Towards Distributed Emerging Pattern Mining on Itemset Streams

Björn Jacobs and Henrik Grosskreutz
Fraunhofer IAIS
{firstname.lastname}@iais.fraunhofer.de
Schloss Birlinghoven
53754 Sankt Augustin

Abstract

In this paper, we present a new approach for mining emerging patterns in itemset streams. Unlike in the classical setting, we do not assume that two datasets are given and that the task is to find itemsets occurring more often in one dataset than in the other. Instead, we look for itemsets that were seldom in the stream previously, but recently occur much more often. Our approach differs from earlier approaches in that it employs a distributed representation of the candidate space, which is scattered over a cluster of machines. This paper describes work in progress, meaning that it concentrates on the approach but lacks experimental evaluation.

1 Introduction

In this paper, we present a new approach to perform *Emerging Pattern Mining* on *data streams*. The task of emerging pattern mining [Dong and Li, 1999] considers the classical itemset setting, where the data to be analyzed takes the form of sets of items. A classical example is market basket analysis, where every data record, or "transaction", consists of the set of items bought together by a particular customer. Another example is Twitter, where a transaction corresponds to an individual tweet and the items correspond to the words within that tweet.

The goal of emerging pattern mining is to "capture significant changes between datasets", and to "capture emerging trends in business or demographic data" [Dong and Li, 1999]. In the market basket example, the emerging patterns would hence be sets of items that are often bought together in one dataset (for example, in recent sales data), but were only seldom bought in another dataset (older sales data). In the Twitter example, the emerging patterns would be words that co-occur often in one dataset (recent tweets), but seldom in another dataset (older tweets).

While classical algorithms operate on a static dataset, we consider the setting where the data takes the form of a stream. That is, we assume that the input is a potentially infinite sequence of transactions – as in the twitter example. Our goal is then to find, at any time, the current set of emerging patterns, that is, the itemsets that occur much more frequently in recent transactions than in older transactions. Ultimately, this allows identifying new emerging trends or topics.

Our approach is based on the idea to keep, in memory, a representation of the whole space of candidate patterns. With every candidate pattern, we store a set of statistics, which are updated whenever a new transaction comes in. As the space of candidate patterns can be extremely large, we aim for a *distributed computation approach*, where the candidate space (and the computation) can be distributed on a cluster of machines. The main intention of this approach is not to speed up the process (which nonetheless can be a positive side-effect), but to scale out horizontal in order to realize very large candidate spaces. To this end, we base our implementation a framework supporting the distribution of data and computation to multiple nodes - in our case the Akka toolkit.

This paper presents work-in-progress. That is, it describes the problem and the intended approach, but it lacks conclusive results. The finalization of the implementation and the evaluation of the system are the topic of an ongoing master thesis. Nevertheless, this paper already provides important contributions. In particular:

- We present our approach and show that its memory requirements are only logarithmic in the number of incoming transactions and linear in the size of the candidate pattern space (Section 4.1);
- We describe how our approach can be distributed on a cluster of computing nodes, which allows dealing with very large candidate spaces (Section 4.2).

The remainder of this paper is structured as follows: After a brief discussion of related work in Section 2 and a review of the task and the standard approaches in Section 3, we present our new approach in Section 4. Subsequently, we describe our prototypical implementation in Section 5, before we conclude in Section 6.

2 Related work

In recent years, a lot of research has been done to deal with change in data streams (also called *concept drift*). A large share of the proposed approaches, however, do not fit to our setting because they make the assumption that the incoming data streams contain a *label* and their goal is to find a classifier for future data (e.g. [Alhammady and Ramamohanarao, 2005; Wang *et al.*, 2005]).

An approach very similar to ours is the work of [Kifer *et al.*, 2004], who introduce a meta algorithm based on a 2-window-approach on unlabeled data streams. The algorithm measures the distance between the probability distributions of the items in each window and announces change if it lies above a certain threshold. The main difference between this approaches and ours is that we distribute the task over a cluster of machines, exploiting distributed memory to deal with very large candidate spaces, and using parallel computation for speed-up. This also distinguishes our approach from the numerous approaches to the related task of

itemset mining over data streams [Cheng *et al.*, 2008]. Another approach for parallel item-set mining is the work of [Li *et al.*, 2008]. It differs from our approach in two central ways. First, the algorithm is not designed to work on data streams and second, it only implicitly represents the search space by distributing the database, where our approach represents the search space explicitly.

3 Preliminaries

In this section, we provide a formal definition of the task of emerging pattern mining, and describe existing approaches.

3.1 Emerging Patterns and Supervised Descriptive Rule Discovery

Emerging pattern mining [Dong and Li, 1999] belongs to a family of tasks known as *supervised descriptive rule discovery* [Kralj Novak *et al.*, 2009], which also includes the tasks of subgroup discovery and contrast set mining. The input consists of a sequence of transactions $T_1, \ldots, T_m$. Every transaction consists of a set of items, i.e. $T_i = (i_{i,1}, \ldots, i_{i,N_i})$ where every item stems from a fixed universe of items.

A pattern P is also a subset of items, i.e. $P = i_1, \ldots, i_n$. We say that a transaction T_i *contains* a pattern P if and only if $P \subseteq T_i$. The (relative) *support* of a pattern P in a sequence of transactions $DB = T_1, \ldots, T_m$, denoted by $supp(P, DB)$ refers to the share of transactions in the sequence containing P.

In the classical setting, one is given two datasets DB_1 and DB_2, and the goal is to find patterns that have a noticeably higher support in DB_2 than in DB_1. This difference in support is measured using some *quality function*, which assigns a real-valued figure to any given pattern. The higher the figure, the more salient the difference in supports is considered. Different quality functions have been proposed in the supervised descriptive rule mining community. One example is the weighted relative accuracy (WRACC) [Lavrac *et al.*, 2004], defined as follows:

$$WRACC(P, DB_1, DB_2) = supp(P, DB_2) \times (1 - p_0) - supp(P, DB_1) \times p_0$$

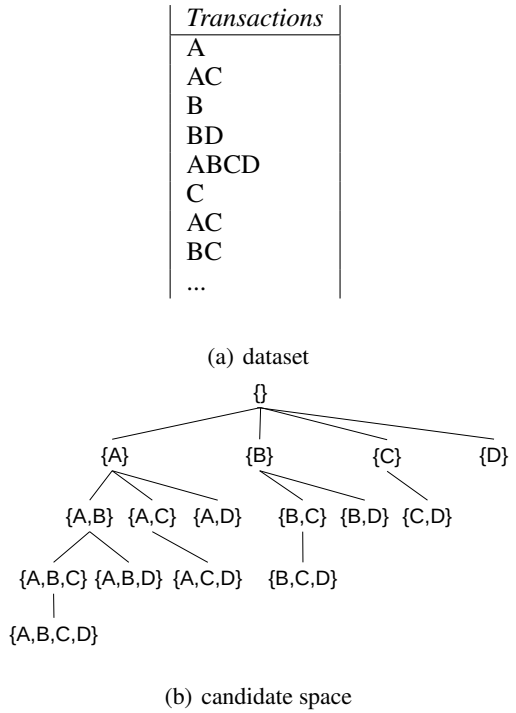
Here p_0 is defined as $\frac{|DB_2|}{|DB_1 \cup DB_2|}$.

Based on these definitions, the task of *top-k emerging pattern mining* is to find the k patterns having highest quality (or, in case of ties, a set of k maximum-quality patterns).

3.2 The Classical Approach To Supervised Descriptive Rule Discovery

The classical computational approach to supervised descriptive rule discovery is to load the whole dataset into memory and to traverse the search space of candidate patterns. Figure 1 will help illustrating this approach: it shows an example built from the 4 items A, B, C and D. Figure 1(a) shows the dataset and the corresponding search space is visualized in Figure 1(b). In this figure, the candidate patterns are arranged in levels, where every pattern in a level has the same cardinality, i.e. is built from the same number of items.

The different supervised descriptive rule discovery algorithms explore the search space in different ways. Some approaches apply some heuristic search and only explore a subset of the search space [Lavrac *et al.*, 2004], while other approaches exhaustively traverse the complete search space, e.g. relying on some tree traversal algorithm



(a) dataset

(b) candidate space

Figure 1: An example dataset and its corresponding candidate space

[Grosskreutz *et al.*, 2008]. Whenever a node is visited, the algorithm computes statistics about the pattern, in particular the support, which allows evaluating the quality of that candidate. The algorithms keeps track of the highest-quality patterns during traversal. Once the traversal ends, the top-quality pattern(s) are returned as result.

From the perspective of this paper, the details of these approaches are less important than the fact that for every candidate, these approaches have to compute statistics based on the *whole sequence of itemsets*. Typically, the algorithms will keep the dataset in memory to speedup this task, possibly relying on some efficient data structures [Han *et al.*, 2000; Atzmüller and Puppe, 2006]. This approach is, however, not applicable with *streams* of potentially infinite length.

3.3 The Sampling Approach of Scheffer and Wrobel

A completely different approach was proposed by Scheffer and Wrobel [Scheffer and Wrobel, 2002]. The main goal of this approach was to reduce the computation time by trading the exact-solution-guarantee for probabilistic guarantees with fixed bounds on confidence and error. To this end, the paper proposes a randomized algorithm which iteratively (1) draws a sample record, and (2) uses that sample to update the statistics of *all* candidate patterns. Once the algorithm can guarantee, with sufficiently high probability, that a candidate will have low or high quality, it is discarded respectively accepted at runtime. The iterations continue until with sufficiently high probability the k best-quality patterns are found.

4 Mining Emerging Patterns in Streams

We will first specify the task we consider, i.e. mining emerging patterns in the setting of data stream: The input consists of

$[t_0$ - $t_1)$	$[t_1$ - $t_2)$	$[t_2$ - $t_3)$	...	$[t_{m-1}$ - $t_m)$
5	8	13	...	27

Figure 2: Statistics stored for every candidate pattern

- an itemset stream, that is a (potentially infinite) sequence of pairs (T, t), where T is a transaction and t a timestamp. We assume the timestamps to be monotonically increasing.
- an integer k, and two timeframes W_{old} and W_{new}.

The output is a mapping, from every time t to the emerging patterns wrt. $DB_{new}(t, W_{new})$ and $DB_{old}(t, W_{new}, W_{old})$. Here, $DB_{new}(t, W_{new})$ consists of all "new" transactions having a timestamp in the interval $(t - W_{new}, t]$, and $DB_{new}(t, W_{new})$ consists of all "old" transaction occurring in the interval $(t - W_{new} - W_{old}, t - W_{new}]$.

4.1 Our Approach

Our approach is based on the idea of [Scheffer and Wrobel, 2002], namely to store a representation of the whole search space, together with statistics about the support of every candidate pattern. Unlike them, however, we are not concerned with probabilistic guarantees, as we do not aim for a randomized algorithm.

Another difference is that here, we are concerned with streaming data, and that our goal is to compute the emerging patterns with respect to two time windows. To this end, we do not just store, for every pattern, the number of occurrences in two datasets, but instead have to store all information required to continuously calculate the support in the two windows.

As storing all incoming transactions together with their timestamp would result in costs which are at least linear in the number of incoming transactions, instead we only store a discretized representation, which only stores the number of occurrences within m smaller time windows. Figure 2 illustrates the idea. In the first time window, the pattern occurred 5 times; in the second, it occurred 8 times, etc. As we are only interest in the frequency of patterns in the interval $(t - W_{new} - W_{old}, t]$, our representation can discard old time windows, which ensures that the size of the representation is limited.

The overall memory requirements for every entry in this data structure is bounded logarithmically in the number of incoming transactions. Overall, this approach has worst-case memory requirements bounded by $O(\log(|\mathcal{T}|) \cdot |\mathcal{P}| \cdot m)$, where $|\mathcal{T}|$ denotes the maximum number of incoming transactions within a timeframe of $(W_{old} + W_{new})$, m denotes the number of time windows used to cover the interval $(t - W_{new} - W_{old}, t]$, and $|\mathcal{P}|$ denotes the size of the candidate pattern space.

4.2 Distributed Computation

One issue with the approach to store a representation of the search space is that this can result in large memory requirements. It is obvious that the size of the space search is exponential in the number of items, and that hence its representation may exceed the main memory of a single machine. For this reason, it would be desirable to have a *distributed* algorithm that can run on a *cluster* of machines. In particular, we aim at an algorithm that can (almost) uniformly distribute the representation of the search space on any given cluster of N machines.

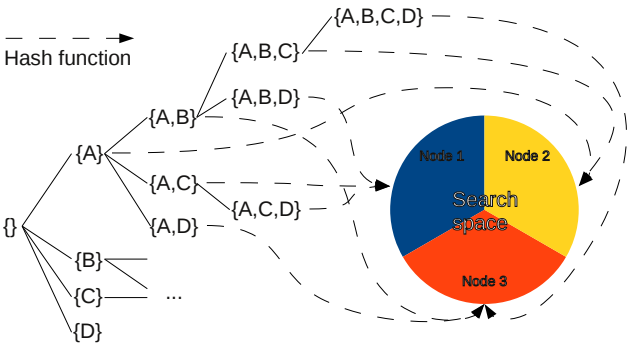


Figure 3: Distribution of the Candidate Space on a Cluster of Machines

Our solution for splitting the search space is illustrated in Figure 3 and works as follows: On the left, the figure shows the candidate pattern space, while on the right the different machines in the cluster are presented. Every pattern from the candidate pattern space is mapped to one of the nodes in the cluster by a hash function. The hash function deterministically determines the target node for each candidate pattern. In the given example the cluster consists of three separate nodes where each node is responsible for storing the statistics of a subset of all patterns. Together, the nodes represent the entire search space. Whenever a certain candidate pattern needs to be updated, the algorithm can quickly identify the responsible node which subsequently updates the pattern's statistics in the search space.

5 Implementation

We will now provide a overview of our implementation, which is based on the Akka toolkit.

5.1 Akka

The Akka toolkit (http://akka.io/) provides a high-level framework for developing distributed applications following the actor model. The user can create actors that implement different behavior. These actors can be distributed to remote nodes and communicate with each other by passing messages. With this, the user can create an topology of data-source and compute nodes, define their relations and the functions that are applied on the data.

Akka was chosen over other frameworks like Hadoop or Storm for different reasons. In contrast to Strom, Akka proved to have a more mature code base and documentation. Hadoop's focus on the other hand lies more on batch data processing than on streaming data processing.

5.2 A topology for Emerging Pattern Mining

Figure 4 describes the central topology of our system for mining Emerging Patterns. The control flow goes from left to right. The leftmost node represents a source of transactions. Regarding the Twitter example, this node would emit a flow of tweets. A transaction dispatcher node connects to the data stream and distributes the received transactions to a set of candidate pattern generator nodes by a broadcast. Every generator node receives the transaction $T$and subsequently creates a set of patterns that consists of all those patterns that are affected by the transaction and are element of its partition of the candidate space.

To control the amount of memory and computation required, we additionally restrict the cardinality of the patterns considered. This is a standard approach in supervised descriptive rule discovery (e.g. [Grosskreutz *et al.*,

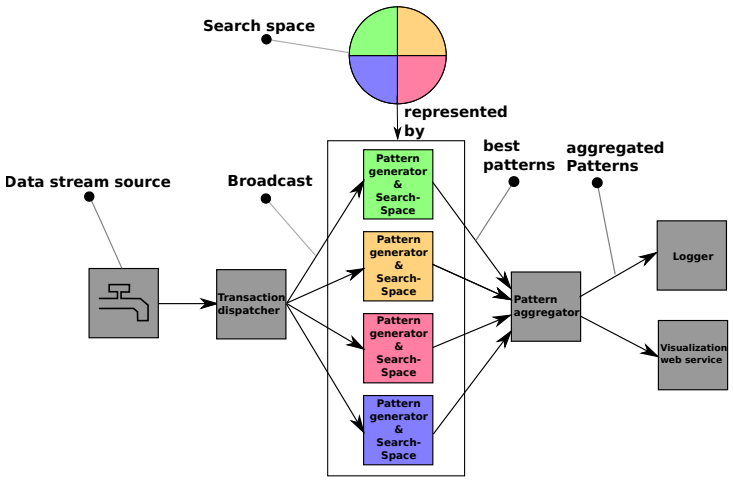


Figure 4: Topology of the system for performing Emerging Patterns Mining

2008]). Lets say we have a set of items in a transaction $T = \{i_1, \ldots, i_n\}$ and an integer value c denoting the maximal cardinality of the patterns to be considered. Then the power-set with limited cardinality c of that transaction is defined as follows:

$$P_c(T) = \{s \subseteq T : |s| < c\}$$

The generator has two central functions. It is responsible for generating patterns and for keeping track of the patterns' statistics. Following to the generation, it updates the statistics in an local data structure.

The node periodically calculates which of the patterns it takes care of have the highest quality value. It transfers this list of best patterns to a subsequent node that receives these lists from all nodes, creating an aggregated list of all global patterns. The two rightmost nodes represent example applications for the output, like logging to a file or presenting the result in a web page.

6 Discussion

In this paper, we have motivated a variation of the task of emerging pattern mining, which operates on data streams. The idea is to search for itemsets which occur more often in recent transactions than in older transactions. We have presented a new approach to this task, which allows distributing the computation and the representation of the candidate pattern space on a cluster of machines. This distributed representation is arguably the most distinguishing feature of our approach, compared to existing approaches to related tasks like frequent itemset mining on streams.

This paper presents work in progress and much remains to be done. In particular, we still lack an empirical evaluation of the approach. This will be the topic of an on-going master thesis, which will answer questions about the scalability of the approach and possible limitations.

Acknowledgments

This publication has been produced in the context of the EU Collaborative Project P-Medicine, which is funded by the European Commission under the contract ICT-2009-6-270089.

References

[Alhammady and Ramamohanarao, 2005] Hamad Alhammady and Kotagiri Ramamohanarao. Mining emerging patterns and classification in data streams. In *Proceedings of the 2005 IEEE/WIC/ACM International Conference on Web Intelligence*, WI '05, 2005.

[Atzmüller and Puppe, 2006] Martin Atzmüller and Frank Puppe. SD-map - a fast algorithm for exhaustive subgroup discovery. In *PKDD*, pages 6–17, 2006.

[Cheng *et al.*, 2008] James Cheng, Yiping Ke, and Wilfred Ng. A survey on algorithms for mining frequent itemsets over data streams. *Knowl. Inf. Syst.*, 16(1):1–27, 2008.

[Dong and Li, 1999] Guozhu Dong and Jinyan Li. Efficient mining of emerging patterns: Discovering trends and differences. pages 43–52, 1999.

[Grosskreutz *et al.*, 2008] Henrik Grosskreutz, Stefan Rüping, and Stefan Wrobel. Tight optimistic estimates for fast subgroup discovery. In *ECML/PKDD (1)*, pages 440–456, 2008.

[Han *et al.*, 2000] Jiawei Han, Jian Pei, and Yiwen Yin. Mining frequent patterns without candidate generation. In *SIGMOD Conference*, pages 1–12, 2000.

[Kifer *et al.*, 2004] Daniel Kifer, Shai Ben-David, and Johannes Gehrke. Detecting change in data streams. VLDB '04, pages 180–191. VLDB Endowment, 2004.

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

[Lavrac *et al.*, 2004] N. Lavrac, B. Kavsek, P. Flach, and L. Todorovski. Subgroup discovery with CN2-SD. *Journal of Machine Learning Research*, 5(Feb), 2004.

[Li *et al.*, 2008] Haoyuan Li, Yi Wang, Dong Zhang, Ming Zhang, and Edward Y Chang. Pfp: parallel fp-growth for query recommendation. In *ACM conference on Recommender systems*. ACM, 2008.

[Scheffer and Wrobel, 2002] Tobias Scheffer and Stefan Wrobel. Finding the most interesting patterns in a database quickly by using sequential sampling. *Journal of Machine Learning Research*, 3:833–862, 2002.

[Wang *et al.*, 2005] Peng Wang, Haixun Wang, Xiaochen Wu, Wei Wang, and Baile Shi. On reducing classifier granularity in mining concept-drifting data streams. ICDM '05. IEEE Computer Society, 2005.