Sets of Robust Rules, and How to Find Them

Jonas Fischer and Jilles Vreeken

[1] Max Planck Institute for Informatics, and Saarland University, Germany
[2] CISPA Helmholtz Center for Information Security, Germany

Abstract. Association rules are among the most important concepts in data mining. Rules of the form $X \to Y$ are simple to understand, simple to act upon, yet can model important local dependencies in data. The problem is, however, that there are so many of them. Both traditional and state-of-the-art frameworks typically yield millions of rules, rather than identifying a small set of rules that capture the most important dependencies of the data. In this paper, we define the problem of association rule mining in terms of the Minimum Description Length principle. That is, we identify the best *set of rules* as the one that most succinctly describes the data. We show that the resulting optimization problem does not lend itself for exact search, and hence propose Grab, a greedy heuristic to efficiently discover good sets of rules directly from data. Through an extensive set of experiments we show that, unlike the state-of-the-art, Grab does reliably recover the ground truth. On real world data we show it finds reasonable numbers of rules, that upon close inspection give clear insight in the local distribution of the data.

1 Introduction

Association rules are perhaps the most important primitive in data mining. Rules of the form $X \to Y$ are not only simple to understand, they are also simple to act upon, and, most importantly, can express important *local* structure in the data. The problem is, however, that there are so many of them, and that telling the interesting from the uninteresting rules has so far proven impossible. Both traditional algorithms based on support and confidence [2], as well as modern approaches based on statistical tests [8] typically discover orders of magnitude more rules than the data has rows – even when the data consists of nothing but noise. In this paper we show how to discover a small, non-redundant set of noise-resistant rules that together describe the data well.

To succinctly express subtly different structures in data, we allow *multiple* items in the consequent of a rule. To illustrate, while rule sets $R_1 = \{A \to B, A \to C\}$ and $R_3 = \{A \to BC\}$ both express that B and C appear frequently in the context of A, the former states they do so independently, while the latter expresses a dependency between B and C. We additionally allow for *patterns*, which are simply rules like $R_4 = \emptyset \to ABC$ and express unconditional dependencies. Real data is often noisy, and hence we allow rules to hold *approximately*. That is, for a transaction $t = ABC$, our models may infer that rule

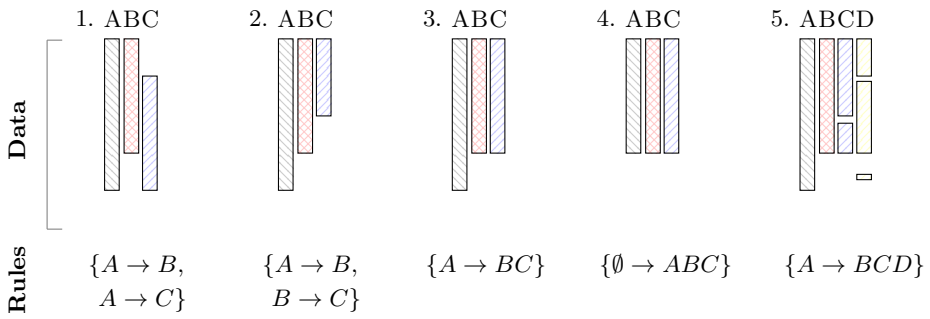


Fig. 1: *Five toy databases with corresponding rule sets* 1) B and C occur in the context of A but independently of each other, 2) C occurs in the context of B, which in turn occurs in the context of A, 3) B and C show strong joint dependence in the context of A, 4) A, B, C show strong unconditional dependence, and 5) a rule with noise, BCD occuring jointly in the context of A.

$R_5 = A \to BCD$ holds, even though item D is not present in t. To determine whether such an inference should be made, as well as to determine the quality of a rule set for given data, we rely on information theory.

In particular, we define the rule set mining problem in terms of the Minimum Description Length (MDL) principle [7]. Loosely speaking, this means we identify the best rule set as that one that compresses the data best. This set is naturally non-redundant, and neither under- nor over-fitting, as we have to pay for every additional rule we use, as well as for every error we make. We formally show that the resulting problem is neither submodular, nor monotone, and as the search space is enormous, we propose Grab, an efficient any-time algorithm to heuristically discover good rule sets directly from data. Starting from a singleton-only model, we iteratively refine our model by considering combinations of rules in the current model. Using efficiently computable tight estimates we minimize the number of candidate evaluations, and as the experiments show, Grab is both fast in practice, and yields high quality rule sets. On synthetic data, Grab recovers the ground truth, and on real-world data it recovers succinct models of meaningful rules. In comparison, the state of the art methods we compare to discover up to several millions of rules for the same data, and are hence hardly useful by itself.

The paper follows the usual structure and we postpone all proofs to the Supplementary Material.

2 Related Work

Pattern mining is arguably one of the most important and well-studied topics in data mining. We aim to give a succinct overview of the work most relevant to ours. The first, and perhaps most relevant proposal is that of association

rule mining [2], where in an unsupervised manner the goal is to find all rules of the form $X \to Y$ from the data that have high frequency and high confidence. As it turns out to be straightforward to distill the high-confidence rules from a given frequent itemset, research attention shifted to discovering frequent itemsets efficiently [15,34,9], and non-redundantly [3,4,18]. Frequency alone is a bad measure of interestingness, however, as it leads to spurious patterns [29]. To alleviate this, statistically sound measures were proposed that can mine patterns with frequencies that deviate significantly from our expectation based on margins [32,23], or richer background knowledge [10,27,5]. Perhaps because it is already difficult enough to determine the interestingness of a pattern, let alone a rule, most proposals restrict themselves to patterns. The key exception is KINGFISHER, which proposes an upper bound for Fisher's exact test that allows to efficiently mine significant dependency rules using the branch-and-bound framework [8]. Notably, however, KINGFISHER can only discover exact rules with a single item consequent. In addition, all these approaches suffer from the problems of multiple test correction, and return all patterns they deem significant, rather than a small non-redundant set.

Less directly related to our problem setting, but still relevant, are supervised methods that discover rules that explain a given target variable Y. Zimmermann and Nijssen [35] give a good general overview. However, unlike Wang et al. [31] and Papaxanthos et al. [20], we are not interested in rules that explain only Y, but rather aim for a set of rules that together explains all of the data well.

Our approach is therewith a clear example of pattern set mining [29]. That is, rather than measuring the quality of individual patterns, we measure quality over a set of patterns [30,6]. Information theoretic approaches, such as MDL and the Maximum Entropy principle, have proven particularly successful for measuring the quality of sets of patterns [30,14]. Most pattern set approaches do not account for noise in the data, with ASSO [16], HYPER+ [33], and PANDA [13] as notable exceptions. However, extending any of the above from patterns to rules turns out to be far from trivial, because rules have different semantics than patterns. PACK [28] uses MDL to mine a small decision tree per item in the data, and while not technically a rule-mining method, we can interpret the paths of these trees as rules. In our experiments we will compare to KINGFISHER as the state-of-the-art rule miner, HYPER+ as a representative of noise resilent pattern miner, and PACK as a pattern miner, which output can be translated into rules.

3 Preliminaries

In this section we discuss preliminaries and introduce notation.

3.1 Notation

We consider binary transaction data D of n-by-m, with $n = |D|$ transactions over an alphabet $\mathcal{I}$ of $m = |\mathcal{I}|$ items. In general, we denote sets of items as $X \subseteq \mathcal{I}$. A transaction t is an itemset, e.g. the products bought by a customer.

We write $\pi_X(D) := \{t \cap X \mid t \in D\}$ for the projection of D on itemset X. The transaction set, or selection, T of itemset X is the set of all transactions $t \in D$ that contain X, i.e. $T_X = \{t_j \in D \mid X \subseteq t\}$. The *support* of an itemset X is then simply the number of transactions in D that contain X, i.e. $\mathit{support}(X) = |T_X|$.

An association rule $X \to Y$ consists of two non-intersecting itemsets, the antecedent or head X, and consequent or tail Y. A rule makes a statement about the conditional occurrence of Y in the data where X holds. If $X = \emptyset$, we can interpret a rule as a pattern, as it makes an statement on where in the whole data the consequent holds. Throughout this manuscript, we will use A, B, C to refer to sets of single items and X, Y, Z for itemsets of larger cardinality.

3.2 Minimum Description Length

The Minimum Description Length (MDL) principle [25] is a computable and statistically well-founded approximation of Kolmogorov Complexity [12]. For given data D, MDL identifies the best model M^* in a given model class $\mathcal{M}$ as that model that yields the best lossless compression. In one-part, or, *refined* MDL we consider the length in bits of describing data D using the entire model class, $L(D \mid \mathcal{M})$, which gives strong optimality guarantees [7] but is only feasible for certain model classes. In practice we hence often use two-part, or, *crude* MDL, which is defined as $L(M) + L(D \mid M)$. Here $L(M)$ is the length of the description of the model, and $L(D \mid M)$ the length in bits of the description of the data using M. We will use two-part codes where we have to, and one-part codes where we can. Note that in MDL we are only concerned with code *lengths*, not materialized codes. Also, as we are interested in measuring lengths in bits, all logarithms are to base 2, and we follow the convention $0 \log 0 = 0$.

4 Theory

To use MDL in practice, we first need to define our model class $\mathcal{M}$, how to describe a model M in bits, and how to describe data D using a model M. Before we do so formally, we first give the intuitions.

4.1 The Problem, Informally

Our goal is to find a set of rules that together succinctly describe the given data. Our models M hence correspond to sets R of rules $X \to Y$. A pattern ABC is simply a rule with an empty head, i.e. $\emptyset \to ABC$. A rule *applies* to a transaction $t \in D$ if the transaction supports its head, i.e. $X \subseteq t$. For each transaction to which the rule applies, the model specifies whether the rule *holds*, i.e. whether Y is present according to the model. We can either be strict, and require that rules only hold when $Y \subseteq t$, or, be more robust to noise and allow the rule to hold even when not all items of Y are part of t, i.e. $Y \setminus t \neq \emptyset$. In this setting, the model may state that rule $A \to BCD$ holds for transaction $t = ABC$, even though $D \notin t$ (see Fig. 1.5). A model M hence needs to specify for every rule

$X \to Y \in R$ a set of transactions $T^M_{Y|X}$ where it asserts that Y holds in the context of X, and, implicitly also $T^M_{\not Y|X}$, the set of transactions where it asserts Y does not hold. Last, for both these we have to transmit which items of Y are actually in the data; the fewer errors we make here, the cheaper it will be to transmit. To ensure that we encode any data D over $\mathcal{I}$, we require that a model M contains at least singleton rules, i.e. $\emptyset \to A$ for all $A \in \mathcal{I}$. It is easy to see that a valid model M can be represented as a directed acyclic graph (DAG). The vertices of the graph correspond to the rules in R, and a vertex $r = X \to Y$ has incoming edges from all vertices $r' = X' \to Y'$ for which $X \cap Y'$ is non-empty. A cycle would prevent us from decoding the data without loss, as we need to know transactions T_X before we can determine whether rule r applies.

We explicitly allow for rules with non-singleton tails, as this allows us to succinctly describe subtly different types of structure. When B happens independently of C in T_A (Fig. 1.1), rule set $R_1 = \{A \to B, A \to C\}$ is a good description of this phenomenon. In turn, when C occurs often – but not always – in T_B, which in turn happens often in T_A (Fig. 1.2) rule set $R_2 = \{A \to B, B \to C\}$ is a good description. To succinctly describe that B and C are statistically dependent in T_A (Fig. 1.3) we need rules with multiple items in its tail, i.e. $R_3 = \{A \to BC\}$. Finally, if A, B, and C frequently occur jointly, but conditionally independent of any other variable, we need patterns to express this, which are just consequents in the context of the whole database $R_4 = \emptyset \to ABC$.

4.2 MDL for Rule Sets

Next, we formalize an MDL score for the above intuition. We start by defining the cost of the data given a model, and then define the cost of a model.

Cost of the data We start with the cost of the data described by an individual rule $X \to Y$. For now, assume we know $\pi_X(D)$ and T_X. We transmit the data over Y in the context of X, i.e. $D_{Y|X} = \pi_Y(T_X)$, in three parts. First, we transmit the transaction ids where model M specifies that both X and Y hold, $T^M_{Y|X}$, which implicitly gives $T^M_{\not Y|X} = T_X \setminus T^M_{Y|X}$. We now, in turn transmit that part of $D_{Y|X}$ corresponding to the transactions in $T^M_{Y|X}$, resp. that part corresponding to $T^M_{\not Y|X}$. We do so using optimal data-to-model codes, i.e. indices over canonically ordered enumerations,

$$L(D_{Y|X} \mid M) = \log \binom{|T_X|}{|T^M_{Y|X}|} + \log \binom{|T^M_{Y|X}| \times |Y|}{\mathbb{1}(T^M_{Y|X})} + \log \binom{|T^M_{\not Y|X}| \times |Y|}{\mathbb{1}(T^M_{\not Y|X})} ,$$

where we write $\mathbb{1}(T^M_{Y|X})$ for the number of 1s in $T^M_{Y|X}$, i.e. $\mathbb{1}(T^M_{Y|X}) = \sum_{t \in T^M_{Y|X}} |t \cap Y| \leq |T^M_{Y|X}| \times |Y|$.

It is easy to see that when the model makes exact assertions on Y holding when X is present, i.e. when $T^M_{Y|X} = T_{Y|X}$, the second term vanishes, and analogously for the third term when $T^M_{\not Y|X} = T_{\not Y|X}$. Both terms vanish simultaneously only when $D_{Y|X} \in \{\emptyset, Y\}^{|D_X|}$. This is trivially the case when Y is a singleton.

The overall cost of the data given the model simply is the sum of the data costs per rule,

$$L(D \mid M) = \sum_{X \to Y \in M} L(D_{Y|X} \mid M)$$

To decode the data, the recipient will of course need to know each rule $X \to Y$. These are part of the model cost.

Cost of the Model To encode a rule, we first encode the cardinalities of X and Y using $L_{\mathbb{N}}$, the MDL-optimal code for integers $z \geq 1$, which is defined as $L_{\mathbb{N}}(z) = \log^* z + \log c_0$, where $\log^* z = \log z + \log\log z + ...$, and c_0 is a normalization constant such that $L_{\mathbb{N}}$ satisfies the Krafft-inequality [26]. We can now encode the items of X, resp. Y, one by one using optimal prefix codes, $L(X) = -\sum_{x \in X} \log \frac{s_x}{\sum_{i \in \mathcal{I}} s_i}$. Last, but not least we have to encode its parameters, $|T^M_{Y|X}|$, $\mathbb{1}(T^M_{Y|X})$, and $\mathbb{1}(T_{Y|X})$. These we encode using a refined, mini-max optimal MDL code. In particular, we use the regret of the NML code length [11] for the class of binomials,

$$L_{pc}(n) = \log \left(\sum_{k=0}^{n} \frac{n!}{(n-k)!k!} \left(\frac{k}{n}\right)^k \left(\frac{n-k}{n}\right)^{n-k} \right) ,$$

which is also known as the parametric complexity of a model class. Kontkanen and Myllymaki [11] showed that this term can be computed in time $O(n)$ in a recursive manner. We obtain the model cost $L(X \to Y)$ for a rule $X \to Y$ by

$$\begin{aligned} L(X \to Y) = L_{\mathbb{N}}(|X|) + L(X) + L_{\mathbb{N}}(|Y|) + L(Y) + \\ L_{pc}(|T_X|) + L_{pc}(|T^M_{Y|X}| \times |Y|) + L_{pc}(|T_{Y|X}| \times |Y|) . \end{aligned}$$

From how we encode the data we can simply ignore the last two terms for rules with $|Y| = 1$. The overall cost of a model M then amounts to

$$L(M) = L_{\mathbb{N}}(|R|) + \sum_{X \to Y \in R} L(X \to Y) ,$$

where we first send the size of rule set R, and then each of the rules in order defined by the spanning tree of the dependency graph.

4.3 The Problem, formally

We can now formally define the problem in terms of MDL.

Definition 1 (Minimal Rule Set Problem). *Given data D over items $\mathcal{I}$, find that rule set R and that set of $\mathcal{T}$ of tid-lists $T^M_{Y|X}$ for all $X \to Y \in R$, such that for model $M = (R, T)$ the total description length,*

$$L(D, M) = L(M) + L(D \mid M)$$

is minimal.

Solving this problem involves enumerating all possible models $M \in \mathcal{M}$. There exist $\sum_{i=0}^{|\mathcal{I}|} \left(\binom{|\mathcal{I}|}{i} \times 2^i \right) = 3^{|\mathcal{I}|}$ possible rules – where the second term in the sum describes all possible partitions of i items into head and tail, and the equality is given by the binomial theorem. Assuming that the optimal $T^M_{Y|X}$ are given, there are generally $2^{3^{|\mathcal{I}|}}$ possible models. The search space does not exhibit any evident structure that can be leveraged to guide the search, which is captured by the following two theorems, the proofs of which are postponed to Appendix A.1.

Theorem 1 (Submodularity). *The search space of all possible sets of association rules 2^Ω, when fixing a dataset and using the description length $L(D, M)$ as set function, is not submodular. That is, there exists a data set D s.t. $\exists X \subset Y \subseteq \Omega, z \in \Omega.\ L(D, X \cup \{z\}) - L(D, X) \leq L(D, Y \cup \{z\}) - L(D, Y)$.*

Theorem 2 (Monotonicity). *The description length $L(D, M)$ on the space of all possible sets of association rules 2^Ω is not monotonously decreasing. That is, there exists a data set D s.t. $\exists X \subset Y \subseteq \Omega.\ f(X) \leq f(Y)$.*

Hence, we resort to heuristics.

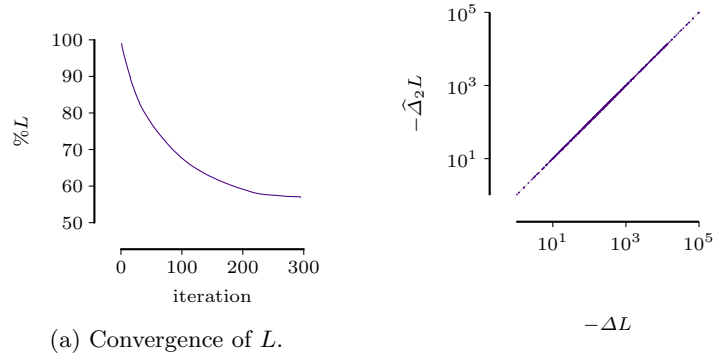
5 Algorithm

In this section we introduce Grab, an efficient heuristic for discovering good solutions to the Minimal Rule Set Problem. Grab consists of two steps, candidate generation and evaluation, that are executed iteratively until convergence of $L(D, M)$, starting with the singleton-only rule set $R_0 = \{\emptyset \to A \mid A \in \mathcal{I}\}$.

Candidate generation From the current rule set R we iteratively discover that refined rule set R' that minimizes the gain $\Delta L = L(D, M') - L(D, M)$. As refinements we consider the combination of two existing rules into a new rule.

We generate candidate refinements by considering all pairs $r_1 = X \to Y, r_2 = X \to Z \in R$, assuming w.l.o.g. $n_{XY} \geq n_{XZ}$, and merging the tails of r_1 and r_2 to obtain candidate rule $r'_1 = X \to YZ$, and merging the tail of r_1 with the head to obtain candidate rule $r'_2 = XY \to Z$. We now construct refined rule sets R'_1 and R'_2 by adding rule r'_1 resp. r'_2. To reduce redundancy, we remove r_2 from both R'_1 and R'_2, and r_1 from R'_1, taking care not to remove singleton rules. We only evaluate those refined rule sets R' for which the corresponding dependency graph is acyclic, and select the one with minimal gain $\Delta L < 0$. For completeness we give the pseudocode as Algorithm 3 in the Supplement.

Gain estimation To avoid naively evaluating the gain ΔL of every candidate, we rely on accurate gain estimatations. In particular, we consider two different estimates; the first estimate is very inexpensive to compute, but overly optimistic as it assumes a perfect overlap between the two rules. The second estimate is computationally more costly, as it requires us to compute the intersection of the tid lists of the two original rules. In practice, however, it is exact (see Fig. 2b).



(a) Convergence of L.

(b) Estimate vs. actual gain for L.

Fig. 2: GRAB *searches efficiently and estimates accurately.* For *DNA* we show (left) the convergence of the relative compression of model M at iteration i against the singleton model M_s, $\%L = \frac{L(D,M)\times 100}{L(D,M_s)}$, and (right) the correlation between estimated and actual gain of all evaluated candidates in real data.

It is easy to see that we need different estimate definitions depending on how we combine two rules r_1 and r_2. In the interest of space, we here consider one case in detail: that of combining singleton rules $r_1 = \emptyset \to A$ and $r_2 = \emptyset \to B$ into $r = A \to B$. For the remaining definitions we refer to Appendix A.5.

Following the general scheme described above, for the first estimate $\widehat{\Delta}_1$ we assume that $T_B \subseteq T_A$. With singleton tails we do not transmit any errors. Thus, we only subtract the old costs for r_2 and send where the new rule r holds, as well as the regret for the matrices,

$$\widehat{\Delta}_1(r) = -\log \binom{n}{n_B} + \log \binom{n_A}{n_B} + L_{pc}(n_A) + L_{pc}(n_B) + L_{pc}(n_A - n_B) - L_{pc}(n) \ .$$

For the tighter, second estimate $\widehat{\Delta}_2$ we instead need to retrieve the exact number of usages of the rule by intersecting the tid lists of merged rules. The change in model costs $L(M)$ by introducing r appearing in $L(M)$ is trivially computable and thus abbreviated by $\hat{L}(M)$. For formerly covered transactions that are not covered by the new rule, we need to send singleton rules with adapted costs. Additionally, we need to subtract the model costs for r_2, in case B is completely covered by r, ensured by the indicator variable I. We hence have

$$\begin{aligned}\widehat{\Delta}_2(r) = &-\log \binom{n}{n_B} + \log \binom{n_A}{|T_A \cap T_B|} + \log \binom{n}{|T_B \setminus T_A|} + \hat{L}(M) + L_{pc}(n_A) \\ &+ L_{pc}(|T_A \cap T_B|) + L_{pc}(n_A - |T_A \cap T_B|) - I(T_B \subseteq T_A) \times L_{pc}(n) \ .\end{aligned}$$

GRAB first computes the first order estimate $\widehat{\Delta}_1$ per candidate, and only if this shows potential improvement, it computes the second order estimate $\widehat{\Delta}_2$.

Out of those, it evaluates all candidates that have the potential to improve over the best refinement found so far. In the next paragraph we describe how to efficiently compute the overall score $L(D, M)$.

Efficiently computing $L(D, M)$ To get the codelength of a rule set with a new candidate, two steps are carried out, which we summarize in Alg. 1. First, the data is covered with the new rule to determine where the rule holds and what error matrices to send. Covering the data is straightforward, but to find the error matrices we have to optimize for the best point to split between additive and destructive noise. We can observe that each rule encoding is independent of every other rule (except singletons), that is, changing the error matrices for one rule does not change the matrices for any other rule as we always encode all transactions where the antecedent is fulfilled.

With this in mind, it is clear that we can optimize the split point for each rule $X \to Y$ separately. Thus, we find a partitioning of T_X into $T^M_{Y|X}$ and $T^M_{\not Y|X}$ that minimizes the contribution of this rule to the overall costs:

$$\begin{aligned}\Delta_{T_X, T^M_{Y|X}, T^M_{\not Y|X}, \mathbb{1}(T^M_{Y|X}), \mathbb{1}(T^M_{\not Y|X})} \;&=\; L_{pc}(|T_X|) \;+\; L_{pc}(|T^M_{Y|X}| \times |Y|) \\ +\; L_{pc}(|T^M_{Y|X}| \times |Y|) \;&+\; \log \binom{|T^M_{Y|X}| \times |Y|}{\mathbb{1}(T^M_{Y|X})} \;+\; \log \binom{|T^M_{\not Y|X}| \times |Y|}{\mathbb{1}(T^M_{\not Y|X})} .\end{aligned}$$

We can also view the problem from a different angle, namely, for each transaction $t \in T_X$ we count how many items of Y are present, which yields a vector of counts B, $B[i] = |\{t \in T_X \mid |t \cap Y| = i\}|$. For fixed split point k, we get the additive and destructive matrix sizes $\mathbb{1}(\cdot)^k$ and transaction set sizes $|\cdot|^k$:

$$\begin{aligned} |T^M_{Y|X}|^k &:= \sum_{i=k}^{|B|+1} B[i] & |T^M_{\not Y|X}|^k &:= \sum_{i=1}^{k-1} B[i] \\ \mathbb{1}(T^M_{Y|X})^k &:= \sum_{i=k}^{|B|} B[i] \times i & \mathbb{1}(T^M_{\not Y|X})^k &:= \sum_{i=0}^{k-1} B[i] \times i \; . \end{aligned}$$

To find the best split k^* we optimize along k using the two equation sets above, which is in time linear in the size of the consequent,

$$k^* = \operatorname*{argmin}_{k=1\ldots|B|} \left(\Delta_{T_X, T^M_{Y|X}, T^M_{\not Y|X}, \mathbb{1}(T^M_{Y|X}), \mathbb{1}(T^M_{\not Y|X})} \right) \; . \tag{1}$$

This yields the best splitpoint k^* for how many items of the consequent are required for a rule to *hold* in terms of our MDL score and thus implicitly gives the error matrices.

Putting everything together, we have GRAB, given in pseudo-code as Alg. 2.

Complexity In the worst case we generate all pairs of combinations of rules, and hence at each step GRAB evaluates a number of candidates quadratic in the size of the rule table. Each evaluation of the $O(3^{2^m})$ candidates requires a

Algorithm 1: COVER

input : Database D, model $M = (R, \mathcal{T})$, refined rule set R'
output : Model $M' = (R', \mathcal{T}')$

$T_{Y|X}^{M'} \leftarrow$ according to Equation (1) ; // Initialize where new rule holds
for $I \in \mathcal{I}$ **do** // For each singleton
  $T_I^{M'} \leftarrow T_I$; // Reset usage to baseline model
  for $\{X \to Y \in R' \mid I \in Y\}$ **do** // For each rule tail containing I
    $T_I^{M'} \leftarrow T_I^{M'} \setminus T_X$; // Remove these transactions from list
return $(R', \{T_I^{M'} \mid I \in \mathcal{I}\} \cup \{T_{U|V}^{M} \in \mathcal{T} \mid U \to V \in R \cap R'\} \cup \{T_{Y|X}^{M'}\})$;

Algorithm 2: GRAB

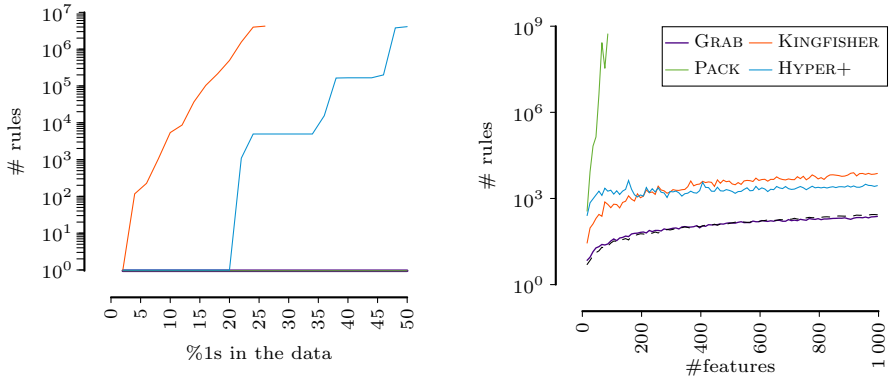
input : Dataset D
output : Heuristic approximation to M

$M \leftarrow \{\emptyset \to \{A\} \mid A \in \mathcal{I}\}$; // Initialize model with singletons
do
  $C \leftarrow$ generateCandidates(D, M);
  $M^* \leftarrow M$; $\Delta^* \leftarrow 0$;
  while C contains a refinement R with $\widehat{\Delta}_2 < \Delta^*$ **do**
    $R' \leftarrow$ refinement $R \in C$ with best $\widehat{\Delta}_2$;
    $M' \leftarrow$ cover(D, M, R') ; // Construct model M'
    $\Delta' \leftarrow L(D, M') - L(D, M)$; // Compute exact gain
    if $\Delta' < \Delta^*$ **then**
      $M^* \leftarrow M'$; $\Delta^* \leftarrow \Delta'$;
  if $M^* \neq M$ **then** // Update best model
    $M \leftarrow M^*$;
while $L(D, M) < L(D, M^*)$;
return M

database cover which costs time $O(n \times m)$, and singleton transaction set update, thus giving an overall time in $O(3^{2^m} \times m \times n)$. However, MDL ensures that the number of mined rules is small as otherwise the codelength of the model would quickly exceed the cost of the singleton model. A more useful statement about runtime is thus given in the following theorems that are based on the size of the output or in other words the number of mined rules. For the proofs, see Appendix A.2.

Theorem 3 (Grab candidate evaluations). *Given that we mine k rules for a given dataset D,* GRAB *evaluates $O((m+k)^3)$ candidates.*

This theorem gives us insight in how many times GRAB calls COVER. For the runtime analysis, we know that in each step i our rule table has size $m + i$ and GRAB has to compute the cover of the newest rule in time $O(n \times m)$ and update the singleton costs in time $O((m + i) \times m \times n)$.



(a) Rules found in pure noise data of different sparsity. GRAB and PACK do not find any rule at all, whereas KINGFISHER and HYPER+ pick up random noise.

(b) Rules found in synthetic data. GRAB closely approximates the ground truth number of planted rules (black, dashed).

Fig. 3: GRAB *recovers ground truth.* Results on random data (left) and synthetic data with planted rules for different data dimensionalities (right).

Theorem 4 (Grab runtime). *Given that we mine k rules for a given dataset D, the overall runtime of* GRAB *is* $O((m+k)^4 \times m \times n)$.

In practice, however, this runtime is never reached both due to our gain estimates and because we only allow to merge rules with the same head.

6 Experiments

In this section we empirically evaluate GRAB quantitatively and qualitatively on both synthetic and real-world data. We implemented GRAB in C++. We make all code and data available for research purposes.[3] All experiments were executed single-threaded on Intel Xeon E5-2643 v3 machines with 256 GB memory running Linux. We report the wall-clock running times.

We compare to state of the art methods for mining statistically significant patterns and association rules.[4] In particular, we compare to HYPER+ [33], which mines noise-resistant patterns, KINGFISHER [8], which is arguably the current state of the art for mining statistically significant rules under the Fisher-exact-test[5], and PACK [28], an MDL-based method that yields a binary tree per item $A \in \mathcal{I}$ of which we can interpret the paths to leafs as rules $X \to A$.

[3] `http://eda.mmci.uni-saarland.de/grab/`

[4] We leave classical support-confidence methods out for the simple fact that even for trivial data APRIORI results in millions of rules (see Appendix A.3).

[5] No relation to the first author.

Synthetic data First, we consider data with known ground truth. As a sanity check, we start our experiments on data without any structure. We draw datasets of 10000-by-100 of d% 1s, and report for each method the average results over 10 independent runs in Fig. 3a. We find that both KINGFISHER and HYPER+ quickly discover up to millions of rules. This is easily explained, as the former relies on statistical significance only, and lacks a notion of support, whereas the latter does have a notion of support, but lacks a notion of significance. PACK and GRAB, however, retrieve the ground truth in all cases.

Next, we consider synthetic data with planted rules. We generate datasets of $n = 20000$ transactions, and vary m from 10 to 1000 items. We generate rules that together cover all features. We sample the cardinality of the heads and tails from a Poisson with $\lambda = 1.5$. To avoid convoluting the ground truth via overlap, or by rules forming chains, we ensure that every item A is used in at most one rule. Per rule, we choose confidence c uniformly at random between 50 and 100%. We then randomly partition the n transactions into as many parts as we have rules, and per part, set the items of the corresponding rule head X to 1, and set Y to 1 for c% of transactions within the part. Finally, we add noise by flipping 1% of the items in the data – we use this low noise level to allow for a fair comparison to the competitors that do not explicitly model noise.

We provide the results in Fig. 3b. We observe that unlike in the previous experiment, here PACK strongly overestimates the number of rules – it runs out of memory for data of more than 92 features. KINGFISHER and HYPER+ both discover over an order of magnitude more rules than the ground truth. GRAB, on the other hand, is the only one that reliably retrieves the ground truth.

Real-World Data Second, we verify whether GRAB also yields meaningful results on real data. To this end we consider 8 data sets over a variety of domains. In particular, from the UCI repository we consider *Mushroom*, *Adult*, *Covtype*, and *Plants*. In addition we use data of Belgium traffic *Accidents*, *DNA* amplification [19], *Mammals* [17], and ICDM *Abstracts* [28]. We give basic statistics in Table 1, and provide more detailed information in Appendix A.6.

We run each of the methods on each data set, and report the number of discovered non-singleton rules for all methods and the average number of items in head and tail for GRAB in Table 1. We observe that GRAB retrieves much more succinct sets of rules than its competitors, typically in the order of tens, rather than in the order of thousands to millions. The rules that GRAB discovers are also more informative, as it is not constrained to singleton-tail rules. This is also reflected by the number of items in the consequent, where the average tail size is much larger than 1 for e.g. *Mammals* and *Plants*, where we find multiple rules with more then 10 items in the consequent.

To qualitatively evaluate the rules that GRAB discovers, we investigate the results on *Abstracts* and *Mammals* in closer detail. For *Abstracts* we find patterns such as $\emptyset \to \{naive, bayes\}$, $\emptyset \to \{nearest, neighbor\}$, $\emptyset \to \{pattern, frequency\}$, and, notably, $\emptyset \to \{association, rule\}$. Further, we find meaningful rules, including $\{high\} \to \{dimension\}$, $\{knowledge\} \to \{discovery\}$, $\{ensembl\} \to$

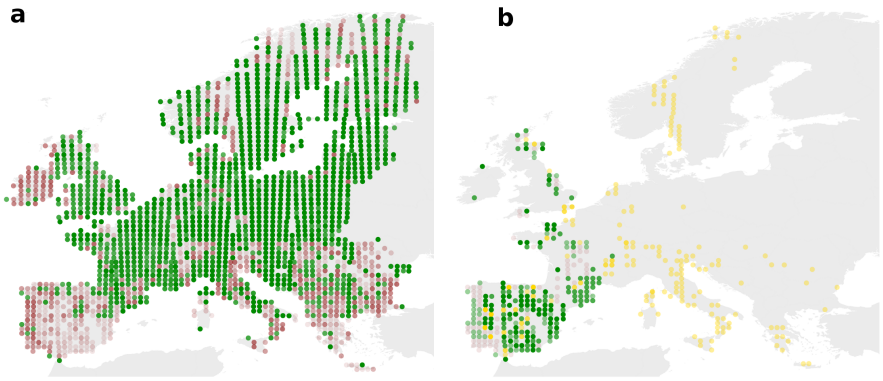


Fig. 4: *Example Rules for Mammals* Shown are the inferred presence (green) and absence (red) of **a** pattern $\emptyset \rightarrow \{$*common squirrel, deer, ermine, marten, mice**$\}$ and **b** rule $\{$*Southwest European cat*$\} \rightarrow \{$*Mediterranean mice*, Iberian rabbit* $\}$. The intensity of the colour indicates how many items of the tail hold – the ideal result is hence dark green and light red. Yellow dots indicate presence of tail items where head does not apply.

$\{$*bagging*, *boosting*$\}$, and $\{$*support*$\} \rightarrow \{$*vector*, *machin*, *SVM*$\}$. All patterns and rules correspond to well-known concepts in the data mining community.

On *Mammals*, Grab finds large patterns such as $\emptyset \rightarrow \{$*red deer, European mole, European fitch, wild boar, marten, mice**$\}$, and $\emptyset \rightarrow \{$*common squirrel, deer, ermine, marten, mice**$\}$, that correspond to animals that commonly occur across Europe, with multiple mouse species (items) indicated by mice*. In addition, it also discovers specific patterns, e.g. $\emptyset \rightarrow \{$*snow rabbit, elk, lynx, brown bear*$\}$, which are mammals that appear almost exclusively in northeastern Europe. We visualized the second rule in Figure 4a to show that the consequent should hold in most of the cases, but not necessarily need to be always present. Moreover, Grab is able to find meaningful rules in the presence of noise, e.g. $\{$*Southwest European cat*$\} \rightarrow \{$*Mediterranean mice*, Iberian rabbit*$\}$, where the rule should only hold in southwest europe. For the rule that Grab discovers this is indeed the case, although the data contains (likely spurious) sightings of Iberian rabbits or Mediterranean mice in Norway (see Fig. 4b) and some sightings of mice alone, along the Mediterranean sea.

6.1 Runtime and Scability

Last, but not least, we investigate the runtime of Grab. We first consider scalability with regard to number of features. For this, in Fig. 5a we give the runtimes for the synthetic datasets we used above. From the figure we see that while Grab is not as fast as Kingfisher and Hyper+, it scales favourably with regard to the number of features. Although it considers a much larger search space, Grab

			Grab			Hyper+	Kingfisher	Pack
Dataset	n	m	$\overline{\lvert X\rvert}$	$\overline{\lvert Y\rvert}$	$\lvert R\rvert$	Number of Discovered Rules		
Abstracts	859	3933	0.9	1.2	29	1508	$42K$	334
Accidents	339898	468	1	1.1	138	$65M$	*mem*	$69M$
Adult	10830	97	1	1.1	27	$26K$	$9K$	$68M$
Covtype	581012	105	1.3	1.1	41	$13K$	$43K$	$286M$
DNA	1316	392	1	1.7	147	49	$140K$	451
Mammals	2183	121	1.5	2	38	*mem*	$\geq 10M$	$2K$
Mushroom	8124	119	1.6	1.5	65	$13K$	$81K$	$7K$
Plants	34781	69	1.2	3.2	20	$6M$	*mem*	910

Table 1: For Grab, the size of the rule set and average size of head $\overline{\lvert X\rvert}$ and tail $\overline{\lvert Y\rvert}$ are given. For the other methods, number of found rules are given, *mem* indicates an aborted run due to memory usage > 256GB.

only needs seconds to minutes. On real data Grab is the fastest method for five of the data sets, and only requires seconds for the other datasets, whereas the other methods take up to hours for particular instances (compare Figure 5b).

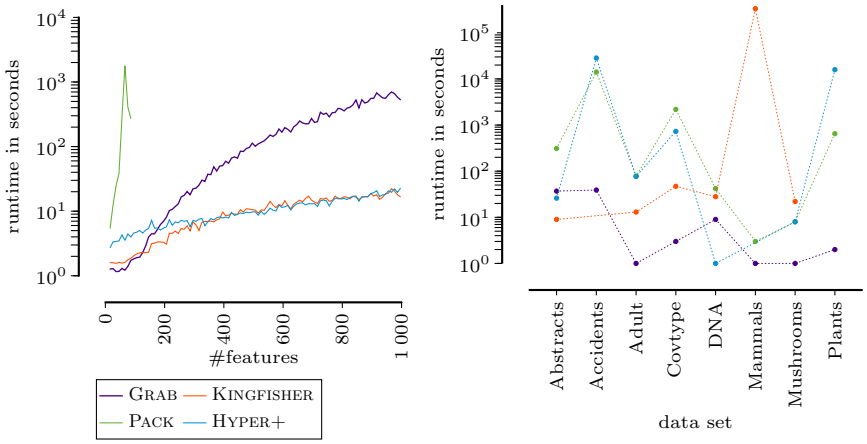
7 Discussion

The experiments show that Grab is fast and returns crisp, informative rule sets. On synthetic data it recovers the ground truth, without picking up noise. On real world data, it retrieves concise and easily interpretable rule sets, as opposed to the state of the art that discovers thousands, up to millions of rules.

The results on the *Mammals* data clearly show Grab recovers known population structures, even in the presence of noise. The results on the ICDM *Abstracts* data are equally good, with rule $\{support\} \to \{vector, machin, svm\}$ as a notable example. In contrast to machine learning, in data mining "support" is ambiguous. In the ICDM abstracts it means the support of a pattern, as well as support vector machines, and the rule expresses this. To verify this, we additionally ran Grab on abstracts from the Journal of Machine Learning Research (JMLR), where it instead recovers the pattern $\emptyset \to \{support, vector, machin, svm\}$.

Thanks to careful implementation and accurate gain estimates, Grab scales very well in the number of transactions, as well as in the number of features. In practice, Grab can consider up to several thousand features in reasonable time. Ultimately, we are interested in bioinformatics applications, and are hence interested in rule set search strategies that scale up to millions of features or more. For similar reasons we are interested in extending Grab towards continuous-valued, and mixed-type data. This we also leave for future work.

Whereas the rules Grab discovers provide useful insight, they are not necessarily actionable; that is only the case when X causes Y. Currently Grab can only reward correlation, and we are interested in extending it towards additionally identifying causal rules from observational data [22].



(a) Runtime on synthetic data.

(b) Runtime on real data.

Fig. 5: *Scalability* On the left side, runtimes are visualized on a logarithmic y-axis for synthetic data of varying number of features (x-axis). On the right, runtimes (logarithmic y-axis) are depicted for 8 real world data sets (x-axis). Kingfisher did not finish on *Accident* and *Plants*, Hyper+ did not finish on *Mammals*.

8 Conclusion

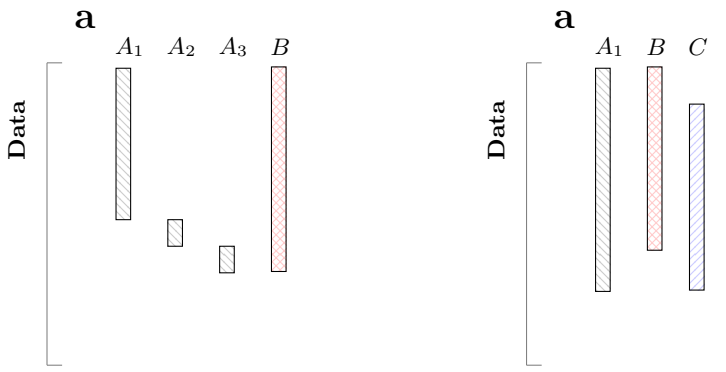
We considered the problem of non-parametrically discovering sets of association rules for a given dataset. We proposed to mine small, non-redundant sets of highly informative noise-resistant rules and patterns, that together succinctly describe the data at hand. To do so, we defined a score based on solid information theoretic grounds, showed the problem does not lend itself for efficient optimization, and proposed Grab, a highly efficient heuristic that greedily approximates the MDL optimal result. Grab is unique in that it can discover both patterns and rules, is noise-resistant and allows rules and patterns to hold approximately, and, can discover rules with non-singleton consequents. Through thorough experiments we showed that unlike the state-of-the-art, Grab is able to recover the ground truth in synthetic data, and discovers small sets of highly meaningful rules from real world data.

References

1. R. Agrawal, T. Imielinksi, and A. Swami. Mining association rules between sets of items in large databases. In *SIGMOD*, pages 207–216. ACM, 1993.
2. R. Agrawal and R. Srikant. Fast algorithms for mining association rules. In *VLDB*, pages 487–499, 1994.
3. R. Bayardo. Efficiently mining long patterns from databases. In *SIGMOD*, pages 85–93, 1998.

4. T. Calders and B. Goethals. Non-derivable itemset mining. *Data Min. Knowl. Disc.*, 14(1):171–206, 2007.
5. T. De Bie. Maximum entropy models and subjective interestingness: an application to tiles in binary databases. *Data Min. Knowl. Disc.*, 23(3):407–446, 2011.
6. J. Fowkes and C. Sutton. A subsequence interleaving model for sequential pattern mining. In *KDD*, 2016.
7. P. Grünwald. *The Minimum Description Length Principle*. MIT Press, 2007.
8. W. Hämäläinen. Kingfisher: an efficient algorithm for searching for both positive and negative dependency rules with statistical significance measures. *Knowl. Inf. Sys.*, 32(2):383–414, 2012.
9. J. Han, J. Pei, and Y. Yin. Mining frequent patterns without candidate generation. In *SIGMOD*, pages 1–12. ACM, 2000.
10. S. Jaroszewicz and D. A. Simovici. Interestingness of frequent itemsets using bayesian networks as background knowledge. In *KDD*, pages 178–186. ACM, 2004.
11. P. Kontkanen and P. Myllymäki. MDL histogram density estimation. In *AISTATS*, 2007.
12. M. Li and P. Vitányi. *An Introduction to Kolmogorov Complexity and its Applications*. Springer, 1993.
13. C. Lucchese, S. Orlando, and R. Perego. Mining top-k patterns from binary datasets in presence of noise. In *SDM*, pages 165–176, 2010.
14. M. Mampaey, J. Vreeken, and N. Tatti. Summarizing data succinctly with the most informative itemsets. *ACM TKDD*, 6:1–44, 2012.
15. H. Mannila, H. Toivonen, and A. I. Verkamo. Efficient algorithms for discovering association rules. In *KDD*, pages 181–192, 1994.
16. P. Miettinen and J. Vreeken. MDL4BMF: Minimum description length for Boolean matrix factorization. *ACM TKDD*, 8(4):A18:1–31, 2014.
17. T. Mitchell-Jones. Societas europaea mammalogica. http://www.european-mammals.org, 1999.
18. F. Moerchen, M. Thies, and A. Ultsch. Efficient mining of all margin-closed itemsets with applications in temporal knowledge discovery and classification by compression. *Knowl. Inf. Sys.*, 29(1):55–80, 2011.
19. S. Myllykangas, J. Himberg, T. Böhling, B. Nagy, J. Hollmén, and S. Knuutila. DNA copy number amplification profiling of human neoplasms. *Oncogene*, 25(55):7324–7332, 2006.
20. L. Papaxanthos, F. Llinares-López, D. A. Bodenham, and K. M. Borgwardt. Finding significant combinations of features in the presence of categorical covariates. In *NIPS*, pages 2271–2279, 2016.
21. N. Pasquier, Y. Bastide, R. Taouil, and L. Lakhal. Discovering frequent closed itemsets for association rules. In *ICDT*, pages 398–416. ACM, 1999.
22. J. Pearl. *Causality: Models, Reasoning and Inference*. Cambridge University Press, 2nd edition, 2009.
23. L. Pellegrina and F. Vandin. Efficient mining of the most significant patterns with permutation testing. In *KDD*, pages 2070–2079, 2018.
24. J. Peters, D. Janzing, and B. Schölkopf. Identifying cause and effect on discrete data using additive noise models. In *AISTATS*, pages 597–604, 2010.
25. J. Rissanen. Modeling by shortest data description. *Automatica*, 14(1):465–471, 1978.
26. J. Rissanen. A universal prior for integers and estimation by minimum description length. *Annals Stat.*, 11(2):416–431, 1983.
27. N. Tatti. Maximum entropy based significance of itemsets. *Knowl. Inf. Sys.*, 17(1):57–77, 2008.

28. N. Tatti and J. Vreeken. Finding good itemsets by packing data. In *ICDM*, pages 588–597, 2008.
29. J. Vreeken and N. Tatti. *Interesting Patterns*, pages 105–134. Springer, 2014.
30. J. Vreeken, M. van Leeuwen, and A. Siebes. KRIMP: Mining itemsets that compress. *Data Min. Knowl. Disc.*, 23(1):169–214, 2011.
31. F. Wang and C. Rudin. Falling rule lists. In *AISTATS*, 2015.
32. G. I. Webb. Discovering significant patterns. *Mach. Learn.*, 68(1):1–33, 2007.
33. Y. Xiang, R. Jin, D. Fuhry, and F. F. Dragan. Succinct summarization of transactional databases: an overlapped hyperrectangle scheme. In *KDD*, pages 758–766, 2008.
34. M. J. Zaki, S. Parthasarathy, M. Ogihara, and W. Li. New algorithms for fast discovery of association rules. In *KDD*, Aug 1997.
35. A. Zimmermann and S. Nijssen. Supervised pattern mining and applications to classification. In *Frequent Pattern Mining*, pages 425–442. Springer, 2014.



(a) Submodularity counterexample. (b) Monotonicity counterexample.

Fig. 6: Data used as counterexamples for submodularity and monotonicity of $L(D, M)$. For the submodularity counterexample (**a**), $n = 1000, n_{A_1} = 500, n_{A_2} = 100, n_{A_3} = 100, n_B = 700$. For the monotonicity counterexample (**b**), $n = 1000, n_A = 850, n_B = 700, n_C = 700$.

A Appendix

A.1 Submodularity and Monotonicity of the Search space

In this section we provide counterexamples to disprove submodularity and monotonicity of the codelength function on the search space of all rule sets.

Submodularity Let us first define submodularity:

Definition 2 (Submodularity). *Given an alphabet Ω and a set function $f : 2^{\Omega} \to \mathbb{N}$, a set function is called submodular iff $\forall X \subset Y \subseteq \Omega, z \in \Omega$. $f(X \cup \{z\}) - f(X) \geq f(Y \cup \{z\}) - f(Y)$.*

This definition of submodularity inherently shows the diminishing returns property of submodular functions, that is the larger the set grows, adding a particular element to it give less increase in the function value. Theoretically, this property can be used to guide the search, but here we show that if we define the alphabet to be all possible rules and the function to be the codelength defined before, $\Omega = \{X \to Y \mid X \subseteq \mathcal{I}, Y \subseteq \mathcal{I} \setminus X\}$ and $f(R) = -L(D, R)$, the codelength is neither submodular nor monotone w.r.t the search space of rules.

For the counterexample, visualized in Fig. 6a, let $n = 1000$ be the database size and $n_{A_1} = 500, n_{A_2} = 100, n_{A_3} = 100, n_B = 700$ the respective usages of the variables. Now let us set $X = \{A_1 \to B\}, Y = \{A_1 \to B, A_2 \to B\}$, and

$z = A_3 \to B$ and compute the codelengths:

$$
\begin{aligned}
f(X) = -\Bigg(& \log \binom{n}{n_{A_1}} + \log \binom{n}{n_{A_2}} + \log \binom{n}{n_{A_3}} + \log \binom{n}{n_B - n_{A_1}} \\
& + L_{pc}(n_{A_1}) + L_{pc}(n_{A_1} \times |B|) + 4L_{pc}(n) \Bigg), \\
f(X \cup \{z\}) = -\Bigg(& \log \binom{n}{n_{A_1}} + \log \binom{n}{n_{A_2}} + \log \binom{n}{n_{A_3}} + \log \binom{n}{n_B - n_{A_1} - n_{A_3}} \\
& + L_{pc}(n_{A_1}) + L_{pc}(n_{A_1} \times |B|) + L_{pc}(n_{A_3}) + L_{pc}(n_{A_3} \times |B|) + 4L_{pc}(n) \Bigg), \\
f(Y) = -\Bigg(& \log \binom{n}{n_{A_1}} + \log \binom{n}{n_{A_2}} + \log \binom{n}{n_{A_3}} + \log \binom{n}{n_B - n_{A_1} - n_{A_2}} \\
& + L_{pc}(n_{A_1}) + L_{pc}(n_{A_1} \times |B|) + L_{pc}(n_{A_2}) + L_{pc}(n_{A_2} \times |B|) + 4L_{pc}(n) \Bigg), \\
f(Y \cup \{z\}) = -\Bigg(& \log \binom{n}{n_{A_1}} + \log \binom{n}{n_{A_2}} + \log \binom{n}{n_{A_3}} + L_{pc}(n_{A_1}) + L_{pc}(n_{A_1} \times |B|) \\
& + L_{pc}(n_{A_2}) + L_{pc}(n_{A_2} \times |B|) + L_{pc}(n_{A_3}) + L_{pc}(n_{A_3} \times |B|) + 3L_{pc}(n) \Bigg).
\end{aligned}
$$

Note that the error matrices are basically zero cost, because in this counterexample we do not have any errors. If we use the definition of submodularity from above and plug in the numbers given above, we get:

$$
\begin{aligned}
& f(X \cup \{z\}) - f(X) \geq f(Y \cup \{z\}) - f(Y) \\
\equiv \log \binom{n}{n_B - n_{A_1}} - \log \binom{n}{n_B - n_{A_1} - n_{A_3}} & \geq \log \binom{n}{n_B - n_{A_1} - n_{A_2}} + L_{pc}(n) \\
& \equiv 716.9 - 464.4 \geq 464.4 + 5.33 \quad \lightning,
\end{aligned}
$$

which is a contradiction.

Monotonicity The next question that can be raised is if the codelength function is monotonically decreasing with respect to the search space. Let us first define Monotonicity formally.

Definition 3 (Monotonicity). *Given an alphabet Ω and a set function $f : 2^{\Omega} \to \mathbb{N}$, a set function is called monotonically decreasing iff $\forall X \subset Y \subseteq \Omega$. $f(X) \geq f(Y)$. Analogously, the function is called monotonically increasing iff $\forall X \subset Y \subseteq \Omega$. $f(X) \leq f(Y)$.*

It is easy to see that the function is not monotonically decreasing, as if we add a rule that only covers parts of the data already explained by a different rule, there are no bits saved because all rules are send independently and thus we just pay additional bits to send the part where the new rule holds. Consider

the example database given in Figure 6b. If we start with a model $X = \{A \rightarrow B, A \rightarrow C\}$, and add a rule such that $Y = X \cup \{A \rightarrow B \cup C\}$, it is easy to see that the codelength is increasing, as we redundantly encode the data B and C with the new rule. The exact calculations are left as an exercise.

A.2 Greedy algorithm complexity

In this section we provide the proofs for the runtime of GRAB in terms of the output size of the problem instance.

Theorem 5 (Grab candidate evaluations). *Given that we mine k rules for a given dataset D,* GRAB *evaluates $O((p+k)^3)$ candidates.*

Proof. In each step of the algorithm we add at most one rule to the rule table. We start initially with all p singletons in the table. At step i we thus generate $(p+i)^2$ many new candidates. If we mine k rules overall, there are $\sum_{i=0}^{k}(p+i)^2$ candidates that need to be evaluated in the worst case. We first do an index shift to obtain $\sum_{j=p}^{p+k} j^2$. We now proof a closed form solution to this sum by rearranging the sum of cubes:

$$\sum_{i=0}^{k} i^3 = \sum_{i=0}^{k} \big((i+1)^3\big) - (k+1)^3$$

Next we expand the right hand side:

$$\sum_{i=0}^{k} i^3 = \sum_{i=0}^{k} \big(i^3\big) + 3\sum_{i=0}^{k} \big(i^2\big) + 3\sum_{i=0}^{k} \big(i\big) + \sum_{i=0}^{k} \big(1\big) - (k+1)^3$$

and rearrange

$$3\sum_{i=0}^{k} i^2 = -3\sum_{i=0}^{k} \big(i\big) - \sum_{i=0}^{k} \big(1\big) + (k+1)^3.$$

Expanding the remaining cubic term and applying Gauss' formula to the one sum we get

$$3\sum_{i=0}^{k} i^2 = -3\frac{k(k+1)}{2} - (k+1) + k^3 + 3k^2 + 3k + 1.$$

Simplifying, we obtain

$$\sum_{i=0}^{k} i^2 = k^3 + \frac{3}{2}k^2 + \frac{k}{2}.$$

We thus evaluate $\sum_{j=p}^{p+k} j^2 = (p+k)^3 + \frac{3}{2}(p+k)^2 + \frac{p+k}{2} - ((p-1)^3 + \frac{3}{2}(p-1)^2 + \frac{p-1}{2}) = O((p+k)^3)$ many candidates. □

With that known, we can now proof the runtime in terms of the input, given that we know that in each step i our rule table of size $p+i$ has to compute the cover of the newest rule in time $O(n \times p)$ and update the singleton costs in time $O((p+i) \times p \times n)$.

Theorem 6. *Given that we mine k rules for a given dataset D, the overall runtime of* GRAB *is $O((p+k)^4 \times p \times n)$.*

Proof. We know that $O((p+i)\times p\times n)$ is the dominating factor in the evaluation. In each round i, we evaluate $(p+i)^2$ candidates in the worst case, for each of which we have time $O((p+i)\times p\times n)$. We thus get a runtime in $O((p+i)^3\times p\times n)$ at round i, hence, given that we mine k rules, an overall runtime of $\sum_{i=0}^{k} O((p+i)^3\times p\times n)$.

We proof now the closed form solution for the sum $\sum_{i=0}^{k} i^3 = \left(\frac{k(k+1)}{2}\right)^2$ by induction.

Base case: trivial.

Induction hypothesis: Assume closed form holds for k=j.

Induction step $j \to j+1$:

$$\begin{aligned}
\sum_{i=0}^{j+1} i^3 &= \sum_{i=0}^{j} \left(i^3\right) + (j+1)^3 \\
&= \left(\frac{j(j+1)}{2}\right)^2 + (j+1)^3 \\
&= \frac{1}{4}\left(j^2(j+1)^2 + 4(j+1)(j+1)^2\right) \\
&= \frac{1}{4}\left((j+1)^2(j^2+4j+4)\right) \\
&= \frac{1}{4}\left((j+1)^2(j+2)^2\right) \\
&= \left(\frac{(j+1)(j+2)}{2}\right)^2.
\end{aligned}$$

Plugging in this closed form into $\sum_{i=0}^{k} O((p+i)^3 \times p \times n)$ similar as before, we thus get an overall runtime of $O((p+k)^4 \times p \times n)$. □

A.3 Apriori on Mushroom

To illustrate the (well-known) impracticality of using Apriori to mine useful association rules, for different support and confidence parameters we show in Fig. 7 the number of rules Apriori discovers on the *Mushroom* data. We see that even at 90% confidence and 10% frequency, i.e. approx. 800 out of 8024 rows of data) we discover close to $10M$ rules, which hardly is a useful result on a dataset of 8 024 transactions over 124 items.

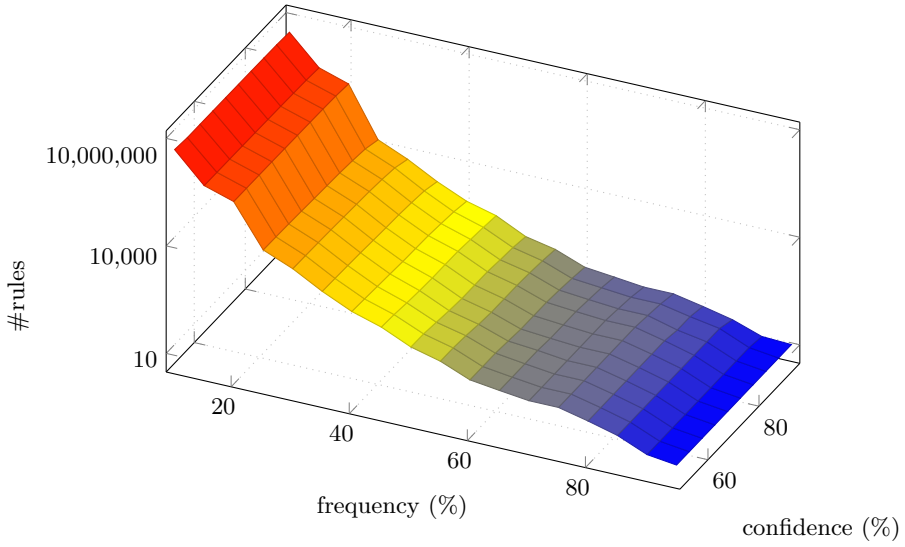


Fig. 7: *Apriori discovers many, many rules* The number of rules Apriori discovers on the *Mushroom* data for varying levels of frequency and confidence.

A.4 Algorithm

The pseudocode for the candidate generation scheme strictly follows the explanation in Sec. 5.

Algorithm 3: generateCandidates (D, M)

input : Database D, model $M = (R, \mathcal{T})$
output : Ordered candidate refinement set C

$G \leftarrow$ dependency graph of rule set R;
$C \leftarrow \emptyset$; // Candidate list
for $r_1 = X \to Y, r_2 = X \to Z \in R, n_Y \geq n_Z$ **do** // Rule pairs with same head
  $r_1' \leftarrow X \to YZ$; $R_1' \leftarrow R + r_1' - r_1 - r_2$; // Merge the tails
  if $\widehat{\Delta}_1(r_1') < 0$ **then** // Check gain estimates
    if $\widehat{\Delta}_2(r_1') < 0$ **then**
      if $G + r_1' - r_1 - r_2$ is acyclic **then**
        $C \leftarrow C \cup (\widehat{\Delta}_2(r_1'), R_1')$
  $r_2' \leftarrow XY \to Z$; $R_2' \leftarrow R + r_2' - r_2$; // Merge the head
  if $\widehat{\Delta}_1(r_2') < 0$ **then** // Check gain estimates
    if $\widehat{\Delta}_2(r_2') < 0$ **then**
      if $G + r_2' - r_2$ is acyclic **then**
        $C \leftarrow C \cup (\widehat{\Delta}_2(r_2'), R_2')$
return C

A.5 Gain Estimates

In this section we provide all formulas necessary to compute first and second tier estimates of the gain. The first and second tier estimate computations are explained throughout the next subsections.

Merging singletons As we encode singleton rules $\emptyset \to A, A \in \mathcal{I}$ and more complex rules $X \to Y,\ |X| > 0 \vee |Y| > 1$ with two different models, we need to have different gain estimates for all combinations of merging different types of rules. In this section, we provide definitions of all estimates for all combinations.

Suppose we have rules $\emptyset \to A, \emptyset \to B,\ A, B \in \mathcal{I}$, we get the gain estimate for $r = \emptyset \to AB$ by considering all possible splitpoints for the error matrices, assuming perfect overlap for the first tier estimate and partial overlap given by

n_{est} for the second tier estimate:

$$
\begin{aligned}
\widehat{\Delta}_1(r) = &- \log \binom{n}{n_{\emptyset \to A}} - \log \binom{n}{n_{\emptyset \to B}} - 2 \times L_{pc}(n) \\
&+ \min \Bigg(\log \binom{2n}{n_A + n_B} + L_{pc}(n) + L_{pc}(2n), \\
&\log \binom{n}{n_A} + \log \binom{2n_A}{n_A + n_B} + L_{pc}(n) + L_{pc}(2n_A) + L_{pc}(2(n - n_A)) \\
&\log \binom{n}{n_B} + \log \binom{2(n - n_B)}{n_A - n_B} + L_{pc}(n) + L_{pc}(2n_B) + L_{pc}(2(n - n_B)) \Bigg), \\
\widehat{\Delta}_2(r) = &- \log \binom{n}{n_{\emptyset \to A}} - \log \binom{n}{n_{\emptyset \to B}} - 2 \times L_{pc}(n) + \hat{L}(M) \\
&+ \min \Bigg(\log \binom{2n}{n_A + n_B} + L_{pc}(n) + L_{pc}(2n), \\
&\log \binom{n}{n_{est}} + \log \binom{2(n - n_{est})}{n_A + n_B - 2n_{est}} \\
&+ L_{pc}(n) + L_{pc}(2n_{est}) + L_{pc}(2(n - n_{est})), \\
&\log \binom{n}{n_A + n_B - n_{est}} + \log \binom{2(n_A + n_B - n_{est})}{n_A + n_B} \\
&+ L_{pc}(n) + L_{pc}(2(n_A + n_B - n_{est})) + L_{pc}(2(n - n_A - n_B + n_{est})) \Bigg),
\end{aligned}
$$

with $n_I = |T_I|$ the support of item I, $n_{X \to Y} = |T^M_{Y|X}|$ the current usage of the singleton rule, and $n_{est} = |T_A \cap T_B|$ the estimated support of the new rule r. We compute the change in the model costs regarding sending information about a rule, that is the terms appearing in $L(M)$, exactly and abbreviate it here with $\hat{L}(M)$.

Similarly, we get the gain estimates for $r = A \to B$:

$$
\begin{aligned}
\widehat{\Delta}_1(r) = &- \log \binom{n}{n_{\emptyset \to B}} - L_{pc}(n) \\
&+ \log \binom{n_A}{n_B} + L_{pc}(n_A) + L_{pc}(n_B) + L_{pc}(n_A - n_B), \\
\widehat{\Delta}_2(r) = &- \log \binom{n}{n_{\emptyset \to B}} - I(T^M_{B|\emptyset} \subseteq T_A) \times L_{pc}(n) + \hat{L}(M) \\
&+ \log \binom{n_A}{n_{est}} + \log \binom{n}{|T^M_{B|\emptyset} \setminus T_A|} \\
&+ L_{pc}(n_A) + L_{pc}(n_{est}) + L_{pc}(n_A - n_{est}).
\end{aligned}
$$

Note that in the first line of the second tier gain estimate, we have the indicator variable I that evaluates to 1 iff $T^M_{B|\emptyset} \subseteq T_A$. That is, we only substract the model costs for the old singleton rule $\emptyset \to B$ if it is not used anywhere else.

Merging a singleton with a pattern If we combine a rule $\emptyset \to X, |X| > 1$ with a singleton rule $\emptyset \to A$, we need to consider three different cases, each of which has a different gain estimate. First, let us consider the case of merging the tails to $r = \emptyset \to XA$.

For the first tier gain estimate we assume a perfect overlap between the dense error matrix of $\emptyset \to X$ and the transactions supporting A. In essence, we assume that A overlaps with the most dense part of $\emptyset \to X$. We can get the splitpoint (Eq. 1) by estimating the new count vector for the first tier gain estimate as follows. We generate a new count vector of size equal to the size of the new tail plus one for the cases where none of the items in the tail hold. Starting from the largest bin (i.e. all items of the tail hold), for each bin we merge all combinations of the count bins of the generating rules that would yield this number of items. E.g. for rules $r_1 = \emptyset \to AB$ and $r_2 = \emptyset \to C$ we have counts $B_1 = [2, 10, 47]$ and $B_2 = [1, 55]$, we merge these two count vectors into $B' = [1, 3, 8, 47]$. With these counts we can then estimate the new splitpoint k' and the corresponding transaction set and error matrix counts $|T^M_{XA|\emptyset}|'$, and $\mathbb{1}'(T^M_{XA|\emptyset})$, as well as $|T^M_{\cancel{XA}|\emptyset}|'$, and $\mathbb{1}'(T^M_{\cancel{XA}|\emptyset})$ for the number of rows and number of 1s in a matrix, respectively. Be aware of the prime as a notation for this approximation of the overlap. For the second tier gain estimate we compute the actual error matrix counts by intersecting the corresponding tid lists for each level according to Eq. 1.

$$
\begin{aligned}
\widehat{\Delta}_1(r) = &- L(\emptyset \to X) - \log \binom{n}{n_{\emptyset \to A}} + \log \binom{n}{|T^M_{XA|\emptyset}|'} \\
&+ \log \binom{|T^M_{XA|\emptyset}|' \times (|X|+1)}{\mathbb{1}'(T^M_{XA|\emptyset})} + \log \binom{|T^M_{\cancel{XA}|\emptyset}|' \times (|X|+1)}{\mathbb{1}'(T^M_{\cancel{XA}|\emptyset})} \\
&+ L_{pc}(n) + L_{pc}\left(|T^M_{XA|\emptyset}|' \times (|X|+1)\right) + L_{pc}\left(|T^M_{\cancel{XA}|\emptyset}|' \times (|X|+1)\right) \\
\widehat{\Delta}_2(r) = &- L(\emptyset \to X) - \log \binom{n}{n_{\emptyset \to A}} + \log \binom{n}{|T^M_{XA|\emptyset}|^{k^*}} + \hat{L}(M) \\
&+ \log \binom{|T^M_{XA|\emptyset}|^{k^*} \times (|X|+1)}{\mathbb{1}(T^M_{XA|\emptyset})^{k^*}} + \log \binom{|T^M_{\cancel{XA}|\emptyset}|^{k^*} \times (|X|+1)}{\mathbb{1}(T^M_{\cancel{XA}|\emptyset})^{k^*}} \\
&+ L_{pc}(n) + L_{pc}\left(|T^M_{XA|\emptyset}|^{k^*} \times (|X|+1)\right) + L_{pc}\left(|T^M_{\cancel{XA}|\emptyset}|^{k^*} \times (|X|+1)\right)
\end{aligned}
$$

The second case for merging $\emptyset \to X$ and $\emptyset \to A$ is that the singleton becomes the new head $r = A \to X$. Here, as before, we assume that tids with X are a

subset of tids of A. For the second tier estimate we get the overlap of X with A by intersecting the tid list of the destructive noise matrix of X and the tid list of A to find out where X holds. Furthermore, we estimate increase of the singleton costs for all singletons $I \in X$ that are not covered any longer, i.e. $T^M_{I|\emptyset} \cup (T_I \setminus T_A)$.

$$
\begin{aligned}
\widehat{\Delta}_1(r) = &+ \log \binom{n_A}{|T^M_{X|\emptyset}|} + \log \binom{|T^M_{X|\emptyset}| \times |X|}{\mathbb{1}(T^M_{X|\emptyset})} + \log \binom{(n_A - |T^M_{X|\emptyset}|) \times |X|}{\mathbb{1}(T^M_{\not X|\emptyset})} \\
&- L(\emptyset \to X) + L_{pc}(n_A) + L_{pc}(|T^M_{X|\emptyset}| \times |X|) + L_{pc}((n_A - |T^M_{X|\emptyset}|) \times |X|) \\
\widehat{\Delta}_2(r) = &- L(\emptyset \to X) + \hat{L}(M) + \sum_{I \in X} \left(- \log \binom{n}{|T^M_{I|\emptyset}|} + \log \binom{n}{|T^M_{I|\emptyset} \cup (T_I \setminus T_A)|} \right) \\
&+ \log \binom{n_A}{n_{est}} + \log \binom{n_{est} \times |X|}{\min\left(\mathbb{1}(T^M_{X|\emptyset}), n_{est} \times |X|\right)} \\
&+ \log \binom{(n_A - n_{est}) \times |X|}{\min\left(\mathbb{1}(T^M_{\not X|\emptyset}) + \max\left(0, \mathbb{1}(T^M_{X|\emptyset}) - n_{est} \times |X|\right), (n_A - n_{est}) \times |X|\right)} \\
&+ L_{pc}(n_A) + L_{pc}(n_{est} \times |X|) + L_{pc}((n_A - n_{est}) \times |X|)
\end{aligned}
$$

The third case of the mixed rule estimates is that X becomes the new head, we thus get $r = X \to A$. The first estimate is straightforward, assuming that the tids of the singleton rule is a subset of occurences of tids of X. For the second tier gain estimate we obtain the usage of the new rule by intersecting the tid lists of the two merged rules as before, we only need to take care of the case that not all singletons occur together with X. In that case we have to send an adapted code for the singleton. The regret term is taken care of by the indicator variable in the first line of the second tier estimate, the data costs are covered by the estimate $|T_A \cap T_X|$ of how much less of A we have to encode with a singleton using r,

$$
\begin{aligned}
\widehat{\Delta}_1(r) = &- \log \binom{n}{n_{\emptyset \to X}} - L_{pc}(n) + \log \binom{n_X}{n_{\emptyset \to X}} \\
&+ L_{pc}(n_X) + L_{pc}(n_{\emptyset \to X}) + L_{pc}(n_X - n_{\emptyset \to X}) \\
\widehat{\Delta}_2(r) = &- \log \binom{n}{n_{\emptyset \to X}} - I(T^M_{X|\emptyset} \subseteq T_X) \times L_{pc}(n) + \hat{L}(M) + \log \binom{n_X}{n_{est}} \\
&+ \log \binom{n}{|T^M_{X|\emptyset} \setminus T_X|} - \log \binom{n}{|T^M_{A|\emptyset}|} + \log \binom{n}{|T^M_{A|\emptyset}| - |T_A \cap T_X|} \\
&+ L_{pc}(n_X) + L_{pc}(n_{est}) + L_{pc}(n_X - n_{est})
\end{aligned}
$$

Merging general rules What remains are the estimates for merging two "proper" rules $X \to Y$ and $X \to Z$. To obtain the first tier gain estimate of merging the tails to $r = X \to YZ$, we assume that the most dense parts of the destructive noise matrices overlap. Hence we can use our approach of before and

estimate the splitpoint k' and counts B' by merging always the counts $B_1[i]$ and $B_2[j]$ with i, j as large as possible, thus getting estimates for the error matrices $|T^M_{YZ|X}|'$, $\mathbb{1}'(T^M_{YZ|X})$ and so on. For the second tier gain estimate we obtain the actual best splitpoint k^* by intersecting the corresponding tid lists for each level pairs, thus getting the actual count array B.

$$
\begin{aligned}
\widehat{\Delta}_1(r) = &- L(X \to Y) - L(X \to Z) + \log \binom{n_X}{|T^M_{YZ|X}|'} \\
&+ \log \binom{|T^M_{YZ|X}|' \times |Y \cup Z|}{\mathbb{1}'(T^M_{YZ|X})} + \log \binom{|T^M_{\cancel{YZ}|X}|' \times |Y \cup Z|}{\mathbb{1}'(T^M_{\cancel{YZ}|X})} \\
&+ L_{pc}(|T^M_{YZ|X}|' \times |Y \cup Z|) - L_{pc}(|T^M_{\cancel{YZ}|X}|' \times |Y \cup Z|) \\
\widehat{\Delta}_2(r) = &- L(X \to Y) - L(X \to Z) + \hat{L}(M) + \log \binom{n_X}{|T^M_{YZ|X}|} \\
&+ \log \binom{|T^M_{YZ|X}| \times |Y \cup Z|}{\mathbb{1}(T^M_{YZ|X})} + \log \binom{|T^M_{\cancel{YZ}|X}| \times |Y \cup Z|}{\mathbb{1}(T^M_{\cancel{YZ}|X})} \\
&+ L_{pc}(|T^M_{YZ|X}| \times |Y \cup Z|) + L_{pc}(|T^M_{\cancel{YZ}|X}| \times |Y \cup Z|) + L_{pc}(n_X)
\end{aligned}
$$

For the other case, where we merge the head to $r = XY \to Z$, we assume for $\widehat{\Delta}_1$ that the transactions where $X \to Z$ holds overlaps perfectly with the area where $X \cup Y$ holds. For the second tier gain estimate we get the exact overlap counts by intersecting the tid lists of each level. We estimate the singleton costs by adding up the potential costs of increasing the singleton counts by the number of transaction not covered anymore, i.e. estimating the new costs by $\sum_{I \in Z} \left(\log \binom{n}{|T^M_{I|\emptyset} \cup (T_I \setminus T_{XY})|} \right)$.

$$
\begin{aligned}
\widehat{\Delta}_1(r) = &- L(X \to Z) + \log \binom{n_{XY}}{\min\left(n_{XY}, |T^M_{Z|X}|\right)} + \hat{L}(M) \\
&+ \log \binom{\min\left(n_{XY}, |T^M_{Z|X}|\right) \times |Z|}{\min\left(\min\left(n_{XY}, |T^M_{Z|X}|\right) \times |Z|, \mathbb{1}(T^M_{Z|X})\right)} \\
&+ \log \binom{\max\left(0, n_{XY} - |T^M_{Z|X}|\right) \times |Z|}{\min\left(\mathbb{1}(T^M_{\not Z|X}), \max\left(0, n_{XY} - |T^M_{Z|X}|\right) \times |Z|\right)} \\
&+ L_{pc}(n_{XY}) + L_{pc}(\min\left(n_{XY}, |T^M_{Z|X}|\right) \times |Z|) \\
&+ L_{pc}(\max\left(0, n_{XY} - |T^M_{Z|X}|\right) \times |Z|) \\
\widehat{\Delta}_2(r) = &- L(X \to Z) + \hat{L}(M) + \log \binom{|T^M_{Z|XY}| \times |Z|}{\mathbb{1}(T^M_{Z|XY})} + \log \binom{|T^M_{\not Z|XY}| \times |Z|}{\mathbb{1}(T^M_{\not Z|XY})} \\
&+ \sum_{I \in Z} \left(- \log \binom{n}{|T^M_{I|\emptyset}|} + \log \binom{n}{|T^M_{I|\emptyset} \cup (T_I \setminus T_{XY})|} \right) \\
&+ L_{pc}(n_{XY}) + L_{pc}(|T^M_{Z|XY}| \times |Z|) + L_{pc}(|T^M_{\not Z|XY}| \times |Z|)
\end{aligned}
$$

A.6 Data

In this section, we briefly summarize the composition of the real datasets used to analyze GRAB.

- **Abstracts** A collection of $m = 3933$ stemmed words (features) for $n = 859$ abstracts of ICDM 2001-2007 (samples).
- **Accidents** A dataset of roughly $n = 340000$ records of traffic accidents in Belgium[6] with binarized meta information about the accident's location and circumstances summing up to $m = 468$ features.
- **Adult** A collection of Census income data with $m = 97$ binary variables providing information about $n = 10830$ persons and if their income exceeds 50K.
- **Covtype** Data about forest cover types for $n = 581012$ small areas along with meta information about the area, such as soil type and slope, building a data set with $m = 105$ binary features.
- **DNA** Microarray measurements of DNA amplification comprising data about copy number amplification for $n = 4590$ cases and $m = 391$ sites (binarized).
- **Mammals** A record of $n = 2183$ geo locations across europe along with longitude an latitude stating for $m = 121$ different mammals if it has been sighted at this location.
- **Mushrooms** A collection of hypothetical samples of $n = 8124$ Mushrooms generated from The Audubon Society Field Guide to North American Mushrooms, with $m = 119$ binary attributes such as cap shape or spore color.
- **Plants** For $m = 70$ states in the United States and Canada, for $n = 22632$ plants information has been gathered, whether the plant appears in the state or not.

[6] `http://fimi.ua.ac.be`