projections into the data points. As indicated earlier, a point is said to cover a projection when it is drawn from the local projection found by the genetic algorithm. (For example, a point covers the projection *3*6 if the second and fourth coordinates correspond to grid ranges 3 and 6.) In the postprocessing phase, we find all the sets of data points that contain the the abnormal projections reported by the algorithm. These points are the outliers and are denoted by $\mathcal{O}$ in the description of Fig. 3.

2.4 Choice of projection parameters

An important issue in the algorithm is to be able to choose the projection parameters k and ϕ. Note that one of the reasons that we are finding outliers by the projections-based method is the sparsity of the data. Thus, for a k-dimensional projection

```
Algorithm Mutation(S, p1, p2)
begin
  for each string s ∈ S do begin
    Let Q be the set of positions in s that are *;
    Flip a coin with success probability p1;
    if the flip is a success, then begin
        convert a random position in Q to a random number
          between 1 and φ;
        convert a random position not in Q to *;
    end
    Define R to be the set of positions in s that are not *;
    Flip a coin with success probability p2;
    if the flip is a success then begin
        Pick a position in R and flip it to a random number
          between 1 and φ;
  end;
  return(S);
end
```

Fig. 6. The mutation algorithm

out of a d-dimensional data set, each subcube represented by a k-dimensional projection contains an expected fraction $1/\phi^k$ of the data. Thus, if we pick $\phi = 10$, then even for a four-dimensional projection the expected number of points in the subcube would only be a fraction 10^{-4} of the whole. Thus, if the data set contains less than 10,000 points, the k-dimensional cubes are expected to contain less than one point. This means that it is not possible to find a cube that has a high sparsity coefficient and covers at least one point. In general, the values of ϕ and k should be picked small enough that the sparsity coefficient of cube containing exactly one point is reasonably negative. At the same time ϕ should be picked high enough that there are sufficient numbers of intervals on each dimension that corresponds to a reasonable notion of locality. Once ϕ has been picked, we determine k by using the following method. We calculate the sparsity coefficient of an empty cube. From Eq. 1, this is given by $-\sqrt{\frac{N}{\phi^k-1}}$. The value of N is predecided by the choice of the data set. Now, it remains to pick k appropriately so that it results in a high enough sparsity coefficient. If the data set were uniformly distributed, then the distribution of data points in each cube could be represented by a normal distribution, and the above sparsity coefficient would be the number of standard deviations by which the actual number of points differed from the expected number of points. For such a case, a choice of sparsity coefficient of -3 would result in 99.9% level of significance that the given data cube contains less points than expected and is hence an abnormally sparse projection. In general, the normal distribution assumption is not true; however, a value of $s = -3$ is a good reference point to decide the value of k. Therefore, we have:

$$\sqrt{\frac{N}{\phi^k - 1}} = -s\,. \tag{2}$$

By expressing the entire equation in terms of k, we obtain $k^* = \lfloor \log_\phi(N/s^2 + 1)\rfloor$. For a real application, a user may wish to test different values of this (intuitively interpretable) parameter s to determine appropriate values of $k = k^*$. The value of $k = k^*$ thus returned is the largest value of k at which abnormally sparse projections may be found before the effects of high dimensionality result in sparse projections by default. The value of $k = k^*$ is also the most informative for the purpose of outlier detection since it is the highest-dimensional embedded space in which useful outliers may be found. We also note that since the integral dimensionality of the space in which outliers are found is lower than what is indicated in Eq. 2, the corresponding sparsity coefficients are also sometimes more negative than the threshold chosen by the user.

We note that since an evolutionary approach was used for outlier detection, the running time of the method is proportional to the size of the data. The bottleneck operation is in the computation of the fitness function. In each iteration, the number of fitness functions to be computed is proportional to data set size. For this reason, the scalability of the algorithm is also proportional to the data set size. At the same time, since the evolutionary algorithm is essentially a generic search heuristic over the data space, it is not possible to propose a reasonably tight estimation of the complexity behavior in closed form.

3 Empirical results

The algorithm was implemented on a 233-MHz machine running AIX 4.1.1 with 100 MB of main memory. We tested the outlier detection on several real data sets obtained from the UCI machine learning repository. These are data sets that are naturally designed for classification and machine learning applications. The data sets were picked in a way so as to result in considerable variability in terms of the the number of attributes. In addition, the data sets were cleaned in order to take care of categorical and missing attributes.

We tested the performance of the method using both the brute-force and the evolutionary technique. As expected, the brute-force technique required considerably more computational resources than the evolutionary search technique for high-dimensional data sets. For one of the high-dimensional data sets (musk data set), the brute-force algorithm was unable to terminate in a reasonable amount of time because of the high dimensionality of the problem. For example, in order to find k-dimensional projections of a d-dimensional problem, there are a total of $\binom{d}{k} \cdot \phi^k$ possibilities. Even for a modestly complex problem with $d = 20, k = 4, \phi = 10$, this results in $7 * 10^7$ possibilities. In the case of the musk data set (which has 160 dimensions), the brute-force algorithm was unable to find even three-dimensional projections. Clearly, as the dimensionality increases, the computational complexity of the problem becomes untenable. The goal of the evolutionary algorithm is to provide outliers that are reasonably comparable with the brute-force algorithm but can be found much more efficiently.

In Table 1, we have illustrated the results for five data sets from the UCI machine learning repository. For all cases, we picked $\phi = 7$ for the different data sets. The values of k was then determined using the relationship $k = \lfloor \log_\phi(N/s^2 + 1)\rfloor$. In each case we found the $m = 20$ best projections and reported the outlier points corresponding to these projections. The value of s picked for all cases was -3. It is evident from these results that the performance of the brute-force technique quickly becomes untenable for very high-dimensional data sets. In fact, for the musk data set with 160 dimensions, the outlier detection algorithm did not terminate in a reasonable

Table 1. Performance for different data sets

Data set	Brute (time)	Gen (time)	Gen^o (time)	Brute (quality)	Gen (quality)	Gen^o (quality)
Breast cancer (14)	1314	35	42	−3.57	−3.07	−3.54
Ionosphere (34)	13115	301	267	−3.12	−2.12	−3.12 (*)
Segmentation (19)	2112	71	43	−3.11	−2.76	−3.11 (*)
Musk (160)	—	954	721	—	−2.07	−2.81
Machine (8)	11	31	12	−3.31	−3.15	−3.31 (*)

amount of time; therefore, we have been unable to report the results for this particular case. This is an important observation since the utility of this technique is primarily for the high-dimensional case.

As discussed earlier, we implemented two crossover mechanisms. The first was a simple two-point crossover mechanism that performs the crossover by exchanging segments of the two strings. We implemented a optimized crossover mechanism that finds good recombinations of solutions in the search space. The results with the optimized mechanism have been superscripted with an ^{o}. Clearly, the optimized mechanism performs substantially better in terms of the quality of the final solution found. This is because the two-point crossover mechanism often resulted in strings that were not in the feasible search space of k-dimensional projections. On the other hand, the optimized crossover solution identified combinations of dimensions that were both feasible and of high quality. We have also reported the average sparsity coefficients of the best 20 (nonempty) projections indicated under the column (quality). In three of the five data sets, the average quality of the best 20 best projections was the same using either the evolutionary or the brute-force algorithm. We have marked these cases with a "*". We note that the brute-force method provides the optimum solution in terms of the sparsity coefficient. However, in most cases, the evolutionary algorithm is almost equally good in finding solutions of reasonable quality. Another interesting observation was that the optimized mechanism was significantly faster than the two-point crossover mechanism for many data sets. Indeed effective solution recombination that is tailored to each specific problem is important in providing high solution quality in a reasonable amount of running time. The results show that the evolutionary algorithm works qualitatively quite well for most of the data sets. The relatively small level of qualitative sacrifice by the evolutionary algorithm method is offset by the huge performance gain over the brute-force method.

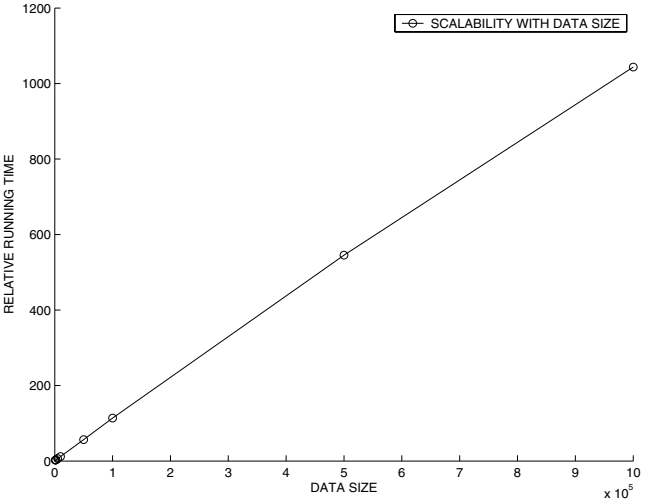

Fig. 7. Scaling of running time with database size (Syn. 1)

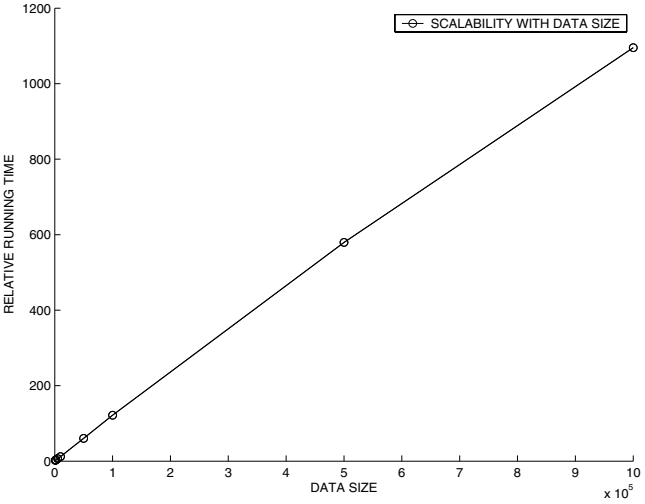

Fig. 8. Scaling of running time with database size (Syn. 2)

3.1 Results with synthetic data sets

We also tested the algorithm with a couple of synthetic data sets. We generated the projected clusters as discussed in detail in [3]. As in [3], we label these data sets synthetic 1 and synthetic 2 respectively. In addition, we superimposed 0.1% additional points onto the data set as outliers. We note that these kinds of outliers are much weaker since they represent the background noise inserted into the data as opposed to actual deviations from normal behavior. Therefore, such outliers are often more difficult to detect in practice. We computed the percentage of outliers that were detected by the algorithm when running this procedure. We also computed the additional percentage of points that had been falsely classified as outliers by the algorithm. The results are illustrated in Table 2. We note that the second percentage (false positives) illustrated in the table is in terms of the number of outliers found by the algorithm. An interesting observation is that, while most of the true outliers were found by the algorithm, the percentage of points that had been chosen falsely as outliers was somewhat higher.

We also tested the scalability of the algorithm to increasing database size and dimensionality. For testing scalability with data set size, we generated synthetic data sets with the

Table 2. Effectiveness with synthetic data

Data set	Percent outliers found	False positive percent
Synthetic 1	98.9%	9.5%
Synthetic 2	99.3%	8.9%

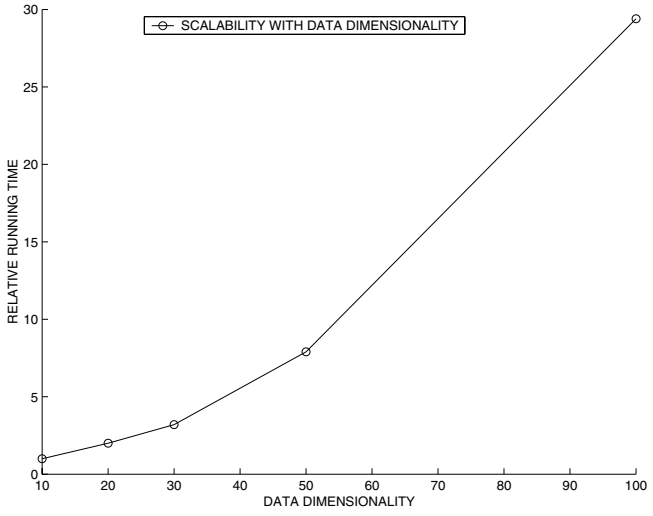

Fig. 9. Scaling of running time with database dimensionality (Syn. 1)

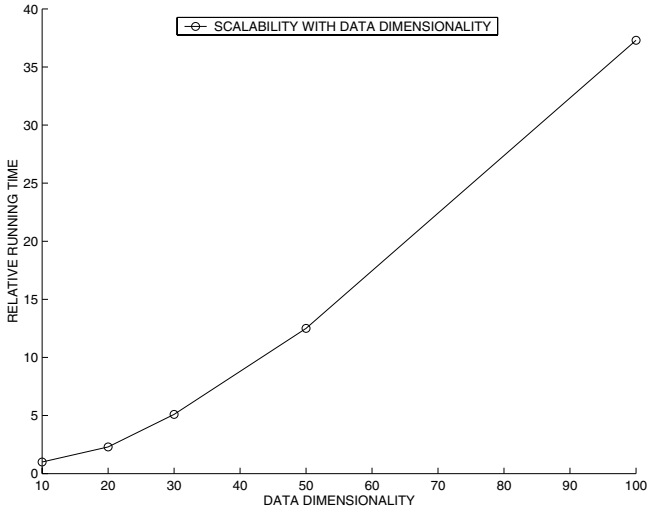

Fig. 10. Scaling of running time with database dimensionality (Syn. 2)

same parameters as the above two data sets discussed except for the data set size parameter that was varied. A similar process was used in order to test scalability with increasing data dimensionality. As illustrated in Figs. 7 and 8, the algorithm showed linear scalability with data set size. In Figs. 9 and 10, we have illustrated the scalability with increasing data dimensionality for the Syn. 1 and Syn. 2 data sets. While the search space with increasing data dimensionality increases exponentially, it is clear from the above charts that the computational complexity increases only slightly worse than linearly. This is because of the fact that the evolutionary algorithm prunes away most of the search space during the exploration process.

Table 3. Arrythmia data set

Case	Class codes	Percent
Commonly occurring Classes ($\geq 5\%$)	01, 02, 06, 10, 16	85.4%
Rare classes ($< 5\%$)	03, 04, 05, 07 08, 09, 14, 15	14.6%

3.2 An intuitive evaluation of results

A qualitative evaluation of the outlier detection algorithm provides challenges because of the subjectivity in defining abnormal behavior. An interesting way to test for qualitative behavior was to look at the actual points found by the outlier detection algorithm and the reason that these points were picked as outliers. One of the interesting data sets in the UCI machine learning repository is the arrythmia data set, which has 279 attributes corresponding to different measurements of physical and heart-beat characteristics that are used to diagnose arrythmia. The data set contains a total of 13 (nonempty) classes. Class 1 was the largest and corresponds to people who do not have any kind of heart disease. The remaining classes correspond to people with diseases of one form or another, some less common than others. For example, class 2 corresponds to isochemic changes in the coronary artery, a relatively common condition. We considered those kinds of class labels that occurred in less than 5% of the data set as rare labels. One way to test how well the outlier detection algorithm worked was to run the method on the data set and test the percentage of points that belonged to one of the rare classes. If the outlier detection works well, we expect such abnormal classes to be overrepresented in the set of points found. These kinds of classes are also interesting from a practical perspective.

We ran the evolutionary algorithm to find all the sparse projections in the data set that correspond to a sparsity coefficient of -3 or less. A total of 85 points contained these projections. When we examined these 85 points, we found that 43 of them belonged to one of the rare classes. Furthermore, many of the points that did not belong to these 43 instances also showed interesting properties such as errors in recording the data (see below). In contrast, when we ran the algorithm in [23] over the data set, we found that only 28 of the 85 best outliers belonged to a rare class. These results were obtained using the 1-nearest neighbor; the results did not change significantly (and in fact worsened slightly) when the k-nearest neighbor was used. The less effective performance of this technique was because of the well-known effects of the data getting spread out sparsely in high dimensionality. In such cases, the sparsity effects of the different dimensions start dominating and it becomes difficult to meaningfully identify points as outliers since the small number of dimensions that show abnormal behavior are often masked by the noise effects of all the other dimensions.

We note that the algorithm in [20] defines outliers in a somewhat similar way to [23] because it uses full-dimensional nearest-neighbor distances; therefore, the noise effects in the results obtained with the algorithm of [23] are also applicable to the technique of [20]. Another alternative is the algorithm discussed in [21], which cannot be implemented efficiently for high-dimensional data; for a 279-dimensional data set this algorithm is not practical.

Even more interesting knowledge was obtained by examining the projections determined by the algorithm. Actually looking at the projections made it possible to find the actual patterns that correspond to abnormal behavior. In many cases, we also found some interesting outliers that are created by errors in recording the data; for example, on examining the patterns we found one record for which the height was 780 cm and the weight was 6 kg. This is obviously not compatible with standard human measurements; therefore, it is clear that there was some error in recording the data. The ability of the outlier detection algorithm to mine the appropriate combination of attributes (out of 279 attributes in this case) is important since such local patterns were not discovered by distance-based algorithms such as those discussed in [20,23].

Another interesting data set on which we tested the outlier detection method was the housing data set, which had 14 attributes concerning housing values in suburbs of Boston. The feature values of this data set corresponded to various factors that influenced housing prices such as crime rate, accessibility to highways, nitric oxide concentrations, distances to centers helping employment search, etc. We picked 13 of these 14 attributes (eliminating the single binary attribute) and then ran the outlier detection algorithm to find interesting three- and four-dimensional projections. An interesting example of an outlier was a record that had a high crime rate (1.628) and high pupil-teacher ratio (21.20) but had low distances (1.4394) to employment centers. The reason that such a record would be an outlier is that localities with high crime rates and high pupil-teacher ratios were also typically far from the employment centers. Another interesting outlier point was a projection that corresponded to low nitric oxide concentration (0.453), high proportion of pre-1940 houses (93.40%), and high index of accessibility to radial highways (8). This was again because the latter two attributes usually corresponded to high nitric oxide concentration. We also found some interesting points that showed informative trends with respect to the housing price. For example, it was usually the case that points with high index of accessibility to radial highways also had high crime rates. We found an interesting outlier point that had a low crime rate (0.04741), modest number of business acres per town (11.93), and also a low median home price (11,900). This was a rather contrarian point since the first two features' values are usually indicative of high housing prices in the rest of the data. Such data points are also useful for a classifier training algorithm since such points that are contrarian to the overall trends can confuse the training process. Thus these outlier detection techniques can also be used to prescreen such points from the data set before applying a classification algorithm.

4 Conclusions

In this paper, we discussed a new technique for outlier detection that is especially suited to very high-dimensional data sets. The method works by finding lower-dimensional projections that are locally sparse and cannot be discovered easily by brute-force techniques because of the number of combinations of possibilities. This technique for outlier detection has advantages over simple distance-based outliers that cannot overcome the effects of the dimensionality curse. We also illustrated how to implement the technique effectively for high-dimensional applications by using an evolutionary search technique. This implementation works almost as well as a brute-force implementation over the search space in terms of finding projections with very negative sparsity coefficients, but at a much lower cost. The techniques discussed in this paper extend the applicability of outlier detection techniques to high-dimensional problems; such cases are most valuable from the perspective of data mining applications.

References

1. Aggarwal CC (2001) Re-designing distance functions and distance based applications for high dimensional data. ACM SIGMOD Rec 30(1):13–18
2. Aggarwal CC et al (1999) Fast algorithms for projected clustering. In: Proceedings of ACM SIGMOD, pp 61–72
3. Aggarwal CC, Yu P (2000) Finding generalized projected clusters in high dimensional spaces. In: Proceedings of ACM SIGMOD, pp 70–81
4. Aggarwal CC, Hinneburg A, Keim DA (2001) On the surprising behavior of distance metrics in high dimensional space. In: Proceedings of ICDT, pp 420–434
5. Aggarwal CC, Orlin JB, Tai RP (1997) Optimized crossover for the independent set problem. Operat Res 45(2):226–234
6. Agrawal R, Gehrke J, Gunopulos D, Raghavan P (1998) Automatic subspace clustering of high dimensional data for data mining applications. In: Proceedings of ACM SIGMOD, pp 94–105
7. Agrawal R, Imielinski T, Swami A (1993) Mining association rules between sets of items in large databases. In: Proceedings of ACM SIGMOD, pp 207–216
8. Arning A, Agrawal R, Raghavan P (1996) A linear method for deviation detection in large databases. In: Proceedings of KDD, pp 164–169
9. Beyer K, Goldstein J, Ramakrishnan R, Shaft U (1999) When is nearest neighbors meaningful? In: Proceedings of ICDT, pp 217–235
10. Breunig MM, Kriegel H-P, Ng RT, Sander J (2000) LOF: identifying density-based local outliers. In: Proceedings of ACM SIGMOD, pp 93–104
11. Chakrabarti K, Mehrotra S (2000) Local dimensionality reduction: a new approach to indexing high dimensional spaces. In: Proceedings of the VLDB conference, pp 89–104
12. Darwin C (1859) The origin of species by natural selection. Available at: http://www.literature.org/authors/darwin-charles/the-origin-of-species/
13. De Jong KA (1975) Analysis of the behaviour of a class of genetic adaptive systems. PhD dissertation, University of Michigan, Ann Arbor, MI
14. Ester M, Kriegel H-P, Sander J, Xu X (1996) A density-based algorithm for discovering clusters in large spatial databases with noise. In: Proceedings of KDD, pp 226–231
15. Goldberg DE (1989) Genetic algorithms in search, optimization and machine learning. Addison-Wesley, Reading, MA
16. Guha S, Rastogi R, Shim K (1998) CURE: an efficient clustering algorithm for large databases. In: Proceedings of ACM SIGMOD, pp 73–84
17. Hinneburg A, Aggarwal CC, Keim DA (2000) What is the nearest neighbor in high dimensional spaces? In: Proceedings of the VLDB conference, pp 506–515
18. Holland JH (1975) Adaptation in natural and artificial systems. University of Michigan Press, Ann Arbor, MI

19. Kirkpatrick S, Gelatt CD, Vecchi MP (1983) Optimization by simulated annealing. Science 220(4589):671–680
20. Knorr E, Ng R (1998) Algorithms for mining distance-based outliers in large data sets. In: Proceedings of the VLDB conference, pp 392–403
21. Knorr E, Ng R (1999) Finding intensional knowledge of distance-based outliers. In: Proceedings of the VLDB conference, pp 211–222
22. Parthasarathy, S Aggarwal CC (2003) On the use of conceptual reconstruction for mining massively incomplete data sets. IEEE Trans Knowl Data Eng 15(6):1512–1531
23. Ramaswamy S, Rastogi R, Shim K (2000) Efficient algorithms for mining outliers from large data sets. In: Proceedings of ACM SIGMOD, pp 427–438
24. Zhang T, Ramakrishnan R, Livny M (1996) BIRCH: an efficient data clustering method for very large databases. In: Proceedings of ACM SIGMOD, pp 103–114