Knowl Inf Syst
DOI 10.1007/s10115-010-0343-7

REGULAR PAPER

Non-derivable itemsets for fast outlier detection in large high-dimensional categorical data

Anna Koufakou · Jimmy Secretan · Michael Georgiopoulos

Received: 7 June 2009 / Revised: 30 April 2010 / Accepted: 4 September 2010

Abstract Detecting outliers in a dataset is an important data mining task with many applications, such as detection of credit card fraud or network intrusions. Traditional methods assume numerical data and compute pair-wise distances among points. Recently, outlier detection methods were proposed for categorical and mixed-attribute data using the concept of Frequent Itemsets (FIs). These methods face challenges when dealing with large high-dimensional data, where the number of generated FIs can be extremely large. To address this issue, we propose several outlier detection schemes inspired by the well-known condensed representation of FIs, *Non-Derivable Itemsets (NDIs)*. Specifically, we contrast a method based on frequent NDIs, *FNDI-OD*, and a method based on the negative border of NDIs, *NBNDI-OD*, with their previously proposed FI-based counterparts. We also explore outlier detection based on Non-Almost Derivable Itemsets (NADIs), which approximate the NDIs in the data given a δ parameter. Our proposed methods use a far smaller collection of sets than the FI collection in order to compute an anomaly score for each data point. Experiments on real-life data show that, as expected, methods based on NDIs and NADIs offer substantial advantages in terms of speed and scalability over FI-based Outlier Detection method. What is significant is that NDI-based methods exhibit similar or better detection accuracy compared to the FI-based methods, which supports our claim that the NDI representation is especially well-suited for the task of detecting outliers. At the same time, the NDI approximation scheme, NADIs is shown to exhibit similar accuracy to the NDI-based method for various δ values and further runtime performance gains. Finally, we offer an in-depth discussion and experimentation regarding the trade-offs of the proposed algorithms and the choice of parameter values.

A. Koufakou (✉)
U.A. Whitaker School of Engineering, Florida Gulf Coast University,
Fort Myers, FL 33965, USA
e-mail: akoufakou@fgcu.edu

J. Secretan · M. Georgiopoulos
School of Electrical Engineering and Computer Science,
University of Central Florida, Orlando, FL, USA

Published online: 08 December 2010

Keywords Outlier detection · Anomaly detection · Frequent itemset mining · Non-Derivable itemsets · Categorical datasets

1 Introduction

The task of detecting outliers in a dataset has received significant attention in many applications, such as network intrusion detection [10] or outlying points in engineering data [11]. Outlier detection methods aim to detect patterns that occur infrequently in the data. One of the most widely accepted definitions of an outlier pattern is provided by [13]: "An outlier is an observation that deviates so much from other observations as to arouse suspicion that it was generated by a different mechanism".

Most of the previous research in outlier detection has focused on datasets that contain numerical attributes or ordinal attributes that can be directly mapped into numerical values. However, in categorical data, methods that assume numerical attributes must first map each categorical value to a numerical value, a task which is not a simple process; for example, mapping a marital status value, married or single, to a numerical value [22].

A second issue is that many current applications for outlier detection involve very large datasets; e.g. Wal-Mart had 460 terabytes of data in 2004 [14], and multi-terabyte time series data are usual in astronomy [29]. Therefore, the proposed methods must scale well with the number of data points and the dataset dimensionality [1,22]. Data points may also be distributed over various geographical sites; this makes the transferring of data difficult, due to data size, as well as ownership and control issues. Therefore, the proposed algorithms must require minimal data scans.

Recently, outlier detection methods for categorical and mixed-attribute data were proposed, (e.g. [20,22]). Some of these methods use the concept of *Frequent Itemset Mining* (FIM). FIM has attracted substantial attention since the seminal paper by [2], which introduced Apriori, today recognized as one of the ten most influential data mining algorithms [26]. FIM extracts patterns or sets of items (categorical values) that co-occur frequently in the dataset. Then, outlier points are likely to be the points that contain relatively few of the frequent patterns (itemsets).

FI-based outlier detection methods [16,20,22] first extract all frequent itemsets from the data and then assign an outlier score to each data point based on the frequent sets it contains. However, FIM algorithms have been known to face problems for *dense data*, e.g. census data [30]. Dense datasets contain many strong correlations and are typically characterized by items with high frequency and many frequent patterns [30]. As a result, there might be a large number of FIs to generate and store. In general, outlier detection in large high-dimensional data with many categorical values will also face issues if based on frequent sets. As we show in Sect. 4, this issue persists even if we constrain the maximum length of generated sets [20,22]. To alleviate this issue, many methods have been proposed such as using condensed representations of the FI collection (CFIs) [9], or approximating the number of FIs in dense data [6].

In this paper, we use a smaller collection of sets instead of frequent sets, in order to detect outliers efficiently given large high-dimensional data. Specifically, we use Non-Derivable Itemsets (NDI) that have been shown to present large gains over FIs [8]. As shown in Sect. 3, the way sets are pruned in the NDI algorithm makes the NDI concept well-suited to the outlier detection algorithm. Furthermore, we also explore the use of a method that approximates the NDIs in the data using a δ parameter, called Non-Almost Derivable Itemsets (NADIs) [28].

We summarize the significance of our work as follows:

- We employ a condensed representation of FIs, NDI, in outlier detection, which has not been proposed before to the best of our knowledge; we believe our findings can easily be transferred to other areas, such as clustering or summarization of categorical data. Moreover, we discuss how NDI as a CFI suits the task of detecting outliers especially well.
- We propose two algorithms for detecting outliers: one based on the frequent NDIs, FNDI-OD, and one based on the negative border of the frequent NDIs, NBNDI-OD. The negative border of FIs consists of all sets X such that X is infrequent but all subsets of X are frequent. We compare the two NDI-based outlier detection methods with their FI-based counterparts given several datasets and parameter values.
- We explore the effect of an approximation scheme for NDIs, based on a δ parameter, called Non-Almost Derivable Itemsets (NADIs). We propose an outlier detection algorithm based on the Frequent NADIs, FNADI-OD. We also conduct experiments to explore the trade-off between accuracy and runtime performance of the outlier detection algorithm for various δ and σ value combinations.
- Our proposed methods are shown to be much faster and have similar detection accuracy compared to their FI-based Outlier Detection counterparts. Specifically, FNDI-OD is significantly faster than the FI method, even with large high-dimensional data and low σ values. More importantly, FNDI-OD exhibits higher or similar detection accuracy rates compared to FI-OD for the data and σ values we tested. Finally, FNADI-OD presents more runtime gains compared to FNDI-OD usually at a small cost of detection accuracy for various δ values. Finally, we offer an in-depth discussion of our findings regarding the advantages and weaknesses of the presented algorithms, and the effects of the parameter values, σ and δ, on the detection accuracy given different datasets.

The organization of this paper is as follows: In Sect. 2, we provide a literature review. In Sect. 3, we describe the FI-based, NDI-based and NADI-based outlier detection algorithms. Section 4 contains our experiments and results. Finally, in Sect. 5, we summarize our work and provide concluding remarks.

2 Previous work

Existing outlier detection work can be categorized as follows.

Statistical-model based methods assume that a specific model describes the distribution of the data [3]. This leads to the issue of estimating a suitable model for each particular dataset and application. Also, as data increases in dimensionality, it becomes increasingly more challenging to estimate the multidimensional distribution [1].

Distance-based methods do not make any assumptions about the data and instead compute pair-wise distances among data points [18]. These methods can easily have quadratic complexity with respect to the number of data points and thus become impractical for large data [22]. A distance-based method based on randomization and pruning in [4] has complexity close to linear in practice. *Clustering* techniques assume that outliers are points that do not belong to formed clusters; however, these methods focus on optimizing the clustering process, not on detecting outliers [18].

Density-based methods identify outliers as those lying in relatively low-density regions. In [7], a *degree* of anomaly, local outlier factor (*LOF*), is assigned to each point based on the local density of the area around the point and the local densities of its neighbors. Although

these methods can detect local outliers that may be missed by distance-based methods, they face a challenge with sparse high-dimensional data.

Other examples of outlier detection work include Support Vector methods [24] among others. Finally, most of these methods require data to be in the same location and cannot easily be applied to geographically distributed data.

Most of these techniques are suitable for numerical or ordinal data that can be easily mapped to numerical values; in the case of categorical data, there is little sense in ordering the data in order to map them to numerical values and compute distances [22]. Another limitation of previous methods is the lack of scalability with respect to number of points and/or data dimensionality. An entropy-based outlier detection method for categorical data is proposed in [15], and a fast and scalable method for categorical data, AVF, is presented in [21].

The methods in [16,20,22] use FIs to assign an outlier score to each data point based on the subsets this point contains. [22] propose a distributed and dynamic outlier detection method for data with both categorical and continuous attributes. For each set of categorical values (itemset), X, the method isolates the data points that contain set X, then calculates the covariance matrix of the continuous values of these points. A point is likely to be an outlier if it contains infrequent categorical sets, or if its continuous values differ from the covariance. *ODMAD* [20], an outlier detection method for mixed-attribute data, can handle sparse high-dimensional continuous data. ODMAD first inspects the categorical space in order to detect data points with irregular categorical values or sets. Second, it sets aside these points and focuses on specific categorical values and the points that contain these values one at a time. ODMAD exhibits significant performance gains and lower memory requirements compared to [22], and its distributed version achieves linear speedup for distributed datasets [19].

The FI-based methods do well with respect to runtime performance and scalability with the tested data. Nevertheless, these techniques rely on mining all FIs and face problems with performance and memory requirements when applied to large high-dimensional data. This is a well-known issue for FIM, not restricted to a specific algorithm (e.g. Apriori [2]). Specifically, FIM algorithms perform well with sparse datasets, such as market-basket data, but face problems for dense data. The resulting FIs might still be numerous, an issue exacerbated when these sets contain millions of items or the σ threshold is too low. In the case of outlier detection, we observed that this happens when a large high-dimensional dataset is used that contains several highly-frequent categorical values.

To solve this issue, much work has been conducted toward *Condensed representations of FIs* (CFIs). The goal of these methods is to generate a smaller collection of representative sets, from which all FIs can be deduced. Many CFIs have been proposed, e.g. *maximal*, *closed*, *δ-free*, *non-derivable* (see [9] for a CFI survey). In this paper, we use Non-Derivable Itemsets (NDIs) [8], and an approximation of NDIs, Non-Almost Derivable Itemsets (NADIs) [28] based on a δ parameter. As we discuss in the next section, the way itemsets are used in previously proposed FI-based outlier detection methods (e.g. [20,22]) indicates that these previous methods can benefit from the NDI generation and pruning process. This is verified in our experimental results, not only in the large runtime performance gains but also in terms of similar outlier detection accuracy rates.

Besides condensed representations, other types of techniques have been proposed and used as a step prior to data mining tasks, such as clustering. For example, [27] uses highly correlated association patterns to filter out noisy objects, resulting in better clustering performance and higher quality associations. *Summary sets* [25] are proposed to summarize categorical data for clustering, while a support approximation and clustering method is presented in [17].

3 Algorithms

As shown in [21], the 'ideal' outlier point in a categorical dataset is one for which each and every value in that point 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. The algorithm in [21] assigns a score to each data point, which reflects the frequency with which each attribute value of the point occurs. In [20], 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 values is infrequent.

We consider a dataset $\mathcal{D}$ which contains n data points, $\mathbf{x}_i$, $i = 1 \ldots n$ (see Table 1 for the notation used in this paper). If each point $\mathbf{x}_i$ has m attributes, we write $\mathbf{x}_i = [x_{i1}, \ldots, x_{il}, \ldots, x_{im}]$, where x_{il} is the value of the l-th attribute of $\mathbf{x}_i$. Note that each point or record in $\mathcal{D}$ has exact dimensionality m, and, in this sense, $\mathcal{D}$ is not the traditional FIM transactional database where the length of each transaction (row) may vary.

Given dataset $\mathcal{D}$, a support threshold σ, and a number of target outliers, k, the goal is to detect the k outlier points in $\mathcal{D}$. The algorithms presented here have two main phases: (1) Extract a collection of patterns or sets in the data and (2) compute an anomaly score for each data point and use the scores to detect outliers. In the following sections, we present the following methods to detect outliers: a method that uses all FIs in the dataset, *FI-OD*; a method that uses the negative border of the FIs, *NBFI-OD*; a method based on frequent Non-Derivable Itemsets (NDIs), *FNDI-OD*; a method that uses the negative border of the NDIs, *NBNDI-OD*; and finally a method that uses the frequent Non-Almost Derivable Itemsets (NADIs), *FNADI-OD*.

3.1 Outlier detection based on frequent itemsets

Since frequent sets correspond to the "common patterns" which are found in many of the data points, outliers are likely to be the points that contain relatively few of these frequent patterns [16,20]. Likewise, normal data points will contain frequent categorical values, or frequent sets of these values.

Table 1 Notation used in this paper

Term	Definition
$\mathcal{D}$	Dataset
n	Number of data points in $\mathcal{D}$
m	Dimensionality of $\mathcal{D}$
r	Number of single distinct items in $\mathcal{D}$
$\mathbf{x}_i$	The i-th point in $\mathcal{D}$, $i = 1 \ldots n$
x_{il}	The l-th value of $\mathbf{x}_i$, $l = 1 \ldots m$
I	Itemset
$\|I\|$	Length of itemset
σ	Minimum support
k	Number of target outliers
δ	Parameter in NADI algorithm
MAXLEN	Maximum length of itemset I

Let $\mathcal{I} = \{i_1, i_2, \ldots, i_r\}$ be a set of r items in a database $\mathcal{D}$. Each data point (row) $\mathbf{x}_i \in \mathcal{D}$ contains certain of these r items. Given X, a set of some items in $\mathcal{I}$, we say that $\mathbf{x}_i$ contains X if $X \subseteq \mathbf{x}_i$. The support of X, $supp(X)$, is the percentage of rows in $\mathcal{D}$ that contain X. We say that X is frequent if it appears at least σ times in $\mathcal{D}$, where σ is a user-defined threshold. The collection of frequent itemsets is denoted by $\mathcal{FI}$:

$$\mathcal{FI} =_{\text{def}} \{X \subseteq \mathcal{I} \mid supp(X) \geq \sigma\}$$

3.1.1 Outlier detection based on FIs: FI-OD

Using the frequent set information, one can define an outlier factor or score for each data point, $\mathbf{x}_i$, $i = 1 \ldots n$. The equation for the outlier score based on FIs is designated as *FIODScore*, given in Eq. 1 below:

$$FIODScore(\mathbf{x}_i) = \sum_{X \subseteq \mathbf{x}_i \wedge X \in \mathcal{FI}} \frac{supp(X)}{|X|} \tag{1}$$

where $|X|$ denotes the length of set X or the number of items (attribute values) in X.

The function in Eq. 1 is based on *FindFPOF* [16]. It assigns a high score to data points that contain many frequent values or frequent sets. The lower the score in Eq. 1 the more likely the point is an outlier. The lowest possible score is 0; this score is assigned to a point that does not contain any frequent values or sets. Note that in [16], the summation term is divided by the total number of frequent sets in $\mathcal{D}$, but this does not affect the detection of outlier points.

Also, in Eq. 1, we divide the support of a set by the length of the set, following the concept in [22]: essentially the longer a set is, the less it contributes to the score of a point. This is because longer frequent sets contain subsets that are themselves frequent, and they have already been added to the score of a point. Finally, as shown in [20,22], we obtain a good outlier detection accuracy by only considering sets of length up to a user-entered *MAXLEN*.

Example 3.1 Let point $\mathbf{x} = [a\, b\, c]$, and $MAXLEN = 3$, the possible subsets of $\mathbf{x}$ are: a, b, c, ab, ac, bc, and abc.

If subset I of $\mathbf{x}$ is frequent, i.e. $supp(I) \geq \sigma$, we increase the score of $\mathbf{x}$ by $supp(I)$ divided by the length of I. In our example, if $supp(ab) = 0.3$ and ab is frequent, $FIODScore(\mathbf{x})$ will increase by $0.3/2 = 0.15$.

The pseudocode for the *FI-OD* method is shown in Algorithm 1. The first step is to mine the frequent sets from the data using a FIM algorithm such as Apriori [2], but other FIM algorithms could be used instead. *FI-OD* goes over each data point $\mathbf{x}_i$ and checks the frequent sets against point $\mathbf{x}_i$. For each set in the frequent itemset lattice that is a subset of point $\mathbf{x}_i$, we update the outlier score corresponding to $\mathbf{x}_i$. If a categorical value (item) a does not belong in $\mathbf{x}_i$, none of a's supersets are checked against $\mathbf{x}_i$, and so on. Finally, the k data points with the lowest score are returned as outliers.

3.1.2 Outlier detection based on the negative border of FIs: NBFI-OD

In contrast to the previous section, the algorithms in [20,22] use the *infrequent* subsets of a point to compute its outlier score. Here we present part of the Outlier Detection for Mixed Attribute Datasets algorithm (*ODMAD*) [20] which assigns a score to a point according to

Input : Database $\mathcal{D}$, target outliers k, minimum support σ, maximum itemset length *MAXLEN*
Output: k detected outliers

```
G = Get FI (D, σ, MAXLEN);
FIODScore[|D|], outliers[k] := ∅;
foreach x_i ∈ D, i ← 0..n do
    foreach set f ∈ G ∧ f ⊆ x_i do
        Update FIODScore[i];
    end
end
outliers[k] ← x_i with min FIODScore;
```

Algorithm 1: FI-OD Pseudocode

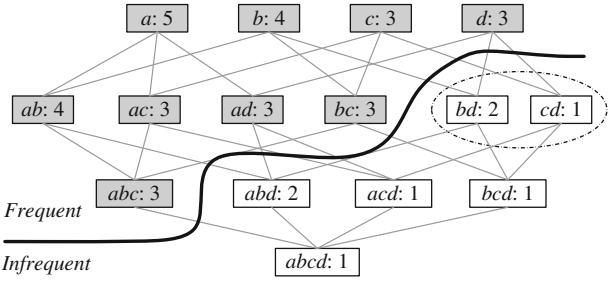


Fig. 1 Example of frequent sets, infrequent sets, and sets on negative border

its categorical and its continuous values. Since in this context we focus on categorical data, we only describe the categorical score function of ODMAD.

The method in [22] generates FIs and then assigns a score to each point $\mathbf{x}_i$ based on the length of all infrequent sets that exist in $\mathbf{x}_i$. However, this method does not take into consideration the frequency of the infrequent values. Enumerating all possible infrequent sets and their support in the data is extremely expensive especially for large data.

Instead of considering all infrequent subsets of a point, ODMAD only considers the sets that belong in the negative border of the frequent sets, $\mathcal{B}^-(FI)$. A set X belongs in $\mathcal{B}^-(FI)$ if all its subsets are frequent but X itself is infrequent:

$$\mathcal{B}^-(FI) =_{\text{def}} \{I \subseteq \mathcal{I} \mid \forall J \subset I : J \in \mathcal{FI} \wedge I \notin \mathcal{FI}\}$$

Figure 1 depicts an example for frequent sets and negative border sets. For example, set *bd* is on the negative border, but set *abd* is not, as it contains subset *bd* that is infrequent.

By only considering the $\mathcal{B}^-(FI)$ sets, we are able to include the frequency of these sets in our outlier detection score. We assign an anomaly score to each data point that depends on each set $d \in \mathcal{B}^-(FI)$ that is also contained in this point. The score is inversely proportional to the support and the length of the infrequent set. We name this algorithm *NBFI-OD* for Outlier Detection based on the Negative Border of FIs. As in the previous section, we stop generation of frequent sets longer than *MAXLEN*. The outlier score is shown in Eq. 2 below:

$$NBFIODScore(\mathbf{x}_i) = \sum_{d \subseteq \mathbf{x}_i \wedge d \in \mathcal{B}^-(FI)} \frac{1}{supp(d) \times |d|} \tag{2}$$

A higher score implies that it is more likely that the point is an outlier. A point with very infrequent single values will get a very high score; a point with moderate infrequent single values will get a moderately high score; and a point whose single values are all frequent and

has a few infrequent subsets will get a moderately low score. The pseudocode for NBFI-OD is shown in Algorithm 2. To construct $\mathcal{B}^-(FI)$, we maintain a table with the pruned candidate sets, i.e. sets which are found infrequent after the Apriori generation process [2].

Another reason for using only the sets in the $\mathcal{B}^-(FI)$ is that we are more interested in either single categorical values that are infrequent, or infrequent sets containing single values that are frequent on their own. Also, considering only these sets, i.e. the ones on the negative border of FIs, makes our method faster than a method that uses all infrequent sets such as [22]. This is shown in the following example.

Example 3.2 Assume we have two points, each with three categorical attributes: $\mathbf{x}_1 = [a\ b\ c]$ and $\mathbf{x}_2 = [a\ b\ d]$. If only single values a and c are infrequent with support equal to 0.5, the score is as follows:

$$NBFIODScore(\mathbf{x}_1) = \frac{1}{supp(a)} + \frac{1}{supp(c)} = 4.$$
$$NBFIODScore(\mathbf{x}_2) = \frac{1}{supp(a)} = 2.$$

Since a and c are infrequent, we do not check any of their combinations with other values because they will also be infrequent. The sets we do not check are ab, ac, ad, bc, cd. However, bd consists of frequent values, b and d, so we check its frequency. Assuming bd is infrequent, and $supp(bd) = 0.4$, we increase the score of x_2:

$$NBFIODScore(\mathbf{x}_2) = 2 + \frac{1}{supp(bd) \times |bd|} = 3.25.$$

This value still reflects that point $\mathbf{x}_1$ is more likely to be an outlier than $\mathbf{x}_2$, since it has two single irregular values and $\mathbf{x}_2$ has a lower score since it contains one irregular or infrequent value and one infrequent combination of values.

This score was shown to have a very good accuracy in [20], and similar findings are shown in the Experiments Section.

3.2 Outlier detection based on Non-Derivable Itemsets

As noted earlier, FI-based methods may face challenges for large high-dimensional data. In the case of such data, many FIs will be generated that are spurious combinations of the same categorical values with the same or similar support. One proposed solution for this issue is the Non-Derivable Itemsets (NDI) representation [8]. The NDI representation is well-suited for the task of outlier detection because it prunes sets whose support can be derived from their

Input : $\mathcal{D}$, k, σ, $MAXLEN$
Output: k detected outliers

1 $\mathcal{B}^-(FI)$ = Get Negative Border of $\mathcal{FI}$ ($\mathcal{D}$, σ,$MAXLEN$);
2 $NBFIODScore[|\mathcal{D}|]$, $outliers[k] := \emptyset$;
3 **foreach** $\mathbf{x}_i \in \mathcal{D}, i \leftarrow 0..n$ **do**
4 **foreach** *set* $d \in \mathcal{B}^-(FI) \wedge d \subseteq \mathbf{x}_i$ **do**
5 Update $NBFIODScore[i]$;
6 **end**
7 **end**
8 outliers[k] $\leftarrow \mathbf{x}_i$ with max $NBFIODScore$;

Algorithm 2: NBFI-OD Pseudocode

subsets (derivable sets). Basically, the NDI algorithm retains sets and support information for 1-sets and 2-sets (i.e. single values and combinations of two values) and prune sets of longer length that do not provide additional information.

3.2.1 Background on NDI

In this section, we present background on the NDI representation and algorithm from [8].

[8] showed that itemsets whose support can be deduced or derived from their subsets, i.e. *derivable sets*, can be pruned, and that this can dramatically reduce the amount of sets generated.

Let a general itemset, G, be a set of items and negations of items, e.g. $G = \{ab\overline{c}\}$. The support of G in this case is the number of rows where items a and b are present while item c is not present. We say that a general itemset $G = X \cup \overline{Y}$ is based on itemset I if $I = X \cup Y$. The deduction rules in [8] are based on the inclusion-exclusion (IE) principle [12]. For example, using the IE principle, we write the following for the support of a general itemset $\{a\overline{bc}\}$:

$$supp(a\overline{bc}) = supp(a) - supp(ab) - supp(ac) + supp(abc).$$

Based on $supp(a\overline{bc}) \geq 0$, we can write the following:

$$supp(abc) \geq supp(ab) + supp(ac) - supp(a).$$

The above is an upper bound on the support of set abc. [8] extend this concept to computer rules in order to derive the upper and lower bounds on the support of itemset I. Let $LB(I)$ and $UB(I)$ be the lower and upper bounds on the support of I. Also, let $R_X(I)$ denote a rule (such as the inequality above) that computes a bound on the support of I based on $X \subset I$. The lower and upper bounds on the support of I are

$$LB(I) = \max\{\delta_X(I),\ |I \backslash X|\ odd\}$$
$$UB(I) = \min\{\delta_X(I),\ |I \backslash X|\ even\}$$

where $\delta_X(I)$ denotes the summation in rule $R_X(I)$ and is shown below:

$$\delta_X(I) = \sum_{X \subseteq J \subset I} (-1)^{|I \backslash J|+1} supp(J).$$

An itemset I is *derivable* if $LB(I) = UB(I)$. An example for set abc is given below.

Example 3.3 The rules $R_X(abc)$ are shown for X equal to $abc, ab, ac, bc, a, b, c, \emptyset$, respectively. The support interval for abc is [1,1]; therefore abc is derivable.

tid	Items			
1	a	$supp(abc)$	$\geq$	0
2	$b; c$		$\leq$	$supp(ab) = supp(ac) = 2, sup(bc) = 3$
3	c		$\geq$	$supp(ab) + supp(ac) - supp(a) = 0$
4	$a; b$		$\geq$	$supp(ab) + supp(bc) - supp(b) = 1$
5	$a; c$		$\geq$	$supp(ac) + supp(bc) - supp(c) = 0$
6	$b; c$		$\leq$	$supp(ab) + supp(ac) + supp(bc)$
7	$a; b; c$			$-supp(a) - supp(b) - supp(c) + supp(\emptyset) = 1.$

Given database $\mathcal{D}$ and threshold σ, the NDI algorithm produces the condensed representation $NDIRep(\mathcal{D}, \sigma)$. The NDI representation contains only the non-derivable frequent sets as defined below:

$$NDIRep(\mathcal{D}, \sigma) =_{\text{def}} \{I \subseteq \mathcal{I} \mid I \in \mathcal{FI} \wedge LB(I) \neq UB(I)\}.$$

The NDI algorithm uses Apriori-gen to generate candidate sets, and then prune infrequent candidates. If a set I is NDI, i.e. $LB(I) \neq UB(I)$, the algorithm needs to count the support of I; if it is found that $supp(I) = LB(I)$ or $supp(I) = UB(I)$, all strict supersets of I are derivable and they can be pruned. The process terminates when candidate sets cannot be generated further.

Several properties of the NDIs were presented in [8], which we briefly summarize below.

Monotocity: If I and J are itemsets, $J \subseteq I$, and J is derivable, then I is derivable.

[8, Theorem 3.1] Given itemsets $I, J \subseteq \mathcal{I}$ and $J \subset I$. The interval width of I will be at most half of the interval width of J:

$$UB(I) - LB(I) \leq \frac{1}{2}(UB(J) - LB(J)). \tag{3}$$

Therefore, the NDI collection cannot be very large, because the width of support intervals, $UB(I) - LB(I)$, decreases exponentially with the cardinality of itemset I. Also, every set I with cardinality larger than the logarithm of the size of database, i.e. $|I| > \log_2(n) + 1$, must be derivable.

3.2.2 Motivation for outlier detection using NDIs

The motivation behind selecting NDIs for outlier detection is mainly to handle high-dimensional categorical data in an efficient manner. As we see in Sect. 4, when detecting outliers using FIs, we are scanning through a much larger number of sets than the ones generated by the NDI algorithm. Also, the sets generated by traditional FIM methods are much longer than the ones generated by NDI. It is shown in [8] that with certain datasets, the number and length of frequent itemsets generated by the Apriori algorithm was so large that the execution of the algorithm had to be terminated. Our experiments also support these results (see Sect. 4).

What is more important is that using only *NDIRep* greatly speeds up the outlier detection process, while not seriously impacting the accuracy of outlier detection. The reasons behind this are based on the score in Sect. 3.1.1, as well as the NDI pruning process.

From Sect. 3.1.1, we know the following: (1) shorter sets are more important to the *FIODScore* than longer sets; and (2) as a consequence of (1), sets longer than a user-entered *MAXLEN* value can be ignored.

Regarding (1), both NDI and Apriori (or other FIM) algorithms generate almost identical sets of length 1 and length 2. Therefore, NDI preserves the shorter sets which are more important to the score. The sets of length greater than 2 not maintained by NDI, i.e. the frequent derivable sets, do not provide significant additional information when compared to their subsets. In fact, as the σ threshold decreases, most of the DIs, especially those of longer length, are increasingly longer combinations of highly frequent items. This implies that it is not necessary to continue adding the support information of every possible combination of these items for the outlierness score, because it will be repeatedly adding the same support number for most of the normal points.

With respect to (2), using *MAXLEN* also speeds up the algorithm. e.g. the collection of sets with length less than 4 is also much smaller than all FIs that can be generated. Still, *MAXLEN*

is another parameter that the user needs to set, whose value may vary for each dataset and application. Even after using *MAXLEN*, the NDI collection is smaller than the FI collection for $MAXLEN > 2$. Therefore, using NDI increases the performance of the algorithm, while, at the same time, freeing the user from the responsibility of choosing an arbitrary value for the maximum length of a set. We further experiment with this parameter in Sect. 4.3.

Another avenue is to use any CFI scheme, e.g. NDI or closed sets [23], to generate all FIs and then proceed with outlier detection based on FIs. However, the issue we are addressing is that a very large number of FIs is generated either way. In this case, if we used a CFI in order to generate all FIs, the computation of the outlier scores for each point would still be a very slow process. Therefore, it is imperative to use a representative set of FIs instead of all FIs for the computation of our score.

On the other hand, other condensed representations could be used instead of *NDIRep* in Eq. 1. Another successful and popular CFI is *closed* sets [23]. A *closed set* is a set with support such that there is no superset with support equal to it [23]. This way, the closed representation is a different collection than *NDIRep*. Although there are datasets for which the closed representation is smaller than *NDIRep*, the former is not as straightforward as the latter to use in Eq. 1. Mainly, the closed representation does not preserve the support information of all single frequent values (items) or shorter itemsets in the data. That is because the closure of a single-item set might be a longer set. For example, given two itemsets, a and $abcw$, $abcw$ might be the closure of a, so the closed representation only contains $abcw$. However, as discussed in the previous section, shorter irregular or infrequent itemsets are more important to the outlier score than longer sets. Therefore, a score function similar to Eq. 1, using the closed representation rather than *NDIRep*, is more likely to assign similar scores to normal as to outlier points. This can be seen in the following example.

Example 3.4 Given the five-point four-dimensional database, $\mathcal{D}$, and $\sigma = 3/5$, we have the following collections:

1	$a\ b\ c\ d$
2	$a\ b\ c\ w$
3	$a\ b\ c\ z$
4	$a\ b\ y\ d$
5	$o\ q\ x\ d$

$\mathcal{FI} = \{a{:}4, b{:}4, c{:}3, d{:}3, ab{:}4, ac{:}3, bc{:}3, abc{:}3\}$

$NDIRep = \{a{:}4, b{:}4, c{:}3, d{:}3, ab{:}4, ac{:}3, bc{:}3\}$

$ClosedRep = \{d{:}3, ab{:}4, abc{:}3\}$

We see that, in this example, *ClosedRep* is smaller than *NDIRep*. Also, there is little difference between *NDIRep* and $\mathcal{FI}$. We note that this example is used to illustrate the suitability of these CFIs for the outlier detection score, and not their efficiency. As we see in Sect. 4, there are significant performance gains from using *NDIRep* instead of $\mathcal{FI}$.

Given the above collections, the scores based on previous sections are as follows—*ClosedOD* refers to Score in Eq. 1 using the frequent closed sets instead of *NDIRep*:

Point	*FIOD*	*FNDIOD*	*ClosedOD*
1	20	19	6
2	17	16	3
3	17	16	3
4	13	13	5
5	3	3	3

According to the above computations, the FI-based and NDI-based score maintain the ordering of these five points. Using *ClosedRep*, the fifth point has the same score as the

second and third points. However, the fifth point has only a single frequent item, d, and its remaining values and their combinations are infrequent, so according to Sect. 3.1, it is more likely to be an outlier than the rest of the points.

Therefore, as the NDI representation preserves more smaller sets and their support, it is straightforward to apply to outlier detection. *NDIRep* also preserves those longer sets that cannot be derived from their subsets, as they do provide new information for the outlier score. Thus, NDI-based outlier detection provides a good trade-off between accuracy and efficiency, which can also be seen in Sect. 4.

3.2.3 Outlier detection based on NDIRep: FNDI-OD

Given a database $\mathcal{D}$ and the *NDIRep*($\mathcal{D}$, σ) set, we propose an algorithm named FNDI-OD to find outliers as follows. For each data point in the database, $\mathbf{x}_i = [x_{i1}, \ldots, x_{im}]$, we assign an outlier score based on the itemsets in *NDIRep*($\mathcal{D}$, σ) that are subsets of $\mathbf{x}_i$. Every time an itemset in *NDIRep*($\mathcal{D}$, σ) is found to be a subset of $\mathbf{x}_i$, we update the score of $\mathbf{x}_i$ as follows:

$$FNDIODScore(\mathbf{x}_i) = \sum_{I \subseteq \mathbf{x}_i \wedge\ I \in NDIRep} \frac{supp(I)}{|I|} \tag{4}$$

where I is a non-derivable frequent set, i.e. it belongs in *NDIRep* (instead of X, a frequent set, in Eq. 1). Therefore, only those frequent sets in *NDIRep* are used to find outliers (thus eliminating the need to use all frequent sets as previously with FI-OD). Finally, the algorithm finds the k lowest scores and labels the respective data points as outliers, where k is a positive integer provided as input. The pseudocode for FNDI-OD is in Algorithm 3.

3.2.4 Outlier detection based on the negative border of NDIRep: NBNDI-OD

As with NBFI-OD, we also propose an outlier detection method based on NDI infrequent sets. This method uses the sets that are generated and then pruned during the NDI algorithm because they are infrequent. These sets are in the negative border of the NDI tree:

$$\mathcal{B}^{-}(NDI) =_{\text{def}} \{I \subseteq \mathcal{I} \mid \forall J \subset I : J \in NDIRep \wedge I \notin NDIRep\}$$

Notice that in the case of NDI, sets may be pruned because they are derivable or because they are infrequent. Therefore, the *NDIRep* negative border contains even less sets than the FI-negative border.

Input : $\mathcal{D}$, k, σ, *MAXLEN*
Output: k detected outliers

```
1 NDIRep = Get Frequent NDIs (D, σ, MAXLEN);
2 FNDIODScore[|D|], outliers[k] := ∅;
3 foreach x_i ∈ D, i ← 0..n do
4 |   foreach set I ∈ NDIRep ∧ I ⊆ x_i do
5 |   |   Update FNDIODScore[i];
6 |   end
7 end
8 outliers[k] ← x_i with min FNDIODScore;
```

Algorithm 3: FNDI-OD Pseudocode

The pseudocode for the Negative Border NDI-OD, NBNDI-OD, is shown in Algorithm 4. The score for NBNDI-OD is given in Eq. 5:

$$NBNDIODScore(\mathbf{x}_i) = \sum_{I \subseteq \mathbf{x}_i \ \wedge \ I \in \mathcal{B}^-(NDI)} \frac{1}{supp(I) \times |I|} \tag{5}$$

3.3 Outlier detection based on approximation of NDIs

3.3.1 Non-almost derivable itemsets (NADIs)

NDIRep includes all frequent itemsets I such that $UB(I) \neq LB(I)$. However, on closer inspection, some of the NDIs have a small support interval, $W(I) = UB(I) - LB(I)$. This means that the lower and upper bound for the support of itemset I are very close. We may also wish to eliminate such sets I with small $WB(I)$, because the support of I is *close* to being derivable from the supports of the subsets of I, $X \subset I$.

In [28], given a user-defined error-tolerance threshold δ, an itemset I is Almost Derivable (ADI) if $UB(I) - LB(I) \leq \delta$. Otherwise, I is called a non almost derivable itemset (NADI). As shown in [28] if set X is an ADI and $X \subset Y$, Y is also ADI.

The NADIRep is the collection of all NADIs that are also frequent:

$$NADIRep(\mathcal{D}, \sigma, \delta) =_{\text{def}} \{I \subseteq \mathcal{I} \mid I \in \mathcal{FI} \ \wedge \ UB(I) - LB(I) \leq \delta\}$$

Property 3.1 If set J is non-almost derivable and $|\delta_X(J) - supp(J)| \leq \delta$, where $\delta_X(J)$ is the closest bound to $supp(J)$, then set $I \supset J$ is almost derivable (ADI).

Proof [8] observe the following: Given an itemset J such that J is a subset of I, and the difference between the support bound of J, $\delta_X(J)$ and the actual support of J, $supp(J)$ is minimal across all such subsets J of I. Then, the difference between the closest bound of J and the support of J actually defines the width of the support interval for set I:

$$|\delta_X(J) - supp(J)| = supp(X\overline{Y}) = UB(I) - LB(I).$$

Given that

$$|\delta_X(J) - supp(J)| \leq \delta$$

we have $UB(I) - LB(I) \leq \delta$ and I is δ-DI. □

This property implies that we stop generating sets after the estimated support, i.e. the closest support bound, of a set J is within δ of the actual support of set J.

Input : $\mathcal{D}$, k, σ, *MAXLEN*
Output: k detected outliers

```
1 B⁻(NDI) = Get Negative Border of NDIRep(D, σ, MAXLEN);
2 NBNDIODScore[|D|], outliers[k] := ∅;
3 foreach x_i ∈ D, i ← 0..n do
4 |   foreach set d ∈ B⁻(NDI) ∧ d ⊆ x_i do
5 |   |   Update NBNDIODScore[i];
6 |   end
7 end
8 outliers[k] ← x_i with max NBNDIODScore;
```

Algorithm 4: NBNDI-OD Pseudocode

Table 2 Dataset details: number of rows, columns, single distinct items, percentage of outliers in the dataset

Dataset	Rows	Columns	Items	Outlier %
BC	483	9	87	8.00
Mushroom	4,644	22	113	9.40
KDD1999-10	98,587	39	1,179	1.30
KDD1999-U2R	97,330	39	1,080	0.05
KDD1999-Entire	98,3550	39	1,542	1.10
pumsb	49,046	74	2,113	–
connect	67,557	43	129	–
Artif-500-50	500,000	50	1,750	–
USCensus1990	2,458,287	86	396	–

3.3.2 Outlier detection based on NADIRep: FNADI-OD

Finally, we propose to use all frequent NADIs instead of frequent NDIs to detect outliers. This algorithm, called FNADI-OD, follows Algorithm 3 with the exception of using *NADIRep* instead of *NDIRep*. The following equation displays the score for FNADI-OD:

$$FNADIODScore(\mathbf{x}_i) = \sum_{I \subseteq \mathbf{x}_i \wedge I \in NADIRep} \frac{supp(I)}{|I|} \tag{6}$$

As shown in [28], the NADI representation contains a smaller number of sets than the NDI representation. However, as the δ value increases, the accuracy of the algorithm is expected to drop. Since we are interested in detecting outliers and not extracting frequent sets, we can allow larger δ values than the ones presented in [28]. As we see in the next section, we can expect similar accuracy to FNDI-OD with larger performance gains in the algorithm's efficiency.

4 Experiments

4.1 Experimental setup

We conducted all experiments on a Pentium 2- GHz processor with 1 GB RAM. We used the NDI code available online,[1] and implemented the rest in C++. Pruned sets (negative border) are stored in a vector, while frequent sets are stored as in Goethals implementation. Also, as we go over each point in $\mathcal{D}$ to compute FI/FNDI-OD scores, we only keep the frequent values in each point and then search only for frequent values in the trie.

4.1.1 Datasets

All datasets were obtained from the UCI Repository [5] and the FIMI repository[2] (see Table 2 for a summary). The datasets that were selected had one or more of the following characteristics: large number of rows; large number of attributes (columns); large number or distinct items; and/or potential for generating large number of frequent sets (i.e. dense dataset).

[1] http://www.adrem.ua.ac.be/goethals/software.

[2] http://fimi.cs.helsinki.fi.

Wisconsin Breast Cancer (*BC*): The attributes in BC are computed from an image of a fine needle aspirate (FNA) of a breast mass, and describe characteristics of the cell nuclei (e.g. radius or texture). The original set contains 444 benign points and 212 malignant. As in [21], to make the dataset more imbalanced, we kept every sixth malignant record, resulting in 39 outliers (malignant), 444 non-outliers (benign).

Mushroom: This set represents samples for 23 species of gilled mushrooms. The set contains 8,124 points and 22 categorical attributes. Mushrooms are poisonous (48.2%) or edible (51.8%). To make the dataset more imbalanced, we kept every tenth poisonous record. The final set contains 113 unique categorical values, 4,644 total points, and 436 outliers (poisonous) or 9.4% of new set.

KDD1999: This set 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, etc. There are three available sets: training set, test set, and 10% of the training set. For our experiments, we used the 10% training set and the entire training set. All KDD1999 sets contain 33 continuous attributes and 8 categorical attributes.

Due to the large number of attacks, we preprocessed the data such that attack points are a small percentage of the data, and these points were chosen randomly from the collection of attack 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 outlier points at random without necessarily preserving the burst length. We followed [20,22] and detected bursts of packets in the data. Our resulting 10% training set contains 98,587 points, 1,309 attacks; our resulting entire training set contains 983,550 rows and 10,769 attacks. We discretized continuous attributes using equal-width discretization (20 intervals) and removed 2 attributes that contained the same value for all records. Both resulting sets have 39 columns; the 10% set has 1,179 distinct categorical values (single items) and the entire training set has 1,542.

KDD1999 (U2R): We also performed experiments with a dataset identical to the KDD1999 10% training set except that the new set only contains attacks of type "U2R". The resulting dataset has 97,330 total points out of which 52 are attacks (of type "U2R").

pumsb: This set is based on census data and it is a benchmark set for evaluating the performance of FIM algorithms on dense datasets. One of its characteristics is the skewed support distribution, e.g. 81.5% of the items in this set have support less than 1%, while 0.95% of them have support greater than 90%. It contains 49,046 rows, 74 attributes, and 2,113 total distinct items. The instances in this set are not labeled, and therefore we only performed experiments to compare the runtime performance of all proposed algorithms.

Connect: This set contains all legal 8-ply positions in the game of connect-4 in which neither player has won yet, and in which the next move is not forced. Connect is also a dense dataset; it contains a total of 67,557 instances with 42 categorical attributes, and a total of 129 distinct items. This dataset does not contain labeled instances.

USCensus1990: This set is a discretized version of the USCensus1990 raw dataset. It contains 2,458,287 data points and 68 categorical attributes. This dataset does not contain labeled instances.

Artificial: We created an artificial dataset based on data generated with the categorical data generator available online.[3] We generated datasets with various numbers of rows (300, 100, 20K, etc.), 50 attributes, and various numbers of values per attribute. We combined the resulting datasets into one dataset with 500 total rows, and 35 distinct values per column. This

[3] http://www.cs.umb.edu/~dana/GAClust/index.html.

dataset also does not contain labeled instances and is used to compare runtime performance for the various algorithms in this paper.

4.1.2 Evaluation

We compare runtime performance of the algorithms using the same data. We also evaluate the accuracy of outlier detection based on the following measures:

- *Correct Detection rate* (*CD*): ratio of the number of outliers correctly identified as outliers over total number of outliers;
- *False Alarm rate* (*FA*): ratio of the number of normal points erroneously flagged as outliers over total number of normal points.

4.2 Results

We ran several experiments with various values for support threshold, σ, the desired number of outliers, k, and the NADI parameter δ. We used *MAXLEN* equal to 4 for all experiments as in [20,22]. We discuss the effects of several parameter values and our algorithms in the Discussion Sect. (4.3).

4.2.1 Detection accuracy results

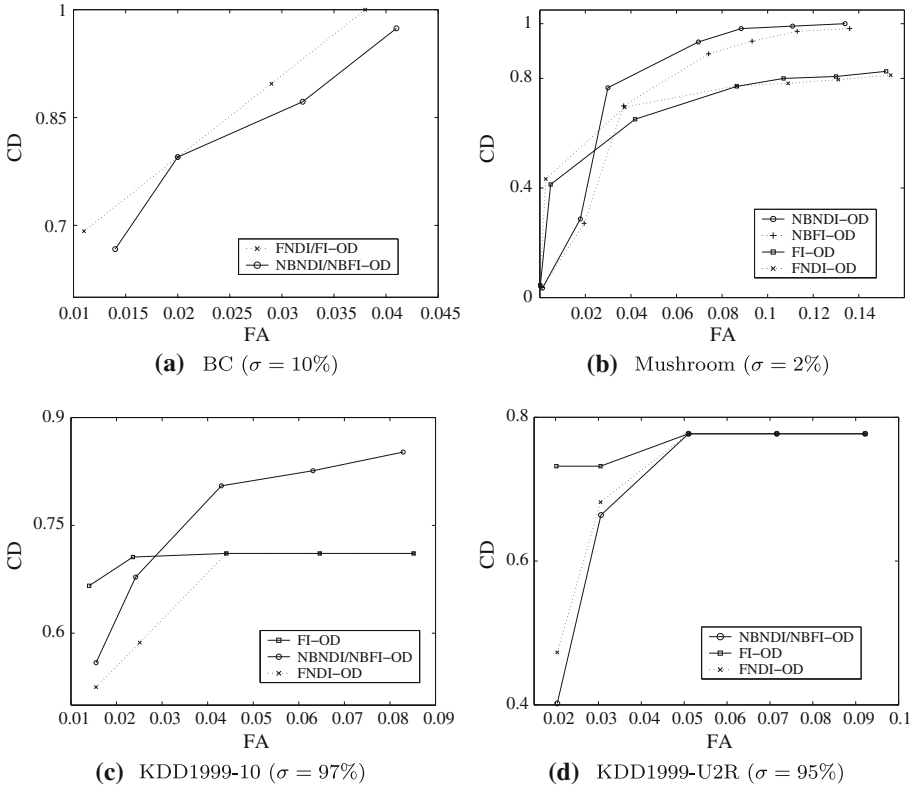
Figure 2 contains the ROC curves for the Breast Cancer (*BC*), KDD1999-10 (all attacks and *U2R* only), and Mushroom datasets. This figure displays the detection accuracy, CD, rates versus the False Alarm, FA, rates as the number of target outliers k increases while the σ is kept constant.

As shown in Fig. 2a, we did not observe a difference in CD or FA rates for the Breast Cancer Dataset between FI-based OD and NDI-based OD. In this case ($\sigma = 10\%$), the FNDI-OD and FI-OD have consistently better rates except for k equal to 40 where all methods have equal rates. Figure 2b shows that the FI- and FNDI-OD methods do better for lower k values but eventually the NBFI- and NBNDI-OD methods have better CD rates for similar FA values. For example, for $k = 1{,}000$, the CD (FA) rate for NBNDI-OD is 100% (13.4%) versus 98.2% (13.6%) for NBFI-OD, 81.2% (15.4%) for FNDI-OD, and 82.6% (15.2%) for FI-OD.

Figure 2c–d contain results for the average detection accuracy for the *KDD1999* datasets based on bursts of attacks that were correctly detected. 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 [22]. In Fig. 2c–d, the negative border (NB) methods have the same detection accuracy rates; for the KDD1999-10 dataset with all of the attacks, the NB methods perform better than the FI/FNDI-based methods, while for the dataset with only the U2R attacks, FI-OD has better rates but eventually the accuracy rates are the same for higher k values.

Table 3 contains the CD and FA results on the Mushroom and KDD1999-10 datasets for various σ values while the target outlier number k is constant. From Table 3a, NDI-based methods have better rates than FI-based methods for the Mushroom set for all σ values except for 50% when they are equal. The best accuracy, 93.3% is obtained for σ equal to 2%, by the NBNDI-OD method.

Table 3b also contains the average detection accuracy for *KDD1999-10* based on bursts of attacks for various σ values. From Table 3b, the methods based on the infrequent sets



(a) BC ($\sigma = 10\%$) **(b)** Mushroom ($\sigma = 2\%$)

(c) KDD1999-10 ($\sigma = 97\%$) **(d)** KDD1999-U2R ($\sigma = 95\%$)

Fig. 2 ROC curves for FNDI-OD, FI-OD, NBFI-OD, and NBNDI-OD as k increases ($MAXLEN = 4$)

perform consistently better than the frequent set methods. Specifically, the NBNDI-OD and NBFI-OD have 80.5% accuracy with 4% false alarm rate. It is important to note that these algorithms detected *all* 22 types of attacks in the KDD1999-10 set, so the *overall attack detection accuracy* is 100% for NBNDI-OD and NBFI-OD. Also, we notice that FI-OD fares better than FNDI-OD for $\sigma \leq 95\%$; however, both methods detect the same 20 out of 22 attacks so they have a total of 91% accuracy w.r.t. attacks.

Overall, the infrequent-based methods, NBNDI-OD and NBFI-OD do better than the methods that use frequent sets for scoring, i.e. FNDI-OD and FI-OD. Also, except for the *KDD1999-10* set, all methods have better accuracy for lower σ values. Section 4.3 contains a discussion regarding these results.

4.2.2 Runtime results

The performance advantage of the NDI-based technique versus the FI-based OD with respect to running time and generated sets is apparent in Tables 4 and 5. These Tables respectively contain the total generated sets for each algorithm (NDIs and FIs), sets on the negative borders ($\mathcal{B}^-(FI)$ and $\mathcal{B}^-(NDI)$), and the time in seconds for NDI-based OD and FI-based OD for each of the two phases of the OD algorithms, for various σ values. The user-entered parameter k has a negligible effect on the runtime performance as shown in [21]. The two

Table 3 Comparison of correct detection (False Alarm) rates of FNDI-OD, FI-OD, NBNDI-OD, and NBFI-OD

σ%	FNDI-OD *CD (FA)*	FI-OD *CD (FA)*	NBNDI-OD *CD (FA)*	NBFI-OD *CD (FA)*
(a) Mushroom ($k = 700$; actual outliers: 392)				
50	**72.0 (9.2)**	**72.0 (9.2)**	57.1 (10.7)	57.1 (10.7)
20	**76.1 (8.8)**	75.0 (8.9)	38.8 (12.6)	25.2 (14.0)
15	**76.4 (8.7)**	75.0 (8.9)	63.3 (10.1)	49.3 (11.5)
10	**76.8 (8.7)**	76.1 (8.8)	54.1 (11.0)	31.2 (13.4)
5	77.3 (8.6)	76.8 (8.7)	**89.0 (7.4)**	73.2 (9.1)
2	77.3 (8.6)	77.1 (8.7)	**93.3 (7.0)**	89.0 (7.4)
(b) KDD1999-10 ($k = 5{,}000$; actual outliers: 1,309)				
99.4	70.1 (4.5)	70.1 (4.5)	**80.5 (4.3)**	**80.5 (4.3)**
97	71.1 (4.4)	71.1 (4.4)	**80.5 (4.3)**	**80.5 (4.3)**
95	59.0 (4.4)	70.6 (4.3)	**80.5 (4.3)**	**80.5 (4.3)**
90	32.8 (4.5)	35.7 (4.5)	**80.5 (4.3)**	**80.5 (4.3)**
75	22.3 (4.5)	26.8 (4.6)	**80.5 (4.3)**	**80.5 (4.3)**
50	25.3 (4.5)	31.0 (4.5)	**80.5 (4.3)**	**80.5 (4.3)**
10	25.9 (4.5)	–	**80.5 (4.3)**	**80.5 (4.3)**

Better rates are in bold

phases are (1) mining the sets (frequent sets for FIM-OD, and NDIs for NDI-OD), and (2) computing the outlier score for each data point to detect the outliers.

In Table 4, the Mushroom dataset reveals the potential for high number of FIs; e.g. for $\sigma = 5\%$, there are 9,069 NDIs versus 44,741 FIs. This is after we set *MAXLEN* to 4. FNDI-OD takes 10 s for the Mushroom set ($\sigma = 5\%$), while FI-OD takes 38 s for the same task.

The situation is exacerbated for the KDD1999 sets (see Table 5), mostly due to the large number of data points, single items, and high dimensionality. For σ equal to 90%, FNDI-OD took 4 min to detect the outliers in the KDD1999-10 dataset, while FI-OD needed 40 min to accomplish the same task. This is because NDI generated only 177 non-derivable frequent sets versus the 1,486 frequent sets generated by Apriori. Moreover, the methods using the infrequent pruned sets (NBNDI-OD and NBFI-OD) are much faster than the FI-based ones. For the same σ value, NBNDI-OD takes less than 3 min and NBFI-OD takes 5 min. Also, as σ decreases, the advantage of using NBNDI-OD over NBFI-OD becomes larger. For example, for $\sigma = 10\%$, NBFI-OD takes 13 min versus 4 min for NBNDI-OD. Also, for $\sigma = 10\%$, FI-OD generates more than 110 thousand sets, and its execution was terminated.

The results for the *pumsb* dataset show that the NDI sets might also face challenges for low support values (see Table 4). For example, for $\sigma = 10\%$, there are more than a million NDIs and more than two million FIs. Furthermore, we had to stop execution of FI- and FNDI-OD for σ equal to 30%, while the NB-based methods had relatively acceptable performance. Overall, this type of situation shows the benefit of using an approximate representation such as NADIs (related results for FNADI-OD are shown in Sect. 4.3.3).

Figure 3 contains an illustration of the runtime performance for FI- and NDI-based algorithms using the Mushroom, KDD1999-Entire, pumsb, and UCSensus1990 datasets. This figure shows the performance of FI-OD decreases much faster than FNDI-OD. Finally, Fig. 4 illustrates the generated sets for each of the algorithms for the Mushroom, KDD1999-Entire,

Table 4 Size comparison between NDI, FI, negative border of NDI, and negative border of FI

Dataset	σ%	NDIs	FIs	$\mathcal{B}^-$(NDI)	$\mathcal{B}^-$(FI)
Mushroom	50	122	535	123	140
Mushroom	20	1,156	5,323	487	729
Mushroom	15	2,068	9,402	765	1,125
Mushroom	10	3,846	18,255	1,323	1,968
Mushroom	5	9,069	44,741	3,004	4,242
Mushroom	2	21,217	115,725	7,187	8,856
KDD1999-10	99.4	177	1,486	1,172	1,244
KDD1999-10	97	459	5,898	1,159	1,191
KDD1999-10	90	1,499	19,628	1,154	1,263
KDD1999-10	75	2,750	29,635	1,156	1,156
KDD1999-10	50	6,518	62,814	1,161	1,190
KDD1999-10	10	13,541	110,955	1,230	1,348
KDD1999-Entire	99	339	3,163	1,534	1,538
KDD1999-Entire	97	672	5,970	1,535	1,540
KDD1999-Entire	90	2,525	19,665	1,556	1,589
KDD1999-Entire	75	4,456	29,635	1,519	1,519
pumsb	75	5,008	12,743	2,130	2,696
pumsb	50	31,286	76,959	2,546	6,542
pumsb	30	200,238	411,819	4,831	19,421
pumsb	10	1,142,137	2,362,314	27,864	73,123
connect	90	199	3,476	141	272
connect	80	348	11,403	199	496
connect	50	1,397	49,039	382	1,275
connect	20	7,574	172,434	2,278	4,966
connect	10	28,799	343,839	7,155	9,217
USCensus1990	95	177	194	506	506
USCensus1990	90	1,727	1,857	872	876
USCensus1990	80	7,386	9,619	1,118	1,119
Artif-500-50	15	150,036	164,226	1,935	1,935
Artif-500-50	10	221,441	240,361	2,160	2,160
Artif-500-50	5	971,616	1,061,486	29,935	29,935

pumsb, connect, UCSensus1990, and Artif-500-50 datasets. This figure shows that for some datasets the negative border of FIs is almost identical to NDIs, such as the KDD1999, USCensus1990, and Artif-500-50 datasets. This is not the case for the dense datasets, such as pumsb and connect datasets: for example, for the pumsb set and σ equal to 10%, the compression ratio of $\mathcal{B}^-(NDI) \div \mathcal{B}^-(FI)$ is 38%.

4.3 Discussion

4.3.1 Performance of OD phases

From Table 5, it is apparent that most of the time of outlier detection is spent in the second phase, i.e. to compute the scores and detect outliers. A good example is the KDD1999-10

Table 5 Runtime performance for NDI- versus FI-based methods

Dataset	σ%	FNDI-OD	FI-OD	NBNDI-OD	NBFI-OD
Mushroom	50	1/1	1/2	1/1	1/1
Mushroom	20	1/3	1/13	1/3	1/4
Mushroom	15	1/4	1/17	1/4	1/6
Mushroom	10	1/7	2/24	1/7	2/11
Mushroom	5	1/9	3/35	1/17	3/26
Mushroom	2	2/12	6/45	2/42	6/52
KDD1999-10	99.4	9/24	23/216	9/136	32/164
KDD1999-10	97	14/60	46/675	15/158	54/146
KDD1999-10	90	25/239	134/2,309	25/135	145/172
KDD1999-10	75	33/436	189/3,220	33/133	192/136
KDD1999-10	50	53/905	459/5,711	53/134	490/145
KDD1999-10	10	97/1,225	–	67/146	631/174
KDD1999-Entire	99	136/500	332/4,016	139/2,086	329/1,730
KDD1999-Entire	97	163/992	522/–	162/1,729	512/1,737
KDD1999-Entire	90	317/4,153	1,373/–	314/1,792	1,335/1,868
KDD1999-Entire	75	455/6,959	1,927/–	443/1,698	1,875/1,697
pumsb	75	49/699	27/344	29/192	49/290
pumsb	50	277/3,879	100/2,059	95/238	280/824
pumsb	30	–	–	545/494	1,050/2,345
pumsb	10	–	–	1,570/2,586	–
connect	90	23/327	4/12	4/15	23/36
connect	80	55/968	5/17	5/23	55/70
connect	50	213/3,049	11/72	10/44	209/174
connect	20	545/6,304	24/228	23/251	546/598
connect	10	743/–	43/522	44/779	758/1,026
USCensus1990	95	531/1,067	571/1174	561/2,662	534/2,329
USCensus1990	90	777/7,244	848/7786	786/5,167	840/5,160
USCensus1990	80	1,605/–	1,979/–	1,601/6,724	1,931/6,682
Artif-500-50	15	2,911/–	3,173/–	2,906/1,263	3,168/1,221
Artif-500-50	10	–	–	3,778/1,374	4,141/1,330
Artif-500-50	5	–	–	6,327/19,966	7,245/19,254

Time is shown in seconds as time for Phase 1 (Set mining)/Time for phase 2 (Outlier Scoring)

set ($\sigma = 75\%$), where the second Phase of FI-OD takes 53 min versus 7 min for FNDI-OD. In contrast, FNDI-OD Phase 1 takes 33 s, versus 3 min for FI-OD Phase 1. As σ decreases, Phase 2 becomes increasingly slower for the frequent set-based methods, FNDI-OD and FI-OD. This is due to the increasingly larger number of sets that have to be compared against every single data point (row). The algorithms based on the negative border sets have usually much faster second phase because the NB sets are overall less, thus the comparisons for each point against all NB sets takes less time. For the KDD1999-10 set, we also observe that the number of pruned sets increases slowly as the σ becomes smaller, which is not true for the Mushroom set.

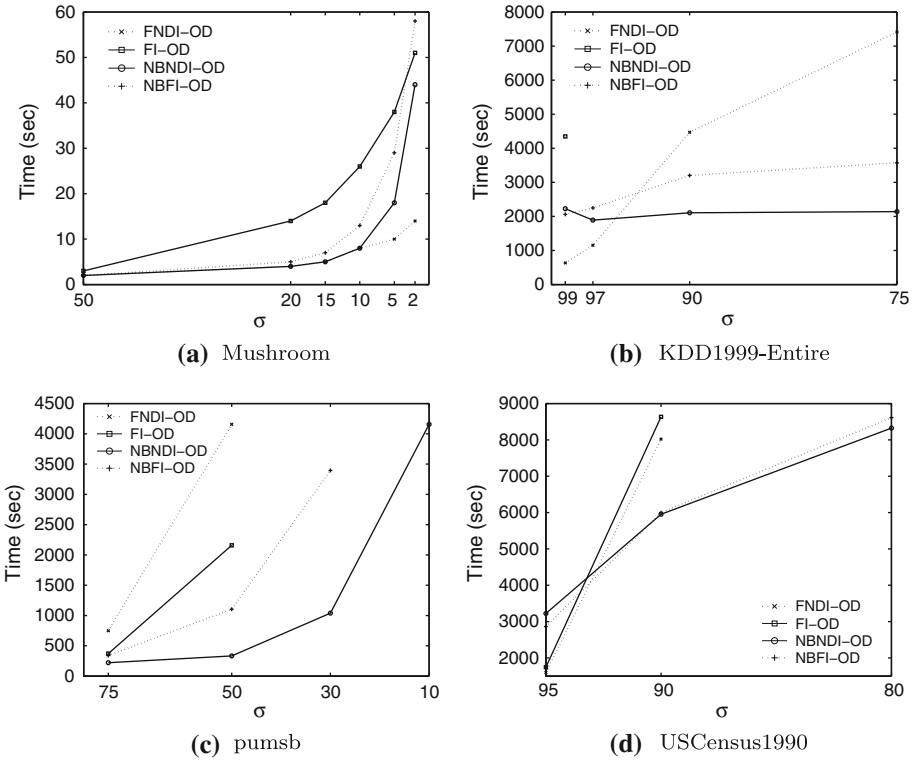


Fig. 3 Runtime performance for FI-OD, FNDI-OD, NBNDI-OD, NBFI-OD as σ decreases ($MAXLEN = 4$)

4.3.2 Restricting the length of generated sets (MAXLEN)

From Table 4, for the KDD1999-10 set ($\sigma = 99.4\%$), FIs with length less than 5 are 1,486 versus a total of 177 NDIs (which is the total NDIs, maximum length of 3). As we can also see in Fig. 3, even with restricting the set length, the FI-OD is still much slower than FNDI-OD as it generates many more sets as σ decreases.

In general, the number of NDI sets generated in total are only slightly larger than the number of NDIs with length less than 5. In contrast, Apriori continues to generate many long derivable itemsets, while NDI prunes these sets and stops at smaller lengths. Therefore, FI-based methods benefit much more from using the *MAXLEN* restriction than NDI-based OD. However, even with using *MAXLEN*, FI-based outlier detection methods are still much slower than NDI-based OD as is also shown in Fig. 3.

In addition, the accuracy of the outlier detection methods may vary depending on the choice of a specific *MAXLEN* value. For example, Fig. 5 depicts the accuracy rates for NBNDI-OD versus NBFI-OD for the Mushroom set and various *MAXLEN* values, and two σ values, 2 and 5%. The best accuracy rates for both algorithms happen in this case for *MAXLEN* equal to 3. However, the CD rate drops significantly for $MAXLEN > 4$ for NBFI-OD while it stays the same for NBNDI-OD. This is because the FI-based methods create many more FIs of longer length that are added to the score, and the difference between outlier score values and normal point score values becomes smaller. This means that the NDI-based methods alleviate the need to choose a suitable *MAXLEN* value for a specific dataset and application.

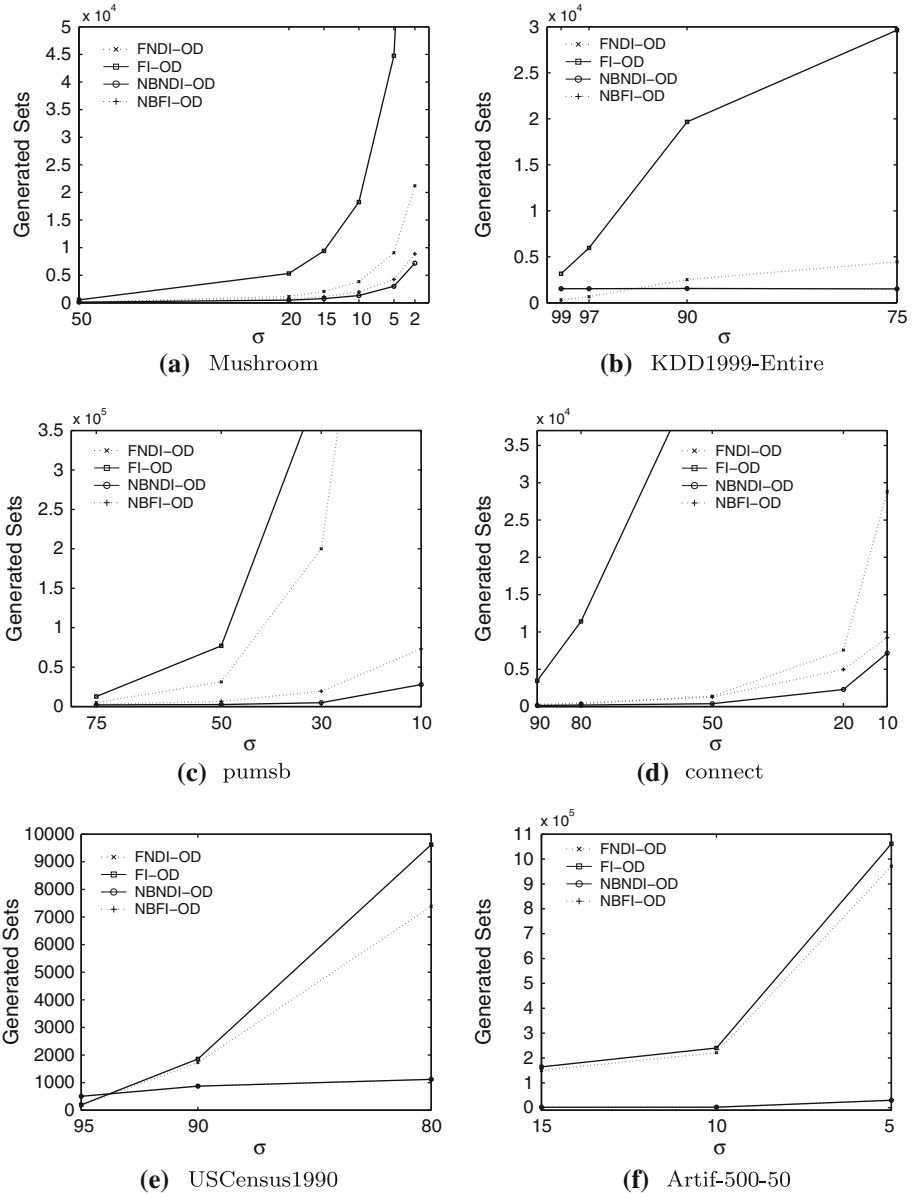


Fig. 4 Generated sets for FI-OD, FNDI-OD, NBNDI-OD, NBFI-OD as σ decreases (*MAXLEN*$= 4$)

The NDI-based OD methods do so also by relying on a much smaller collection of sets than the set of FIs. We also note that for *MAXLEN* $= 3$ and $\sigma = 5\%$, there are 9,076 FIs and 5,105 NDIs, so NDI is still almost half of the FI collection even for lower *MAXLEN* values.

4.3.3 Impact of δ on accuracy and runtime

Table 6a contains accuracy and runtime results for FNADI-OD with several δ values versus FI-OD/FNDI-OD for the Mushroom set and σ equal to 2%. FNADI-OD has similar

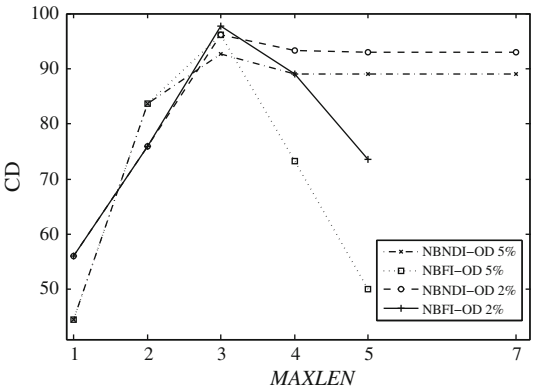


Fig. 5 Effect of *MAXLEN* on the correct detection rates for NBNDI-OD versus NBFI-OD for the mushroom set and two σ values

Table 6 Comparison of Correct Detection (False Alarm) rates, generated sets, and runtime performance in seconds for FI-OD, FNDI-OD, and FNADI-OD

Algorithm	δ	CD (FA)	Sets	Runtime
(a) Mushroom ($k = 700$, $\sigma = 2\%$; actual outliers: 436)				
FI-OD	–	77.1 (8.7)	115,725	51.1
FNDI-OD	–	77.3 (8.6)	21,217	14.2
FNADI-OD	2%	77.5 (8.6)	8,626	6.5
FNADI-OD	5%	77.5 (8.6)	3,659	3.2
FNADI-OD	10%	77.3 (8.6)	1,241	1.4
FNADI-OD	20%	76.1 (8.8)	441	0.8
(b) KDD1999-10 ($k = 5{,}000$, $\sigma = 97\%$; actual outliers: 1,309)				
FI-OD	–	71.1 (4.4)	5,898	721
FNDI-OD	–	71.1 (4.4)	459	74
FNADI-OD	0.02%	71.1 (4.4)	146	24
FNADI-OD	0.05%	71.1 (4.4)	116	21
FNADI-OD	0.10%	71.1 (4.4)	59	18
FNADI-OD	0.15%	61.6 (4.4)	49	12

accuracy to FNDI-OD for δ less than 20%. For example, for δ equal to 10%, FNADI-OD achieves slightly better accuracy than FNDI-OD (and better accuracy than FI-OD). It does so while generating only 1,241 sets and finishing execution in slightly over 1 s versus 8,626 NDIs (14 s) and 115,725 FIs (51 s). Accuracy (correct detection) results for FNADI-OD and FNDI-OD for the Mushroom set are shown in Fig. 6 for various δ and σ values ($\delta = 0$ reflects FNDI-OD). As seen in this figure, the accuracy of FNADI-OD is very close to FNDI-OD for most combinations of the parameter values.

Table 6b compares FNADI-OD with FNDI-OD/FI-OD for the KDD1999-10 set and $\sigma = 97\%$. FNADI achieves the same accuracy as FNDI-OD and FI-OD for δ less than 0.15% (or 150). However, for δ equal to 0.1%, FNADI-OD takes 18 s to finish the task, while FNDI-OD takes 74 s, and FI-OD takes more than 12 min. These results indicate that FNADI-OD closely approximates the performance of FNDI-OD, for a range of δ values, while using a smaller number of sets, and thus offers higher runtime performance gains.

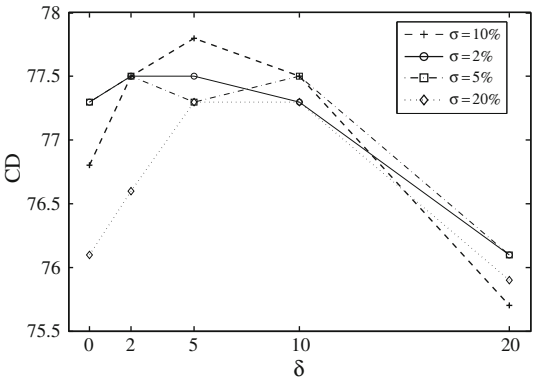
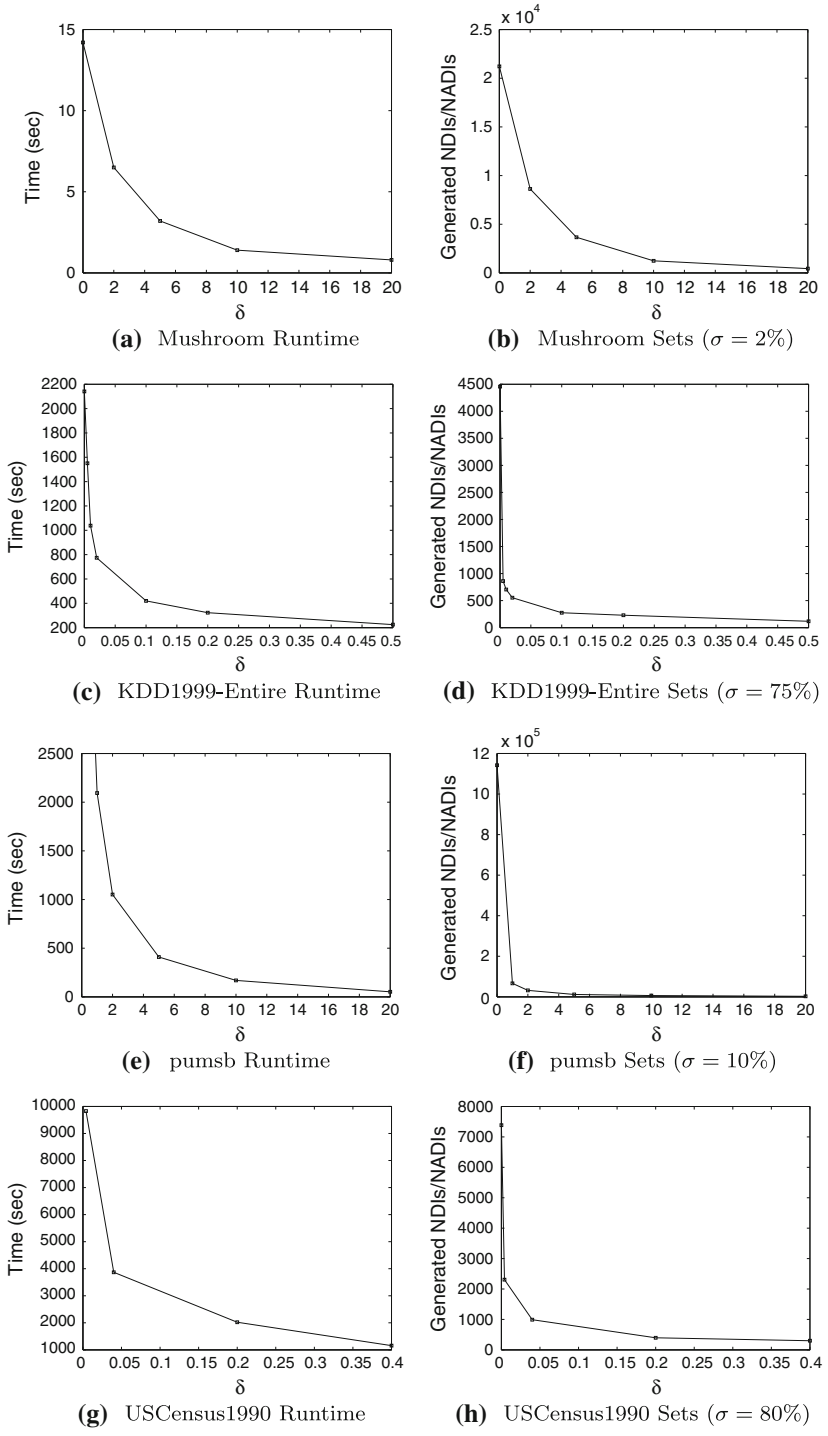


Fig. 6 Correct detection for FNADI-OD versus FNDI-OD for the mushroom set and various δ values ($\delta = 0$ implies FNDI-OD)

Table 7 Comparison of generated sets and runtime performance in seconds for FNDI-OD, and FNADI-OD -δ is a percent of the number of data rows

Dataset	σ%	Algorithm	δ	Sets	Runtime
pumsb	10	FNDI-OD	–	1,142,137	–
pumsb	10	FNADI-OD	1	66,729	2,094
pumsb	10	FNADI-OD	2	31,857	1,051
pumsb	10	FNADI-OD	5	11,814	409
pumsb	10	FNADI-OD	10	6,381	169
pumsb	10	FNADI-OD	20	2,728	52
KDD1999-Entire	75	FNDI-OD	–	4,456	2,141
KDD1999-Entire	75	FNADI-OD	0.005	861	1,551
KDD1999-Entire	75	FNADI-OD	0.010	705	1,038
KDD1999-Entire	75	FNADI-OD	0.020	553	774
KDD1999-Entire	75	FNADI-OD	0.100	275	420
KDD1999-Entire	75	FNADI-OD	0.200	229	323
KDD1999-Entire	75	FNADI-OD	0.500	118	225
Artif-500-50	10	FNDI-OD	–	221,441	–
Artif-500-50	10	FNADI-OD	0.3	19,250	2726
Artif-500-50	10	FNADI-OD	1	1,276	343
USCensus1990	80	FNDI-OD	–	7,386	–
USCensus1990	80	FNADI-OD	0.004	2,307	9,829
USCensus1990	80	FNADI-OD	0.04	991	3,868
USCensus1990	80	FNADI-OD	0.2	396	2,021
USCensus1990	80	FNADI-OD	0.4	301	1,157

Table 7 compares generated sets and runtime for the pumsb, KDD1999-Entire, Artif-500-50, and UCencus1990 datasets. A pictorial representation of these results is shown in Fig. 7. As shown in Table 7 and Fig. 7, small values of δ have a large impact on the runtime of the algorithm. For example, for the pumsb set and $\delta = 5\%$, FNADI-OD has 11,814 sets versus more than 1.1 million NDI sets; moreover, we had to stop the execution of FNDI-OD due



(a) Mushroom Runtime **(b)** Mushroom Sets ($\sigma = 2\%$)

(c) KDD1999-Entire Runtime **(d)** KDD1999-Entire Sets ($\sigma = 75\%$)

(e) pumsb Runtime **(f)** pumsb Sets ($\sigma = 10\%$)

(g) USCensus1990 Runtime **(h)** USCensus1990 Sets ($\sigma = 80\%$)

Fig. 7 Comparison of runtime and generated NADIs vs. NDI as δ increases (sets shown for $\delta = 0$ are NDIs; δ shown as a percent of number of rows in the dataset)

to the amount of time it took to run, while FNADI-OD took about 7 min to run for the same parameter values.

For the Artif-500-50 set and $\sigma = 10\%$, there are 221,441 NDIs, 240,361 FIs, and 2,160 sets on the negative border of FI or NDI (see Table 4), while there are only 1,276 NADI sets for $\delta = 1\%$ (see Table 7). Furthermore, FNADI-OD takes 5 min to finish versus 86 min for NBNDI-OD. The large difference in runtime is the fact that the Artif-500-50 set has a large number of rows (500 thousand), and that the sets on the negative border are saved in a vector instead of the tree structure of the NADIs.

4.3.4 NB-based versus FI-based outlier detection

From Table 3, the NB-based methods seem to have better accuracy than the corresponding FI-based methods. Our assumption is that outliers contain some irregular or infrequent characteristics or values. FI-based scores cannot include these irregular values for high σ values as only the frequent values of a data point are included in its FI-based score. However, we may face a dataset where certain values exist in 90–99% of the points. As almost all points contain these values and their combinations, the FI-based score has no way of distinguishing outliers from normal points for $\sigma = 90\%$. This is in fact what happens for the Mushroom set for a high σ value.

On the other hand, when σ is low, the FI-based score includes irregular values. However, it also includes many of the normal or frequent values. E.g. for $\sigma = 10\%$, sets with 10% frequency are added to the score as well as sets with 90% frequency. This decrease in the σ value might lead to an increase in accuracy as for the Mushroom set (see Table 3a). At the same time, the number of FIs that we need to compare with each point becomes much larger that makes the second phase of the algorithm much slower as seen in Table 5. The NB-based methods for low σ values capture the irregularity of the outliers in the score which leads to better accuracy, usually with the added benefit of a much faster second phase.

The NDI-based method, NBNDI-OD, ensures that the negative border remains relatively small even without the *MAXLEN* restriction. Also, the accuracy rates achieved by NBNDI-OD do not fluctuate for different *MAXLEN* values as is the case with NBFI-OD and the Mushroom data (see Fig. 5).

4.3.5 Which σ works best for a given dataset?

Our main assumption is that outlier points are a small percentage of the dataset $\mathcal{D}$. Therefore, in order for the outlier score to capture the irregularity of the values in the outlier points and successfully detect the outliers we must use a small σ value. This is apparent for the Mushroom data, where better accuracy rates are achieved as σ decreases (see Table 3a) for all methods. In fact, the NB-based methods achieve a much higher accuracy rate for $\sigma < 10\%$. For higher σ values, there are many frequent single values whose combinations are still frequent, a known characteristic of a dense dataset. Therefore, the NB-methods do not capture enough infrequent or irregular characteristics of a point to distinguish the normal from the outlier points.

However for the KDD dataset, the NBFI and NBNDI methods have the same accuracy for all σ values we tested. The reason for this is that the values in the KDD1999-10 dataset are either very frequent or very infrequent as shown in Table 8. A similar number of single values are found infrequent whether σ is equal to 90 or 10%. The negative border of FIs/NDIs also stays relatively the same for different σ values as shown in Table 4, thus the accuracy rates remain the same for different σ values.

Table 8 Frequency of distinct values (items) in mushroom and KDD1999-10 sets

Item frequency range%	Mushroom		KDD1999-10	
	Number of items	%	Number of items	%
[90–100]	3	2	27	2
[50–100]	14	12	36	3
[20–100]	34	30	40	3
(0–10]	63	56	1,135	96
(0–1]	15	13	1,020	86
Total items	113		1,179	

5 Conclusions

Recently, outlier detection techniques were proposed for categorical and mixed-attribute datasets [16,20,22] based on Frequent Itemset Mining (FIM) [2]. These methods aim to extract all frequent sets, or common patterns, in the data and then identify as outliers the points that contain few of these patterns. Even though these methods have been shown to perform well, they face significant challenges for large high-dimensional data where a large number of frequent sets is generated. In this paper, we use *Non-Derivable Itemsets (NDI)*, a condensed representation of FIs presented in [8], in order to detect outliers more efficiently. The NDI-based outlier detection method employs only the non-derivable frequent sets, or NDIs, contained in a point to compute an anomaly score for the point. We also propose an outlier detection method using a δ-based approximate NDI collection, Non-Almost Derivable Itemsets (NADIs) [28].

Specifically, we propose outlier detection schemes based on frequent NDI sets (FNDI-OD), based on the negative border of Frequent NDIs (NBNDI-OD), and based on frequent Non-Almost Derivable Sets (FNADI-OD). Our experiments show that the NDI-based methods present significant runtime advantages compared to their FI-based counterparts. The runtime advantage of NDIs is still apparent even after a user-entered *MAXLEN* parameter is used to restrict the length of generated FIs as in [22]. Overall, FNDI-OD has similar accuracy and false alarm rates compared to FI-OD, while NBNDI-OD achieves the best rates for lower σ values. Moreover, the accuracy of FI-based methods can vary widely based on the choice of the *MAXLEN* value, which is not an issue for NDI-based methods. On the other hand, the approximate method, FNADI-OD, is faster than FNDI-OD, remains feasible for dense data and low σ values, and exhibits accuracy very close to the one achieved by FNDI-OD for a range of δ values.

Future research includes extending our ideas for data with both categorical and continuous attributes, and for distributed datasets.

Acknowledgments This work was supported in part by NSF grants: 0341601, 0647018, 0717674, 0717680, 0647120, 0525429, 0806931, 0837332.

References

1. Aggarwal C, Yu P (2001) Outlier detection for high dimensional data. ACM SIGMOD Record 30(2): 37–46
2. Agrawal R, Srikant R (1994) Fast algorithms for mining association rules in large databases. In: Proceedings int'l conference on very large data bases, pp 487–499

3. Barnett V (1978) Outliers in statistical data. John Wiley and Sons, New York
4. Bay S, Schwabacher M (2003) Mining distance-based outliers in near linear time with randomization and a simple pruning rule. In: Proceedings ACM SIGKDD int'l conference on knowledge discovery and data mining, pp 29–38
5. Blake C, Merz C (1998) UCI Repository of machine learning databases. http://archive.ics.uci.edu (Accessed Sep 2008)
6. Boley M, Grosskreutz H (2009) Approximating the number of frequent sets in dense data. Knowl Inf Syst 21(1):65–89
7. Breunig M, Kriegel H, Ng R, Sander J (2000) LOF: identifying density-based local outliers. ACM SIGMOD Record 29(2):93–104
8. Calders T, Goethals B (2007) Non-derivable itemset mining. Data Min Knowl Discov 14(1):171–206
9. Calders T, Rigotti C, Boulicaut J (2004) A survey on condensed representations for frequent sets. LNCS Constraint-Based Min Inductive Databases 3848:64–80
10. Dokas P, Ertoz L, Kumar V, Lazarevic A, Srivastava J, Tan P (2002) Data mining for network intrusion detection. In: Proceedings NSF workshop on next generation data mining, pp 21–30
11. Fan H, Zaiane O, Foss A, Wu J (2009) Resolution-based outlier factor: detecting the top-n most outlying data points in engineering data. Knowl Inf Syst 19(1):31–51
12. Ganter B, Wille R (1999) Formal concept analysis. Springer, Berlin
13. Hawkins D (1980) Identification of outliers. Chapman and Hall, London
14. Hays C (2004) What Wal-Mart knows about customers habits. The New York Times
15. He Z, Deng S, Xu X, Huang J (2006) A fast greedy algorithm for outlier mining. In: Proceedings Pacific-Asia conference on knowledge and data discovery, pp 567–576
16. He Z, Xu X, Huang J, Deng S (2005) FP-Outlier: frequent pattern based outlier detection. Comp Sci Inf Syst 2(1):103–118
17. Jea K, Chang M (2008) Discovering frequent itemsets by support approximation and itemset clustering. Data Knowl Eng 65(1):90–107
18. Knorr E, Ng R, Tucakov V (2000) Distance-based outliers: algorithms and applications. Int'l J Very Large Data Bases VLDB 8(3):237–253
19. Koufakou A, Georgiopoulos M (2010) A fast outlier detection strategy for distributed high-dimensional data sets with mixed attributes. Data Min Knowl Discov 20(2):259–289
20. Koufakou A, Georgiopoulos M, Anagnostopoulos G (2008) Detecting outliers in high-dimensional datasets with mixed attributes. In: Int'l conference on data mining DMIN, pp 427–433
21. Koufakou A, Ortiz E, Georgiopoulos M, Anagnostopoulos G, Reynolds K (2007) A scalable and efficient outlier detection strategy for categorical data. In: IEEE int'l conference on tools with artificial intelligence ICTAI, pp 210–217
22. Otey M, Ghoting A, Parthasarathy S (2006) Fast distributed outlier detection in mixed-attribute data sets. Data Min Knowl Discov 12(2):203–228
23. Pasquier N., Bastide Y, Taouil R, Lakhal L (1999) Discovering frequent closed itemsets for association rules. In: Proceedings 7th Int'l conference on database theory ICDT, pp 398–416
24. Tax D, Duin R (2004) Support vector data description. Mach Learn 54(1):45–66
25. Wang J, Karypis G (2006) On efficiently summarizing categorical databases. Knowl Inf Syst 9(1):19–37
26. Wu X, Kumar V, Ross Quinlan J, Ghosh J, Yang Q, Motoda H, McLachlan G, Ng A, Liu B, Yu P, Zhou Z, Steinbach M, Hand D, Steinberg D (2008) Top 10 algorithms in data mining. Knowl Inf Syst 14(1):1–37
27. Xiong H, Pandey G, Steinbach M, Kumar V (2006) Enhancing data analysis with noise removal. IEEE Trans Knowl Data Eng 18(3):304–319
28. Yang X, Wang Z, Bing L, Shouzhi Z, Wei W, Bole S (2005) Non-almost-derivable frequent itemsets mining. In: Proceedings int'l conference on computer and information technology, pp 157–161
29. Yankov D, Keogh E, Rebbapragada U (2008) Disk aware discord discovery: finding unusual time series in terabyte sized datasets. Knowl Inf Syst 17(2):241–262
30. Zaki M, Hsiao C (2005) Efficient algorithms for mining closed itemsets and their lattice structure. IEEE Trans Knowl Data Eng 17(4):462–478

Author Biographies

Anna Koufakou received a B.Sc. in Computer Informatics at the Athens University of Economics and Business in Athens, Greece, in 1997, and a M.Sc. and a Ph.D. in Computer Engineering at the University of Central Florida, Orlando, Florida, in 2000 and 2009, respectively. Her research interests include Mining of Large Data Sets, Distributed Data Mining, Outlier Detection, and Frequent Itemset Mining. She is currently an Assistant Professor in Computer Science at the U.A. Whitaker School of Engineering, Florida Gulf Coast University, Fort Myers, Florida.

Jimmy Secretan holds Ph.D. M.S., and B.S. degrees in Computer Engineering from the University of Central Florida, Orlando, Florida. He was a National Science Foundation (NSF) Graduate Research Fellow. He has published in the areas of distributed and privacy preserving data mining, grid computing, machine learning, evolutionary algorithms and web-based systems. He currently works as principal scientist at ad summos, inc., in Celebration, Florida.

Michael Georgiopoulos received the Diploma in EE from the National Technical University in Athens, M.S. and Ph.D. degrees in EE from the University of Connecticut, Storrs, CT, in 1981, 1983 and 1983, respectively. He is currently a Professor in the School of EECS, University of Central Florida, Orlando, FL. His research interests lie in the areas of Machine Learning and applications with special emphasis on neural network and neuro-evolutionary algorithms. He has published more than 60 journal papers and more than 180 conference papers in a variety of conference and journal venues. He has been an Associate Editor of the IEEE Transactions on Neural Networks from 2002–2006, and he is currently serving as an Associate Editor of the Neural Networks journal. He served as the General Chair of S+SSPR 2008, the satellite workshop affiliated with ICPR 2008. He is currently serving as the Technical Co-Chair of IJCNN 2011.