Intelligent Pattern Mining via Quick Parameter Evaluation (Extended Abstract)

Mario Boley
Fraunhofer Institute IAIS, Sankt Augustin, Germany
mario.boley@iais.fraunhofer.de

Abstract

When embedded in a larger workflow, pattern mining algorithms can be required to produce huge outputs. In this situation the usual trial-and-error parameter twiddling is infeasible, because every mining run consumes a considerable amount of time. Instead an intelligent mining process is needed that is able to provide good parameter choices on its own. To realize this vision, quick parameter evaluation algorithms are needed that can give insights into the size and structure of a pattern mining's result set before the actual mining is started. The design of such procedures is a challenging algorithmic problem. As a first step towards a possible solution we empirically evaluate an exemplary implementation of a sampling approach.

1. Introduction

Data mining tasks increasingly often appear embedded in larger workflows with an automatized subsequent processing of their results. An example for this situation is feature extraction for classification tasks dealing with huge scientific datasets, but it can also occur in other computing environments. Pattern mining algorithms from plain flat table domains (e.g. Apriori [1], FP-growth [6]) up to structured ones (e.g. gSpan [13], WARMR [10]) are common for such applications. Since the subsequent processing is performed automatically, miners are often required to produce substantially more output than for instance in the explorative data analysis setting. In this situation known algorithms suffer from a lack of intelligence: On the one hand they usually require the specification of several technical parameters (frequency threshold for frequent pattern mining, minimum confidence for association rules, time window size for sequence mining, etc.). On the other hand they provide no guidance for how these parameters are to set for a given input database. Moreover, the often applied trial-and-error setting of parameters is very inefficient in the described scenario, because when huge outputs are acceptable the corresponding mining time can also be long even with an efficient miner.

We contrast this unsatisfactory situation with the vision of *intelligent pattern mining algorithms*, which are able to forecast enough of their resulting output in order to either set the technical parameters themself or to offer a *small* set of reasonable settings to the user. As a consequence, users would only have to make choices that they can be expected to have a clear idea of. We will give a detailed application scenario of this concept in Section 2 and argue that only *quick parameter evaluation algorithms* can provide a general method capable of fulfilling all of its requirements. The unifying property of those algorithms is that they can compute useful quantities of a mining algorithm's output set in a short amount of time *that does not depend on the size of that output*. We discuss theoretical limitions for the design of such algorithms and possible techniques to evade them in Section 3. Then we briefly evaluate a straightforward implementation of one of those techniques in Section 4. Our experiment reveals encouraging results that give some evidence for the general producibility of intelligent pattern mining via quick parameter evaluation.

To the best of our knowledge, so far there is no general approach dealing with the problem of intelligent and efficient parameter choosing in pattern mining. In the context of feature selection Cheng et al. [3] have recently investigated the effect of the frequency parameter on the classification quality and proposed one possible strategy for setting this threshold. In general, finding methods that can quickly evaluate the great variety of pattern mining parameters of existing algorithms will be a challenge for researcher in the coming years.

2. Practical Analysis

In order to work out the characteristic requirements of intelligent pattern mining we have to specify this vision in more detail. For that reason we now introduce an application scenario dealing with a semi-automatized classification task from computational chemistry. Here pattern mining is used to extract a feature set to be used by a Support Vector Machine (SVM). A more detailed discussion of this setting can for instance be found in [4, 7]. It is important to point out that the scenario does not rely on any specifics from chemistry. Thus it can easily be transferred to other scientific domains and in fact also to other automatized workflows that use pattern mining.

Application Scenario: Feature Extraction *We have a learning task dealing with a chemical dataset containing around two million molecules. About hundred of them are labeled according to a binary attribute, which is the target of our learning task. Moreover, we want to use an SVM to attack our problem. Since we can compute the kernel matrix efficiently even for a huge number of features if the corresponding support vectors are sparse, we would like to attain a feature set containing roughly a million elements such that no instance possesses more than a few hundred features. We regard the molecules as labeled graphs and use an intelligent constraint based closed frequent subgraph miner to extract the features.*

Entering our request for approximately a million patterns in the output set, the miner tells us that we come closest to our request with support thresholds of 98 corresponding to 800,000 closed frequent subgraphs and 97 corresponding to 1,300,000. For our second condition we ask the miner to estimate the average number of patterns contained in one molecule. It turns out that this quantity is close to 20,000 even for a support threshold of 98. We decide to add a minimum size constraint to the mining task in order to reduce this number. Now we have a second technical parameter to set: the minimum size threshold, i.e. the minimum number of edges a subgraph has to contain in order to be listed in the result set. The miner tells us that with this threshold set to twelve we reach an average number of contained patterns of 150 from a retained total of 600,000 patterns. Each of the necessary computations so far has been done in a few seconds. Now we are happy with the parameter settings and start the actual mining process, which will enumerate the feature set in approximately four hours according to the miner's prediction.

There is a number of key observations/requirements in this scenario: The pattern mining is part of a bigger workflow and its result is further processed automatically. For that reason a *huge output size* is acceptable and often even desired in order to make use of all of the available computation power. Still there is a limit to the output size that guarantees the feasibility of the surrounding workflow. Thus, we have an *output size constraint*. Furthermore, there is a *content constraint* to the result that is at least of equal importance as the output size constraint. For the time requirements we see that for a fixed time budget we want to use as much time as possible for the actual mining process and consequently minimize the time needed to determine settings. Thus, a *short setup time* relative to the mining time is desired. A final observation is that the specified constraints are rough. They define rather magnitudes than exact limits. We are dealing with *soft bounds*. A very simple yet important general remark for pattern mining algorithms is that they scale at least linear with the number of output patterns, because every result pattern has to be printed at some point. For that reason their time complexity is analyzed with respect to the size if the output it has to produce (see [9] for an introduction to output sensitive complexity analysis). Similar observations would still hold if we transform the sketched workflow to be fully automatized by removing the user feedback loop. Thus, they are characteristic for intelligent pattern mining. As a next step, we can now discuss possible solutions for these requirements.

A practice often applied in frequent pattern mining is stopping every run as soon as the computation time has exceeded a certain threshold corresponing to an output size constraint or the elements listed so far violate content constraints. But consider the supposed ideal case of an algorithm running in output linear time. Even then every rerun requires at least time proportional to the maximum number of result elements tolerable. The huge output size requirement implies that this *parameter twiddling* approach would contradict the short setup time requirement and thus is infeasible. In fact, we see that the antagonistic requirements of huge outputs and short setup time already imply the need for algorithms that run in a time *independent of the output size*. Moreover, these algorithms should give insights into the size and structure of a pattern miner's result set. While the task of estimating the result set's size is already well defined, we get varying algorithmic tasks for varying content constraints. Thus, we generally call algorithms satisfying the above specification *quick parameter evaluation algorithms*. Finally note that approximation algorithms are perfectly acceptable in this context, because we are dealing with soft bounds.

3. Theoretical Limits and Loopholes

For a theoretical analysis of the very general quick parameter evaluation task it is a good starting point to look at the special case of frequent itemset mining. This is due to two reasons. First, there is already a large body of theoretical research devoted to this topic, and second, most of the other pattern mining tasks are generalizations of frequent itemset mining. This means that they can be used to simulate frequent itemset mining, and thus negative complexity results propagate up to them.

Quickly checking whether a frequency threshold satisfies an output size constraint implies to count efficiently the resulting family of frequent itemsets. This *counting problem* was shown in [5] to be #**P**-hard (see [12] for an introduction to #**P**). Also the related problem of counting the number of maximal frequent itemsets is #**P**-hard as shown in [14]. For other quantities that could help to estimate the shape of a resulting frequent itemset family, and thus evaluate a frequency threshold with respect to content constraints, the computation was shown to be **NP**-hard: the maximum cardinality of a frequent itemset in [5] and the minimum cardinality of an infrequent itemset in [2]. However, as we observed in Section 2 approximative solutions are sufficient for our quest. A first indication that this might help is the fact that the greedy algorithm approximates a minimum cardinality infrequent itemset within a logarithmic factor [2].

In the realm of counting, several hard problems have efficient randomized approximation algorithms based on sampling techniques. In particular with the help of the Markov chain Monte Carlo method (see for instance [8] for an introduction) fully polynomial randomized approximation schemes (FPRAS) were designed. An FPRAS is an algorithm that produces values within factors $(1+\epsilon)$ and $(1-\epsilon)$ of the target value with probability $1-\delta$ in a time polynomial in the size of the input, $1/\epsilon$, and $1/\delta$. The Markov chain Monte Carlo method is based on drawing samples from large sets using a Markov chain that quickly converges to its stationary distribution. See [11] for a survey on the use and analysis of Markov chains in this context. We demonstrate this technique with an exemplary implementation for frequent itemset counting in Section 4. In general sampling is perhaps the most promising approach towards quick parameter evaluation, because for most of the pattern mining tasks it is possible to quickly draw samples from the output set without fully computing it. This might lead to algorithms with a time complexity independent of the output size.

4. Randomized Counting

This section is devoted to the empirical evaluation of a simple algorithm that uses the Markov chain Monte Carlo method for frequent itemset mining. The goal is to attain a procedure that quickly evaluates a frequency threshold with respect to the number of frequent itemsets it produces. Our parameter evaluation task can formally be described as: *Given* a set of items $I=\{a_1,\ldots,a_n\}$, a transactional dataset $\mathcal{T}$, and a frequency threshold t, *estimate* the number of frequent sets $|\mathcal{F}|$ with $\mathcal{F}=\{X\subseteq I\colon |\{T\in\mathcal{T}\colon X\subseteq T\}|\geq t\}$.

It follows a brief description of our algorithm. Denote by $\mathcal{F}_i$ the reduced family of frequent itemsets that contain only items from $\{a_1,\ldots,a_i\}$. For statistical reasons instead of directly approximating the number $|\mathcal{F}|$ we estimate the factors (resp. their reciprocals) of the product

$$|\mathcal{F}|=\frac{|\mathcal{F}_n|}{|\mathcal{F}_{n-1}|}\times\cdots\times\frac{|\mathcal{F}_{16}|}{|\mathcal{F}_{15}|}\times|\mathcal{F}_{15}|$$

by drawing a sample S from $|\mathcal{F}_i|$. The i-th reciprocal $|\mathcal{F}_{i-1}|/|\mathcal{F}_i|$ is then approximated by the fraction of elements of S that are also an element of $\mathcal{F}_{i-1}$. The factor $|\mathcal{F}_{15}|$ is counted exhaustively by DFS enumeration. The sampling from $|\mathcal{F}_i|$ is done using a simple Markov chain that starts in the empty set. Subsequently, when in the current set X, with probability $1/2$ it chooses an item $a\in\{a_1,\ldots,a_i\}$ with uniform probability, which it removes if $a\in X$ or adds if $a\notin X$ and $X\cup\{a\}\in\mathcal{F}$, and otherwise stays in X.

As a test instance we randomly generated a dataset containing 200 transactions over 25 items. Each transaction was created using the following process: With probability $9/10$ uniformly choose a new item and include it into the transaction, otherwise stop. This results in a very dense dataset with a high expected number of frequent itemsets.

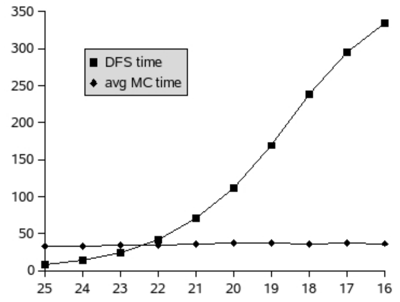

Figure 1.

For each frequency threshold $t\in\{16,\ldots,25\}$ we counted[1] the number of frequent itemsets once with an exhaustive DFS enumeration and a hundred times with the Monte Carlo (MC) algorithm. The DFS enumeration can be expected to scale similar to frequent itemset mining algorithms belonging to its family like for instance the FP-growth algorithm [6], which is basically linear to the number of frequent sets. The time of the MC algorithm does not depend on the number of frequent sets. It is essentially determined by the size of the dataset and the desired accuracy, which in turn is determined by the used sample size and the number of steps performed in each Markov chain simulation. Since we did not change any of these, the time consumed by the MC computation varied only by ±2 seconds for each frequency threshold. Consequently, the performace gap between exhaustive enumeration and randomized counting increased rapidly with the decreasing of the

[1]For our experiment we used a Python 2.4 implementation running on an Intel Pentium D 3GHz with 2GB memory under SUSE Linux 10.0.

frequency threshold. See Figue 1 for an illustration of the consumed time[2] in seconds (y-axis) for each threshold (x-axis).

threshold	# freq. sets	med. ϵ	11th worst ϵ
25	264,401	0.17	0.38
24	512,258	0.22	0.41
23	1,009,016	0.27	0.51
22	1,988,294	0.28	0.57
21	3,827,797	0.29	0.55
20	7,074,945	0.31	0.61
19	12,264,238	0.21	0.54
18	19,351,987	0.2	0.43
17	26,724,911	0.13	0.31
16	31,850,954	0.13	0.29

Table 1.

Turning to the accuracy we observed that the average relative approximation quality did not exceed 36%. In fact this quantity was smaller for most frequency thresholds. See Table 1 for a list of all frequent itemset numbers and the median of achieved approximation ratios, which are measured with their absolute difference to 1. Moreover, discarding the ten worst results for each series, we see that the worst remaining run still provides meaningful estimations (rightmost column).

Summing up we can say: For cases in which the cost of frequency counting is dominated by the number of frequent itemsets straightforward randomized counting produces reasonable approximations in a fraction of the time needed for exhaustive enumeration.

5. Conclusion

We discussed a challenge to pattern mining posed by applications that require a huge mining output. Such applications can for instance be found in classification tasks of large scientific datasets but are also likely to occur in other situations where there is a subsequent machine processing. Since in this situation the lack of intelligence of current mining algorithms cannot be compensated by trial-and-error parameter setting, quick parameter evaluation procedures are needed. These are algorithms that can compute quantities about the shape and size of a pattern mining's result set in a time independent of the size of this set. This independence of the output size points towards sampling techniques as possible solutions.

We gave an example implementation of the Markov chain Monte Carlo method for counting the number of frequent itemsets. It produces promising results in the artificial environment of datasets generated uniformly at random. However, the approach has two major flaws: The mixing time of the used Markov chain has no good bounds for the general case, and the computation requires an often repeated counting of frequencies. Both can easily lead to infeasible setup times for huge real-world datasets.

We did not yet touch the majority of other parameters that come up in the more advanced mining tasks. Finding efficient evaluation procedures for them is an interesting and wide open challenge. Also the combination of two or more parameters is likely to introduce a lot of extra difficulty to the task.

[2]For the MC algorithm the average time is listed.

References

[1] R. Agrawal, H. Mannila, R. Srikant, H. Toivonen, and A. I. Verkamo. Fast discovery of association rules. In *Advances in Knowledge Discovery and Data Mining*, pages 307–328. AAAI/MIT Press, 1996.

[2] M. Boley. On approximating minimum infrequent and maximum frequent sets. In *Proceedings of the Tenth International Conference on Discovery Science (to appear)*. Springer, 2007.

[3] H. Cheng, X. Yan, J. Han, and C.-W. Hsu. Discriminative frequent pattern analysis for effective classification. In *ICDE*, pages 716–725, 2007.

[4] M. Deshpande, M. Kuramochi, and G. Karypis. Frequent sub-structure-based approaches for classifying chemical compounds. In *ICDM*, pages 35–42, 2003.

[5] D. Gunopulos, R. Khardon, H. Mannila, S. Saluja, H. Toivonen, and R. S. Sharm. Discovering all most specific sentences. *ACM Trans. Database Syst.*, 28(2):140–174, 2003.

[6] J. Han, J. Pei, Y. Yin, and R. Mao. Mining frequent patterns without candidate generation: A frequent-pattern tree approach. *Data Min. Knowl. Discov.*, 8(1):53–87, 2004.

[7] T. Horváth, T. Gärtner, and S. Wrobel. Cyclic pattern kernels for predictive graph mining. In *Proceedings of the 10th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, page 158167. ACM Press, New York, 2004.

[8] M. Jerrum and A. Sinclair. The markov chain monte carlo method: An approach to approximate counting and integration. In D. Hochbaum, editor, *Approximation Algorithms for NP-hard Problems*, pages 482–520. PWS Publishing, 1997.

[9] D. S. Johnson, C. H. Papadimitriou, and M. Yannakakis. On generating all maximal independent sets. *Inf. Process. Lett.*, 27(3):119–123, 1988.

[10] R. D. King, A. Srinivasan, and L. Dehaspe. Warmr: a data mining tool for chemical data. *Journal of Computer-Aided Molecular Design*, 15(2):173–181, 2001.

[11] D. Randall. Rapidly mixing markov chains with applications in computer science and physics. *Computing in Science and Engineering*, 8(2):30–41, 2006.

[12] L. G. Valiant. The complexity of computing the permanent. *Theor. Comput. Sci.*, 8:189–201, 1979.

[13] X. Yan and J. Han. gspan: Graph-based substructure pattern mining. In *ICDM*, pages 721–724, 2002.

[14] G. Yang. Computational aspects of mining maximal frequent patterns. *Theor. Comput. Sci.*, 362(1-3):63–85, 2006.