Finding Contrast Patterns for Mixed Streaming Data

Rohan Khade
George Mason University
Fairfax, VA, USA
rkhade@gmu.edu

Jessica Lin
George Mason University
Fairfax, VA, USA
jessica@gmu.edu

Nital Patel
Intel Corporation
Chandler, AZ, USA
nital.s.patel@intel.com

ABSTRACT

Contrast set mining identifies patterns in the data that can best distinguish between groups. Most of the existing work focuses on categorical and batch data, and they do not scale well for large datasets. In this work, we focus on finding contrast patterns for *mixed* (quantitative and categorical) and *streaming* data. We adapt a discretization methodology, Supervised Dynamic and Adaptive Discretization, to identify meaningful bin boundaries. We then use the discretization result to find contrast patterns on streaming data. In order to achieve this, we identify frequent items and then contrast them to a group of interest. To handle potential concept drift, we propose an update strategy to keep the frequent items relevant. In addition, our algorithm samples feature combinations based on their "sampling" score and user feedback to reduce the search space as well as retrieving more interesting patterns.

1 INTRODUCTION

As the amount of data being collected during the semiconductors manufacturing process of increases, the time required to analyze the data and provide relevant feedback also increases. There is a growing need to develop machine learning algorithms to deliver fast feedback to the engineers so that adjustments in the manufacturing line can be made in a timely manner. While the behavior of manufacturing data is often predictable, at times there exist anomalies such as a low yield for certain batch of products. To find the possible cause(s) of this low yield, one approach is to create a model and compare this batch to a normal batch. To achieve this, an engineer needs to know what to look for (the number of features and instances are large), and even after that he/she needs considerable amount of time for analysis. Our intent is to quickly and automatically learn complex patterns in the "population" data (known normal data) and then use these learned patterns to detect differences in the groups. We note here that our goal is data understanding rather than prediction. In prediction such as classification, the goal is to predict an outcome early on in the manufacturing process to reduce costs such as testing. The algorithm typically works like a black box where the user may not understand why the algorithm predicted an outcome. However, in this paper we design a feedback system in which the goal is to identify interpretable patterns that might provide the users helpful insights on potential causes of the differences. Another key difference is that for a prediction algorithms, the "classes" such as "good chips" and "bad chips" need to be known or available in advance in order to build a model. In our proposed work, we seek to detect patterns from a single group of data (the "population"), which will then be compared with incoming data to detect potential differences.

There are several issues that need to be addressed. First, the data may contain both categorical and continuous features, and the features may have high order of interaction between them. Multiple features could contribute to a contrast simultaneously. In this work, we leverage our recently proposed binning strategy, Supervised Dynamic and Adaptive Discretization (SDAD) [8, 9], to address the challenge of dataset containing continuous features. The discretization technique automatically determines the size and number of bins needed in order to find meaningful contrast patterns.

The second challenge we face is the massive data size. A large amount of data is generated daily. To obtain an understanding of the "normal" behavior of the data, we create a frequent itemset database using historical manufacturing data that are known to be normal. However, this data can change as recipes for different chips change. The frequent itemset database must adaptively learn new patterns, update existing ones and discard outdated ones. This is necessary because the representation of the "population" (historical) data needs to stay relevant to the current state of the database. With the updated representation of the population, we can quickly identify potential contrasts between that and a group of interest. If we try to build a database with only a small sample of the population data (e.g. data collected from one day), the data may be biased to the specific machine settings and may not be representative of the population. If the data is too large, e.g. a month worth of data, the data may contain outdated information that bias the results. Building an incremental database of the population itemsets over time can be more efficient and can potentially identify more meaningful patterns. Our algorithm is also easily parallelizable. While this is out of scope for this work, we refer interested readers to [8, 10] for various ways of achieving parallelization to find frequent itemsets.

Moreover, as data streams are monitored, the groups to be compared are not known in advance. Traditional contrast set mining algorithms require the contrasting groups to be known, and then find itemsets which differ significantly across groups. Consider two cases. First, suppose a tester detects a large number of faulty chips on a particular day, and we want to know if there is a difference between chips arriving at this tester and the others. One approach, taken by traditional algorithms, would be to make all the tester IDs as the 'group' label and then find patterns that strongly separate the tester of interest from the rest. As the second scenario, if an oven gives particularly high yield, we would like to know if we can improve the yield of the other ovens. Keeping the oven ID as the 'group' feature, the contrast set algorithm finds itemsets that are different between this oven and the rest. Since the "groups" are not known in advance, our proposed method identifies all frequent itemsets in the population. Once the "sample" of interest (e.g. the oven resulting in high yield) is found, the difference in supports between the population and sample is calculated. This led us to develop an algorithm which identifies frequent patterns[8]. Using frequent itemsets we can identify contrast sets between a sample (data of interest to an engineer) and the population.

The main contributions of the proposed work are as follows:

(1) Given historic data (the “population") and a current data sample, we propose an algorithm that can find contrast sets between any subset of the streaming data and the population quickly. Based on frequent itemsets mining, our approach avoids a lot of re-computation when new groups are identified and new contrasts are needed.
(2) The proposed algorithm leverages user feedback to calculate a sampling score, which improves the patterns returned and reduces computation costs.
(3) We propose an updating strategy that keeps discovered patterns current, discards outdated ones and adds emerging new patterns.

The following section discusses related work. Sections 3 discusses the background on contrast sets and the search strategy adapted in this paper. In section 4, we describe an adaptive and dynamic discretization algorithm for quantitative data that improves upon existing discretization algorithms. We propose an updating strategy for finding contrasts on streaming data, as well as a strategy to use user feedback to compute the sampling probability. In Section 5, preliminary experimental results show that using our approach, we are able to discover the same contrasts that existing algorithms discover, but much more efficiently. Section 6 concludes the paper.

2 RELATED WORK

We discuss related work in frequent set mining and contrast set mining. However, due to the maturity of these areas, the discussion is far from being comprehensive. To the best of our knowledge, there is no other algorithm that has the exact same end goal as ours.

Contrast-set mining has been formally defined as “conjunctions of features and values that differ meaningfully in their distribution across groups” [1]. It can be viewed as a variant of association rule mining. While association rule mining discovers rules or patterns that describe or explain the current situation, contrast-set mining finds patterns that differentiate groups of data by identifying features and values (or conjunctions thereof) that differ meaningfully across them [1]. Knowing the features that characterize the discrepancies across various groups can help users understand the fundamental differences among them, and make independent decisions on those groups accordingly. Therefore, contrast-sets are often presented as sets of rules. The authors in [1] provide the following classic example: on comparing different education groups by asking "What are the differences between people with PhD and Bachelor degrees?âĂİ we might find that P(occupation = sales | PhD) = 2.7 while P(occupation = sales | Bachelor) = 15.8. For manufacturing fault analysis, we might have a query such as "Given two sets of operation sequences: G ("good batch") and B ("bad batch"), what are the differences between them?" The authors in [1] proposed an algorithm, STUCCO (Searching and Testing for Understandable Consistent COntrasts), to find such contrast sets. STUCCO employs efficient search through the space of contrast sets based on another rule mining algorithm, Max-Miner [2]. To assess the meaningfulness of the difference in support values across groups, they use chi-square test on the null hypothesis that the support value is independent of group membership.

Another approach, proposed by [17], discovers that existing commercial rule-finding system, Magnum Opus [18], can successfully perform the contrast-set mining task. The authors conclude that contrast-set mining is a special case of the more general rule-discovery task. A good survey on contrast sets, emerging patterns and subgroup discovery algorithms is provided in [13]. CIGAR [7], apart from using the pruning criterion from STUCCO, adds support, correlation and difference in correlation between the parent and child contrasts. The authors claim that itemsets with a support less than the minimum support may be uninteresting to the analysts. We adapt this assumption for the streaming part of our algorithm. The authors also claim that if the itemset have low correlation, we may find spurious contrasts due to outliers. The authors in [4] use user feedback to sample the patterns and improve the patterns displayed to the user; however, they do not explicitly handle continuous and streaming data.

In [13] we see that Contrast Set Mining, Emerging Pattern Mining and Subgroup Discovery are compatible and hence techniques developed in one domain can potentially be used in another. There is considerably more work done for finding subgroups in numerical domains. The algorithms presented in [11, 15] finds bins for subgroups for mixed data and is implemented in an open source tool Cortana. These techniques usually use an initial discretization method and then merge spaces based on an interest measure. An interesting algorithm described in [6], finds bins for continuous attributes for the problem of subgroup discovery. The algorithm uses optimistic estimates and horizontal pruning to prune the search space. This heavily relies on pruning based on finding the top-k subgroups. Finding all initial split points (exhaustive search in [6]) is expensive but if the initial partitions are not exhaustive but rather frequency or entropy based, the algorithm may miss interesting patterns that occur lower down the tree due to multivariate interactions. The techniques mentioned above are developed for static databases, however the pruning techniques described may be helpful if the groups are known in advance (which does not apply for our application). The current trend of recent algorithms tend to use sampling to improve efficiency and quality of patterns [3–5]. We use some of the sampling techniques explained in these papers and incorporate user feedback to enhance it. This helps improve efficiency and potentially display more meaningful results to the user.

Although our main goal is contrast set mining, we tackle the streaming part of our work using frequent itemsets mining. Frequent itemset mining is typically the first step of association rule mining. This is also where the main computational complexity issues occur. Most existing work in frequent set mining focuses on batch processing and improving the efficiency of the algorithm. However, in many real world applications, new data is continuously generated, e.g. manufacturing processes, sensors etc. General frequent itemset mining algorithms require multiple passes over the database. However, this is not possible in a streaming scenario since data come in high volumes. If we miss a pattern, it may not be possible to go back and check the past data. Since speed is an important issue, some trade-offs in accuracy may be needed. In addition, the space required to store information for future runs and current knowledge may be large.

In general, many streaming algorithms adapt the window model [14]. In landmark window, the goal is to find itemsets between a fixed start point s and current time t. The other option is a sliding window where we are only interested in itemsets found in a time frame. For example, if the window width is w, the goal is to find itemsets in the time $[t - w + 1, t]$. In many cases newer data are more important than older data, in which case a damped window model assigns higher weights to more recent data by defining a decay rate. Time Tilted windows are used to find frequent itemsets over a set of windows. Importance is given to newer data and hence granularity is adjusted as data arrive. Since our goal is to find the general behavior of the data, we use a sliding window strategy, assigning equal weights to all the data.

There are two types of streaming algorithms to find streaming frequent itemsets, true positive algorithms and true negative algorithms. A true positive algorithm does not allow any frequent itemset to be missed. The seminal work by Manku and Motwani [12], called *lossy counting*, uses a user-defined parameter ϵ, and the result contains no itemsets with a support lower than δ - ϵ. The main disadvantage of this approach is that the number of itemsets increases exponentially to guarantee the lower bound. On the other hand, true negative algorithms [19] do not find false positives, but have a higher probability of finding true frequent itemsets. The main limitations of these algorithms are that it is difficult to implement them in parallel; they cannot handle continuous features; and the space requirement may be too high if the data are very large. In this work we use a true negative algorithm; however, as will be discussed later, we can estimate the support of itemsets are missed in the stream of data.

3 BACKGROUND

3.1 Contrast Sets

Bay [1] formally defines contrast set mining as follows. Let A = {a_1, a_2, ..., a_l} be a set of all attribute values for the entire dataset. Let C be a set of all combinations of A. An *itemset c* is a subset of C. Let G = {g_1, g_2, ..., g_m} be a set of groups. Each instance belongs to only one group. If $|g_i|$ is the number of instances in group i and c_i is the itemset of interest in group i, then **support** s_{ic} of an itemset c in group i is the number of instances in g_i that contain c:

$$s_{ic} = \frac{count(c_i)}{|g_i|} \tag{1}$$

An itemset c is **large** between two groups i and j if

$$abs(s_{ic} - s_{jc}) > \delta \tag{2}$$

and **significant** if

$$\chi^2_{ij}(c) < \alpha \tag{3}$$

where α and δ are user-defined parameters.

An itemset is considered a contrast if it is large and significant. Three pruning strategies are used to reduce the search space. (1) An itemset is pruned if it does not have a support over δ in any group (*minimum deviation size* pruning). (2) If its expected occurrence is less than 5 since statistical tests are not significant at that level. (3) By calculating the upper bound value of χ for the itemset's children. To reduce the number of false positives, the value of α is adjusted according to Bonferroni's adjustment explained in [1]. We note here that the pruning strategies used by STUCCO cannot be used in our application since we do not know the groups in advance. However, as will be seen later, we only find itemsets with support greater than δ for the streaming data, and if the group of interest has a support greater than δ, we estimate its support from the population sample.

3.2 Search Strategy

To find combinations of attributes, we need a search strategy. To this end, we adapt the OPUS [16] search tree for our algorithm. However, some modifications are needed since the original tree is designed for categorical attributes. Figure 1 shows the modified

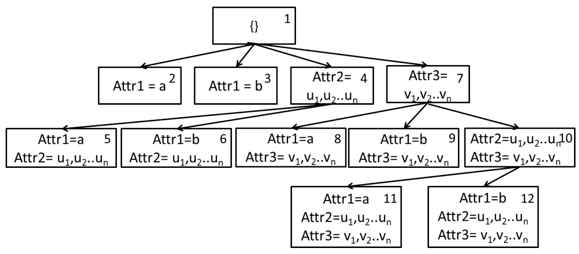
{} 1
Attr1 = a 2
Attr1 = b 3
Attr2= u1,u2..un 4
Attr3= v1,v2..vn 7
Attr1=a Attr2= u1,u2..un 5
Attr1=b Attr2= u1,u2..un 6
Attr1=a Attr3= v1,v2..vn 8
Attr1=b Attr3= v1,v2..vn 9
Attr2=u1,u2..un Attr3= v1,v2..vn 10
Attr1=a Attr2=u1,u2..un Attr3= v1,v2..vn 11
Attr1=b Attr2=u1,u2..un Attr3= v1,v2..vn 12

Figure 1: OPUS Search Tree for Mixed Data

OPUS tree so that it is compatible with continuous attributes and attribute-value pairs instead of directly handling itemsets. Consider an example with one categorical attribute *Attr1* having possible values a and b, and two continuous attributes *Attr2* and *Attr3* having values u_1, u_2 ... u_n and v_1, v_2 ... v_n respectively where n is the number of instances. The order in which the nodes are traversed is indicated by the number in each node. In this example, node 2 with itemset $\{a\}$ is first explored. If it can be asserted that node 2 can be pruned, then we do not need to explore nodes 5, 8 and 11. It should be noted that the traversal order is the reverse of depth-first search. The reason that we use this approach instead of a depth-first approach is to maximize pruning. Although breadth-first search can also be used, we opt for OPUS since the storage overhead is less at each level.

3.3 Streaming Frequent Itemsets / Patterns

Frequent Itemsets (FI's) are itemsets which are present in the dataset with a frequency greater than a user-defined threshold. The formal definition follows closely to that of contrast sets but notice the subtle differences. Using the definition of c from the earlier paragraph, let T be a dataset of transactions. If $|T|$ is the number of rows in the dataset then **support** s of an itemset c is the number of times c occurs in T:

$$s_c = \frac{count(c)}{|T|} \tag{4}$$

A streaming dataset τ is a sequence of indefinite number of datasets of transactions {T_1, T_2, T_3, ...}. Let $T_{k,l}$ = {T_k, T_{k+1},... T_l} where $k < l$. The support s of an itemset c in a time window $[k, l]$ is:

$$s_{c_{[k,l]}} = \frac{count(c_{[k,l]})}{|T_{k,l}|} \tag{5}$$

4 STREAMING CONTRAST SETS

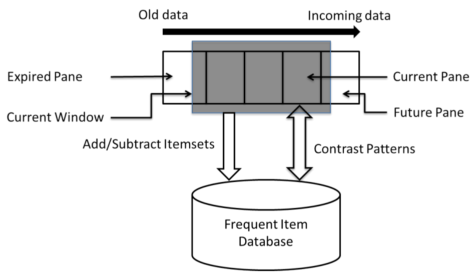

Figure 2: Sliding window for Data Streams

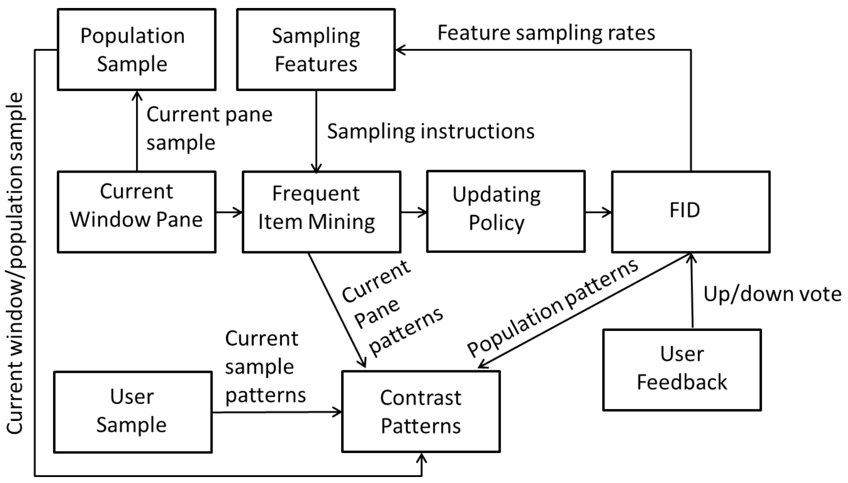

Figure 3: Data flow

We start our discussion by explaining some of the basic terminology used in this section. As shown in Figure 2, data continuously arrive in a data stream. Each block of data is called a **pane**, and a group of panes that are relevant is called a **window**. In Figure 2, suppose a pane represents a day's worth of data, then the window size, λ=3, represents 3 days. Itemsets that are no longer frequent in this window can be discarded.

Figure 3 provides a high-level view of our proposed algorithm. In the next few subsections we discuss what is computed in each block and what information is communicated by each arrow. Simply stated, we pre-compute the frequent itemsets for the current window pane from the population data. This is a time-consuming task, given the massive size of the database. The frequent itemsets discovered from the population data will be saved in a **Frequent Itemset Database** (FID from here on). We note here that this database is not static; it is updated with the arrival of new data as shown in the figure. A data structure such as a prefix tree (like Figure 1) can store this information. When a new **window pane** arrives, frequent itemsets from the current pane will be computed, using techniques explained in a later section. These frequent itemsets are then compared with the population frequent itemsets computed previously to obtain potential **Contrast Patterns**. If the new pane does not conform to the patterns found in the previous panes, the engineers can be notified in a timely manner.

The user can also provide a sample dataset that he or she wishes to compare against the population data, or generate the sample from the archived data on the fly as seen in the **User Sample** block in Figure 3. The frequent itemsets from the sample data are extracted and they will be compared against the itemsets retrieved from the FID for the population data to find **Contrast Patterns**.

Sometimes a pattern that is found in the Current Window Pane or the User Sample may not be in the FID since its support in the population may be below the threshold. To overcome this, we keep a **Population Sample** if the algorithm encounters such a situation. The supports of the population are then estimated from this sample.

The intuition behind our algorithm is that the patterns for the population data represent the "normal" behavior of the population, and comparing the set of patterns from sample data against this "normal prototype" would reveal any differences between the two datasets. Such discovery could potentially identify anomalies in the current or sample data, and shed some light on the causes of the anomalies (by examining the rule differences).

After finding the frequent itemsets for the current pane, the algorithm updates the FID with the new frequent itemsets found using our **Updating Policy**. As new data come in, the distribution and relationships may change among the features. To keep the FID relevant, we need to remove patterns that are no longer relevant and add emerging patterns. If the algorithm finds a new pattern, we delay adding it to the database until we keep finding the same pattern in multiple iterations and if an existing pattern is no longer significant, it is phased out. The user can keep track of these changes in a timely manner.

The **User Feedback** block lets the user decide if a pattern is subjectively interesting. This in turn is fed to the **Sampling Features** block which improves the efficiency and displays more interesting patterns.

In addition to offering the ability to compute the contrast-set in real-time given the sample data, another advantage that our approach offers is that the contrasting patterns are not restricted to having only the values of a feature representing group membership. We will provide more detail on the algorithm in the next section.

4.1 Finding Frequent Itemsets for Current Window Pane

While traversing the OPUS search tree explained in the previous section, if we reach a node with only categorical features, finding frequent itemsets for the current window pane for categorical variables is straightforward, i.e., we just need the support count of the itemsets in the current pane. Therefore, we will concentrate on itemsets that contain continuous features. To this end, we use Supervised Dynamic and Adaptive Discretization for frequent itemsets (SDAD-FI) [8]. We note that any frequent itemset mining algorithm that can handle mixed attributes can be applied here; however, we opt for SDAD-FI since it can handle datasets with mixed attributes and is parallelizable. We also show in [8, 9] that SDAD is able to find better quality bins for frequent itemset mining and association rule mining in a reasonable amount of time compared to other state of the art binning and rule mining algorithms. As opposed to standard discretization techniques such as equi-width or equi-frequency, SDAD-FI does not need to

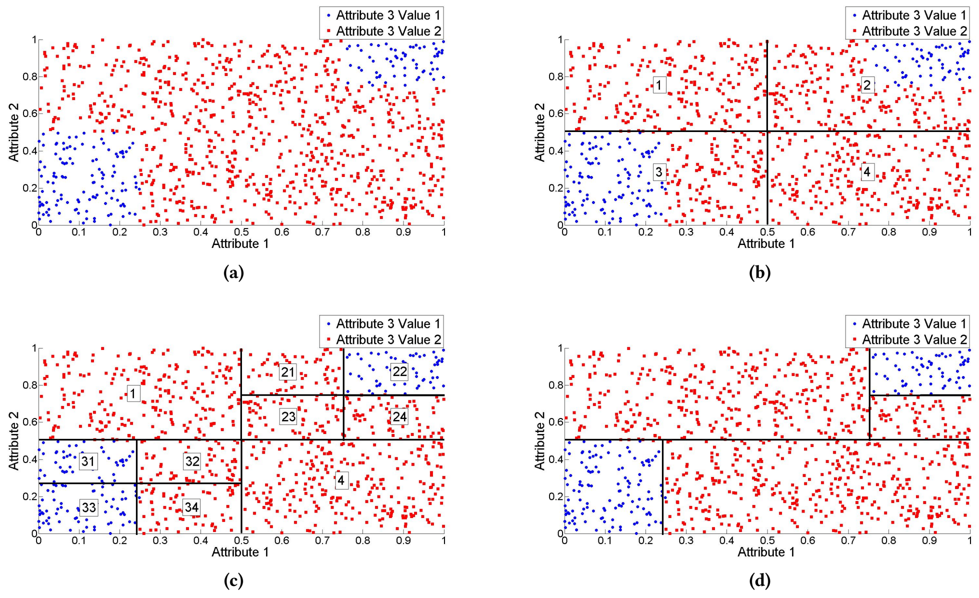

Figure 4: (a) Simulated dataset with 3 features (b) After first partition between feature 1, 2 and 3 (c) After second partition between feature 1, 2 and 3 (d) After merging

know the number of bins to partition in advance, nor the sizes of the bins. It tries to find the "best" bins with respect to the distribution and relationships between features.

The main idea of SDAD-FI is to first employ a top-down approach where the algorithm iteratively partitions the space between features and calculates an interest measure with the goal to obtain regions that satisfy a user-defined homogeneous threshold. We use the interest measure *purity* as defined in [8]. The intuition behind purity is to find the most homogeneous space of the current set of features. We will describe the algorithm with an example. Consider Figure 4a. This dataset contains three features: Attributes 1, 2 and 3. Attributes 1 and 2 are continuous and Attribute 3 is categorical with 2 values indicated by the color and shape of the markers in the scatter plot. In Figure 4b, SDAD-FI divides the continuous features into 4 spaces. Space 1 and 4 (numbers shown in figure) are homogeneous, i.e. it contains only one categorical feature value. However, space 2 and 3 are not. SDAD-FI again divides those spaces as shown in Figure 4c and now finds pure spaces. Since dividing pure spaces do not find "better" bins, the algorithm stops, and this concludes the top-down process.

The next step is to merge contiguous and similar spaces in a bottom-up fashion. Similar spaces are those that are not significantly different according to the χ-squared test. SDAD-FI starts with the smallest contiguous spaces in terms of *n-volume* (area in this case). It first merges space 21 and 23, then 31 and 33, and then 32 and 34. It further merges the combined space of 21 and 23 into space 1, and 32 and 34 into space 4. The final bins are shown in Figure 4d.

Obviously, the example shown is an ideal case. In real world datasets, we usually do not find such clear cut boundaries. However, this example illustrates the flow of the algorithm. In general, SDAD-FI is a greedy algorithm and continuous features are split until splitting does not result in statistically significant differences between the spaces (instead of looking for pure bins). The example discusses a case in which categorical and continuous features co-exist in the itemset. However, SDAD-FI also works when there are only continuous features in the itemset. In this case, the splitting and merging operations are as explained above, with the modification that the interest measure is now the correlation between the continuous features. See [8] for more details.

4.2 Updating Policy and FID

If an itemset contains only categorical values, the Frequent Itemset Database (FID) is updated if a difference in the supports of a Frequent Itemset (FI) is detected between the current pane and the FID using the following update policy. Consider the example in Figure 2. Let the minimum support be δ. If the size of the window is λ, then in Figure 2 we have used λ=3. Now suppose there is an FI in the FID with support s_{FID}, and the same FI is found in the current pane with support $s_{current}$, we calculate the approximate support of the itemset $s_{expired}$ in the expired pane from the population sample (explained later). The support of the FI in the FID can then be updated using a simple weighted average technique.

$$s_{new} = s_{FID} + \frac{s_{current}}{\lambda} - \frac{s_{expired}}{\lambda} \tag{6}$$

If a new FI is found in the current pane with a support greater than δ, but it is not in the FID, then applying the updating strategy explained above would not work since (λ-1)*s_{FID}/λ would be zero and $s_{current}/\lambda$ may not be big enough to be greater than δ to be added to the FID. For example, suppose δ = 0.1. If in the current pane we find an itemset with support of 0.9, and if λ = 10, then s_{new} would be 0.09. As a result, the FI would not be able to enter the FID even if it is found consistently after a certain point. To overcome this, we approximate the support (s_{FID} in the above formula) from the **population sample** and keep a temporary database of new FI's.

The itemset is stored in a temporary database until the itemset attains a support greater than the minimum support. After this point, we say the itemset is indeed frequent and add it to the FID, as well as deleting it from the temporary database. There is a possibility that the current itemset is an anomaly and is not truly frequent. In this case, the itemset needs to be purged from the temporary database. Let the number of panes, starting from the first pane the itemset was found to be frequent, be n. The algorithm keeps track of the actual support (s_{act_supp}) as soon as the itemset was discovered as frequent, and it is calculated by

$$s_{act\_supp} = \frac{(n-1) * s_{act\_supp}}{n} + \frac{s_{current}}{n} \tag{7}$$

As long as $s_{act\ supp}$ is above the minimum support threshold, the algorithm keeps the itemset in the temporary database.

We use this approach to add or purge itemsets because our goal is to maintain a consistent FID which is representative of the population. For example, if an FI is in only one pane, it may be an outlier and not representative of the population. However, if an itemset appears regularly in multiple panes, but is absent for some reason for a short period of time, the algorithm should not purge it out. We note here that keeping the population sample to estimate the support of unknown itemsets reduces the number of frequent patterns needed to be stored like in [11, 19], resulting in less computation. Although the storage overhead is higher to keep the population sample, it serves another purpose. It also helps when the user wants to find contrast patterns on the fly, and the itemset is not present in the FID.

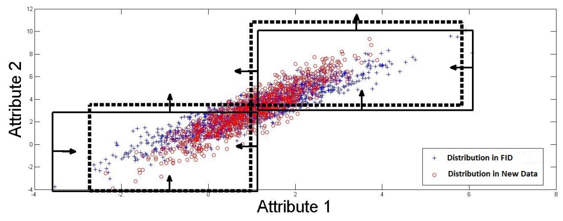

Figure 5: Bin movement

The method explained can be used for frequent itemsets containing only categorical features. Updating FI's with continuous features needs a different approach. Binning is an expensive operation and, for each pane, it is unlikely to have exactly the same bin boundaries even if the distributions are statistically the same because of noise in the data. Instead of calling SDAD-FI at every pane, first, the continuous variables of the current pane are binned by re-using the bin boundaries of the current FID. If the supports between the FID and current pane are not significantly different using the χ-squared test, the bins in the FID are kept unchanged. However, if that is not the case, we discretize the itemset in the current pane by running SDAD-FI. At this stage, we check if there is a slight concept drift or if a completely new relationship is learned between the features. The algorithm conducts a test to check if the new bin boundaries overlap with the current boundaries, and if so, by how much. To do this, there has to be an overlap in every axis. For example, suppose the current itemset has 2 continuous features (the bin boundaries in this case form rectangles, see Figure 5), $x1$ and $x2$ are the vectors of the x coordinates, and $y1$ and $y2$ vectors of the y coordinates of the 2 rectangles, then there is an overlap between the rectangles

if $(((min\_x1 \leq min\_x2 \ \&\& \ min\_x2 \leq max\_x1) \ || \ (min\_x2 \leq min\_x1 \ \&\& \ min\_x1 \leq max\_x2)) \ \&\&$

$((min\_y1 \leq min\_y2 \ \&\& \ min\_y2 \leq max\_y1) \ || \ (min\_y2 \leq min\_y1 \ \&\& \ min\_y1 \leq max\_y2))$

In general, to decide if there is significant overlap, the ratio of the n-volume of the overlapping space to the minimum n-volume of the original space and the new space is calculated. This is a user-defined value (we use 0.8 in our experiments). If there is significant overlap, the algorithm moves the current boundaries towards the new boundaries as shown in Figure 5. The figure shows the distribution of the data points in the FID and the current pane represented by the blue and red points respectively. The bold rectangles represent the bin boundaries of the FID, and the dotted lines are the new bins boundaries found. The arrows indicate the directions of movement of the boundaries of the new bins. The movement is again calculated by using a weighted average method, with the weight determined by the number of days the original pattern was in the FID. For example, if the number of days the pattern was in the database is n and the perpendicular distance to the new hyperplane is x, then the actual distance the current hyperplane moves is x/n. We note here that each plane (in this case line segment) moves independently of each other. The number of days in the FID is then incremented by 1, and the support is approximated using Equation 6 by calculating the supports of the new bins. More specifically, we add the support of the current pane and subtract the support of the expired pane calculated from the population sample.

If the new boundaries of the current pane are significantly different from the original boundaries, the new itemsets with the new bins are stored in the temporary database. In the next panes, the algorithm checks if the data conforms to these new bins or if this behavior was just an anomaly. This procedure is similar for itemsets containing just categorical features.

4.3 Sampling Patterns and User Feedback

During the manufacturing process, the interaction between features are not expected to change drastically. We can exploit this predictability by using a sampling method to decide which itemset should be checked in each pane. Sampling helps reduce the search space for each run (and potentially display more interesting contrasts to the user). This is achieved by calculating a "sampling" score for each itemset in the database. We identified three factors that should contribute to this score.

(1) Number of days in database
(2) Change in support
(3) User feedback

4.3.1 Number of days in database. In the manufacture of semiconductors, we do not expect the interaction between the features to change drastically on a daily basis once the recipe for a chip is fixed. However, concept drift is possible due to aging of machines or inherent properties of the process. This is potentially interesting for our algorithm. Patterns that have been found consistently tend to be stable and show very little variability on the daily basis and hence it is less useful to calculate its interest measure in every pane. Patterns that are newly discovered (i.e. the patterns in the temporary database) tend to have more variability and need to be sampled at a higher rate. If n is the number of days since a pattern is found, and λ is the window size, the $score_{nd}$ is calculated by

$$score_{nd} = 1 - \frac{n}{\lambda} \tag{8}$$

This linear equation produces results ranging from 0 to 1. The score is inversely proportional to the number of days the pattern is present in the database.

4.3.2 Change in support. If a pattern is newly discovered, not found any more or has a high difference in support from what was found previously, the algorithm should sample this pattern at a higher rate. We calculate the $score_s$ as

$$score_s = \frac{abs(s_{FID} - s_{current})}{max(s_{FID},\ s_{current})} \tag{9}$$

This score also ranges from 0 to 1, with 1 meaning that the pattern is new or is not found in the current window. It is also proportional to the change in support.

4.3.3 User feedback. Pattern mining algorithms tend to find spurious patterns due to the bias of the interest measures or due to the domain knowledge of the user (some patterns may be obvious or inherently not interesting). To overcome this bias, the proposed algorithm lets the user up vote or down vote a pattern displayed. Doing so allows the algorithm to learn more important combinations of features over time. With the user voting information, the algorithm will sample these interesting combinations at a higher rate so as to be less likely to miss changes of these features. Let v_{max} be the maximum number of up votes a pattern can receive, v_{min} be the maximum number of down votes a pattern can receive, and v_{up} and v_{down} be the numbers of up and down votes a pattern actually receives from the users, respectively, then

$$score_u = \frac{v_{min} + (v_{up} - v_{down})}{v_{max} + v_{min}} \tag{10}$$

Different users may have different preferences on the types of patterns that are interesting. Hence, we cannot remove the pattern from the FID when a user down votes a pattern. Rather, the algorithm needs to keep track of the votes and not display such pattern again to the same user.

If **w** and **score** are the vectors representing the weight for each score and score calculated, then

$$score_{total} = \mathbf{w} \cdot \mathbf{score} \tag{11}$$

Once the score is calculated, we use a logistic function to calculate the probability ϕ of sampling the itemset

$$\phi = A + \frac{K - A}{1 + e^{score_{total} - mean(score)}} \tag{12}$$

where A and K are the lower and upper bound of the probability of sampling. The expected number of panes after which a pattern's interest measures are calculated is $1/\phi$

4.4 Finding Contrast Patterns and Population Sample

Given a sample, we wish to compare against the population. To this end, we explore the search space of the sample using OPUS search. At each node, the algorithm checks the FID to see if it is an itemset that has been previously found. If it is, and the itemset has only categorical values, then comparing them is straightforward. Let λ be the window size and σ be the average number of tuples present in each pane, then the total number of itemsets (needed for χ-squared calculation) is given by:

$$N = \lambda * \sigma \tag{13}$$

Furthermore, let s_c be the support of itemset c in the database, the number of times c is present in the database is:

$$N_c = s_c * N \tag{14}$$

After finding the counts of an itemset, the algorithm can proceed to check whether the itemset is large and significant as explained earlier in the background section. Notice here that we are only interested in itemsets that are more frequent in the sample than the population.

If the itemset is present in FID and contains at least one continuous feature, the algorithm discretizes the continuous features using the bin boundaries previously found and recorded in the FID for these same features. Once discretized, the algorithm can treat the features as categorical features, and can continue as explained above. If the itemset is not large and significant, this suggests that the distribution and relationship between features is not different in the sample.

A problem arises if the distribution or relationship between features change, or if the itemset is not frequent in the population but found in the sample (since that information is potentially lost). To overcome this, we keep a population sample **d** that fits in memory. At each window pane, the algorithm randomly samples k instances of the current transactions. If the algorithm has not finished running for at least (λ) panes, k instances are added to the end of **d**. Otherwise, when a new window pane arrives, the algorithm removes the first k instances and adds the current sample to the end. Therefore, the maximum number of tuples in **d** is $k * \lambda$. Now, if an itemset which is frequent in the current window pane is not found in the FID, the algorithm can go to **d** and calculate the approximate support. Again, if the itemset contains only categorical features, this is straightforward. If the itemset contains continuous features, the algorithm discretizes the samples itemset using SDAD-FI, and uses the bin boundaries to calculate the support of those bins from **d**.

5 EXPERIMENTAL EVALUATION

Experimental Setup: For all the experiments, we keep *min pr* difference 0.1 (for merge step in SDAD-FI), λ =5, α = 0.05 and δ = 0.1 (same as [1]). The datasets used to quantitatively compare the performance of the algorithms are shown in Table 1. Each dataset is divided into 5 random partitions. The parameters A and K are set to 0.2 and 0.8 respectively, which means that each itemset has a minimum of 0.2 and maximum of 0.8 probability to be tested in each pane. For user feedback, we randomly up vote or down vote an itemset with probability 0.1. The population sample is capped at 10% of each partition. A minimum support of 0.05 is used for experiments in Table 3.

5.1 Public Datasets

These are static datasets. By creating random partitions, we should find similar distributions of the features in each partition. We are not aware of any algorithm that works similarly to ours, or finds contrast patterns for mixed and streaming data. Hence, the goal of these experiments is to verify that the proposed approach is able to find contrast patterns comparable to the actual contrasts found if the entire static dataset were used. We compare our algorithm with STUCCO on the entire dataset (since it is a batch algorithm), by binning the continuous attributes using the bins found by SDAD-FI, and then running STUCCO on the discretized data.

Table 1: Public Datasets

Dataset	Sample	No. of instance Dataset/ Sample	No. of Features/ Continuous Features
Adult	Doctorate	48842/594	13/5
Spambase	Spam	4601/1813	57/57
Shuttle	High	54489/8903	9/9
Credit Card	Yes	29998/6635	24/23
Census Income	Above 50K	199523/12382	39/11

Table 2: Average Time in seconds to find Frequent Patterns in each Pane for Public Datasets

Dataset	Support=0.05	Support=0.1	Support =0.2	Support=0.3
Adult	17.36	16.49	10.37	10.69
Spambase	27.18	23.88	24.11	26.09
Shuttle	9.13	11.02	7.68	11.03
Credit Card	29.41	28.08	23.7	32.56
Census Income	232.25	189.77	154.98	157.08

Table 3: Quantitative Analysis of Contrast Sets for Public Datasets

Dataset	Time to find Contrast using SDAD FI (seconds)	Time to find Contrast on entire dataset (seconds)	True Positive	False Positive	False Negative
Adult	3.09	65.02	40	14	26
Spambase	121.59	952.87	27	2	2
Shuttle	4.25	62.33	9	0	0
Credit Card	45.25	482.74	8	0	0
Census Income	148.72	1654.85	228	0	24

The 5 partitions are fed into the algorithm sequentially and the FID is updated. Finally, the "User Sample" (shown in the table as sample of interest) is compared to the FID.

In Table 2, we report the average time it takes to find frequent patterns for each pane for different minimum supports. The results indicate that not only the minimum support affects the running time, but the number of items needed to be updated in the FID also matters. We notice that for some datasets, even increasing the minimum support increases the running time, which is counter intuitive. We can compare the first column (Support = 0.05) of this table with running SDAD-FI on the entire dataset in Table 3. The total running time is actually greater (average time multiplied by 5) because of the time require for updating and computation, however, we see that once the algorithm pre-computes the frequent itemsets, it can find contrast patterns quickly (see first 2 columns of Table 3). We also compare the difference between the contrasts found using the windowing procedure and the actual difference in support by showing the true positives, false positives and false negatives.

The results show that our approach finds contrasts comparable to contrasts found when using the entire dataset. It also shows that the time it takes to find the contrasts is much faster at run time than using the entire dataset.

5.2 Simulated Dataset

We simulated a dataset which is a simple but typical scenario in our application, semiconductor manufacturing. This dataset gives an overall idea of what we are trying to achieve. The dataset contains 4 features: the date, the oven id, temperature and a

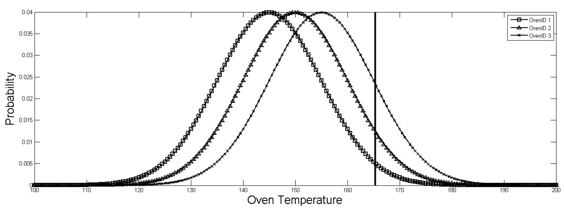

Figure 6: Simulated dataset temperature data distribution

Table 4: Frequent-1 Items found in Population by SDAD-FI for Simulated Streaming Dateset

Frequent Item	Support
Response0	0.93
Response1	0.07
Temperature <= 150.08	0.5
Temperature > 150.08	0.5
Oven2	0.33
Oven1	0.33
Oven3	0.33

Response which indicates a pass or fail for a particular test. Each instance represents a chip and has a unique id. In the simulated data, we have 3 oven ids, and for each oven the temperature is controlled individually and follows a normal distribution. We simulate this for 10 batches, each having 600 instances, 200 for each oven. The probability distribution of the data is shown in

Table 5: Contrast Sets for Simulated Streaming Dataset

Serial Number	Contrast Set	Support for Population	Support for Sample
1	$Temperature <= 150.08$	0.5	0.31
2	$Temperature > 150.08$	0.5	0.69
3	$Temperature <= 150.08$	0.5	0
4	$Temperature > 150.08$	0.5	1
5	Oven3	0.33	0.58
6	Oven1	0.33	0.11
7	$Temperature <= 150.08$	0.5	0.45
8	$Temperature > 150.08$	0.5	0.55
9	Response 1	0.07	0.11
10	Response 0	0.93	0.89

Table 6: Contrast Sets for Semiconductor Manufacturing Streaming Dataset

Serial Number	Contrast Set	Support for Sample	Support for Population
1	$Tester - KAWM$	0.51	0.27
2	$Tester - KAWM$ *and Burn in Flag* $- 80$	0.51	0.25
3	*Unit Location at CAM* $- JVFT$ *and Entity at Deflux* $- VAKM$	0.55	0.33
4	*CAM Location* $- 70$	0.50	0.33
5	$10.6435 <=$ *CAM PEAK TEMP STD* $<= Inf$	0.55	0.42
6	$91.5456 <=$ *CAM DIE TIME ABOVE AVG* $<= Inf$ *and PRODUCT* $- YBDF$	0.47	0.33

Figure 6. Ovens with IDs 1, 2 and 3 have mean temperatures 145, 150 and 155 respectively, and each has a standard deviation of 10. Instances that failed the test have Response = 1, while those that passed have Response = 0. The probability of failure follows a sigmoid distribution where the mean of the sigmoid is 165, as shown in Figure 6. The minimum probability of failure is 0.05, and maximum is 0.9. Hence the probability of failure for an instance is given by

$$\phi = 0.05 + \frac{0.9 - 0.05}{1 + e^{temperature - 165}} \tag{15}$$

Using this scenario, we try to answer 3 questions: (1) We notice that the oven with $OvenID = 1$ yields a high number of chips that passed the test. What is the difference between the daily runs of $OvenID = 1$ and the population? (2) On similar grounds as (1) but globally, given a sample of chips that have failed the test, can we find meaningful differences? (3) If, on a particular day, we change the temperature, can we provide a timely feedback on how it is affecting the Response? We note here that to answer the questions above, a traditional contrast set algorithm needs to be run thrice on the entire data, which is time consuming since the data is large in a real world scenario. Table 4 shows the Frequent 1-Items found.

For the first scenario, we took a sample that contains only $OvenID = 1$. In Table 5, the first 2 contrasts are the ones that are significant at $level = 1$. As the contrasts show, OvenID=1 is operating at a lower temperature.

In the second scenario, looking at contrasts 3 to 6 we notice that the bad samples are baking at a higher temperature, and Oven3 is failing much more than average. Looking at the contrasts, the analyst can recommend operating the oven at a lower temperature.

In the third scenario, we increased the mean of each oven by 2, keeping the standard deviation at 10 for a particular batch. When the algorithm runs daily, it finds that this batch has a higher rate of failure than usual and can notify the analyst as soon as something goes wrong. From the contrast, we see again that the increase in temperature might be the cause of the high rate of failure.

All the contrasts shown find very simple contrasts but provide useful feedback to the analyst. Some of the frequent 2-itemsets found are $Temperature <= 165.02$ *and Response* $= 1$, $Temperature > 165.02$ *and Response* $= 0$. In a real life scenario, contrasts can be more complex, and bin boundaries can be fuzzy. The frequent itemset mining algorithm is capable of finding these rules [8]. In initial runs, the date feature was listed as strong contrasts. This was down voted by the user and soon the algorithm learned it was not an important feature, hence reducing its sample rate to the minimum.

5.3 Semiconductor Manufacturing Dataset

We conducted experiments on real production data of a certain product group. Note that the data was cleaned and modified to remove any proprietary information, although relationships between attributes remain unchanged. The goal of this experiment is to see whether one can quickly identify the root cause of why parts are failing a specific test. The dataset consists of 15 daysâĂŹ worth of data. Each day consists of 3,000 to 20,000 instances of individual units, and each instance has 174 features. We built an FID, and we then contrasted the failing units to this database. The contrasts found are shown in Table 6.

The contrasts found shed some light on the potential causes of the problems. In Table 6 contrasts 1 and 2 show that most of the failed parts are routed through the same tester and had been flagged for burn in. Contrasts 3 and 4 show there was a higher likelihood a particular chip attach tool was used (CAM Location) and location of the unit in the tray (towards the back of the oven) was used. This may explain contrasts 4 and 5 which state that the chip attach reflow temperatures were on the higher end of the spectrum. So in summary, this situation exhibits a marginality between the chip attach process, the presence of burn-in and potential marginality at that specific test equipment. With this information, engineers can make changes at these particular stages of the manufacturing line, which may in turn improve the process yield.

As mentioned earlier, we do not expect the distribution and relationships between the features to change a lot on daily basis, and hence the sampling procedure will help improve the running time. The first pane of the algorithm is run without any sampling. Since the FID is empty, all bin boundaries have to be calculated. The average time for the first run is 2456 seconds. After the first pane, the average time for a pane to update the FID with the sampling procedure explained earlier is 182 seconds, and without sampling it is 515 seconds. This improvement in the running time is significant, but the contrasts found do not differ significantly. We calculated the difference in supports between the population and sample for the top 25 contrasts in the experiment without sampling. The corresponding contrasts from the experiment with sampling were extracted. As expected, the continuous attribute bins were slightly shifted when compared between the 2 experiments. The absolute average difference of differences of supports for population and sample between the 2 experiments is 0.017 with a standard deviation of 0.041. Their distributions are not significantly different at $\alpha = 0.05$ according to the Wilcoxon signed rank test.

6 CONCLUSION

In this paper, we propose a method to find Contrast sets in mixed and streaming data by extending a well-studied approach. Using a binning strategy that automatically finds the size and number of bins for the continuous features, we are able to find meaningful contrasts even when higher order interactions occur. An updating strategy was discussed to keep patterns relevant. To find contrast patterns, we first find the frequent patterns for the population, which is then contrasted to the sample of interest. This algorithm does not need the group information in advance. The main motivation to find these types of contrast patterns in streaming data is to provide timely feedback to analyst during the semiconductor manufacturing process. The next step for our work is to improve the parallelization capacity for our algorithm to run on even larger datasets. To achieve this, we plan to distribute the work evenly among the cluster, based on the estimates we calculate from previous iterations.

REFERENCES

[1] Stephen D Bay and Michael J Pazzani. 2001. Detecting group differences: Mining contrast sets. *Data mining and knowledge discovery* 5, 3 (2001), 213–246.

[2] Roberto J Bayardo Jr. 1998. Efficiently mining long patterns from databases. *ACM Sigmod Record* 27, 2 (1998), 85–93.

[3] Guillaume Bosc, Chedy Raïssy, Jean-François Boulicaut, and Mehdi Kaytoue. 2016. Any-time diverse subgroup discovery with monte carlo tree search. *arXiv preprint arXiv:1609.08827* (2016).

[4] Vladimir Dzyuba and Matthijs van Leeuwen. 2017. Learning what matters–Sampling interesting patterns. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining*. Springer, 534–546.

[5] Vladimir Dzyuba, Matthijs van Leeuwen, and Luc De Raedt. 2017. Flexible constrained sampling with guarantees for pattern mining. *Data Mining and Knowledge Discovery* 31, 5 (2017), 1266–1293.

[6] Henrik Grosskreutz and Stefan Rüping. 2009. On subgroup discovery in numerical domains. *Data mining and knowledge discovery* 19, 2 (2009), 210–226.

[7] Robert J Hilderman and Terry Peckham. 2005. A statistically sound alternative approach to mining contrast sets. In *Proceedings of the 4th Australia Data Mining Conference (AusDM-05)*. 157–172.

[8] Rohan Khade, Jessica Lin, and Nital Patel. 2015. Frequent Set Mining for Streaming Mixed and Large Data. In *Machine Learning and Applications (ICMLA), 2015 IEEE 14th International Conference on*. IEEE, 1130–1135.

[9] Rohan Khade, Nital Patel, and Jessica Lin. 2015. Supervised Dynamic and Adaptive Discretization for Rule Mining. In *SDM Workshop on Big Data and Stream Analytics*.

[10] Ming-Yen Lin, Pei-Yu Lee, and Sue-Chen Hsueh. 2012. Apriori-based frequent itemset mining algorithms on MapReduce. In *Proceedings of the 6th international conference on ubiquitous information management and communication*. ACM, 76.

[11] Michael Mampaey, Siegfried Nijssen, Ad Feelders, and Arno Knobbe. 2012. Efficient algorithms for finding richer subgroup descriptions in numeric and nominal data. In *Data Mining (ICDM), 2012 IEEE 12th International Conference on*. IEEE, 499–508.

[12] Gurmeet Singh Manku and Rajeev Motwani. 2002. Approximate frequency counts over data streams. In *Proceedings of the 28th international conference on Very Large Data Bases*. VLDB Endowment, 346–357.

[13] Petra Kralj Novak, Nada Lavrač, and Geoffrey I Webb. 2009. Supervised descriptive rule discovery: A unifying survey of contrast set, emerging pattern and subgroup mining. *Journal of Machine Learning Research* 10, Feb (2009), 377–403.

[14] Kostas Patroumpas and Timos Sellis. 2006. Window specification over data streams. In *International Conference on Extending Database Technology*. Springer, 445–464.

[15] Matthijs van Leeuwen and Arno Knobbe. 2012. Diverse subgroup set discovery. *Data Mining and Knowledge Discovery* 25, 2 (2012), 208–242.

[16] Geoffrey I Webb. 1995. OPUS: An efficient admissible algorithm for unordered search. *Journal of Artificial Intelligence Research* 3 (1995), 431–465.

[17] Geoffrey I Webb, Shane Butler, and Douglas Newlands. 2003. On detecting differences between groups. In *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 256–265.

[18] Geoffrey I Webb and Songmao Zhang. 2005. K-optimal rule discovery. *Data Mining and Knowledge Discovery* 10, 1 (2005), 39–79.

[19] Jeffery Xu Yu, Zhihong Chong, Hongjun Lu, and Aoying Zhou. 2004. False positive or false negative: mining frequent itemsets from high speed transactional data streams. In *Proceedings of the Thirtieth international conference on Very large data bases-Volume 30*. VLDB Endowment, 204–215.