NASSAU: Description Length Minimization for Boolean Matrix Factorization

Sanjar Karaev

Max-Planck Institute for Informatics
Campus E 1 4, D-66123 Saarbrücken, Germany
skaraev@mpi-inf.mpg.de

Abstract. Boolean Matrix Factorization (BMF) is an important tool in data mining that in many cases allows to increase interpretability for binary data. In BMF one decomposes a given binary matrix into a Boolean product of binary factors such that some cost function is minimized.
In this work we consider the description length of the data as the cost, which has been proven effective in uncovering true structure of the data and removing the noise. The argument is that structured data is easier to compress than the noise, and hence simpler models should be favored. We introduce a new BMF algorithm, that we call `NASSAU`, which traces back its history and correct its earlier mistakes. As turned out in our experiments, this approach performs reasonably well for both real-world and synthetic data.

Keywords: Matrix factorization, Boolean matrices, MDL principle, Random walks

1 Introduction

The amount of data that needs to be analyzed in today's world is enormous, rendering it impossible to explain it all without making simplifying assumptions. Typically, we strive to find a comparatively small number of patterns that occur in the data frequently and provide a good explanation for it.

Matrix factorization is a very common tool in data mining, allowing to extract a small number of frequent patterns. The matrix factorization problem is, given a matrix, find its decomposition into two or more factors that satisfy some constraints. Perhaps the most famous type of matrix factorization is the Singular Value Decomposition (SVD) [2]. However, despite its many useful mathematical properties, its results can sometimes be hard to interpret. For example, explaining negative values in the factors of SVD might be a problem. A very common approach to increase interpretability is to require the factors to have the same type as the input data, like for example in Nonnegative Matrix Factorization (NMF) [3], which restricts the factors to be nonnegative real matrices.

A special class of matrix decomposition is the Boolean Matrix Factorization problem (BMF) [1]. In BMF both the input matrix and the factors are binary,

and the matrix product is Boolean. The motivation for this is analogous to the nonnegativity restriction in NMF – we force the factors to be binary because so is the input, and Boolean product is natural on binary data. There are different forms of BMF depending on the cost function used.

In this work we use the Minimum Description Length principle (MDL) [4], which is very useful in tackling the problem of the trade-off between fitting the data well and having a simple model. In general, the more complexity we allow in the model, the better we can fit the data. However, having high model order comes at the cost of fitting the noise.

In this paper we present `NASSAU`, a new BMF algorithm that is designed to minimize the description length. Unlike the majority of the previously proposed BMF algorithms, it can correct its previous mistakes. `NASSAU` is quite robust to subtractive noise, which is especially beneficial for real-world data as in many domains there could be zeros simply due to the lack of observation.

2 Related Work

In BMF one decomposes a given binary matrix into the Boolean product of two binary matrices such that some cost is minimized. Perhaps the most intuitive and frequently used scoring function is the one that counts how many 0 - 1 errors were made in the reconstructed data. This objective was used in one of the first algorithms proposed for solving the BMF problem – `Asso` [1]. However, this is not the only reasonable choice for the cost function. For example, Miettinen and Vreeken [5] suggested to use the description length of the data as an alternative cost. In this case the problem becomes an application of the Minimum Description Length principle (MDL) [4]. It postulates that one should favor a model yielding the shortest description of the data. The intuition is that structured data is much more easily compressible than noise, and thus models providing shorter description of the data better capture its essence. In [5] the authors combined the MDL principle with the `Asso` algorithm [1] to tackle the model order selection problem in BMF. However, the `Asso` algorithm minimizes the number of $0-1$ errors, and MDL was only run as a postprocessing step to compare the results for different ranks and select the best one.

Lucchese, et al. [6] proposed an algorithm that they call `PANDA+`, which is capable of optimizing with respect to several different objectives, including the description length. However, they use a different encoding of the data than the one considered in this work (for details see [6] and [5]). Also the way `PANDA+` works is very different from our approach. In particular, it expects solid core patterns is the data, which it then tries to extend.

`PANDA+` is an extended version of `PANDA` algorithm, which was introduced in the authors' previous work [7].

Another area of research that has a strong link to BMF is tiling transaction databases [13]. A tile is a submatrix consisting only of ones. The objective is to cover as many ones as possible with tiles, without covering any zeros. The

problem is similar to BMF in that the original data is covered with rank-1 matrices full of ones, but unlike BMF, covering zeros with ones is not allowed.

3 Notation

Let $B \in \{0,1\}^{n \times k}$ and $C \in \{0,1\}^{k \times m}$. We denote by $B \circ C$ the Boolean product of matrices B and C. We will also require some notation for manipulating rows and columns of matrices. For any matrix A, we denote its i-th row by A_i and its j-th column by A^j. The matrix obtained by removing A^j (A_j) is denoted by A^{-j} (A_{-j}). In addition, if A is of size n-by-m and there is a column vector c of size n-by-1 and a row vector r of size 1-by-m, then we denote by $[A, c]$ and $\begin{bmatrix} A \\ r \end{bmatrix}$ matrices obtained by joining A with c and r respectively.

Let $A \in \mathbb{R}^{n \times m}$, $B \in \mathbb{R}^{n \times k}$, and $C \in \mathbb{R}^{k \times m}$ be binary matrices, and let $\langle B, C \rangle$ be a Boolean decomposition of A. Then we call B and C *factors* of this decomposition and for any $1 \leq l \leq k$, the rank-1 matrix formed by the vector pair $\langle B^l, C_l \rangle$, a *block*.

Let again $\langle B, C \rangle$ be a Boolean decomposition of a binary matrix A. We denote by $L(A, B, C)$ the description length of A with factors B and C.

4 BMF with MDL

In this work we study the Boolean Matrix Factorization problem with description length as a scoring function. We start this section by introducing the Minimum Description Length principle, which we then use to formulate the BMF problem.

4.1 Minimum Description Length (MDL) and encoding BMF

Following the MDL principle [4], we use the description length of the data as our cost function. MDL is a formalization of the Occam's razor, which states that faced with two competing models that describe the data equally well, one should choose the simpler one. In MDL the complexity of the model is expressed as the code length needed for the lossless compression of the data with this model. It is known [14] that structured data usually compresses much better than the noise, which makes MDL a usefool tool for noise removal.

The idea to use MDL as a scoring function for BMF comes from [5], where the authors introduces several possible encodings for the BMF problem. In this work we use the encoding that was deemed the best in the above paper, which the authors call *data to model encoding*. Here we will describe the main ideas behind this encoding scheme. More detailed explanation can be found in [5].

Assume that we are given a binary n-by-m matrix A and its decomposition $A \approx B \circ C$, where $B \in \{0,1\}^{k \times m}$ and $C \in \{0,1\}^{k \times m}$. The description length of this factorization can be represented as follows

$$L(A, B, C) = \alpha + L(B, C) + L(A \,|\, B, C).$$

Here α is a constant term that does not depend on a particular factorization (see [5] for details on how it is computed), $L(B, C)$ is the description length of the model (factors B and C) and $L(A \mid B, C)$ is the description length of the data (matrix A) given the model. $L(B, C)$ can be split into two parts $L(B, C) = L(B) + L(C)$, where each summand corresponds to one factor. Each column of B is independently encoded as a binary vector. Since there are $\binom{n}{|B^i|}$ binary strings that have the same length and number of ones as B^i, it can now be identified by two integers: one encoding the number of nonzero elements in B^i, (maximum n) and the other encoding the index of B^i among all binary strings having the same profile (maximum $\binom{n}{|B^i|}$). Thus, encoding B^i requires $\log n + \log \binom{n}{|B^i|}$ bits [5]. Since B has k column we have

$$L(B) = k \log(n) + \sum_{i=1}^{k} \log \binom{n}{|B^i|} . \tag{1}$$

C can be encoded analogously to the above by encoding its rows as binary strings.

It now remains to encode the description of the data given the model, or the error matrix $E = XOR(A, B \circ C)$.

It can be split it into positive and negative parts, E^+ and E^-, where $E^+_{ij} = 1$ if and only if $A_{ij} > (B \circ C)_{ij}$ and $E^-_{ij} = 1$ if and only if $A_{ij} < (B \circ C)_{ij}$ [5]. In order not to assume any structure in the error matrix, E^+ and E^- are encoded as binary strings in the same way as columns of B. This yields

$$L(E^+) = \log(mn - |B \circ C|) + \log \binom{mn - |B \circ C|}{|E^+|}$$

and

$$L(E^-) = \log(|B \circ C|) + \log \binom{|B \circ C|}{|E^-|} .$$

4.2 Boolean Matrix Factorization (BMF)

In this work we study the following problem.

Definition (Boolean matrix factorization). *Given a binary matrix $A \in \{0, 1\}^{n \times m}$, the Boolean Matrix Factorization (BMF) problem is to find binary factor matrices B and C such that the total description length of the data when represented as a Boolean product of B and C is minimized. We denote the description length of a Boolean decomposition of matrix A into factors B and C by $L(A, B, C)$.*

Note that a more traditional way of defining the BMF problem is to look for a decomposition of a given rank (see e.g. [5]). We do not require the rank as an input, but rather aim to find the best decomposition regardless of it, when finding the right rank becomes a byproduct.

5 Algorithm

In this section we present a new algorithm, which we call `NASSAU` (Algorithm 1), for solving the BMF problem. In a nutshell it works as follows: start with an empty factorization, then iteratively add blocks until there is no more gain in the description length, and then go through the earlier blocks and attempt to improve them.

The basic building block for `NASSAU` is a routine called `FindBlock` (algorithm 3) which, given the input matrix A and current factors B and C, finds (approximately) an optimal block to be added to the factors. It requires a set of candidate column vectors on the input, which it uses to start building potential blocks to be added to the current factorization. It then chooses the one that yields the best change to the description length. Subsection 5.3 describes this process in detail.

We run `FindBlock` repeatedly until the description length does not decrease anymore. Periodically we update all the blocks found up to this point to fix some of the suboptimal decisions made in the past. This is done using the routine `CyclicUpdates` (Algorithm 2), which goes though the current factors updating one block at a time with `FindBlock`. Finally, when adding a new block does not improve the cost, we already have a reasonable approximation of the optimal rank of the data. Next, we run `CyclicUpdates` yet again to make final fixes to the obtained blocks.

`NASSAU` accepts several parameters that control its execution. The first parameter t represents the initial temperature for the `CyclicUpdates` function (it is explained in Subsection 5.2). It is updated using another parameter τ. Parameter θ is used within `FindBlock` routine and is explained in Subsection 5.3. Finally, M determines how frequently we update current blocks.

5.1 Finding candidates

The candidate vectors for `FindBlock` are found using restarted random walks on the bipartite graph corresponding to the input matrix. A restarted random walk starts from a certain node in the graph and on each iteration either returns to the origin with probability ϵ or visits one of the neighbors of the current node with probability $(1 - \epsilon)/d$, where d is the number of neighbours. Candidate vectors are then obtained by thresholding the stationary solution of the corresponding Markov chain.

5.2 `CyclicUpdates`

`CyclicUpdates` is used in `NASSAU` to improve the found factors by backtracking the decision history and updating previous blocks with `FindBlock`. The motivation for this is that some of the blocks might have become redundant after we have found new ones. Note that the objective is not strictly decreasing – `FindBlock` is a heuristics and is not guaranteed to improve the current block. However, we might still want to keep the update to avoid getting stuck in a local

minimum. We apply a technique similar to simulated annealing – we always accept an update if it decreases the cost, otherwise we might still accept it with a probability proportional to current temperature. This is different from the standard simulated annealing in that we use a deterministic approach to find the next step and only use randomization to decide whether to actually apply it.

5.3 `FindBlock`

`FindBlock` finds one more block to be added to the current factorization. It aims to minimize the number of 0 - 1 errors rather than the description length. The reason why we use this objective is that direct rank-1 optimization of the description length is a complicated task due to its complex nature, and 0 - 1 error proved to be a good proxy for it in these settings.

`FindBlock` starts with a set of candidate column vectors. For each candidate it finds a corresponding row vector such that together they would form a good block in terms of 0 - 1 error (line 9). Observe that once an elements has been covered, it does not matter if we cover it again. Hence, we should disregard already covered elements when computing the score. This is done by introducing a binary weight matrix W (line 4) that has ones only at the positions corresponding to not yet covered elements of A. Note that setting element $c_l = 1$ corresponds to using b to cover l-th column of A, whereas setting $c_l = 0$ corresponds to covering it with all zeros. `FindBlock` accepts parameter θ that controls when the algorithm would cover a column of A with vector b. We set c_l to 1 if and only if using b to cover A^l would result in an error that is better than the error when using a vector of all zeros by a factor of at least θ. We then fix c and find b in the same fashion (line 10). These alternating updates are repeated until convergence conditions are satisfied. We then compare the blocks obtained starting from different candidates and choose the one yielding the best cost.

6 Experiments

We performed synthetic and real-world tests with the proposed algorithm, and compare it with two of the most successful BMF algorithms `Asso` [1] and `PANDA+` [6]. In addition we ran the same experiments with a truncated version of the `NASSAU` algorithm that does not perform updates to the found blocks (that is it stops when adding blocks does not improve the score). This version is denoted by `NASSAUnc` throughout this section.

6.1 Synthetic Data

We evaluated `NASSAU`, as well as its truncated form `NASSAUnc`, on synthetic data and also compared the results to those of `Asso` [1] and `PANDA+` [6] algorithms. `Asso` depends on a user provided parameter to threshold the association matrix. To obtain more accurate results, we ran it with the parameter ranging from 0 to 1 with step 0.05, and then chose the best solution.

Algorithm 1 NASSAU

Input: matrix $A \in \{0,1\}^{n \times m}$, $0 < t < 1$, $0 < \tau < 1$, $0 < \theta < 1$, $M > 0$ – integer
Output: Factors $B \in \{0,1\}^{n \times k}$ and $C \in \{0,1\}^{k \times m}$
function NASSAU(A, t, τ, θ, M)
    Initialize: $B \leftarrow 0^{n \times 0}$, $C \leftarrow 0^{0 \times m}$,
    $Candidates \leftarrow GetCandidates(A)$ ▷ Random walks.
    $[b, c] \leftarrow FindBlock(A, B, C, Candidates, \theta)$
    $Bnew \leftarrow [B, b]$, $Cnew \leftarrow \left[\begin{smallmatrix} C \\ c \end{smallmatrix}\right]$
    while $L(A, Bnew, Cnew) < L(A, B, C)$ **do**
        $[b, c] \leftarrow FindBlock(A, B, C, Candidates)$
        $B \leftarrow Bnew$, $C \leftarrow Cnew$
        $Bnew \leftarrow [B, b]$, $Cnew \leftarrow \left[\begin{smallmatrix} C \\ c \end{smallmatrix}\right]$
        if M rounds since last update **then**
            $[Bnew, Cnew] \leftarrow CyclicUpdates(A, Bnew, Cnew, Candidates, 0, \theta)$
    while not converged **do**
        $[B, C] \leftarrow CyclicUpdates(A, B, C, Candidates, t, \theta)$
        $t \leftarrow t * \tau$
    return B, C

Algorithm 2 CyclicUpdates

Input: matrices $A \in \{0,1\}^{n \times m}$, $B \in \{0,1\}^{n \times k}$, $C \in \{0,1\}^{k \times m}$, $Candidates \in \{0,1\}^{n \times s}$, $t > 0$, $0 < \theta < 1$
Output: Factors $Bbest \in \{0,1\}^{n \times k}$ and $Cbest \in \{0,1\}^{k \times m}$
function CYCLICUPDATES(A, B, C, $Candidates$, t, θ)
    $Bbest \leftarrow B$, $Cbest \leftarrow C$
    for $l = 1$ **to** k **do**
        $[b, c] \leftarrow FindBlock(A, B^{-l}, C_{-l}, Candidates, \theta)$
        $Bnew \leftarrow [B^{-l}, b]$, $Cnew \leftarrow \left[\begin{smallmatrix} C_{-l} \\ c \end{smallmatrix}\right]$ ▷ Replace current block.
        $d = L(A, Bnew, Cnew)$
        if $d < L(A, B, C)$ **then**
            $B \leftarrow Bnew$, $C \leftarrow Cnew$
        else
            $B \leftarrow Bnew$, $C \leftarrow Cnew$ with probability t
        if $d < L(A, Bbest, Cbest)$ **then**
            $Bbest \leftarrow B$, $Cbest \leftarrow C$
    return $Bbest$, $Cbest$

Algorithm 3 FindBlock

Input: matrices $A \in \{0,1\}^{n \times m}$, $B \in \{0,1\}^{n \times k}$, $C \in \{0,1\}^{k \times m}$, $Candidates \in \{0,1\}^{n \times s}$, $0 < \theta < 1$
Output: block $\langle bbest, cbest \rangle$ with $bbest \in \{0,1\}^{n \times 1}$ and $cbest \in \{0,1\}^{1 \times m}$
function FindBlock($A, B, C, Candidates, \theta$)
    $W \leftarrow \mathbf{1} - B \circ C$ ▷ Weight matrix.
    $bbest \leftarrow 0^{n \times 1}$, $cbest \leftarrow 0^{1 \times m}$
    for $i = 1$ **to** s **do**
        $b \leftarrow Candidates^i$ ▷ Initialize b with i-th candidate column.
        repeat
            $c \leftarrow$ row vector with $c_l = 1$ iff $\sum_{W_{jl}=1} |A_{jl} - b_j)| < \theta \sum_{W_{jl}=1} A_{jl}$
            $b \leftarrow$ column vector with $b_l = 1$ iff $\sum_{W_{lj}=1} |A_{lj} - c^j| < \theta \sum_{W_{lj}=1} A_{lj}$
        until stopping criteria are satisfied
        if $L(A, [B, b], \left[\begin{smallmatrix} C \\ c \end{smallmatrix}\right]) < L(A, [B, bbest], \left[\begin{smallmatrix} C \\ cbest \end{smallmatrix}\right])$ **then**
            $bbest \leftarrow b$, $cbest \leftarrow c$
    return $bbest$, $cbest$

The experimental setup was as follows. We first generated random binary factors, then multiplied them using the Boolean matrix product to obtain matrices of size 600 by 400 with inner dimension of 15 and density 0.08. Then after adding some noise, we ran the algorithms on the noisy matrices and measured the obtained description length.

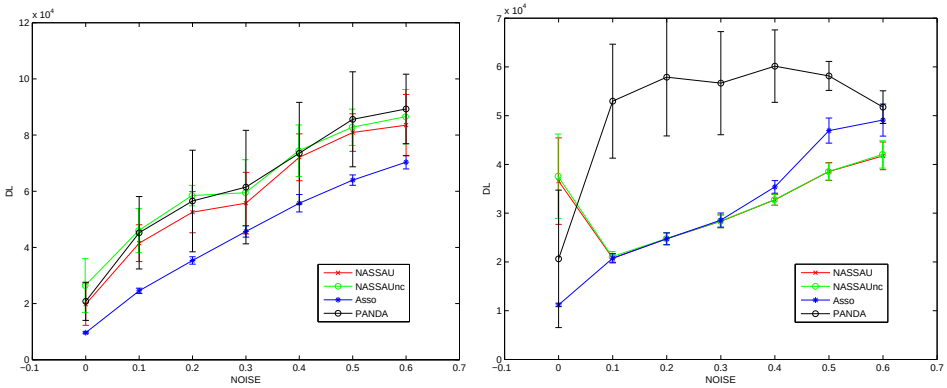
Additive noise. The purpose of this test is to find how robust the algorithms are to additive noise, that is when 0s are turned to 1s. We used various noise levels, ranging from 0 to 60% with respect to the number of ones in the input matrix. We also added a very low level (3%) of destructive noise, which was kept constant throughout the test. The results are presented in Figure 1a.

Destructive noise. The varying destructive noise test has the identical setup to the above , except that we now turn 1s to 0s. The level of noise is again measured relative to the number of ones in the input data. Analogously to the previous test, we added 3% of additive noise. The results are shown in Figure 1b.

From the plots it is obvious that `Asso` is more robust to additive noise than other algorithms. On the other hand, `NASSAU` outperforms all other methods for high levels of destructive noise. `PANDA+` performs slightly worse than `NASSAU` for the additive noise test. However, high levels of subtractive noise deteriorates its results relatively quickly compared to other methods. Comparing performances of `NASSAU` and its `NASSAUnc`, we see that while updates were clearly useful for additive noise, in case of destructive noise the improvement was very small.

6.2 Real-World Data

We performed experiments on real-world datasets and compared the results of `NASSAU`, `NASSAUnc`, `Asso`, and `PANDA+` algorithms. We tested all the algorithms



(a) Varying additive noise. (b) Varying destructive noise.

Fig. 1. Varying noise test. The vertical axis represents the obtained description length and the horizontal axis stands for the level of noise. Markers represent the means over 10 instances and the error bars have width of twice the standard deviation.

on the following datasets: Paleo – fossil records[1], Dialects – presence data of dialects across Finnish municipalities [9], [10], Newsgroups – an excerpt from the 20Newsgroups dataset[2] containing news posts in a bag-of-words representation, and Mammals – a presence data of different mammals within geographical areas of 50×50 kilometers in Europe [8]. For `Asso`, same as in the synthetic experiments, we chose the best scoring thresholding parameter. The obtained results are collected in Table 1. It can be seen that `NASSAU` obtains lower costs than the other two methods for all the datasets tested. A possible explanation for that is that it is less vulnerable to destructive noise, which is very likely to occur in many of the real-world datasets. For example in the Mammals dataset many of 0s could actually represent the fact that some species have not been observed in some area due to, for instance low population, rather than the absence of this species. In other words, in this case the data contains many false negatives, and `NASSAU` is quite robust against them. Also for two out of four datasets updates run by `NASSAU` gave it a substantial edge over its truncated form.

7 Conclusions

In this work we introduced a new algorithm for the BMF problem that directly optimizes the description length of the data. The algorithm we propose is nonhierarchical and is capable of fixing errors it has previously made. Based on experiments with both real-world and synthetically generated data, we can

[1] NOW public release 030717, available at http://www.helsinki.fi/science/now/ [Fortelius et al. 2003].
[2] http://people.csail.mit.edu/jrennie/20Newsgroups/

Table 1. Comparison between the description length obtained for various real-world datasets by `Asso`, `PANDA+`, `NASSAU`, and `NASSAUnc` algorithms.

Algorihtm	Paleo	Dialects	Newsgroups	Mammals
Asso	18556	209713	65955	215209
PANDA+	19728	292120	67120	234710
NASSAU	17831	176017	65680	179939
NASSAUnc	17931	230120	66198	196970

conclude that is is competitive with the best existing BMF methods. In particular, it is very robust to high levels of destructive noise (more so than all other methods that we tested). Moreover, it obtained better results on all real-world experiments that we conducted.

References

1. Miettinen, P.: Matrix decomposition methods for data mining: Computational complexity and algorithms. Ph.D. thesis, University of Helsinki (2009)
2. Golub, G. H. and Van Loan, C. F.: Matrix computations. JHU Press. (1996)
3. Lee D. D. and Seung, H. S.: Learning the parts of objects by non-negative matrix factorization. Nature, 401:788-791. (1999)
4. Rissanen, J.: Modeling by shortest data description. Automatica 14, 1, 465-471. (1978)
5. Miettinen, P. and Vreeken, J.: Model order selection for Boolean matrix factorization. Proceedings of the 17th ACM SIGKDD international conference on Knowledge Discovery and Data Mining. (2011)
6. Lucchese, C., Orlando, S., and Perego R.: A unifying framework for mining approximate top-k binary patterns. IEEE Transactions on Knowledge and Data Engineering. (2013), to appear
7. Lucchese, C., Orlando, S., and Perego, R.,: Mining top-k patterns from binary datasets in presence of noise. SDM, SIAM, pp. 165-176. (2010)
8. Mitchell-Jones, A., Amori, G., Bogdanowicz, W., Krystufek, B., Reijnders, P. H., Spitzenberger, F., Stubbe, M., Thissen, J., Vohralik, V., and Zima, J.: The atlas of european mammals. Academic Press. (1999)
9. Embleton, S. M. and Wheeler, E. S.: Finnish dialect atlas for quantitative studies. Journal of Quantitative Linguistics 4, 1-3, 99-102. (1997)
10. Embleton, S. M. and Wheeler, E. S.: Computerized dialect atlas of finnish: dealing with ambiguity. Journal of Quantitative Linguistics 7, 3, 227-231. (2000)
11. Cover, T. M. and Thomas, J. A.: Elements of information theory. Wiley-Interscience, New York. (2006)
12. Vereshchagin, N. and Vitanyi, P.: Kolmogorov's structure functions and model selection. IEEE Transactions on Information Technology 50, 12, 3265-3290. (2004)
13. Geerts, F., Goethals, B., and Mielikinen, T.: Tiling databases. Discovery Science. Springer Berlin Heidelberg. (2004)
14. Grünwald, P.: A tutorial introduction to the minimum description length principle. MIT Press. (2005)