BMaD – A Boolean Matrix Decomposition Framework

Andrey Tyukin, Stefan Kramer, and Jörg Wicker

Johannes Gutenberg-Universität Mainz, Staudingerweg 9, D-55128 Mainz, Germany
tyukiand@students.uni-mainz.de, {kramer,wicker}@informatik.uni-mainz.de

Abstract. Boolean matrix decomposition is a method to obtain a compressed representation of a matrix with Boolean entries. We present a modular framework that unifies several Boolean matrix decomposition algorithms, and provide methods to evaluate their performance. The main advantages of the framework are its modular approach and hence the flexible combination of the steps of a Boolean matrix decomposition and the capability of handling missing values. The framework is licensed under the GPLv3 and can be downloaded freely at http://projects.informatik.uni-mainz.de/bmad.

1 Introduction

The goal of a Boolean matrix decomposition (BMD) is to represent a given Boolean matrix as a product of two or more Boolean factor matrices. It is a well-known and researched problem with a wide range of applications [2], e.g. in multi-label classification [9], clustering [7], bioinformatics [10], or pattern mining [6]. In this demo, we introduce BMaD system, that understands BMD as a three step process. This division into three steps is inspired by the work of Pauli Miettinen [3,5,4] and uses and extends this work.

The presented implementation differs from previous ones in three points. First, the BMD is implemented as a modular algorithm, where each step can be carried out using multiple algorithms. Hence, different modules from different previous publications can be freely combined to a new BMD method. Second, all implemented matrix decomposition methods support missing values in the data. Published algorithms for BMD can be easily extended to support this. In most cases, no modification of the algorithms were necessary, yet no previous implementation supported missing values in the data. This became necessary in previous work, when BMD was applied to multi-label classification, where some labels can be set to unknown [9]. Third, BMaD is implemented in Java, which makes it easy to use with WEKA [1] and run on many systems out of the box. Methods to load WEKA instances directly into BMaD are provided.

While BMaD does not introduce a completely new algorithm, it provides an easy way to combine state-of-the-art steps of BMD to use established methods for new ways of BMD. Due to its capability to handle missing values, it was already used in previous publications [9] and has the potential to be used in future research, e.g. for developing machine learning algorithms using BMD, or any applications using BMD that benefit from a Java implementation.

Previous implementations did not provide such a wide range of BMD methods in one framework. Additionally, so far there is no implementation capable of handling missing values and no implementation available in Java.

2 Boolean Matrix Decomposition

Let $\mathbb{B}$ be the two-element Boolean algebra, i.e. the set $\{0,1\}$ equipped with binary operations $\wedge$ (AND), $\vee$ (OR), and the unary operation $\neg$. First, we define the *Boolean matrix product*. Let $A \in \mathbb{B}^{h\times m}$ and $B \in \mathbb{B}^{m\times w}$ be two Boolean matrices for $h, w, m \in \mathbb{N}$. Their Boolean product $A \otimes B \in \mathbb{B}^{h\times w}$ is defined as:

$$A \otimes B := \left[\bigvee_{k=1}^{m} A_{i,k} \wedge B_{k,j} \right]_{i,j} .$$

Real-world data often contains missing values (which we denote by '?'). BMaD accepts Boolean matrices with missing values as input for the decomposition algorithms. Given a Boolean matrix $A \in (\mathbb{B} \cup \{?\})^{h\times w}$ and a parameter $d \in \{1,\ldots,h\}$, the goal is to find factor matrices $C \in \mathbb{B}^{h\times d}$ and $B \in \mathbb{B}^{d\times w}$, such that the reconstruction $C \otimes B$ is as close to the original matrix A as possible. More precisely, the *reconstruction error* E is defined as:

$$E(A,\tilde{A}) := \#\{(i,j) \in \{1,\ldots,h\} \times \{1,\ldots,w\} : A_{i,j} \neq ?\,\text{and}\, A_{i,j} \neq \tilde{A}_{i,j}\}$$

for $A \in (\mathbb{B} \cup \{?\})^{h\times w}$ and $\tilde{A} \in \mathbb{B}^{h\times w}$, that is, the number of entries in $\tilde{A}$ that differ from *known* values of the matrix A. If all entries of A are known, this error is just the L^1 norm of the real-valued difference between $\tilde{A}$ and A. Using these definitions, a more precise formulation of the BMD problem is as follows: given $A \in (\mathbb{B} \cup \{?\})^{h\times w}$, and compression dimension $d \in \{1,\ldots,h\}$, find matrices $C \in \mathbb{B}^{h\times d}$ and $B \in \mathbb{B}^{d\times w}$, such that the reconstruction error $E(A, C \otimes B)$ is minimized.

The problem is known to be NP-complete for Boolean matrices without unknown values [3,5]. By obvious reduction, this also holds for the problem presented here. Hence, we do not attempt to solve the problem exactly, but instead consider a family of heuristics suitable for finding approximate solutions.

3 Algorithms and Implementation

We provide an implementation of a family of modular algorithms, which include several algorithms previously proposed by Miettinen *et al.* [3,5,4]. As previously, let $A \in \mathbb{B}^{w\times h}$, $d \in \{1,\ldots,h\}$ and $c > 0$ be the input parameters. Algorithms representable in BMaD consist of three subalgorithms, which are more or less independent of each other.

Table 1. Example program using BMaD. First, the single modules are created, then they are used to decompose the matrix. Finally, the reconstruction error is calculated on the decomposition.

```
// initialize the modules
CandidateGenerator generator = new AssociationGenerator(0.2);
BasisSelector selector = new GreedySelector();
Combinator combinator = new DensityGreedyCombinator();
// import matrix from WEKA instances objet
BooleanMatrix original = new BooleanMatrix(instances);
// decompose matrix using variable dim as dimension
BooleanMatrixDecomposition bmd = new
    BooleanMatrixDecomposition(generator, selector, combinator);
Tuple<BooleanMatrix, BooleanMatrix> t = bmd.decompose(original,
    dim);
BooleanMatrix c = t._1, b = t._2;
// generate reconstruction by Boolean multiplication
BooleanMatrix reconstruction = c.booleanProduct(b);
// calculate the relative reconstruction error
double reconstructionError =
    original.relativeReconstructionError(reconstruction,
    onesWeight);
```

Candidate generation First, a set of *potential* basis patterns is generated from the matrix A. It consists of rows of same width as matrix A.

- ***Identity*** All rows of A are declared to be candidates [5].
- ***Association*** Candidates are generated using pairwise associations [5].
- ***Intersection*** For each pair of rows of A, the entrywise minimum is a candidate [8].

Basis selection The size of the set generated in the first step is usually much larger than the parameter d. Hence we have to sort out the less meaningful patterns and retain exactly d candidate patterns that are included into the basis matrix B (**b**asis, second factor). Most of the subalgorithms discussed here also generate a coarse approximation of the matrix C at this step. In the next step, one can obtain the final version of the matrix C by either refining the approximation, or building C from scratch.

- ***Greedy Algorithm*** The error is minimized in a greedy manner [5].
- ***Local Search (with minor variations)*** Similar to the Greedy Algorithm but it iterates over $k \in \{1, \ldots, d\}$, replacing the k-th basis row of B [5].

Boolean combination In this step, the matrix C (**c**ombination, first factor) is constructed. The goal is to represent rows of the original matrix A as *Boolean combinations* of the basis patterns from the matrix B. Clearly, each row of the matrix A can be represented independently, hence it is enough to specify how to calculate entries of one single row of C.

- ***Iter*** Iterates multiple times over entries of C, and uses the change in reconstruction error to check if flipping an entry decreases the error [5].

Cover-greedy algorithm Start with an empty matrix C and repeatedly search a basis row ρ that maximizes the change in reconstruction error.

Density-greedy algorithm Basis rows with fewer 1s are preferred to rows with more 1s.

The API of BMaD is straightforward (an example call is given in Table 1). For each module, a class exists, for each step, a class must be initialized and used. The classes provide the appropriate methods to perform the given step of the BMD. Additionally, methods to compute errors or visualize matrices and errors are implemented (examples of visualizations are shown on the web site `http://projects.informatik.uni-mainz.de/bmad` and in the demo).

4 Conclusion

This demo presents a modular framework for BMD, implementing it as a modular algorithm. Each step of the algorithm can be carried out by several modules, providing a flexible implementation of the BMD. It is implemented in Java and provides support for missing values in the data set and a WEKA interface. In the demo, we will present a step-by-step tutorial how to use BMaD, showing the possibilities of it and visualizing the results.

References

1. Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., Witten, I.H.: The WEKA data mining software: an update. ACM SIGKDD Explorations Newsletter 11(1), 10–18 (2009)
2. Lu, H.: Boolean matrix decomposition and extension with applications. PhD thesis, Rutgers University (2011)
3. Miettinen, P.: The Boolean column and column-row matrix decompositions. Data Mining and Knowledge Discovery 17(1), 39–56 (2008)
4. Miettinen, P., et al.: Matrix decomposition methods for data mining: Computational complexity and algorithms. PhD thesis, University of Helsinki (2009)
5. Miettinen, P., Mielikainen, T., Gionis, A., Das, G., Mannila, H.: The discrete basis problem. IEEE Transactions on Knowledge and Data Engineering 20(10), 1348–1362 (2008)
6. Shen, B.-H., Ji, S., Ye, J.: Mining discrete patterns via binary matrix factorization. In: Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 757–766. ACM (2009)
7. Streich, A.P., Frank, M., Basin, D., Buhmann, J.M.: Multi-assignment clustering for Boolean data. In: Proceedings of the 26th Annual International Conference on Machine Learning, pp. 969–976. ACM (2009)
8. Vaidya, J.: Boolean matrix decomposition problem: Theory, variations and applications to data engineering. In: 2012 IEEE 28th International Conference on Data Engineering (ICDE), pp. 1222–1224. IEEE (2012)
9. Wicker, J., Pfahringer, B., Kramer, S.: Multi-label classification using Boolean matrix decomposition. In: Proceedings of the 27th Annual ACM Symposium on Applied Computing, pp. 179–186. ACM (2012)
10. Zhang, Z.-Y., Li, T., Ding, C., Ren, X.-W., Zhang, X.-S.: Binary matrix factorization for analyzing gene expression data. Data Mining and Knowledge Discovery 20(1), 28–52 (2010)