

Detecting Outliers in Categorical Data Sets Using Non-Derivable Itemsets

Michelle Fox^a, Gary Gramajo^b,
Anna Koufakou^c and Michael Georgiopoulos^d

^aSchool of EECS, Milwaukee School of Engineering, Milwaukee, WI;

^bSchool of Mathematics, Florida State University, Tallahassee, FL;

^cSchool of EECS, University of Central Florida, Orlando, FL;

^dSchool of EECS, University of Central Florida, Orlando, FL

ABSTRACT

Outlier Detection is a research field with many applications, such as detecting credit card fraud or network intrusions. Most previous research focused on numerical data and pair-wise distances among data points to detect outliers. Nevertheless, most categorical data sets lack straightforward mapping to numerical values and approaches that rely on computing distances do not apply so easily. Recently, a few outlier methods were proposed for categorical datasets using the concept of Frequent Itemsets (FIs). The number of generated FIs can be far too high, especially in the case of large, dense datasets, containing a high number of categorical values. There has been much research towards summarizing and/or condensing the FIs in a dataset to address this issue. However these ideas have not been applied directly to the field of outlier detection. In this report, we explore the effect of using a condensed representation of Frequent Itemsets, called Non-Derivable Itemsets (NDI), on the accuracy and efficiency of an outlier detection method. Our experimental results indicate that NDI-based Outlier Detection offers significant gains in terms of speed and scalability over a FI-based outlier detection, while maintaining comparable detection accuracy.

Keywords: Outlier Detection, Categorical Data, Frequent Itemset Mining, Non-Derivable Itemsets, Condensed Representations

1. INTRODUCTION

Outlier detection is a research field with many applications, such as credit card fraud detection,¹ or discovering network intrusion.² In contrast to the aim of traditional data mining (i.e. to mine common or frequent patterns in the dataset), outlier detection approaches focus on detecting patterns that occur infrequently in the dataset.³ One of the most widely accepted definitions of an outlier pattern is provided by Hawkins:

An outlier is an observation that deviates so much from other observations as to arouse suspicion that it was generated by a different mechanism.⁴

The majority of the existing research efforts in outlier detection have focused on datasets with a specific attribute type, mainly assuming that attributes are only numerical and/or ordinal. In the case of data with categorical attributes, techniques which assume numerical data need to first map the categorical values to numerical values, a task which is not a straightforward process (e.g., the mapping of a marital status attribute (married or single) to a numerical attribute).⁵ A second issue is that many applications for mining outliers require the mining of very large datasets (e.g. terabyte-scale data⁶). This leads to the need for outlier detection algorithms which must scale well with the size and dimensionality of the dataset.⁷ Data may also be scattered across various geographical areas, which implies that transferring data to a central location and then detecting outliers is impractical, due to the size of the data, as well as data ownership and control issues. Thus, the

Further author information: (Send correspondence to M.G.):

A.K.: E-mail: akoufako@mail.ucf.edu; M.G.: E-mail: mighaelg@mail.ucf.edu, Tel.: 1 407 823 5338

algorithms designed to detect outliers must minimize the number of data scans, as well as the need for excessive communication and required synchronization.

Recently a few outlier detection methods have been proposed for categorical datasets.^{7,8} Some of these methods use *Frequent Itemset Mining* (FIM). FIM entails the process of finding sets of items that co-occur frequently in the data. Let $I = \{i_1, i_2, \dots, i_m\}$ be a set of items, m is the number of items in the database, D a set of transactions (rows or records). Each transaction T is a set of items such that $T \subseteq I$. Each transaction is associated with a unique identifier, called its *TID*. Given a set of some items in I , X , we say that a transaction T contains X , if $X \subseteq T$. The support of X , $sup(X)$, is the number of transactions T in D that contain X . We say that a set X is frequent if it appears no less than *minsup* times in our dataset D , where *minsup* is a user-defined threshold.

The FIM-based outlier detection methods^{5,8,9} assume that they can first extract all frequent itemsets from the data, up to a user entered maximum length, and then assign an outlier score to all data points based on the subsets the points contain. For the frequent set extraction step, these algorithms can employ an Apriori-like method. However, even though Apriori-inspired algorithms perform well with sparse data sets, such as market-basket data, where the frequent patterns are of short length, they face problems for dense data.¹⁰ If the *minsup* threshold is low or there is a high number of items in the dataset, there might be far too many frequent itemsets to store or, in the case of distributed data, exchange between the different sites. To alleviate this issue many methods have been proposed, for example condensed representations such as *maximal sets*¹¹ or *closed sets*.¹⁰

The focus of this paper is to apply a condensed representation method to efficiently mine FIs in dense datasets, as part of an outlier detection algorithm for categorical datasets. Specifically, we will use Non-Derivable Itemsets (NDI) which have shown to improve efficiency and scalability over traditional FIM Apriori-like methods by several orders of magnitude when applied to dense datasets.¹²

2. PREVIOUS WORK

2.1 Outlier Detection

The existing outlier detection work can be categorized as follows.

Statistical-model based methods assume that a specific model describes the distribution of the data.¹³ General limitations include obtaining the suitable model for each particular dataset and application, and lack of scalability with respect to data dimensionality.⁵

Distance-based methods¹⁴ essentially compute distances among data points, thus become quickly impractical for large datasets (e.g., a nearest neighbor method has quadratic complexity with respect to the number of dataset points). For example, Knorr et al.¹⁴ define a point as an outlier if at least $p\%$ of the points in the dataset lie further than distance D from it. These methods exhibit high computational complexity (e.g. nearest neighbor based methods have quadratic complexity with respect to the dataset size) rendering them impractical for really large datasets. Several methods may be employed to make the k -NN queries faster (to achieve linear or logarithmic time), such as an indexing structure (e.g. KD-tree, or X-tree); however these structures have been shown to break down as the dimensionality grows.¹⁵ Bay and Schwabacher¹⁵ propose a distance-based method based on randomization and pruning and claim its complexity is close to linear in practice. Distance-based methods require data to be in the same location or large amounts of data to be transferred from different locations, which makes them impractical for distributed data.

Clustering techniques can be employed to first cluster the data, so that outliers are the points that do not belong to formed clusters. In general, all clustering-based methods rely on the clusters to define outliers, thus focus in optimizing clustering, not outlier detection.¹⁵

Density-based methods estimate the density distribution of the data and identify outliers as those lying in relatively low-density regions. For example, Breunig et al.¹⁶ assign a *degree* of outlierness to each data point, the local outlier factor (*LOF*), based on the local density of the area around the point and the local densities of its neighbors, relying on a user-given minimum number of points. Although these methods are able to detect outliers not discovered by the distance-based methods, they face a challenge with sparse high-dimensional data.

Other examples of outlier detection work include Support Vector methods,¹⁷ and Replicator Neural Networks,¹⁸ among others.

Most of the aforementioned techniques are geared towards numerical data and thus are more appropriate for numerical datasets or ordinal data that can be easily mapped to numerical values; in the case of categorical data, there is little sense in ordering categorical values, then mapping them to numerical values and computing distances, e.g., distance between two values such as TCP Protocol and UDP Protocol.⁵ Another limitation of previous methods is the lack of scalability with respect to number of points and/or dimensionality of the dataset. Outlier detection techniques for categorical datasets have recently appeared in the literature (e.g. entropy-based method,¹⁹ FIM-based method⁸). The methods proposed by He et. al.,¹⁹ Otey et. al.⁵ and later Koufakou et. al.,⁹ use FIM in order to assign an outlier score to each data point based on the frequency of the subsets this point contains. Koufakou et. al.⁷ experimented with three representative outlier detection methods for categorical data, and proposed AVF (Attribute Value Frequency), a simple, fast, and scalable method for categorical sets. Otey et al.⁵ presented a distributed and dynamic outlier detection method to address categorical and continuous attributes.⁵ Otey’s method combines categorical and continuous spaces as follows: for each possible categorical itemset, X , they isolate the data points that contain set X , then calculate the covariance matrix of the continuous values of these points. A point is then likely to be an outlier if it contains infrequent categorical sets, or if its continuous values differ from the covariance. Koufakou et al.⁹ proposed ODMAD, an outlier detection method for mixed attribute data, that can handle sparse high-dimensional continuous data. ODMAD first inspects the categorical space in order to detect data points with irregular categorical values. Secondly, it sets aside these points and focuses on specific categorical values and the points that contain these values one at a time. ODMAD exhibited linear performance with respect to the number of data points and number of numerical dimensions, and significant performance improvements over Otey’s method.

2.2 Frequent Itemset Mining (FIM)

Agrawal introduced the first efficient FIM algorithm, Apriori,²⁰ whose central property is: “If A is a frequent itemset, then every subset of A is a frequent itemset”. What ensues from this statement is that, once a set has been found infrequent, not only can it be pruned, but all the itemsets that contain it can be pruned as well, as they are also infrequent. For example, if set ab has been found to be non-frequent, then itemsets abc , abd , $abcd$, etc., that contain set ab , cannot be frequent, so they too can be pruned. More detail about Apriori can be found in the Algorithm Section.

There are several algorithms in the literature to improve different aspects of Apriori, e.g.: DIC,²¹ which dynamically counts candidates of varying length to reduce number of scans; DHP,²² which collects approximate counts to rule out candidates that cannot possibly be frequent; FP-Growth,²³ which stores the transactions in an FP-tree, a trie structure where every item contains a list of all the transactions that contain that item. Also there are several ARM implementations online.^{24–26}

2.3 Condensed Representations of FIs (CFIs)

Even with candidate pruning in Apriori, the resulting frequent itemsets might still be numerous, an issue exacerbated when these sets contain millions of items. Apriori-inspired algorithms perform well with sparse data sets, such as market-basket data, where the frequent patterns are short, but they face problems for dense data.¹⁰ Dense datasets are those sets that contain many strong correlations, e.g. census data.²⁷ In addition, setting the *minsup* threshold too low, might result in a very high number of frequent itemsets, especially when there are thousands or even millions of items in the dataset.

To solve this issue much work has been conducted towards *condensed representations of FIs* (CFIs).^{10,11} The goal of these methods is to find a much smaller group of representative sets, from which one can deduce all FIs. Many CFIs have been proposed, which we describe in the following section.

2.3.1 Maximal and Closed Sets

A *maximal set* is a frequent set that is not a subset of any other frequent itemset.¹¹ A *closed set* is a set with support such that there is no superset with support equal to it.¹⁰ Let M be the set of maximal sets and C be the set of closed sets. Even though in theory M and C can be exponential in the number of items, in practice,

C can be orders of magnitude smaller than the number of frequent sets (especially for dense datasets), while M can be orders of magnitude smaller than C .

Closed sets are lossless in the sense that the exact frequency of all FIs can be determined from C , while M leads to a loss of information (since subset frequency is not kept).¹⁰ Once a given support threshold is known as well as all frequent closed itemsets with their supports, the rest of the frequent itemsets and their supports can be generated. If an itemset I has at least one superset in the set of all frequent closed itemsets (*FreqClosed*), then the $supp(I) = supp(cl(I))$ where $cl(I)$ is the smallest superset of I in *FreqClosed*. The reader is reminded that the Apriori Principle states that if an itemset is frequent, then all of its subsets are frequent. Thus, the itemset I in this case is frequent.

2.3.2 δ -free Sets

Boulicaut et. al.²⁷ proposed the δ -free sets, which can be used as a δ -adequate representation for itemsets. Essentially, if we know that the association rule $ABC \rightarrow D$ is nearly an exact rule (i.e. it is true for “most” of the dataset), we can approximate the frequency of set $ABCD$ using the frequency of ABC .

In order to provide a more thorough background, we will briefly describe what δ -free sets are based on. Thus, it is important to describe what a δ -strong rule is. A δ -strong rule is an association rule of the form $X \Rightarrow^\delta a$, where $X \subseteq \mathcal{I}$, $a \in \mathcal{I}/X$, and δ is a natural number. The rule is valid if it is not violated in more than δ transactions.

An itemset Y is a δ -free set if and only if there is no valid δ -strong rule $X \Rightarrow^\delta a$ such that $X \subset Y$, $a \in Y$, $a \notin X$. Given any database \mathcal{D} , if the set of all frequent δ -free sets is stored in a set, this set can be used to approximate the support of the frequent non- δ -free sets.²⁷ Thus, a condensed representation of the frequent itemsets is possible.

2.3.3 Non-Derivable Sets

For an itemset I , lower bounds and upper bounds on the support of I can be calculated based on the supports of its subsets. That is, for all $X \subseteq I$, we can calculate the bounds on I . An itemset I is a *derivable itemset* if and only if the *greatest lower bound* is not equal to the *least upper bound* on the support of I .¹² Otherwise I is considered *non-derivable*.¹²

Any itemset not in the remaining NDI representation is either infrequent or derivable or both. The bounds $LB(I)$ and $UB(I)$ are calculated for I . If they are not equal, the itemset I is infrequent. Otherwise, the bounds of the subsets of I can be calculated. If none of the subsets turns out to be infrequent, then the support for I can be found and I is frequent. It is essential to emphasize that non-derivable itemsets cannot be very large. The reason being is that the interval length of their support, $w(I) = UB(I) - LB(I)$, decreases exponentially with the cardinality of I .¹²

2.4 Other methods

Besides condensed representations, other types of patterns have been proposed and used as a step prior to a data mining tasks such as clustering. For example,²⁸ used highly-correlated association patterns called hyperclique patterns²⁹ as a cleaning step to filter out noisy objects, resulting in better clustering performance and higher quality associations, while in³⁰ the authors use hyperclique patterns to detect off-topic documents. The authors in³¹ proposed summary sets in an effort to summarize categorical data for clustering; summary sets were proven to be closed sets. Finally, in³² a support approximation and clustering method was proposed to mine FIs in the dataset.

3. ALGORITHMS

As shown by Koufakou et al.,⁷ the ‘ideal’ outlier in a categorical dataset is one for which each and every value of its values is extremely irregular (or infrequent). The infrequent-ness of an attribute value can be measured by computing the number of times this value is assumed by the corresponding attribute in the dataset. Koufakou et al.⁷ assigns a score to each data point, which reflects the frequency with which each attribute value of the point occurs. In⁹ this notion of ‘outlierness’ is extended to cover the likely scenario where none of the single values in an outlier point are infrequent, but the co-occurrence of two or more of its attribute values is infrequent.

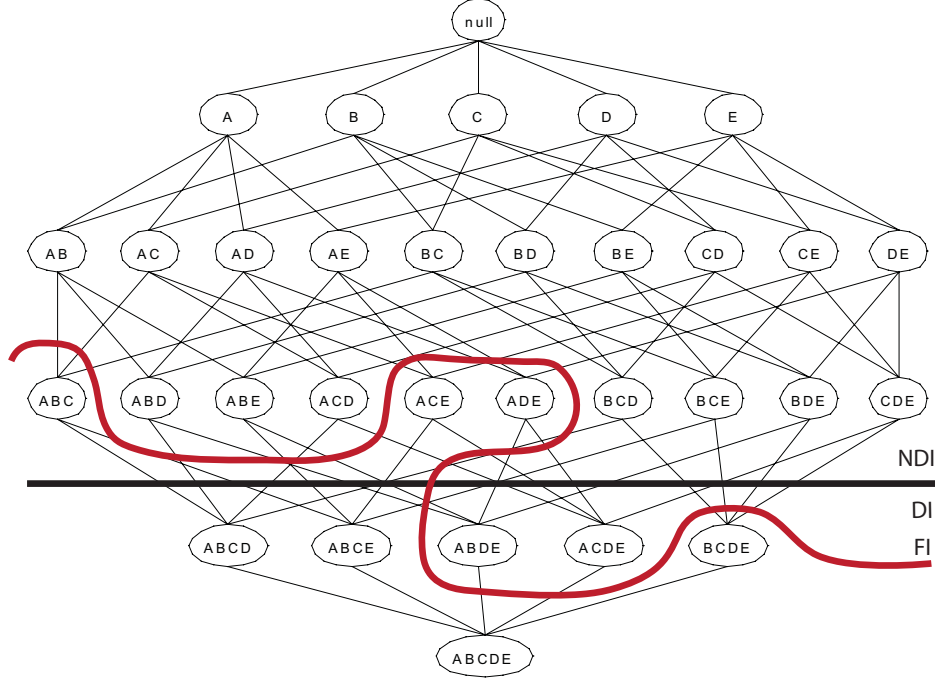


Figure 1. Representation of NDI on a lattice

In order to extract the infrequent itemsets in the dataset, an Apriori-like algorithm can be employed. Alternatively, we propose to use a condensed representation of FIs, called Non-Derivable Itemset (NDI) algorithm, to extract the NDI sets in order to detect outliers.

In the following we describe the Apriori Algorithm, the Non-Derivable Itemset Algorithm, and Outlier Detection using the aforementioned techniques.

3.1 Apriori Algorithm

3.1.1 Overview

In this section, we give a brief overview of the Apriori algorithm.²⁰ The Apriori algorithm returns all of the frequent itemsets, given a database and a user-entered parameter, called minimum support.

3.1.2 Apriori Algorithm Description and Pseudocode

Association Rule Mining (ARM) is a very popular data mining technique, which involves the extraction of relationships which exist among the data. Initially intended to tackle *market basket analysis*, i.e. to explore consumer trends in terms of their purchases,²¹ ARM has been proven useful in a plethora of other applications, for example merchandise stocking,³³ and climate prediction.³⁴ An association rule is an implication of the form “{diapers, milk} \Rightarrow {baby powder},” which means that people who buy diapers and milk also tend to buy baby powder.

Let $I = \{i_1, i_2, \dots, i_m\}$ be a set of items, where m is the number of items in the database, and \mathcal{D} a set of transactions, where each transaction \mathcal{T} is a set of items such that $\mathcal{T} \subseteq I$. Each transaction is associated with a unique identifier, called its TID. Given a set of some items in I , \mathcal{X} , we say that a transaction \mathcal{T} contains \mathcal{X} , if $\mathcal{X} \subseteq \mathcal{T}$. An association rule is an implication of the form $\mathcal{X} \Rightarrow \mathcal{Y}$, where $\mathcal{X} \subset I, \mathcal{Y} \subset I$ and $\mathcal{X} \cap \mathcal{Y} = \emptyset$. In this case, \mathcal{X} is called the *antecedent* or the *body*, and \mathcal{Y} is called the *consequent* or the *head* of the rule.

The support of an item set \mathcal{X} is the percentage of those transactions in \mathcal{D} which contain \mathcal{X} . For example, let \mathcal{D} be the set of all “baskets” or “carts” of products bought by the customers of a grocery store, on a given day. If \mathcal{U} is the set of all transactions that contain all items in \mathcal{X} , then we have:

$$\text{sup}(\mathcal{X}) = \frac{|\mathcal{U}|}{|\mathcal{D}|} * 100\%$$

where $|\mathcal{U}|$, $|\mathcal{D}|$ are the number of elements in \mathcal{U} and \mathcal{D} , respectively.

An itemset \mathcal{X} is *frequent* if its support is not less than the user-specified minimum support (*minsup*). A frequent itemset is maximal if it is not a subset of any other frequent itemset. The set of all maximal frequent itemsets is denoted as MFI, and is also called positive border. The knowledge of MFI is sufficient to know all frequent itemsets, but not their exact support.

The confidence of a rule $\mathcal{R} = (\mathcal{X} \Rightarrow \mathcal{Y})$ is the support of the set of all items that appear in the rule divided by the support of the antecedent of the rule, i.e. the percent of transactions in \mathcal{D} that contain \mathcal{X} also contain \mathcal{Y} . The equation for that is:

$$\text{conf}(\mathcal{R}) = \frac{\text{sup}(\mathcal{X}, \mathcal{Y})}{\text{sup}(\mathcal{X})} * 100\%$$

Finally, the support of rule $\mathcal{R} = (\mathcal{X} \Rightarrow \mathcal{Y})$ is the percentage of those transactions in \mathcal{D} that contain $\mathcal{X} \cup \mathcal{Y}$.

Given a set of transactions \mathcal{D} , the *problem of mining association rules* is to generate all association rules that have support greater than the *minsup* and confidence greater than the user minimum confidence (*minconf*). Mining of association rules can be decomposed into two distinct phases:

- (a) find frequent itemsets, i.e. those that pass the *minsup*, and
- (b) generate the confident, or interesting rules which contain the frequent itemsets.

The first problem of finding the frequent itemsets, which is referred to as Frequent Itemset Mining (FIM), has been identified as the most complex of the two. A naive solution would be to form all possible itemsets, and then calculate their support by going once through the data. This of course would not be computationally feasible, as it would result in 2^m itemsets, i.e. exponential in m , where m is the number of items in the database, resulting in thousands or millions of itemsets even for reasonable m values. On the other hand, the search space of all association rules contains exactly $3^{|m|}$ different rules, however, when the frequent itemsets are given, this space immediately reduces in size. Specifically, for every frequent itemset I , there exists at most $2^{|I|}$ rules of the form $\mathcal{X} \Rightarrow \mathcal{Y}$ such that $\mathcal{X} \cup \mathcal{Y} = I$.

The main idea behind all algorithms for mining frequent (or large) itemsets, can be described as follows: in the first pass over the data, count the support of itemsets of length 1, and determine which have minimum support (large or frequent). Then, in the next data pass, use only those itemsets that were found to be large or frequent in the previous pass, in order to generate new potentially large itemsets, called *candidate* itemsets, and count the support for these candidate itemsets. This continues with the candidate itemsets that pass the minimum support being used as a seed for the next pass, and so on. This loop is ended when no new large itemsets can be found.

Most research on Association Rule Mining has been based on the seminal paper by Agrawal,²⁰ where the Apriori algorithm was introduced. The difference of the Apriori algorithm from the prior research is in the way it generates the aforementioned candidate itemsets, and also in the way the candidates are counted in each pass.²⁰ The central Apriori property is: “If A is a frequent itemset, then every subset of A is a frequent itemset”. What ensues from this statement is that, once a set of items has been found infrequent, i.e. does not pass the minimum support threshold, not only can it be pruned, but *all* the itemsets that contain it can be pruned as well, as none of them can be frequent either. For example, if set ‘AB’ has been found to be non-frequent, then itemsets ‘ABC’, ‘ABD’, ‘ABCD’, etc. that contain it, cannot be frequent, so they too can be pruned. The pseudocode for Apriori from²⁰ is designated as Algorithm 1:

Step **apriori-gen** (see Algorithm 1 pseudocode) is comprised of 2 steps: *Join and Prune*. It returns a superset of the set of all large ($k-1$) itemsets. Essentially, two itemsets, I and J , which contain the same $k-2$ elements except for one item are combined. Suppose the two items that differ between set I and set J are i_1 and i_2 , then I and J can be joined to form a new set if and only if $i_1 < i_2$.

```

1.1  $L_1 = \{\text{singleton itemsets}\};$ 
1.2 for  $k \leftarrow 2$  to  $L_{k-1} \neq \emptyset$  do
1.3   |  $C_k = \text{Apriori-gen}(L_{k-1});$ 
1.4 end
1.5 forall transaction  $t \in \mathcal{D}$  do
1.6   |  $C_t = \text{subset}(C_k, t);$ 
1.7   | forall candidates  $c \in C_t$  do
1.8     |  $c.\text{count}++;$ 
1.9   | end
1.10  |  $L_k = \{c \in C_k | c.\text{count} > \text{minsup}\};$ 
1.11 end

```

Algorithm 1: Pseudocode for Apriori Algorithm

```

2.1 forall itemsets  $c \in C_k$  do
2.2   | forall  $(k-1)$ -subsets  $s$  of  $c$  do
2.3     | if  $s \notin L_{k-1}$  then
2.4       | delete  $c$  from  $C_k;$ 
2.5     | end
2.6   | end
2.7 end

```

Algorithm 2: Pseudocode for the pruning step of the Apriori Algorithm

Next, we prune all itemsets c in C_k such that some $(k-1)$ -subset of c is not in L_{k-1} (see Algorithm 2).

As an example, let L_3 be $\{123\}, \{124\}, \{134\}, \{135\}, \{234\}$. After the join step, we have that $C_4 = \{\{1234\}, \{1345\}\}$. The prune step will delete $\{1345\}$ because the itemset $\{145\}$ is not in L_3 , so we will be left with only $\{1234\}$ in C_4 .

3.1.3 Apriori Computational Complexity

The most computationally complex step is that of counting the support (code lines 4 – 8 of Algorithm 1), as each candidate must be compared against every transaction data, and candidate generation checks the entire database transaction set. Therefore for one single generation the running time is $O(|T| * |C| * |t|)$, if the subset function can be implemented in constant time $|t|$. The improvement exhibited by Apriori is in the number of candidates C that is generated and counted in each pass: after the third pass ($k > 3$), C_k gets really close to L_k , which means that for $k > 3$ almost all candidate itemsets turn out to be frequent itemsets.

3.2 Non-Derivable Itemset Algorithm

3.2.1 Overview

The Non-Derivable (NDI) algorithm³⁵ is generally based on the Apriori algorithm. As explained in the previous section, the Apriori algorithm returns the frequent itemsets given a user-specified minimum support. The NDI algorithm employs certain deduction rules for the support of each itemset to return a compact or condensed representation for all frequent itemsets (FIs). In other words, instead of storing each FI, the NDI stores a representative subset of the set of all FIs. The purpose behind this is to allow a more efficient and scalable traversal of the data, which is what we propose to exploit in this paper.

3.2.2 Deduction Rules

The NDI algorithm aims to improve the efficiency of a Frequent Itemset Mining algorithm. This is done by pruning itemsets whose support can be deduced or derived from their subsets: these sets are called *derivable*. NDI makes use of rules to derive the support of itemsets in order to find and prune the derivable itemsets.

It is important to know what a *generalized itemset* is, before understanding the deduction rules that are used in NDI. A generalized itemset is a set of items and negations of items. For example, let $G = \{abc\}$. The support

of G in this case is the number of transactions where items a and b occur together while item c is not present in those transactions.

The intuition behind the deduction rules comes from the inclusion-exclusion principle. For example, consider the itemset $t = \{\overline{abc}\} \subset \mathcal{D}$ where \mathcal{D} is a database. The itemset set t can be represented as the shaded portion of Figure 2.

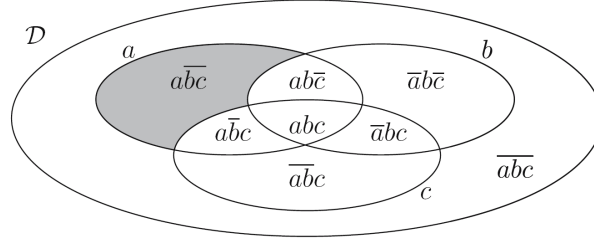


Figure 2. Example of inclusion-exclusion principle for $\{\overline{abc}\}$

Figure 2 can be summarized as: $\text{supp}(\{\overline{abc}\}) = \text{supp}(a) - \text{supp}(ab) - \text{supp}(ac) + \text{supp}(abc)$.

Essentially, when determining the support of t , the support of abc is subtracted twice as we subtract the support of ab and ac . Thus, we add it back into the calculation for the support of t .

Let I be a subset of the set of all itemsets \mathcal{I} . Since the minimum number of times that an itemset can occur is zero, the $\text{supp}(I) \geq 0$. Using the generalized itemsets of I , it is possible to compute an interval of possible values for the $\text{supp}(I)$.

The following theorem will be used to generate the rules for I . The rule generated by using the following theorem is denoted $R_X(I)$.

THEOREM 3.1.

Let $X \subseteq I \subseteq \mathcal{I}$, and $Y = I/X$, then,

$$\text{supp}(I) \leq \sum_{X \subseteq J \subset I} (-1)^{|I/J|+1} \text{supp}(J) \quad \text{if } |I/X| \text{ is odd.}$$

$$\text{supp}(I) \geq \sum_{X \subseteq J \subset I} (-1)^{|I/J|+1} \text{supp}(J) \quad \text{if } |I/X| \text{ is even.}$$

In order to get an intuitive understanding of this theorem, let $I = \{a, b, c\}$. Since X must be a subset of I , we can let $X = \{a\}$. In this case, $|I/X| = |\{a, b, c\}/\{a\}| = |\{b, c\}| = 2$ which is even. Thus, by the theorem above, the X chosen will generate a rule that is a lowerbound on the support of I .

The fact that $|I/X|$ is even or odd will affect the power of -1 . The requirement on J is that it be a proper subset of I while always containing X . Thus, X will determine which subsets J are permissible and thus the cardinality $|I/J|$.

An example is provided below:

The deduction rules for the itemset $I = \{a, b, c\}$ are shown below.

The rules generated above are $R_X(abc)$ for X , respectively, abc , ab , ac , bc , a , b , c , and \emptyset . These deduction rules permit the computation of the greatest lower bound, denoted $\mathbf{LB}(I, \mathcal{D})$, and the least upper bound, denoted $\mathbf{UB}(I, \mathcal{D})$, for the support of I and a database \mathcal{D} . In this case, $[\mathbf{LB}(I), \mathbf{UB}(I)] = [0, 1]$.

<i>tid</i>	<i>Items</i>
1	<i>a</i>
2	<i>b</i>
3	<i>c</i>
4	<i>a, b</i>
5	<i>a, c</i>
6	<i>b, c</i>
7	<i>a, b, c</i>

Figure 3. A database of transactions

$$\begin{aligned}
R_{abc} &: \text{supp}(I) \geq 0 \\
R_{ab} &: \text{supp}(I) \leq \text{supp}(ab) = 2 \\
R_{ac} &: \text{supp}(I) \leq \text{supp}(ac) = 2 \\
R_{bc} &: \text{supp}(I) \leq \text{supp}(bc) = 2 \\
R_a &: \text{supp}(I) \geq \text{supp}(ab) + \text{supp}(ac) - \text{supp}(a) = 0 \\
R_b &: \text{supp}(I) \geq \text{supp}(ab) + \text{supp}(bc) - \text{supp}(b) = 0 \\
R_c &: \text{supp}(I) \geq \text{supp}(ac) + \text{supp}(bc) - \text{supp}(c) = 0 \\
R_\emptyset &: \text{supp}(I) \leq \text{supp}(ab) + \text{supp}(ac) + \text{supp}(bc) - \text{supp}(a) - \\
&\quad \text{supp}(b) - \text{supp}(c) + \text{supp}(\emptyset) = 1
\end{aligned}$$

Figure 4. Calculating the support interval for the itemset $I = \{a, b, c\}$

3.2.3 Pseudocode for the NDI Algorithm

We give a brief description of the NDI algorithm in Algorithm 3. As stated before, the NDI algorithm receives a database and minimum support as input. In order to find the frequent itemsets, the NDI algorithm uses the Apriori algorithm. The Apriori algorithm generates candidate sets that are then pruned in order to find the frequent itemsets. Once the frequent itemsets are found, the NDI algorithm determines whether each frequent itemset is derivable or not, by checking whether the least upper bound and greatest lower bound are equal. If the two bounds are equal, then the itemset is derivable and is thus not stored in $\text{NDIRep}(\mathcal{D}, \sigma)$. Then, the NDI algorithm calls the Apriori algorithm again and the process is repeated until no more candidate sets can be generated by the Apriori. It then outputs the NDI set, $\text{NDIRep}(\mathcal{D}, \sigma)$. We note that in,¹² they state that the cardinality of $\text{NDIRep}(\mathcal{D}, \sigma)$ is bounded by the logarithm of the size of the database.

The NDI algorithm uses the variable $\text{Gen}\{\}$ in order to store the itemsets (from the data set) that will be passed as input into the Apriori algorithm. It uses the variable $\text{Pre}C_{\ell+1}$ to store the itemsets generated by the Apriori algorithm. It uses the variable C_ℓ to store the candidate sets generated by the Apriori algorithm that are non-derivable. It uses the variable F_ℓ to store the non-derivable itemsets in C_ℓ that are frequent.

3.3 Outlier Detection based on Frequent Itemsets (Apriori-OD)

The pseudocode for an outlier detection method based on frequent itemsets is shown in Algorithm 4. The first step is to mine the frequent sets from the data. Using this information, one can define an outlier factor or score for each data point, x_i , where $i = 1 \dots n$, and n is the total number of data points. The algorithm goes over each data point in order to check the subsets of the point. For each frequent set found in x_i , the anomaly or outlier score corresponding to this data point is updated, based on the information obtained from the frequent itemsets contained in x_i . Finally, the k data points with the lowest score are returned. The equation for the outlier score is given in 3.1:

$$\text{FIMODScore}(x_i) = \sum_{f \subset x_i, \text{supp}(f) \geq \sigma} \frac{\text{sup}(f)}{|D|} \quad (3.1)$$

```

input :  $\mathcal{D}$ , threshold  $\sigma$ 
output:  $\text{NDIRep}(\mathcal{D}, \sigma)$ 

3.1  $\ell := 1$ ;  $\text{NDIRep} := \{\}$ ;  $C_1 := \{\{i\} | i \in \mathcal{I}\}$ ;
3.2 forall  $I \in C_1$  do
3.3   |  $I.l := 0$ ;  $I.u := |\mathcal{D}|$ ;
3.4 end
3.5 while  $C_\ell$  is not empty do
3.6   | Count the supports of all candidates in  $C_\ell$  in one pass;
3.7   |  $F_\ell := \{I \in C_\ell | \text{supp}(I, \mathcal{D}) \geq \sigma\}$ ;
3.8   |  $\text{NDIRep} := \text{NDIRep} \cup F_\ell$ ;
3.9   |  $\text{Gen} := \{\}$ ;
3.10  forall  $I \in F_\ell$  do
3.11    | if  $\text{supp}(I, \mathcal{D}) \neq I.l$  and  $\text{supp}(I, \mathcal{D}) \neq I.u$  then
3.12      | |  $\text{Gen} := \text{Gen} \cup \{I\}$ ;
3.13    | end
3.14  end
3.15   $\text{Pre}C_{\ell+1} := \text{AprioriGenerate}(\text{Gen})$ ;
3.16   $C_{\ell+1} := \{\}$ ;
3.17  forall  $I \in \text{Pre}C_{\ell+1}$  do
3.18    | Compute bounds  $[l, u]$  on the support of  $I$ ;
3.19    | if  $l \neq u$  and  $u \geq \sigma$  then
3.20      | |  $I.l := l$ ;  $I.u := u$ ;  $C_{\ell+1} := C_{\ell+1} \cup \{I\}$ ;
3.21    | end
3.22  end
3.23   $\ell := \ell + 1$ ;
3.24 end

```

Algorithm 3: Pseudocode for the NDI Code

```

input : Data set  $\mathcal{D}$ ,  $k$ : target number of outliers, Minimum support:  $\sigma$ 
output:  $k$  detected outliers

4.1  $G = \text{Return Frequent Itemsets}(\mathcal{D}, \sigma)$ ;
4.2 foreach  $x_i \in \mathcal{D}, i \leftarrow 0$  do
4.3   | foreach frequent itemset  $f$  in  $G$  do
4.4     | | if  $f \subset x_i$  then
4.5       | | Update  $\text{FIMODScore}[i]$ ;
4.6     | | end
4.7   | end
4.8 end

```

Algorithm 4: Pseudocode for OD based on FIs

3.4 Outlier Detection based on NDI Frequent Itemsets (NDI-OD)

Given a database \mathcal{D} and the $\text{NDIRep}(\mathcal{D}, \sigma)$ set, we implement an algorithm to find outliers. For each data point (or transaction) in the database, x_i , we assign an outlier score to a point x_i based on the itemsets in $\text{NDIRep}(\mathcal{D}, \sigma)$ that are subsets of x_i . So every time an itemset in $\text{NDIRep}(\mathcal{D}, \sigma)$ is found to be a subset of a data point from the dataset, we update the score for the data point accordingly. Thus, only those frequent itemsets in $\text{NDIRep}(\mathcal{D}, \sigma)$ are used to find outliers (and so eliminating the need to use all the frequent itemsets as in the previous section). Finally, the algorithm will find the k lowest scores and label the respective data points as outliers, where k is a positive integer provided as input. In Section 3.5.3 we discuss why the NDIRep is suitable for the outlier detection problem.

In our algorithm, we use a data structure $NDIODScore[]$ to store the score of each data point. We also use a data structure $outliers[]$ to store the k outliers. The function $getScore()$ takes a data point (or transaction) x_i from our database as a parameter. It returns the score value of that data point. The variable $currentscore$ stores the value of a data point's score. We provide pseudocode of the NDI-OD algorithm in Algorithm 5. As stated at the beginning of Section 3, an outlier may be a point that contains single values that are infrequent, or the co-occurrence of 2 or more of its values is infrequent. The outlier score below will assign a high score to points with many frequent values or frequent subsets. The score will assign 0 to a point that does not contain any frequent subsets. This score is similar to the one defined by He et al.²³ The equation for the outlier score is given in 3.2 where I is a NDI frequent set instead of f which is a frequent set in 3.1:

$$NDIODScore(x_i) = \sum_{I \subset x_i, sup(I) \geq \sigma, I \in NDIRep} \frac{sup(I)}{|D|} \quad (3.2)$$

```

input : NDIRep( $\mathcal{D}, \sigma$ ),  $k$ : target number of outliers
output:  $k$  outliers

5.1 NDIRep = Return NDI sets ( $\mathcal{D}, \sigma$ );
5.2 NDIODScore[ $|\mathcal{D}|$ ] =  $\emptyset$ ;
5.3 outliers[ $k$ ] =  $\emptyset$ ;
5.4 foreach  $x_i \in \mathcal{D}$ ,  $i \leftarrow 0$  do
5.5   | foreach  $I \in NDIRep(\mathcal{D}, \sigma)$  do
5.6   |   | if  $I \subset x_i$  then
5.7   |   |   | update NDIODScore[ $i$ ];
5.8   |   |   end
5.9   |   end
5.10 end
5.11 for  $i \leftarrow 0, i < k$  do
5.12 | outliers[ $i$ ] =  $\mathcal{D}[i]$ ;
5.13 end
5.14 foreach  $x_i \in \mathcal{D}$ ,  $i \leftarrow 0$  do
5.15 | currentScore = getScore( $x_i$ );
5.16 | for  $j \leftarrow 0, j < k$  do
5.17 |   | if currentScore < getScore(outliers[ $j$ ]) then
5.18 |   |   | outlier[ $j$ ] =  $x_i$ ;
5.19 |   |   end
5.20 |   end
5.21 end

```

Algorithm 5: Pseudocode for OD based on NDIs

3.5 Comparative Discussion of NDI Algorithm and Apriori Algorithm

3.5.1 Overview

In this section we summarize the advantages the NDI algorithm has over the Apriori algorithm in finding outliers. Specifically, we argue that the NDI-OD algorithm provides a more efficient and scalable approach than the Apriori-OD algorithm.

3.5.2 Apriori Principle

We begin by stating the Apriori Principle:

THEOREM 3.2.

(Apriori Principle). If an itemset is frequent, then all of its subsets must also be frequent.

This principle can be easily illustrated with an example. Suppose $\{a, b, c\}$ is a frequent itemset. Any transaction that contains $\{a, b, c\}$ must also contain its subsets, $\{a, b\}$, $\{b, c\}$, $\{b, c\}$, $\{a\}$, $\{b\}$, $\{c\}$. Thus, its subsets must also be frequent.

The converse to the Apriori Principle is that if an itemset is infrequent, then all of its supersets are also infrequent. This property is used in the Apriori algorithm in order to prune infrequent sets and thus to reduce the computational costs. Therefore, this principle helps reduce the number of candidate sets generated and counted by the algorithm.

3.5.3 Some Important Properties of NDI

We state an important property of NDI below, the proof is provided in:^{12,35}

1. (Monotonicity) If I and J are itemsets, $J \subseteq I$, and J is derivable, then I is derivable.

Since the NDI algorithm uses the Apriori algorithm, it is implicitly using the Apriori Principle. In addition, given the property stated above, the NDI algorithm is capable of pruning the data even more. Note that in the NDI algorithm, the candidate set is pruned before it is checked for frequency. That is, we prune all itemsets that are derivable. By the property stated above, this means we are also removing supersets of these derivable itemsets so that they are not considered in the next iteration of the loop. Then, each itemset in the reduced candidate set is checked for its frequency.

When detecting outliers using frequent itemsets generated by the Apriori algorithm, we are scanning through a much bigger amount of sets than that generated by the NDI algorithm. As a matter of fact, Calders and Goethals showed^{12,35} that with certain datasets, the number of frequent itemsets generated by the Apriori algorithm was so large that they had to terminate execution of the algorithm. Using only the sets generated by the NDI algorithm will not greatly affect the accuracy of the outlier detection technique. This is because the sets not maintained by NDI, i.e. the derivable sets, do not provide more information than their subsets. In fact, most of the derivable sets, especially those of longer length, are spurious combinations of extremely frequent items. This implies that it is not necessary for the outlier score to add up the support information of every possible combination of these sets because it will be adding up the same support a large number of times for most of the normal points.

Thus, even though we are not using all of the frequent itemsets, the NDI-OD algorithm will provide a good trade-off between accuracy and efficiency. This is acceptable given the large number of frequent itemsets that can be generated by Apriori-like algorithms, while maintaining a comparable accuracy.

4. EXPERIMENTS

4.1 Experimental Setup

We conducted all our experiments on a workstation with a Pentium 2.61 GHz processor and 2 GB of RAM. We used the NDI implementation available online by Goethals et al.²⁶

4.1.1 Datasets

Wisconsin Breast Cancer: This dataset has 699 points and 9 attributes.³⁶ The attributes in this set are computed from a digitized image of a fine needle aspirate (FNA) of a breast mass. They describe characteristics of the cell nuclei present in the image (e.g. radius, texture, etc). Each record is labeled as either benign or malignant. The original dataset contains 569 records, of which 357 benign and 212 malignant. As previously,⁷ we only kept every sixth malignant record, resulting in 39 outliers (malignant) and 444 non-outliers (benign).

KDD1999 Dataset: This dataset³⁶ represents connections to a military computer network and multiple intrusions and attacks by unauthorized users. The raw binary TCP data were processed into features such as connection duration, protocol type, number of failed logins, etc. There are three available datasets: a training set, a test set, and a set with 10% of the training set. For our experiments we used the 10% of the training

set. All the KDDCup 1999 sets contain 33 continuous attributes and 8 categorical attributes. We discretized the continuous attributes using equal-width discretization with 20 intervals. Due to the large number of attacks in these datasets, we preprocessed the datasets such that attack points are around 2% of the dataset, and these points were chosen randomly from the collection of attack data-points. Network traffic packets tend to occur in bursts for certain intrusions. While we preserved the proportions of the various attacks in the data, we selected our outlier points at random without necessarily preserving the length of the bursts. We followed the same concept as by Otey et al.,⁵ and detected bursts of packets in the data set. Our processed dataset contains 98,587 data points and 1,309 attacks. We removed 2 columns (attributes) that contained the same categorical value for all records as it would not affect the detection of outliers. The resulting dataset has 39 columns and 1179 distinct categorical values (single items).

4.1.2 Evaluation

We evaluate both algorithms, Apriori-OD and NDI-OD, based on the following two measures:

- *Correct detection rate*, which is the number of outliers correctly identified by each approach as outliers:

$$CD = \frac{\text{Number of outliers correctly detected as outliers}}{\text{Total number of outliers in dataset}} \quad (4.1)$$

- *False alarm rate*, reflecting the number of normal points erroneously identified as outliers

$$FA = \frac{\text{Number of normal points incorrectly identified as outliers}}{\text{Total number of normal points in dataset}} \quad (4.2)$$

We also compare the running time performance of the two algorithms using the same datasets.

4.2 Results

We ran several experiments with various values for *minsup*, and *k*, the desired number of outliers. The values for *k* were selected to be close to the actual number of outliers in the dataset (i.e. 39 for the Breast cancer dataset and 1309 for the KDD 1999 dataset). The *minsup* is represented as a percentage of the total number of data points in the database. Table 1 contains the accuracy results of Apriori-OD and NDI-OD using the Breast Cancer Dataset and Table 2 contains the accuracy results on the KDD 1999 Dataset. Tables 3 and 4 show the runtime performance for NDI-OD versus Apriori-OD for Breast Cancer and KDD 1999 datasets respectively. Specifically, Table 2 contains the actual number of outliers detected by each algorithm, the accuracy percentage and the false alarm percentage. We did not observe a difference in accuracy or false alarm rates for the Breast Cancer Dataset between Apriori-OD and NDI-OD. For this dataset, *minsup* less than or equal to 20% gives the best accuracy and false alarm rates.

Table 2 contains the accuracy and false alarm rates for the KDD 1999 dataset for *minsup* equal to 98,000 or 99.4%. The reason for not including results for other *minsup* values is that Apriori generated a very high number of frequent itemsets and we had to terminate the program. For example, for *minsup* equal to 90%, Apriori was generating more than 85,000 total frequent itemsets while at the 5-th pass. Table 2 contains the average accuracy based on detected bursts of attack. Essentially, if we detect one point in a burst of packets as an outlier we mark all points in the burst as outliers and we count the burst as detected, similar to Otey et al.⁵ As we can see in Table 2, Apriori-OD detects more bursts for *k* less than 2000. NDI-OD has the same accuracy with Apriori-OD for *k* equal to 2000, while the FA rate is still very low for both algorithms (1.4%).

The significant advantage of the NDI-OD technique versus the Apriori-OD with respect to running time is apparent in Tables 3 and 4. For instance, for *minsup* equal to 98,000 or 99.4% NDI-OD took 24 seconds to detect the outliers in the KDDCup 1999 dataset, while Apriori-OD needed 19 minutes to accomplish the same task. This is because NDI generated only 177 non-derivable frequent sets versus the 6,349 frequent sets generated by Apriori. Figure 5 contains a pictorial representation of the runtime advantage of NDI-OD for the KDD 1999 dataset and *minsup* = 99.4%.

<i>minsup</i>	<i>k</i>	NDI-OD			Apriori-OD		
		Outliers Detected	<i>CD</i>	<i>FA</i>	Outliers Detected	<i>CD</i>	<i>FA</i>
5%	39	32	0.8205	0.0158	32	0.8205	0.0158
	48	35	0.8974	0.0293	35	0.8974	0.0293
	56	39	1.0000	0.0383	39	1.0000	0.0383
10%	39	31	0.7949	0.0180	31	0.7949	0.0180
	48	35	0.8974	0.0293	35	0.8974	0.0293
	56	39	1.0000	0.0383	39	1.0000	0.0383
20%	39	30	0.7692	0.0203	30	0.7692	0.0203
	48	36	0.9231	0.0270	36	0.9231	0.0270
	56	39	1.0000	0.0383	39	1.0000	0.0383
30%	39	30	0.7692	0.0203	30	0.7692	0.0203
	48	34	0.8718	0.0315	34	0.8718	0.0315
	56	38	0.9744	0.0405	38	0.9744	0.0405
50%	39	29	0.7436	0.0225	29	0.7436	0.0225
	48	33	0.8462	0.0338	33	0.8462	0.0338
	56	38	0.9744	0.0405	38	0.9744	0.0405
75%	39	28	0.7179	0.0248	28	0.7179	0.0248
	48	33	0.8462	0.0338	33	0.8462	0.0338
	56	34	0.8718	0.0495	34	0.8718	0.0495

Table 1. Accuracy for NDI-OD versus Apriori-OD for Breast Cancer dataset varying *minsup* and *k* (Actual Outliers: 39).

<i>k</i>	NDI-OD		Apriori-OD	
	<i>CD</i>	<i>FA</i>	<i>CD</i>	<i>FA</i>
1309	0.53	0.0092	0.65	0.0075
1500	0.64	0.0109	0.65	0.0095
1720	0.64	0.0118	0.65	0.0117
2000	0.70	0.0141	0.70	0.0141

Table 2. Accuracy for NDI-OD versus Apriori-OD for *KDD1999* dataset (*minsup* = 99.4%; actual outliers: 1309).

<i>minsup</i>	NDI-OD	Apriori-OD
5%	0.234	0.516
10%	0.203	0.360
20%	0.141	0.219
30%	0.094	0.141
50%	0.078	0.109
75%	0.031	0.032

Table 3. Runtime performance (in seconds) for NDI-OD versus Apriori-OD for Breast Cancer dataset (*k* = 56).

<i>minsup</i>	Time		Generated sets	
	NDI-OD	Apriori-OD	NDI-OD	Apriori-OD
99.4%	24	1156	177	6349
90%	192	--	1513	> 85000
75%	369	--	2845	--
50%	878	--	7262	--
10%	1325	--	16818	--

Table 4. Runtime performance (seconds) and Generated Sets for NDI-OD versus Apriori-OD for *KDD1999* dataset (*k* = 2*K*)

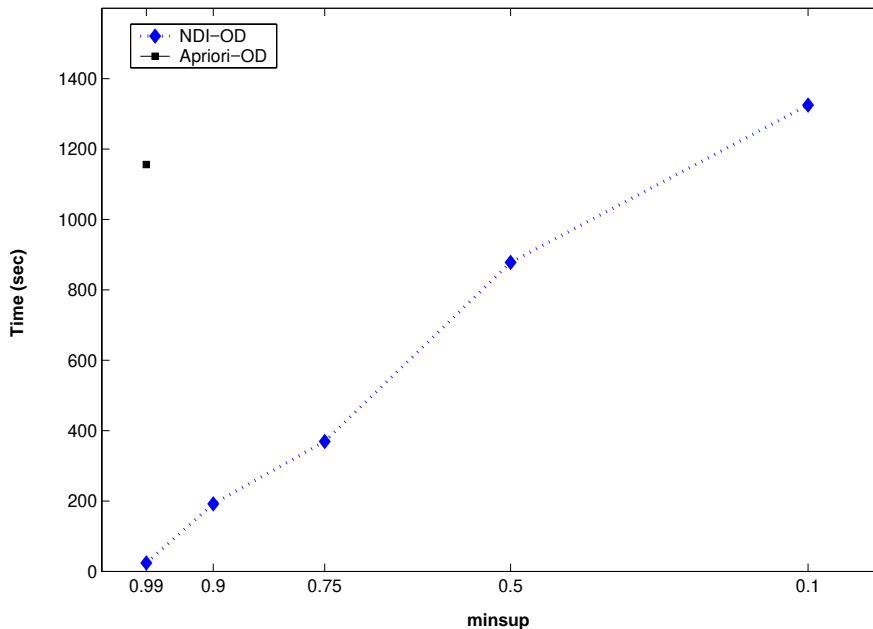


Figure 5. Runtime performance (seconds) for NDI-OD versus Apriori-OD for *KDD1999* dataset

It is noteworthy that the purpose of experimenting with the *KDD1999* dataset is to contrast the accuracy of Apriori-OD and NDI-OD for a large high-dimensional categorical dataset. The detection accuracy and performance might improve slightly by using a different discretization scheme for the continuous features. Alternatively, methods intended for mixed attributes^{5,9} have been shown to have higher accuracy and runtime performance for the *KDD1999* set (e.g. 81% average detection accuracy for the same dataset).

5. CONCLUSIONS

Outlier Detection techniques for categorical datasets have employed Frequent Itemsets,²⁰ in order to identify those points containing irregular patterns. In this paper, we explored applying Non-Derivable Itemsets³⁵ to Outlier Detection to improve the speed and scalability especially for dense datasets. Our experiments show that our proposed method, NDI-OD, performs very well compared to the Apriori-based OD technique, and provides significant runtime advantages. Specifically, for one of the datasets, *KDDCup 1999*,³⁶ Apriori-OD can only finish execution for a very high *minsup* value due to the amount of generated frequent sets, while NDI-OD finishes execution quickly, even for low *minsup* values. Future research includes further improving the speed and scalability of NDI-OD and extending for distributed datasets.

ACKNOWLEDGMENTS

This material is based upon work/research supported in part by the National Science Foundation under Grant No. 0647120 and Grant No. 0647018. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

REFERENCES

- [1] Bolton, R. and Hand, D., “Statistical fraud detection: A review,” *Statistical Science* **17**(3), 235–255 (2002).
- [2] Dokas, P., Ertöz, L., Kumar, V., Lazarevic, A., Srivastava, J., and Tan, P., “Data mining for network intrusion detection,” *Proc. NSF Workshop on Next Generation Data Mining* (2002).

- [3] Lee, W., Stolfo, S., and Mok, K., “A data mining framework for building intrusion detection models,” *IEEE Symposium on Security and Privacy* **132** (1999).
- [4] Hawkins, D., [*Identification of Outliers*], Chapman & Hall (1980).
- [5] Otey, M., Ghoting, A., and Parthasarathy, S., “Fast Distributed Outlier Detection in Mixed-Attribute Data Sets,” *Data Mining and Knowledge Discovery* **12**(2), 203–228 (2006).
- [6] Hays, C., “What Wal-Mart knows about customers habits,” *The New York Times* (2004).
- [7] Koufakou, A., Ortiz, E., Georgiopoulos, M., Anagnostopoulos, G., and Reynolds, K., “A Scalable and Efficient Outlier Detection Strategy for Categorical Data,” *19th IEEE International Conference on Tools with Artificial Intelligence, ICTAI 2007*, **2** (2007).
- [8] He, Z., Xu, X., Huang, J., and Deng, S., “FP-Outlier: Frequent Pattern Based Outlier Detection,” *Computer Science and Information System* **2**(1), 103–118 (2005).
- [9] Koufakou, A., Georgiopoulos, M., and Anagnostopoulos, G., “Detecting Outliers in High-Dimensional Datasets with Mixed Attributes,” *2008 International Conference on Data Mining* (2008).
- [10] Zaki, M. and Hsiao, C., “Efficient Algorithms for Mining Closed Itemsets and Their Lattice Structure,” *IEEE Transactions on Knowledge and Data Engineering*, 462–478 (2005).
- [11] Agarwal, R., Aggarwal, C., and Prasad, V., “Depth first generation of long patterns,” *Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining*, 108–118 (2000).
- [12] Calders, T. and Goethals, B., “Non-derivable itemset mining,” *Data Mining and Knowledge Discovery* **14**, 171–206 (February 2007).
- [13] Barnett, V., [*Outliers in Statistical Data*], John Wiley & Sons New York (1978).
- [14] Knorr, E., Ng, R., and Tucakov, V., “Distance-based outliers: algorithms and applications,” *VLDB Journal, International Journal on Very Large Data Bases* **8**(3), 237–253 (2000).
- [15] Bay, S. and Schwabacher, M., “Mining distance-based outliers in near linear time with randomization and a simple pruning rule,” *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, 29–38 (2003).
- [16] Breunig, M., Kriegel, H., Ng, R., and Sander, J., “LOF: identifying density-based local outliers,” *ACM SIGMOD Record* **29**(2), 93–104 (2000).
- [17] Tax, D. and Duin, R., “Support Vector Data Description,” *Machine Learning* **54**(1), 45–66 (2004).
- [18] Hawkins, S., He, H., Williams, G., and Baxter, R., “Outlier Detection Using Replicator Neural Networks,” *Proc. 4th International Conference on Data Warehousing and Knowledge Discovery*, 170–180 (2002).
- [19] He, Z., Xu, X., and Deng, S., “A Fast Greedy Algorithm for Outlier Mining,” *Advances in Knowledge Discovery and Data Mining* (2005).
- [20] Agrawal, R. and Srikant, R., “Fast algorithms for mining association rules in large databases,” *Proceedings of the 20th International Conference on Very Large Data Bases*, 487–499 (1994).
- [21] Brin, S., Motwani, R., Ullman, J., and Tsur, S., “Dynamic itemset counting and implication rules for market basket data,” *Proc. 1997 ACM SIGMOD international conference on Management of data*, 255–264 (1997).
- [22] Park, J., Chen, M., and Yu, P., “An effective hash-based algorithm for mining association rules,” *Proceedings of the 1995 ACM SIGMOD international conference on Management of data*, 175–186 (1995).
- [23] Han, J., Pei, J., Yin, Y., and Mao, R., “Mining Frequent Patterns without Candidate Generation: A Frequent-Pattern Tree Approach,” *Data Mining and Knowledge Discovery* **8**(1), 53–87 (2004).
- [24] Bodon, F., “A fast apriori implementation,” *Proceedings of the IEEE ICDM Workshop on Frequent Itemset Mining Implementations* **90** (2003).
- [25] Borgelt, C., “Efficient Implementations of Apriori and Eclat,” *Workshop of Frequent Item Set Mining Implementations* (2003).
- [26] Goethals, B., *NDI Implementation*. <http://www.adrem.ua.ac.be/~goethals/software> (2005).
- [27] Boullicaut, J., Bykowski, A., and Rigotti, C., “Approximation of frequency queries by means of free-sets,” *Proc. PKDD Int. Conf. Principles of Data Mining and Knowledge Discovery*, 75–85 (2000).
- [28] Xiong, H., Pandey, G., Steinbach, M., and Kumar, V., “Enhancing Data Analysis with Noise Removal,” *IEEE Transactions on Knowledge and Data Engineering*, 304–319 (2006).
- [29] Xiong, H., Tan, P., and Kumar, V., “Hyperclique pattern discovery,” *Data Mining and Knowledge Discovery* **13**(2), 219–242 (2006).

- [30] Hu, T., Xu, Q., Yuan, H., Hou, J., and Qu, C., “Hyperclique Pattern Based Off-Topic Detection,” *Lecture Notes in Computer Science* **4505**, 374 (2007).
- [31] Wang, J. and Karypis, G., “On efficiently summarizing categorical databases,” *Knowledge and Information Systems* **9**(1), 19–37 (2006).
- [32] Jea, K. and Chang, M., “Discovering frequent itemsets by support approximation and itemset clustering,” *Data & Knowledge Engineering* **65**(1), 90–107 (2008).
- [33] Wong, P., Whitney, P., and Thomas, J., “Visualizing association rules for text mining,” *Proceedings of IEEE Symp. on Information Visualization*, , 120–123 (1999).
- [34] Zhang, Z., Wu, W., and Huang, Y., “Mining dynamic interdimension association rules for local-scale weather prediction,” *Proceedings of the 28th Annual International Computer Software and Applications Conference, COMPSAC 2004*, **2** (2004).
- [35] Calders, T. and Goethals, B., “Mining all non-derivable frequent itemsets,” *Proc. PKDD Int. Conf. Principles of Data Mining and Knowledge Discovery* , 74–85 (2002).
- [36] Blake, C. and Merz, C., *UCI repository of machine learning databases*. <http://archive.ics.uci.edu/ml/> (1998).