Outlier Detection for High Dimensional Data

Charu C. Aggarwal
IBM T. J. Watson Research Center
Yorktown Heights, NY 10598
charu@us.ibm.com

Philip S. Yu
IBM T. J. Watson Research Center
Yorktown Heights, NY 10598
psyu@us.ibm.com

ABSTRACT

The outlier detection problem has important applications in the field of fraud detection, network robustness analysis, and intrusion detection. Most such applications are high dimensional domains in which the data can contain hundreds of dimensions. Many recent algorithms use concepts of proximity in order to find outliers based on their relationship to the rest of the data. However, in high dimensional space, the data is sparse and the notion of proximity fails to retain its meaningfulness. In fact, the sparsity of high dimensional data implies that every point is an almost equally good outlier from the perspective of proximity-based definitions. Consequently, for high dimensional data, the notion of finding meaningful outliers becomes substantially more complex and non-obvious. In this paper, we discuss new techniques for outlier detection which find the outliers by studying the behavior of projections from the data set.

1. INTRODUCTION

An outlier is defined as a data point which is very different from the rest of the data based on some measure. Such a point often contains useful information on abnormal behavior of the system described 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 which make implicit assumptions of relatively low dimensionality of the data. These methods do not work quite as well when the dimensionality is high and the data becomes sparse.

Many data-mining algorithms in the literature find outliers as a side-product of clustering algorithms [2, 3, 5, 15, 18, 27]. However, these techniques define outliers as points which do not lie in clusters. Thus, the techniques implicitly define outliers as the background noise in which the clusters are embedded. Another class of techniques [7, 10, 13, 22, 23, 25] defines outliers as points which are neither a part of a cluster nor a part of the background noise; rather they are specifically points which behave very differently from the norm. Outliers are more useful based on their diagnosis of data characteristics which deviate significantly from average behavior. In many applications such as network intrusion detection, these characteristics may provide guidance in discovering the causalities of the abnormal behavior of the underlying application. The algorithms of this paper determine outliers based only on their deviation value.

Many algorithms have been proposed in recent years for outlier detection [7, 8, 10, 22, 23, 25, 26], but they are not methods which are specifically designed in order to deal with the curse of high dimensionality. The statistics community has studied the concept of outliers quite extensively [8]. In these techniques, the data points are modeled using a stochastic distribution, and points are determined to be outliers depending upon their relationship with this model. However, with increasing dimensionality, it becomes increasingly difficult and inaccurate to estimate the multidimensional distributions of the data points. Two interesting algorithms [22, 25] define outliers by using the full dimensional distances of the points from one another. This measure is naturally susceptible to the curse of high dimensionality. For example, consider the definition by Knorr and Ng [22]: *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 [25], this method is sensitive to the use of the parameter λ which 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 other point [9]. Thus, if we pick λ slightly smaller than the shell radius, then all points are outliers; if we pick λ slightly larger, 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 which can then be defined as outliers. Aside from this, the data in real applications is very noisy, and the abnormal deviations may be embedded in some lower dimensional subspace. The deviations in this embedded subspace cannot be determined by full dimensional measures such as the L_p-norm because of the noise effects of the other dimensions. The work in [25] introduces the following definition of an outlier: *Given a k and n, a point p is an outlier if the distance to its kth nearest neighbor is smaller than the corresponding value for no more than $n-1$ other points.* Al-

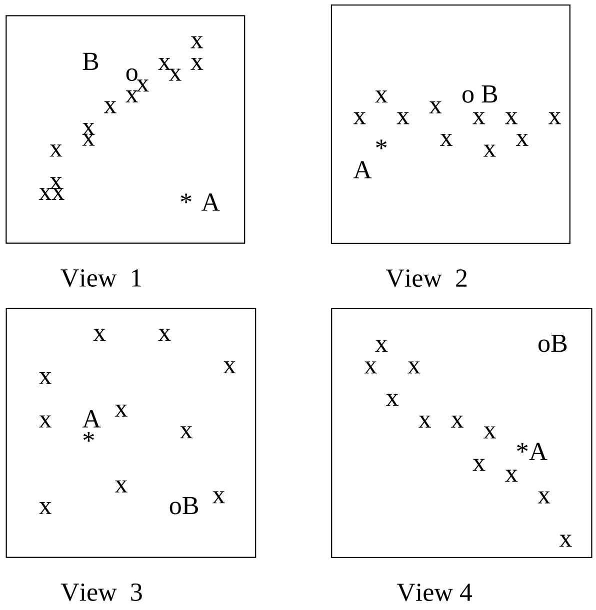

Figure 1: The 2-dimensional views 1 and 4 expose outliers A and B, whereas the views 2 and 3 do not. Full dimensional measures become increasingly susceptible to the sparsity and noise effects of high dimensionality.

though the definition in [25] has some advantages over that provided in [22], it is also not specifically designed to work for high dimensional problems. This is again because of the sparse behavior of distance distributions in high dimensionality, in which the actual values of the distances are similar for any pair of points.

An interesting recent technique finds outliers based on the densities of local neighborhoods [10]. This technique has some advantages in accounting for local levels of skews and abnormalities in data collections. In order to compute the outlier factor of a point o, the method in [10] computes its local reachability density using the average smoothed distances to points in the locality of o. Unfortunately, this is again a difficult computation to perform meaningfully in high dimensionality, in which the concept of locality becomes difficult to define because of data sparsity. 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, 22, 25] 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 then meaningful clusters cannot be found in the data [2, 3, 5, 11]; similarly it is difficult to detect abnormal deviations.

For problems such as clustering and similarity search, it has been shown [1, 2, 3, 5, 11, 19] that by examining the behavior of the data in subspaces, it is possible to design more effective algorithms. This is because different localities of the data are dense with respect to different subsets of attributes. The same insight is true for outliers, because in typical applications such as credit card fraud, only the subset of the attributes which are actually affected by the abnormality of the activity are likely to be useful in detecting the behavior.

In order to better explain our point, let us consider the example illustrated in Figure 1. In the above example, we have shown several 2-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, and 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 outliers effectively 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 use different features or views.

Thus, the problem of outlier detection is similar to a large number of other problems in the data mining literature which lose their algorithmic effectiveness for the high dimensional case. Previous work has not focussed on this critical problem for the high dimensional case, and has used relatively straightforward proximity measures [10, 22, 25] which are more applicable to low dimensional versions. 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. Recent work [23] has discussed the concept of intensional knowledge of distance-based outliers in terms of subsets of attributes. This technique provides excellent interpretability by providing the reasoning behind why a point may be considered an outlier. On the other hand, the technique uses a roll-up/drill-down method which tends to be quite expensive for high dimensional data.

1.1 Desiderata for High Dimensional Outlier Detection Algorithms

In this section, we provide further intuition on the desiderata for effective high dimensional outlier detection algorithms. In order to work effectively, high dimensional outlier detection algorithms should satisfy the following properties:

- They should devise techniques to handle the sparsity problems in high dimensionality effectively.
- They should provide interpretability in terms of the reasoning which creates the abnormality. If possible, the probabilistic level of significance with which this reasoning applies should be determined.
- Proper measures must be identified in order to account for the physical significance of the definition of an outlier in k-dimensional subspace. For example, a distance based threshold for an outlier in a k-dimensional subspace is not directly comparable to one in $(k+1)$-dimensional subspace.
- The outlier detection algorithms should continue to be computationally efficient for very high dimensional problems. If possible, algorithms should be devised which avoid a combinatorial exploration of the search space.
- The algorithms should provide importance to the local data behavior while determining whether a point is an outlier.

We note that some of the above aims have been achieved by different methods [7, 10, 22, 23, 25] though none of them work effectively for the high dimensional case.

In this paper, we discuss a new technique for outlier detection which 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.

1.2 Defining Outliers in Lower Dimensional Projections

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

1.3 Defining Abnormal Lower Dimensional Projections

An abnormal lower dimensional projection is one in which the density of the data is exceptionally lower than average. In order to define such projections, we first perform a grid discretization of the data. Each attribute of the data is divided into ϕ equi-depth ranges. Thus, each range contains a fraction $f = 1/\phi$ of the records. The reason for using equi-depth ranges as opposed to equi-width ranges is that different localities of the data have different densities [10]. We would like to take this into account while finding outliers. These ranges form the units of locality which we will use to define the lower dimensional regions which are unreasonably sparse.

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

Let us assume that there are a total of N points in the database. 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. Under this assumption, the number of points in a cube can be approximated by a normal distribution because of the central limit theorem. Then the expected fraction and standard deviation of the points in a k-dimensional cube is given by $N \cdot f^k$ and $\sqrt{N \cdot f^k \cdot (1 - f^k)}$. 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 which 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 an intuitive idea of the level of significance for a given projection.

1.4 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) which are unusually sparse. This is not unexpected: unlike problems such as large itemset detection [6] where one is looking for large aggregate patterns, the problem of finding subsets of dimensions which are sparsely populated has the flavor of finding a needle in haystack, since one is looking for patterns which 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 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 examining any subset or superset projection. One solution is to change the measure in order to force better closure or pruning properties. This can however 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 which 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. Such methods have recently been used quite successfully for the problem of high dimensional nearest neighbor search [19].

2. EVOLUTIONARY ALGORITHMS FOR OUTLIER DETECTION

In this section, we will discuss the algorithms which 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 which are sparse. These patterns are then used in order to determine the points which are outliers. We discuss two algorithms for outlier detection: a naive brute force algorithm which is very slow at finding the best patterns because of its exhaustive search of the entire space, and a much faster evolutionary algorithm which is able to quickly find hidden combinations of dimensions in which the data is sparse. The total dimensionality of the data is denoted by d. We assume that one of the inputs to the algorithm is the dimensionality k of the projection which is used in order to determine the outliers. Aside from this, the algorithm uses as input the number m of projections to be determined.

The brute-force algorithm is illustrated in Figure 2. The algorithm works by examining all possible sets of k-dimensional candidate projections (with corresponding grid ranges) and retaining the m projections which have the most negative sparsity coefficients. In order to actually determine the candidate projections, the method uses a bottom-up recursive algorithm in which $(i+1)$-candidate cubes are determined by concatenating the candidate i-projections with all $d \cdot \phi$ possible sets of 1-dimensional projections and their grid-ranges (denoted by Q_1). The concatenation operation is illustrated in Figure 2 by $\oplus$. Note that for a given cube, it only makes

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

Figure 2: The Brute-force Technique

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

Figure 3: The Outlier Detection Algorithm

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 ith side proportional to $p - r(i)$;
    Add the solution corresponding to the ith side to S';
    end;
  Replace S by S';
  return(S);
end

Figure 4: The Selection Criterion for the Genetic Algorithm

```
Algorithm Crossover(S)
begin
  Match the solutions in the population pairwise;
  for each pair of solutions s1, s2 thus matched do
     begin
     (s, s') = Recombine(s1, s2);
     Replace s1 and s2 in the population by s and s';
     end;
  return(S);
end

Algorithm Recombine(s1, s2)
begin
  Q = Set of positions in which either s1 or s2 is *;
  R = Set of positions in which neither s1 nor s2 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 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 s derives it; }
  return(s, s');
end
```

Figure 5: The Crossover Algorithm

```
Algorithm Mutation(S, p1, p2)
begin
  for each string s ∈ S do begin
    Let Q be the set of positions in s which 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 as the set of positions in s which 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;
  end;
  return(S);
end
```

Figure 6: The Mutation Algorithm

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 in R_k with most negative sparsity coefficients in $\mathcal{F}$ are retained. The set of points in the data which contain the corresponding ranges for the projections are reported as the final set of outliers.

As we shall see in later sections, the algorithm discussed in Figure 2 is computationally untenable for problems of even modest complexity. This is because of the exponentially increasing search space of the outlier detection problem. In order 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

Evolutionary Algorithms [20] are methods which imitate the process of organic evolution [12] in order to solve parameter optimization problems. The fundamental idea underlying Darwinian evolution is that in nature, resources are scarce and this leads to a competition among the species. Consequently, 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, nature occasionally throws in a variant by the process of mutation, so as to ensure 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 which 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 [21] 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 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 string representations 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, the individuals in the population become more and more genetically similar to each other. This phenomenon is referred to as **convergence**. Dejong [14] defined convergence of a *gene* as the stage at which 95% of the population had the same value for that gene. The population is said to have converged when all genes have converged.

The application of 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 [4], and it is often a non-trivial task to design the recombinations, selections and mutations which work well for a given problem. In the next section, we will discuss the details of the evolutionary search procedures which work effectively for the outlier detection problem.

2.2 The Evolutionary Outlier Detection Algorithm

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 ith 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 4-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 dimension 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 earlier. The evolutionary search technique starts with a population of p random solutions and iteratively used the processes of selection, crossover and mutation in order 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 [14] convergence criterion in order 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 Figure 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 [17] 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 the favor of higher ranked solutions. This is called rank selection and is often more stable than straightforward fitness proportional methods which 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 set of children strings which 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 ith 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 Figure 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 3-dimensional projections in 5-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 2-dimensional and 4-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 those which are viable in terms of feasibility. In order to take care of this, we create an optimized crossover process which 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 as 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.

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.

In order 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 [4] 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 which 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. In order 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 in order to find a solution recombinant for the $(k-k')$ Type III positions. In order to find the remaining positions, we always extend the string with the position which results in the string with most negative sparsity coefficient 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 which combines the good aspects of both parent strings. The crossover technique is a key process in evolutionary algorithms which is not available in hill climbing or simulated annealing methods; by doing so, it is possible to create new strings in the population which combine features of both parent solutions. 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 Figure 5.

- **Mutation:** We perform mutations of two types:
Type I: Let Q be the set of positions in the string which are $*$. Then we pick a position in the string which 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 which 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 mutation algorithm is illustrated in Figure 6.

2.3 Postprocessing Phase

At termination, the algorithm is followed by a postprocessing phase. In the postprocessing phase, we find all the sets of data points which contain the the abnormal projections reported by the algorithm. (For example, a point covers the projection *3*6 if the 2nd and 4th coordinates correspond to grid ranges 3 and 6.) These points are the outliers and are denoted by $\mathcal{O}$ in Figure 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 out of a d-dimensional data set, each subcube represented by a k-dimensional projection contains an expected fraction of $1/\phi^k$ of the data. Thus, if we pick $\phi = 10$, then even for a 4-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 which has 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 number 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 Equation 1, this is given by $-\sqrt{\frac{N}{\phi^k-1}}$. The value of N is pre-decided 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 of course the normal distribution assumption is not true; however, a value of $s = -3$ is a good reference point in order to decide the value of k. Therefore, we have:

$$\sqrt{\frac{N}{\phi^k - 1}} = -s \qquad (2)$$

By expressing the entire equation in terms of k, we obtain $k^* = \lfloor \log_\phi(N/s^2 + 1) \rfloor$. Note that the rounding process often makes the effective sparsity coefficient slightly more negative than that chosen by the user. For a real application, a user may wish to test different values of this (intuitively interpretable) parameter s in order 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.

3. EMPIRICAL RESULTS

The algorithm was implemented on a 233MHz 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 which 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 size of the data and 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 3-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 which 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. These data sets were picked in order to test the behavior of the method for different dimensionalities. In each case we found the $m = 20$ best projections and reported the outlier points corresponding to these projections. It is evident from these results that the performance of the brute force technique quickly becomes untenable for large data sets. In fact, for the musk data set which had 160 dimensions, the outlier detection algorithm did not terminate in a reasonable 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 which performs the crossover by exchanging segments of the two strings. We implemented optimized crossover mechanism which 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 which were not in the feasible search space of k-dimensional projections. On the other hand, the optimized crossover solution identified combinations of dimensions which were both feasible and of high quality. We have also reported the average sparsity coefficients of the best 20 (non-empty) projections indicated under the column (quality). In 3 of the 5 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 the importance of effective solution recombination which 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.

3.1 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 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 which are used in order to diagnose arrythmia. The data set contains a total of 13 (non-empty) 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 which occurred in less than 5% of the data set as rare labels. The corresponding class distribution is illustrated in Table 2. 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 which belonged to one of the rare classes. If the outlier detection works well, we expect that such abnormal classes would be over-represented in the set of points found. These kinds of classes are also interesting from a practical perspective.

We ran the evolutionary algorithm in order to find all the sparse projections in the data set which correspond to a

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 (*)

Table 2: Class Distribution of Arrythmia Data Set

Case	**Class Codes**	**Percentage of Instances**
Commonly Occuring Classes ($\geq 5\%$)	01, 02, 06, 10, 16	85.4%
Rare Classes ($< 5\%$)	03, 04, 05, 07, 08, 09, 14, 15	14.6%

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 which did not belong to these 43 instances also showed interesting properties such as errors in recording the data (see below). In contrast, when the ran the algorithm in [25] over the data set, we found that only 28 of 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 which show abnormal behavior are often masked by the noise effects of all the other dimensions.

We note that the algorithm in [22] defines outliers in a somewhat similar way to [25], because it uses full dimensional nearest neighbor distances; therefore, the noise effects in the results obtained with the algorithm of [25] are also applicable to the technique of [22].

Even more interesting knowledge was obtained by examining the projections determined by the algorithm. By actually looking at the projections it was possible to find the actual patterns which correspond to abnormal behavior. In many cases, we also found some interesting outliers which 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 kilograms. 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 [22, 25].

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 correspond to various factors which influenced housing prices such as crime rate, accessibility to highways, nitric oxides concentration, distances to employment centers etc. We picked 13 of these 14 attributes (eliminating the single binary attribute). Then we ran the outlier detection algorithm in order to find interesting 3- and 4-dimensional projections. An interesting example of an outlier was a record which 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 pupil-teacher ratios were also typically far off from the employment centers. Another interesting outlier point was a projection which correspond 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 correspond to high nitric oxide concentration. We also found some interesting points which 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 which 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 points which are contrarian to the overall trends can confuse the training process. Thus, these outlier detection techniques can also be used in order to pre-screen such points from the data set before applying a classification algorithm.

4. CONCLUSIONS

In this paper, we discussed a new technique for outlier detection which is especially suited to very high dimensional data sets. The method works by finding lower dimensional projections which 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 which 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.

5. REFERENCES

[1] C. C. Aggarwal. Re-designing Distance Functions and Distance Based Applications for High Dimensional Data. *ACM SIGMOD Record*, March 2001.

[2] C. C. Aggarwal et al. Fast Algorithms for Projected Clustering. *ACM SIGMOD Conference Proceedings*, 1999.

[3] C. C. Aggarwal, P. Yu. Finding Generalized Projected Clusters in High Dimensional Spaces. *ACM SIGMOD Conference Proceedings*, 2000.

[4] C. C. Aggarwal, J. B. Orlin, R. P. Tai. Optimized Crossover for the Independent Set Problem. *Operations Research* 45(2), March 1997.

[5] R. Agrawal, J. Gehrke, D. Gunopulos, P. Raghavan. Automatic Subspace Clustering of High Dimensional Data for Data Mining Applications. *ACM SIGMOD Conference Proceedings,* 1998.

[6] R. Agrawal, T. Imielinski, A. Swami. Mining Association Rules Between Sets of Items in Large Databases. *ACM SIGMOD Conference Proceedings*, 1993.

[7] A. Arning, R. Agrawal, P. Raghavan. A Linear Method for Deviation Detection in Large Databases. *KDD Conference Proceedings*, 1995.

[8] V. Barnett, T. Lewis. Outliers in Statistical Data. *John Wiley and Sons*, NY 1994.

[9] K. Beyer, J. Goldstein, R. Ramakrishnan, U. Shaft. When is Nearest Neighbors Meaningful? *ICDT Conference Proceedings*, 1999.

[10] M. M. Breunig, H.-P. Kriegel, R. T. Ng, J. Sander. LOF: Identifying Density-Based Local Outliers. *ACM SIGMOD Conference Proceedings*, 2000.

[11] K. Chakrabarti, S. Mehrotra. Local Dimensionality Reduction: A New Approach to Indexing High Dimensional Spaces. *VLDB Conference Proceedings*, 2000.

[12] C. Darwin. *The Origin of the Species by Natural Selection.* Published, 1859.

[13] D. Hawkins. *Identification of Outliers*, Chapman and Hall, London, 1980.

[14] K. A. De Jong. Analysis of the Behavior of a Class of Genetic Adaptive Systems. Ph. D. Dissertation, University of Michigan, Ann Arbor, MI, 1975.

[15] M. Ester, H.-P. Kriegel, J. Sander, X. Xu. A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise. *KDD Conference Proceedings*, 1996.

[16] J. J. Grefenstette. Genesis Software Version 5.0. Available at http://www.santafe.edu.

[17] D. E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning.* Addison Wesley, Reading, MA, 1989.

[18] S. Guha, R. Rastogi, K. Shim. CURE: An Efficient Clustering Algorithm for Large Databases. *ACM SIGMOD Conference Proceedings*, 1998.

[19] A. Hinneburg, C. C. Aggarwal, D. A. Keim. What is the nearest neighbor in high dimensional spaces? *VLDB Conference Proceedings*, 2000.

[20] J. H. Holland. *Adaptation in Natural and Artificial Systems.* University of Michigan Press, Ann Arbor MI 1975.

[21] S. Kirkpatrick, C. D. Gelatt, M. P. Vecchi. Optimization by Simulated Annealing. *Science* (220) (4589): pages 671-680, 1983.

[22] E. Knorr, R. Ng. Algorithms for Mining Distance-based Outliers in Large Data Sets. *VLDB Conference Proceedings*, September 1998.

[23] E. Knorr, R. Ng. Finding Intensional Knowledge of Distance-based Outliers. *VLDB Conference Proceedings*, 1999.

[24] R. Ng, J. Han. Efficient and Effective Clustering Methods for Spatial Data Mining. *VLDB Conference Proceedings*, pages 144-155, 1994.

[25] S. Ramaswamy, R. Rastogi, K. Shim. Efficient Algorithms for Mining Outliers from Large Data Sets. *ACM SIGMOD Conference Proceedings*, 2000.

[26] S. Sarawagi, R. Agrawal, N. Meggido. Discovery Driven Exploration of OLAP Data Cubes. *EDBT Conference Proceedings*, 1998.

[27] T. Zhang, R. Ramakrishnan, M. Livny. BIRCH: An Efficient Data Clustering Method for Very Large Databases. *ACM SIGMOD Conference Proceedings*, 1996.