Department of Computer Science
Series of Publications A
Report A-2009-0

Computing the Stochastic Complexity of Simple Probabilistic Graphical Models

Tommi Mononen

To be presented in ... text of a long permission notice. Text of a long permission notice. Text of a long permission notice. Text of a long permission notice. Text of a long permission notice. Text of a long permission notice.

University of Helsinki
Finland

Contact information

Postal address:
Department of Computer Science
P.O. Box 68 (Gustaf Hällströmin katu 2b)
FI-00014 University of Helsinki
Finland

Email address: postmaster@cs.Helsinki.FI (Internet)

URL: http://www.cs.Helsinki.FI/

Telephone: +358 9 1911

Telefax: +358 9 191 51120

ISSN 1238-8645
ISBN 000-00-0000-0 (paperback)
ISBN 000-00-0000-0 (PDF)
Computing Reviews (1998) Classification: G.1.0, G.2.1, G.3, H.1.1
Helsinki 2009
Example Printing House

Computing the Stochastic Complexity of Simple Probabilistic Graphical Models

Tommi Mononen

Department of Computer Science
P.O. Box 68, FI-00014 University of Helsinki, Finland
tommi.mononen@cs.helsinki.fi

PhD Thesis, Series of Publications A, Report A-2009-0
Helsinki, ?? month ?? 2009, 57 + 54 pages
ISSN 1238-8645
ISBN 000-00-0000-0 (paperback)
ISBN 000-00-0000-0 (PDF)

Abstract

Minimum Description Length (MDL) is an information-theoretic principle that can be used for model selection and other statistical inference tasks. There are various ways to use the principle in practice. One theoretically valid way is to use the normalized maximum likelihood (NML) criterion. Due to computational problems, this approach has not been used very often. This Thesis presents efficient floating-point algorithms that make it possible to compute the NML for Multinomial, Naive Bayes and Bayesian tree models. None of the presented algorithms rely on asymptotic analysis and with the first two model classes we also discuss how to compute exact rational number solutions.

Computing Reviews (1998) Categories and Subject Descriptors:
G.1.0 General : numerical algorithms
G.2.1 Combinatorics
G.3 Probability and Statistics: statistical computing
H.1.1 Systems and Information Theory

General Terms:
statistical modelling, machine learning, data analysis

Additional Key Words and Phrases:
information theory, Bayesian networks, minimum description length,

generating function, algorithm

Acknowledgements

Does not exist yet!

Contents

Original Publications

This Thesis is based on the following publications, which are referred to in the text as Papers 1-5.

1. Tommi Mononen and Petri Myllymäki. On the Multinomial Stochastic Complexity and its Connection to the Birthday Problem. In *Proceedings of the International Conference on Information Theory and Statistical Learning*, ITSL'08 (Las Vegas, Nevada, USA), pages 17-22, CSREA Press, 2008.

2. Tommi Mononen and Petri Myllymäki. Computing the Multinomial Stochastic Complexity in Sub-Linear Time. In *Proceedings of the European Workshop on Probabilistic Graphical Models*, PGM'08 (Hirtshals, Denmark), pages 209-216, 2008.

3. Tommi Mononen and Petri Myllymäki. On Recurrence Formulas for Computing the Stochastic Complexity. In *Proceedings of the International Symposium on Information Theory and its Applications*, ISITA'08 (Auckland, New Zealand), pages 281-286, IEEE, 2008.

4. Tommi Mononen and Petri Myllymäki. Fast NML Computation for Naive Bayes Models. In *Proceedings of the 10th International Conference on Discovery Science*, DS'07 (Sendai, Japan), pages 151-160, Springer, 2007.

5. Tommi Mononen and Petri Myllymäki. Computing the NML for Bayesian Forests via Matrices and Generating Polynomials. In *Proceedings of the 2008 IEEE Information Theory Workshop*, ITW'08 (Porto, Portugal), pages 276-280, IEEE, 2008.

Chapter 1

Introduction

In this chapter, first we give an informal introduction to the topic area of this Thesis. After that we summarize the main contributions of the author.

1.1 Motivation

Bayesian Networks are versatile probability models that can be used e.g. in prediction and modelling tasks [14, 31]. Models can be constructed either by hand using only prior knowledge or the best model can be found using an algorithm working on given observed data. In the latter case we are talking about machine learning — thus a computer is automatically searching in a set of models that describes the data best. However, in the prediction case we want our model to also predict properties of unseen future data correctly. Thus machine learning algorithms need a scoring (model selection) criterion that correctly evaluates the true goodness of a model.

For Bayesian networks there exists a Bayesian scoring criterion called BDeu [5], which is often used for selecting the best Bayesian network for the given data. The score can be considered to be a state-of-the-art score for the purpose. However, it is not purely objective, as there is a free hyperparameter called equivalent sample size (ESS). Traditionally this parameter is often set to 1. Due to recent development of exhaustive search algorithms, it has been empirically observed that the value of this parameter affects the criterion heavily [42]. One possible solution is of course to try to prove some theoretically valid method for determining the value of this parameter [44]. However, in the following we instead take a different path and consider an entirely different criterion that has its roots in information theory. This information-theoretic criterion can be considered more objective as it has no free parameters that need to be tuned.

The minimum description length (MDL) principle originates from the ideas of *Kolmogorov complexity*. Kolmogorov complexity measures the complexity of strings [27]. The complexity of a string is the length of the shortest description (program) that produces the string and stops after that. In real life, the Kolmogorov complexity cannot be computed for an arbitrary string, because the found description cannot be proved to be the optimal one: as the description set consists of all the possible programs in the world, there may always be a program that gives even a shorter description than the found one. The MDL principle, on the other hand, says that you are allowed to constrain the set of possible probability models (programs) [36, 13]. This constrained set can be for example a Bayesian network structure with free parameters. Now we can take a structure and our observed data compressed (described) with the structure. Then in this fixed set we are able to find the parameter setting that minimizes (in a certain sense to be explained later) the length of the description. This minimum length is called the *stochastic complexity*.

The intuition behind this complexity measure is quite straight forward. If we have a very complex model, it can give a short description for the data, but the description of the model itself is complicated (long). On the other hand, if we have a simple model, it gives a long description for the data. Hence, adding a model and the observed data into the same package forces us to find the optimal complexity of the model. This also ties up the model complexity to be dependent on data length. For big data, we can allow a model to be more complex, because the complex model describes the data part more efficiently. But for small data, a model has to be simple, because otherwise the description of the model increases the length of the whole description. Thus this kind of a criterion has internal over- and under-fitting control.

There still remains the question of actual formulation of the stochastic complexity. Several versions have been proposed by Rissanen during the last decades [33, 34, 35]. At first the definition was based on the so-called two-part-code, where the model complexity and the data complexity are defined independently from each other. Then next version was based on the marginal likelihood definition that takes an average solution over all models (the BDeu score mentioned above is an example of marginal likelihood scores). The latest definition is to use the normalized maximum likelihood (NML) distribution (Shtarkov distribution [41]), which has been proven to be worst-case optimal [35, 13]. Hence, the selected model gives the shortest code among our set of models for worst-case data.

The NML-based stochastic complexity criterion has given very good

results in many application areas. In human genome compression the NML code provides a state-of-the-art method [25, 26, 46]. In image denoising the best NML method is almost as good as the best methods [38]. There exists a histogram estimation method based on the NML that seems to produce very believable histograms [21]. Finally, the factorized normalized maximum likelihood (fNML) criterion for Bayesian networks has been reported to beat the BDeu criterion [43]. The fNML is computational simplification of the true stochastic complexity and therefore the fNML criterion can also be seen as some kind of an approximation of the NML.

The hardest problem when utilizing the stochastic complexity approach is how to overcome computational difficulties: normalized maximum likelihoods are hard to compute, because they involve a normalization term which requires summing over all possible data tables that are of the same size as the observed data table. There are three options: to compute the sum exactly, to use sampling or to use asymptotic approximation. In this Thesis we present new efficient methods for computing the normalizing sums using the exact approach. We also present various new approaches that may eventually lead to computationally even more efficient methods. Even though the sampling approach may finally be the only option in the case of complex models and small data sets, the exact results also support this research, giving at least in some cases a yardstick to which approximations can be compared. The asymptotic approximation approach is probably not usable with small data sets, because in these cases the results cannot be guaranteed to be the correct one or even close enough to the correct one. Therefore for comparison purposes we must know the exact values to avoid pointless and tremendous analytic analyses that just prove the accuracy not to be very good.

The scope of this Thesis is to develop a computational method for computing the stochastic complexity of certain simple probabilistic graphical models. The work contains no comparative analysis between different model selection criteria, but focuses only on stochastic complexity. The true practical value of this work will show up later. In this Thesis we first give efficient algorithms that compute the normalizing sum for a single multinomial variable. The fastest algorithm is sub-linear. The exact method can be considered to be efficient enough for almost all practical uses. After this we formulate an algorithmic framework for the naive Bayes case. For normal data sizes the algorithms based on this framework are also fast enough. The last case is Bayesian Trees. For practical purposes the algorithm is fast enough only for trees with binary variables.

1.2 Main Contributions

For easy access, below we summarize the main contributions of each paper. In the following the list numbers refer to publication numbers.

1. The normalizing sum of a multinomial variable can be represented using a confluent hypergeometric function. The new form appears to be very fast to compute. This computationally simple representation can also be used with mathematical software packages. We show that the form is closely connected to certain moments of the famous birthday problem.

2. Relying on the form in Paper 1, we use the hypergeometric representation to derive a sub-linear algorithm for computing the multinomial stochastic complexity with fixed precision. We also have to assume that the sufficient statistics are precomputed. The algorithm is based on a relatively good upper bound that we derive in the paper.

3. We show that the known generating function for computing the multinomial stochastic complexity is actually a family of marginal generating functions. We demonstrate that in general we have a bivariate generating function, derive representation for the other marginal generating function family and give implications to recurrence formulas. We also suggest that the same kind of bivariate generating functions exists in the case of more complex models, based on our empirical results.

4. We derive a generating function that can be used for computing stochastic complexities of Naive Bayes Models. The generating function explains the previously known recurrence formulas and gives a new framework for designing faster algorithms for the task.

5. An algorithm for computing the stochastic complexity of a Bayesian forest using matrices is presented. Computation is made more efficient using a generating polynomial approach with polytopes and reusing already computed components. To the author's best knowledge, the algorithm is still the fastest known method for exact computation of normalized maximum likelihood for Bayesian forests.

The author of this Thesis made the main contribution in all of these papers.

The rest of this Thesis is organized as follows: The next chapter gives the required definitions and preliminary information that is essential for understanding the chapters that follow. Chapter 3 summarizes computational

main results with respect to a single multinomial variable. The results are collected from papers 1-3. Computation of the stochastic complexity for Naive Bayes models is presented in Chapter 4 and it is based on papers 1, 3 and 4. The stochastic complexity of the Bayesian tree model is considered in Chapter 5. The main results are from Paper 5 and some minor discussion originates from Paper 3. The concluding chapter ties up all the results in a single package. The original publications are reprinted at the end of this Thesis.

Chapter 2

Information Theory and Models

In this chapter we introduce all the required basic concepts and mathematical theory that is necessary for understanding the main results of this Thesis. The first two sections establish the starting points of the research. The third section introduces the mathematical machinery used.

2.1 Information Theory, Stochastic Complexity and Modelling

Information theory is a theory of communication over a channel [7]. The basic setting is the following: A sender wishes to send information using the channel to some receiver. The sender wants to encode the data he is sending in such a way that the receiver will be able to decode it and read the original message. The sender wants to send as few bits as possible, hence achieve the best possible compression for data. However, in the true world channels are usually noisy, thus they are generating errors to the data. This means that the sender has to merge extra bits to the sent message, so that the receiver can infer the original compressed data even if the channel has mutilated the compressed data. The study of these issues belongs to classical information theory. Nowadays the ideas of information theory have broadened to various application areas, such as into statistical inference. In this Thesis we will not consider noise or other limitations that a channel might cause, but we focus on the source coding problem. Thus, we have fixed-length strings — our data — generated by some source, and the sender has to encode the data using some coding method known by the receiver. The sender encodes data using some code words so that more frequent patterns existing in data should have shorter code words than less frequent ones. In order to achieve any compression, the length of frequent

code words should obviously be shorter than the corresponding substrings in data. On the other hand, infrequent code words are allowed to be longer than original substrings.

A *prefix code* is such that in a set of code words, there is no code word that is the prefix of another code word. Prefix codes have a very pleasant property that we can just concatenate code words without any external bits indicating the ending of a code word. In our statistical inference framework we are only interested in lengths of the code words, not the actual code words. The following inequality [7] defines the relationship between code word lengths and existence of prefix codes:

Theorem 2.1 *(Kraft Inequality): For any countable infinite set of code words that form a prefix code, the codeword lengths $L_C(x)$ satisfy the Kraft inequality*

$$\sum_x 2^{-L_C(x)} \leq 1 \tag{2.1}$$

Conversely, given any code lengths satisfying the Kraft inequality, we can construct a prefix code.

A prefix code is complete, if there does not exist a shorter prefix code. This means that a prefix code is complete if and only if the left hand side of the Kraft inequality is 1.

We argued above that frequent patterns should have short codes. We can say that the probability of a pattern is relative to the frequency of that pattern and define

$$L_C(x) = -\log P(x). \tag{2.2}$$

Our code word lengths are not integer values any more, but in fact this does not make a big difference as argued in [13]. We also see that this code can be interpreted to be complete by previous definitions.

Now finally we are ready to make a big leap and consider an information-theoretic approach to probabilistic modeling. Let $x^n \in \mathcal{X}^n$, where x^n is a sequence of length n and $\mathcal{X}^n$ is the set of all sequences. A parametric probabilistic model assigns a probability distribution over these sequences. Each instantiation of the parameters defines a different probability distribution. For each data sequence x^n, there exists a parameter instantiation (distribution) which gives for this particular sequence the maximal probability — these parameters are called the *maximum likelihood parameters* for this data. However, note that taking the maximum likelihood for each data sequence does not constitute a probabilistic model as the sum of the probabilities is greater than 1, which means that the Kraft inequality condition

is not fulfilled. However, there exists an easy solution: we can normalize each individual maximum likelihood with a sum over all maximum likelihoods (for all the data sets) and get the *normalized maximum likelihood* (NML) distribution [13]. This distribution is also known as the Shtarkov distribution.

Definition 1 *The normalized maximum likelihood distribution using a parametric model $\mathcal{M}$ is*

$$P_{NML}(\mathbf{x}^n \mid \mathcal{M}) = \frac{P(\mathbf{x}^n \mid \hat{\theta}(\mathbf{x}^n), \mathcal{M})}{\sum_{\mathbf{y}^n} P(\mathbf{y}^n \mid \hat{\theta}(\mathbf{y}^n), \mathcal{M})}, \tag{2.3}$$

where $\hat{\theta}(\mathbf{x}^n)$ is a set of maximum likelihood parameters of the model $\mathcal{M}$ for data $\mathbf{x}^n$. The stochastic complexity (code length) can now be defined as

$$SC(\mathbf{x}^n \mid \mathcal{M}) = -\log P_{NML}(\mathbf{x}^n \mid \mathcal{M}). \tag{2.4}$$

Let us look at the properties of the above definition. First, now each sequence is mapped to some code word, and the length of this code word is given by (2.4). However, there cannot be a single model that is best for all data sets, but we are trying to get as close as we can with this kind of a model [13]. For each data sequence, the maximum likelihood gives obviously the theoretical limit we can try to reach (it cannot be exceeded, as it is the maximum). A model (distribution) is considered to be *universal*, if it gives almost as high probability for all the data sets as the best model (i.e. the maximum likelihood model) for each data set gives. *Redundancy* is the difference of code lengths between the best model P and our model $\bar{P}$ for given data. By selecting the data that maximizes this redundancy, we get the worst-case redundancy $RED_{\max}$. A model $\bar{P} = \{\bar{P}^{(1)}, \bar{P}^{(2)}, \dots\}$ is universal, if

$$\lim_{n\to\infty} \frac{RED_{\max}(\bar{P}^{(n)}, P)}{n} = 0. \tag{2.5}$$

This means that redundancy can increase only sub-linearly with respect to data size. Notice also that a universal model is considered to be a sequence of distributions — one for each data size.

The normalized maximum likelihood gives the smallest worst-case redundancy among all the universal models [13]. It is therefore a minimax optimal universal model and hence the worst case optimal. This property is in fact very desirable, because with the worst case data we lose the least against the best model. We also know that most of the sequences are incompressible, so this worst-case optimality can also be seen as average-case optimality.

We can use the stochastic complexity in model selection by computing it with different parametric models. We compare these code lengths and select the model, which has the shortest code length for the observed data. The stochastic complexity then favors a simple good fitting model, thus it is obeying the Occham's razor principle: simple models are better. The search problem is still left: how to find the best model from a huge set of models. However, stochastic optimization methods and search algorithms are not a subject of this Thesis and therefore in the sequel we omit further discussion on that topic.

There exists yet a very interesting view point that favors the usage of the normalized maximum likelihood in modelling: it is called *indistinguishability* [3, 13]. In practice if we have very little data, we cannot reliably say which one of the models (distributions) generated it. As we get more and more data, we can rule out an increasing set of models. Generally speaking, two models are indistinguishable, if we cannot rule the other out given the data.

The volume of indistinguishable distributions around a given distribution is shrinking as the data size increases. In the limit indistinguishability leads essentially to the same penalization as MDL (while truth lies in the family) and this complexity can be interpreted to be related to a fraction of distributions in the space of distributions that lie close to the truth. Thus, a simple model has only a small amount of parameter settings that can bring it close to the truth, as a complex model has many parameter settings that bring it close to the truth. As we penalize according to the volume of indistinguishable distributions, we are again ending up with the Occham's razor principle.

2.2 Bayesian Models

We adopt the language from statistics. A binary variable is a two-valued variable and a multi-valued variable is called a multinomial variable. If the variables are i.i.d. (independent and identically distributed), and we compute the sum of binary variables, we have as a result a binomial variable defining the binomial distribution. On the other hand a sum of statistical multinomial i.i.d. variables is called a multinomially distributed variable and it defines the multinomial distribution. This terminology may cause some confusion, which we try to avoid.

We have only one data table and we are not considering different orderings of the rows that produce the same relative frequencies, the observed ordering is enough. Hence, for a single variable (Figure 2.1) with L values

and observed data points $x^n = (x_1, \ldots, x_n)$ the likelihood is

$$P(x^n \mid \mathcal{M}_{mult(L)}) = \theta_1^{h_1} \cdots \theta_L^{h_L}, \tag{2.6}$$

where h_k is a number of points assigned to the kth value [22] and θ_k is the probability of the corresponding value. This does not define a multinomial distribution (the multinomial coefficient is missing), because then we would actually take all the data sets that have the same relative frequencies, and we want only to compute the probability of the observed one. Probabilities θ_k can be assigned several ways, but if we compute the observed relative frequencies of each variable value, we get the highest possible likelihood for our observed data. We denote these parameters by $\hat{\theta}_1, \ldots, \hat{\theta}_L$ and call them maximum likelihood parameters.

Definition 2 *The maximum likelihood for the observed data in the multinomial case is*

$$P(x^n \mid \hat{\theta}(x^n), \mathcal{M}_{mult(L)}) = \left(\frac{h_1}{n}\right)^{h_1} \cdots \left(\frac{h_L}{n}\right)^{h_L}. \tag{2.7}$$

Hence, this is the numerator of the NML for one node Bayesian network (single multinomial variable). The denominator will be presented in Chapter 3.

The *naive Bayes model* can be used for classification and clustering tasks. The model has a class variable and m predictor variables of a multinomial type (Figure 2.1). When represented as a Bayesian network, the model is a two-layer tree where the class variable is the root node Y_0 and predictor variables are leaf nodes $Y_1, \ldots, Y_m$ that are independent of each other given the value of the root variable [17]. Thus the joint probability factorizes as

$$P(\mathbf{y}) = P(y_0) \prod_{i=1}^{m} P(y_i \mid y_0), \tag{2.8}$$

where y_i is the value of the corresponding variable Y_i. In the previous single variable case, data was just a vertical vector of length n. Now we have a table that has n rows and $m+1$ columns. We denote it by $\mathbf{x}^n = (\mathbf{x}_1, \ldots, \mathbf{x}_n)$, where each $\mathbf{x}_j$ is a vector $(x_{j,0}, \ldots, x_{j,m})$. The maximum likelihood parameters correspond to the observed relative frequencies, unconditional with the root and conditional with the leaf variables.

Definition 3 *The maximum likelihood for the Naive Bayes model can be computed using the formula*

$$P(\mathbf{x}^n \mid \hat{\theta}(\mathbf{x}^n), \mathcal{M}_{NB}) = \prod_{k=1}^{L} \left(\frac{h_k}{n}\right)^{h_k} \prod_{i=1}^{m} \prod_{v=1}^{K_i} \left(\frac{f_{ikv}}{h_k}\right)^{f_{ikv}}, \tag{2.9}$$

where h_k is the number of vectors assigned to the kth value of the root variable and f_{ikv} is the number of vectors, where the root variable (parent) value is k and ith predictor variable has the value v [22].

As already mentioned, the Naive Bayes is actually a very simple two-level tree (see Figure 2.1). If we have more levels, we get more complicated trees.

A *Bayesian tree* is a directed acyclic graph, where each node has only one parent node (Figure 2.1). We call node A the parent of node B, if node B has an incoming directed arch from node A. Hence, every tree has only one root node. For this model the joint probability factorizes as

$$P(\mathbf{y}) = \prod_{i=1}^{s} P(y_i \mid y_{g(i)}), \tag{2.10}$$

where s is the number of variables and $g(i)$ is the function that returns the index of the parent node of node i.

Definition 4 *The maximum likelihood for Bayesian trees is*

$$P(\mathbf{x}^n \mid \hat{\theta}(\mathbf{x}^n), \mathcal{M}_{tree}) = \prod_{i=1}^{s} \prod_{k=1}^{K_{g(i)}} \prod_{v=1}^{K_i} \left(\frac{f_{ikl}}{f_{g(i),k}} \right)^{f_{ikl}}, \tag{2.11}$$

where $f_{g(i),k}$ is the number of vectors assigned to the kth value of the parent node of i and $K_{g(i)}$ is the number of values of the parent node of node i.

A *Bayesian forest* is a set of Bayesian trees. The maximum likelihood of data can be computed taking the product of maximum likelihoods of the trees in the forest. We only mention that there exist more complex structures called Bayesian networks that are directed acyclic graphs. However, we are not computing stochastic complexity for these structures in this Thesis and therefore we do not define them formally. The scope of this Thesis is purely computational and there are very good introductory texts on Bayesian networks, thus we are not broadening the view more than necessary. Readers interested in the subject can revise for example [17].

This concludes the introduction of the model families. Stochastic complexity formulas for each of these models are defined in the corresponding chapters.

2.3 Mathematical Tools for Computation

Now we start presenting mathematical tools that are utilized to make computation of the NML denominators efficient for the previous models.

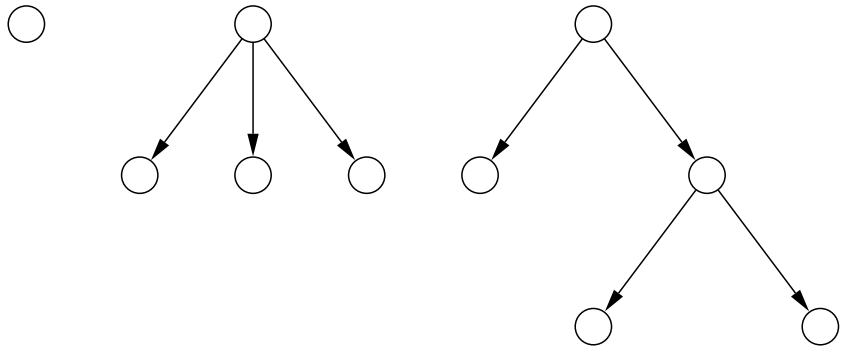

Figure 2.1: Multinomial (left), Naive Bayes (middle) and Bayesian tree (right) models. All graphs together can be interpreted as a forest with three trees.

2.3.1 Generating Functions and Polynomials

Generating functions are powerful tools used in combinatorics and many other areas [11]. The basic idea is that we have two presentations for our target family of functions. The first one is a formal power series presentation, i.e. a generating function presentation, and the other one is a closed-form presentation for the series (does not necessarily exist). We may switch between these presentations and manipulate the form that happens to allow a particular operation more easily. We are actually interested in only the coefficients of a formal power series. These are the functions or values that we want to compute.

In a single variable case we are interested in two different kinds of generating functions: an ordinary generation function (OGF) and an exponential generating function (EGF). In the following we list the necessary properties of both.

The *ordinary generating function* is a formal power series

$$G(z) = \sum_{k=0}^{\infty} a_k z^k, \tag{2.12}$$

and we are interested in the coefficients $a_0, a_1, \ldots$, which are the quantities we want to compute. We denote a sequence of coefficients by $(a_n) = (a_0, a_1, \ldots)$ and the functions of k that gives coefficients by $a(k)$. The variable z is kind of a dummy variable. We usually never evaluate this function $G(z)$ by setting z to some value. If there is a closed-form presentation for the above series, which we have in many interesting cases, we may utilize it as well.

Let us go through some operations we can apply to ordinary generating functions [11, 12]. We use the standard notation for coefficient extraction: $a_k = [z^k]G(z)$. If we multiply two ordinary generating functions $G(z)$ and $F(z)$ and get

$$\sum_{k=0}^{\infty} c_k z^k = G(z)F(z) = (\sum_{k=0}^{\infty} a_k z^k)(\sum_{k=0}^{\infty} f_k z^k), \tag{2.13}$$

then the mth coefficient of the resulting series is defined by a *discrete convolution formula*:

$$c_m = \sum_{k=0}^{m} a_k f_{m-k}. \tag{2.14}$$

Hence, we achieve the resulting ordinary generating function by computing the discrete convolution between sequences of coefficients (Cauchy product). Using the same formula we can easily compute the powers of the generating function $G(z)$. We can achieve any power L by doing $\mathcal{O}(\log L)$-discrete convolutions and using a combinatorial trick [22]: first take the convolution of $G(z)$ with itself to get $G(z)^2$. After this take the convolution of $G(z)^2$ with itself to get $G(z)^4$. This way we finally achieve any $L = 2^i$ and the general case also goes similarly.

The *exponential generating function* is a formal power series

$$EG(z) = \sum_{k=0}^{\infty} b_k \frac{z^k}{k!}, \tag{2.15}$$

where we are interested in coefficients $b_0, b_1, \dots$ and denote the sequence of coefficients by $(b_n) = (b_0, b_1, \dots)$. The functions of k that give coefficients are denoted by $b(k)$.

We use the standard notation for coefficient extraction: $b_k = [z^k]EG(z)$. Notice that we rule out here the factorial term in the denominator, so we are not extracting ordinary formal power series coefficients, but the exponential ones. The resulting coefficients after multiplication of two exponential generating functions $EG(z)$ and $EF(z)$ are defined by the binomial convolution formula:

$$d_m = \sum_{k=0}^{m} \binom{m}{k} b_k h_{m-k}, \tag{2.16}$$

where (h_n) is the coefficient sequence of $EF(z)$. As in the ordinary case, we can also raise $EG(z)$ to higher positive integer powers by doing binomial convolution several times.

As we are interested in computational issues, we may want to use computationally more simple operations for ordinary generating functions. We can write $a_k = \frac{b_k}{k!}$ and this way present an exponential generating function as the ordinary generating function. Thus even if we are dealing with exponential generating functions, we may handle these as ordinary ones. For example this way we do not have to use the binomial convolution formula, but we can just utilize the ordinary convolution formula. We can also take a product of ordinary and exponential generating functions handling both as ordinary ones.

The final operation we go through for single variable generating functions is the *Lagrange inversion formula* (LIF)[37]. The idea is to write a composition in such a way that we do not have the composition anymore. Let $A(z)$ be any formal power series and $B(z) = b_1 z + b_2 z + \cdots$ any formal power series with $b_1 \neq 0$. Thus $B(z)$ has to be an invertible formal power series. Then the following is true:

$$[z^n]A(\overline{B}(z)) = [z^n]A(z)B'(z)\left(\frac{B(z)}{z}\right)^{-n-1}, \tag{2.17}$$

where $\overline{B}(z)$ is the compositional inverse of $B(z)$. Hence, we can transform the composition into a product and still get the same coefficients as before. The basic idea is that the right-hand side gives something that we can handle more easily: if we know how to represent the coefficients of $A(z)$ and the coefficients of $B'(z)(B(z)/z)^{-n-1}$, then we get the coefficients of the original composite function by just using some convolution formula. Thus if the complicated expression involving an inner function gives something simple, for which we know the coefficient presentation, we achieve our goal easily. There also exist other versions of the Lagrange inversion formula, but as we use only this one later, we will not go through the other versions.

There exist many problems that can be solved with a one-variable function, but also many problems, which cannot. Therefore sometimes we also need bivariate generating functions or even multivariate generating functions [11] (the latter case is not relevant in this Thesis). We define only a single exponential version of the bivariate generating function, and leave ordinary and double exponential versions undefined.

Definition 5 *The* bivariate exponential generating function *is a bivariate formal power series:*

$$BG(z,u) = \sum_{k=0}^{\infty}\sum_{l=0}^{\infty} r_{k,l}\frac{z^k}{k!}u^l. \tag{2.18}$$

We can describe these coefficients in the form of an infinite table. This table has marginals and we define that each row and each column is generated by some marginal generating function. Furthermore these functions form a family of horizontal generating functions and a family of vertical generating functions. Each of these functions is a formal power series of a single variable. The *horizontal generating function* (one-parameter) family is

$$MG_k(u) = \sum_{l=0}^{\infty} r_{k,l} u^l \tag{2.19}$$

and the *vertical generating function* (one-parameter) family is defined by

$$MEG^{\langle l \rangle}(z) = \sum_{k=0}^{\infty} r_{k,l} \frac{z^k}{k!}. \tag{2.20}$$

Hence, k is the row index and l is the column index, and to get a corresponding marginal generating function from either of the families, we have to fix either of the indices to some value. We can expand the presented mathematical operations of ordinary and exponential generating functions in a canonical way to these marginal generating functions.

We define *multivariate generating polynomials* to be multivariate polynomials, whose coefficients encode some desired information. We can denote

$$PG_j(\mathcal{X}) = \sum_{\mathbf{x} \in \mathcal{X}} a_{\mathbf{x}} z_1^{x_1} z_2^{x_2} \cdots z_j^{x_j}, \tag{2.21}$$

where $\mathbf{x} = (x_1, \ldots, x_j)$ are vectors in a finite set of positive integer-valued vectors $\mathcal{X}$ and $a_{\mathbf{x}}$:s are the coefficients. Variables z_1 to z_j are dummy variables similar to the ones of generating functions.

In a way the generating polynomials glue both presentations together. They are truncated (finite) formal power series as well as closed-form representations of themselves. For these polynomials, we can do operations such as take a product between two multivariate polynomials. This kind of multiplication corresponds to a higher order convolution operation. Thus, actually we are only interested in doing convolution operations between different quantities, but for representational reasons we need all this machinery to simplify notation.

2.3.2 Umbral Calculus

In the previous section we introduced some basic generating function forms. There is yet one form that we need to present. We define a generating

function of the form

$$\sum_{k=0}^{\infty} s_{k,x} \frac{z^k}{k!} = A(z)e^{x\overline{B}(z)}, \tag{2.22}$$

where

$$A(z) = a_0 + a_1 z + a_2 z^2 + \cdots \qquad (a_0 \neq 0) \tag{2.23}$$

and

$$\overline{B}(z) = b_1 z + b_2 z^2 + \cdots \qquad (b_1 \neq 0), \tag{2.24}$$

where we then denote the (compositional) inverse of $B(t)$ by overline [37]. We use inverse series $\overline{B}(z)$ in the definition, because in Paper 1 we mainly need $B(t)$ and we do not have to then use overlines. Hence, two-variable coefficients $s_{k,x}$ form the sequence $(s_{0,x}, s_{1,x}, \dots)$ called the *Sheffer sequence*. The sequence is defined by the right-hand side. This means that a series expansion of a closed form that is of the given form, defines a Sheffer sequence. Using the definition in the previous chapter we can interpret this generating function to be actually a vertical generating function family of the exponential type. However, for this generating function has been developed a theory of its own, called umbral calculus [37, 8]. Several important polynomials belong into the Sheffer class: e.g. Hermite, Laguerre, Bernoulli and Abel polynomials.

We defined the Sheffer sequence using (2.22). If $A(z) = 1$, we call the sequence of coefficients an *associated sequence*. This case is simpler and in fact this is the one we need. The sequence $(s_{n,x})$ is said to be associated to the *functional* $B(t)$. This can in our framework be considered to be a fancy way of saying that for the given class of exponential generating functions we get coefficients mainly determined by the function $B(t)$.

We only need one umbral calculus computational rule, Umbral composition [37]. As we mentioned in the previous section, we can handle a composite function using the Lagrange inversion formula. However, for this family of generating functions there is also a direct way to infer the coefficients of the composite function.

Proposition 2.1 *(Umbral composition)*

If $(p_{n,x})$ *is associated to* $M(t)$ *and*
$(q_{n,x})$ *is associated to* $N(t)$ *then*

$(\sum_{k=0}^{n} q_{n,k} p_{k,x})$ *is associated to* $N(M(t))$*, where* $q_{n,x} = \sum_{k=0}^{n} q_{n,k} x^k$.

Hence, if we can represent each coefficient of the outer series as a finite formal power series of x^k, where $k = 0 \dots n$, then we have a way to describe coefficients of the composite function.

2.3.3 Hypergeometric Series and Functions

Generalized hypergeometric series is a formal power series, where the ratio of successive terms defines a rational function. If the series converges, we call it the generalized hypergeometric function. But before we can formally define these concepts, we have to define the so called shifted factorials [37]: The *falling factorials* are

$$x^{\underline{k}} = x(x-1)\cdots(x-k+1) \tag{2.25}$$

and the *rising factorials* are of the form

$$x^{\overline{k}} = x(x+1)\cdots(x+k-1). \tag{2.26}$$

At this point we actually need only rising factorials to present hypergeometric series, but falling factorials are utilized later in the Thesis and therefore we defined them also at the same time. The *generalized hypergeometric series* is

$$\sum_{k=0}^{\infty} c_k \frac{z^k}{k!} = \sum_{k=0}^{\infty} \frac{a_1^{\overline{k}} a_2^{\overline{k}} \cdots a_p^{\overline{k}}}{b_1^{\overline{k}} b_2^{\overline{k}} \cdots b_q^{\overline{k}}} \frac{z^k}{k!}, \tag{2.27}$$

where p is the number of rising factorial terms in the numerator and q is the number of rising factorials in the denominator [12]. In the general setting a_i and b_j can be for example complex numbers, but in our combinatorial task we need only integer values later. We can denote the above using the standard notation

$${}_pF_q\left(\left.\begin{matrix} a_1, a_2, \ldots, a_p \\ b_1, b_2, \ldots, b_q \end{matrix}\right| z\right). \tag{2.28}$$

Perhaps the most important property of the generalized hypergeometric series is that the ratio of successive coefficients (of exponential function) is a rational function:

$$\frac{c_{k+1}}{c_k} = \frac{(k+a_1)(k+a_2)\cdots(k+a_p)}{(k+b_1)(k+b_2)\cdots(k+b_q)}. \tag{2.29}$$

We have been discussing generalized hypergeometric series, because mathematical software packages have implementations for the general form. The word "generalized" has been added for historical reasons. If we simply say hypergeometric function or hypergeometric series, we mean the function

$${}_2F_1\left(\left.\begin{matrix} a_1, a_2 \\ b_1 \end{matrix}\right| z\right). \tag{2.30}$$

and its series expansion. Solutions for the hypergeometric differential equation

$$z(1-z)y'' + (b_1 - (a_1 + a_2 + 1)z)y' - a_1 a_2 y = 0 \tag{2.31}$$

can be described using hypergeometric functions. This equation has three singular points. If two of three points merge, solutions can be given using *confluent hypergeometric functions* ${}_1F_1$ and ${}_2F_0$ [2, 1]. The latter function class is the one we will be using later, although we do not have to use the differential equation connection. Hence, we are using the series that can be written as

$${}_2F_0\left(\left.\begin{matrix}a_1, a_2\\ —\end{matrix}\right| z\right) = \sum_{k=0}^{\infty} a_1^{\overline{k}} a_2^{\overline{k}} \frac{z^k}{k!}. \tag{2.32}$$

If some of the a_i:s are negative integer values, then the series is finite and converges. This happens in our case, and we can therefore talk about hypergeometric functions.

2.3.4 Recurrence Equations and Non-Holonomic Functions

Recurrence equations define the relation between coefficients of some series. Although there exist many different types of recurrence equations, in this Thesis we are only interested in the linear ones. Let our function of interest be

$$G(z) = \sum_{k=0}^{\infty} a_k z^k, \tag{2.33}$$

which is a formal power series. We define a *linear homogeneous recurrence equation* to be

$$p_0(i)a_i + p_1(i)a_{i+1} + \cdots + p_r(i)a_{i+r} = 0, \tag{2.34}$$

where $p_k(i)$ is the kth polynomial in one variable and r is a finite positive integer value [32, 47]. Some of the functions $p_k(i)$ must be non-zero. We denoted coefficients of the series by a_i. The above equation must apply for all coefficients in the sequence. By solving a_{i+r} from the above equation, we get a recurrence formula, which can be used for computing coefficients of the series. However, the r first coefficients (initial values) must be computed first, before the recurrence formula can be used.

We have already given one example of a linear homogeneous recurrence of the first order: generalized hypergeometric functions. The ratio of successive terms is a rational function. We can easily see that (2.29) can be written in the form of a recurrence equation.

We define that, if there exists a finite homogeneous linear recurrence for a coefficient sequence, it is *P-recursive* [28]. On the other hand, if we have the corresponding series, then there exists a *linear differential equation*

$$q_m(z)G^{(m)}(z) + \cdots + q_2(z)G''(z) + q_1(z)G'(z) + q_0(z)G(z) = 0 \quad (2.35)$$

with a finite number of terms and the series (function) is *D-finite*. It happens that in a single variable case a closed-form is D-finite if and only if the coefficient sequence is P-recursive. Notice that the smallest possible orders of these two equations do not have to be the same. In some cases they are and in other cases they are not.

We call a function and its coefficient sequence holonomic, if they are D-finite and P-recursive. The opposite term is *non-holonomic*, which means there does not exist either a recurrence or differential equation in the single variable case.

The function $G(z)$ has been so far an ordinary generating function. However, we know that $G(z)$ is holonomic if and only if its exponential counterpart (EGF) is holonomic [4]. Here we have to notice that in both cases we are talking about coefficients of a formal power series, thus coefficients are a_k and $\frac{a_k}{k!}$.

In the multivariate case the situation is a bit more complicated. The following introduction is based on [28]. First we define the single variate case another way: A single variable formal power series is holonomic, if the infinite set of all derivatives of $G(\mathbf{z})$ span a finite dimensional vector space over the field of rational functions in $\mathbf{z}$. Now the multidimensional version is analogous, but instead of derivatives we talk about partial derivatives. This leads to the following:

Proposition 2.2 *$G(\mathbf{z})$ is D-finite if and only if $G(\mathbf{z})$ satisfies a system of linear partial differential equations, one for each $j = 1, \ldots, m$, of the form*

$$\left\{ q_{j,m_j}(\mathbf{z}) \left(\frac{\partial}{\partial z_j}\right)^{m_j} + q_{j,m_j-1}(\mathbf{z}) \left(\frac{\partial}{\partial z_j}\right)^{m_j-1} + \cdots + q_{j,0}(\mathbf{z}) \right\} G(\mathbf{z}) = 0, \quad (2.36)$$

where $q_{i,m_h}(\mathbf{z})$ is a multivariate polynomial and $\mathbf{z} = (z_1, \ldots, z_m)$.

Hence, we have a linear differential equation with respect to every variable. This leads to a thought that maybe we have the same kind of relationship in the multivariate case between differential and recurrence equations as in the single variable case. Unfortunately this is not the case, but we can still create a relationship by setting additional restrictions for the set of admissible recurrence equations. This leads to the following system of recurrences:

Proposition 2.3 *Let the sequence $a_{\mathbf{i}}$, $\mathbf{i} = (i_1, \ldots, i_m)$ satisfy a system of recursions, one for each $j = 1, \ldots, m$, of the form*

$$p_0^{(j)}(i_j)a_{\mathbf{i}} + \sum_{l=1}^{r_j} p_l^{(j)}(i_1, \ldots, i_m)a_{i_1,\ldots,i_j-l,\ldots,i_m} = 0, \tag{2.37}$$

where the $p_0^{(j)}$ are nonzero polynomials of one variable. Then the sequence of $a_{i_1,\ldots,i_m}$ is holonomic.

So, we have the restriction that the first polynomials $p_0^{(j)}(i_j)$ are just polynomials of one variable. If we do not make this restriction, we may have a valid system of recurrences, but they do not have the corresponding holonomic formal power series. However, for the opposite direction we do not need any restrictions.

Proposition 2.4 *If the sequence $a_{\mathbf{i}}$ is holonomic, then it satisfies a system of recurrences, one for each $j = 1, \ldots, m$, of the form*

$$\sum_{l=0}^{r_j} p_l^{(j)}(i_1, \ldots, i_m)a_{i_1,\ldots,i_j-l,\ldots,i_m} = 0. \tag{2.38}$$

We can see that the relationship is asymmetric.

The nice thing about these recurrence equations is that they can be seen as recurrence equations of marginal generating families. In the two-variable case the two recurrence equations are valid of course for horizontal and vertical generating function families.

We still need one important property of multivariate holonomic power series: If a multivariate formal power series is holonomic, then all sections of it are, too. The *section of* $G(z)$ is a power series with fewer variables, where some of the original variables are fixed to a certain value:

$$G^{1,\ldots,s}_{i_{s+1},\ldots,i_m}(z_1, \ldots, z_s) = \sum_{i_1,\ldots,i_s} a_{i_1,\ldots,i_m} z_1^{i_1} \cdots z_s^{i_s}. \tag{2.39}$$

Hence, if we find one section that is non-holonomic, then the original formal power series is also non-holonomic.

2.3.5 Polytopes

We are utilizing polytopes together with the previously presented generating polynomials. As we take a product of two multivariate generating polynomials with all terms positive, we get a multivariate polynomial, which

has more terms than the original ones. We are interested only in some coefficients of this new polynomial and therefore other terms may not be used at all. We want to minimize the computational effort and avoid computing unnecessary terms. Polytopes are multidimensional convex bodies, which we can use as "cages": we have to compute all the terms inside a cage, but none of the outside. In the following we define the problem more rigorously.

Each term of the multivariate generating polynomial can be mapped into a unique point of the space $\mathbb{N}^k$. For example, if we take a term

$$a_{\mathbf{x}} z_1^{x_1} z_2^{x_2} \cdots z_k^{x_k}, \tag{2.40}$$

we can map it by setting the value $a_{\mathbf{x}}$ to the point $(x_1, \ldots, x_k)$. Using this method we can map all the terms of the given multivariate generating polynomial. The multiplication is defined by the ordinary multidimensional convolution formula

$$c_{\mathbf{y}} = \sum_{x_1=0}^{y_1} \cdots \sum_{x_k=0}^{y_k} a_{\mathbf{x}} \cdot b_{\mathbf{y}-\mathbf{x}}, \tag{2.41}$$

where $c_{\mathbf{y}}$ is the coefficient of the product polynomial. Terms $a_{\mathbf{x}}$ and $b_{\mathbf{y}-\mathbf{x}}$ are coefficients of the polynomials to be multiplied. Now the idea is to say that we need to know only coefficients of integer points $(y_1, \ldots, y_k)$ that belong to some set $\mathcal{S}$. We can select a multidimensional convex body so that each of the points in set S is inside the body that we call a polytope.

We have two different ways to describe a polytope [48]. The first one is to define a convex hull over a finite set of points ($\mathcal{V}$-polytope). The second method is to define a bounded $\mathcal{H}$-polyhedron, which is called $\mathcal{H}$-polytope. The $\mathcal{H}$-polyhedron can be defined via an intersection of a finite number of closed half spaces:

$$\{f_{1,i} y_1 + f_{2,i} y_2 + \cdots + f_{k,i} y_k \leq s \mid i = 1 \ldots r\}, \tag{2.42}$$

where some of the multipliers $f_{j,i}$ may be identically zero and s has some boundary value. The number of half spaces is some number r. Depending on the case, either description can be more simple than the other. Also we need different algorithms depending on which one of the presentations we choose. We should always select the presentation that leads to a simpler algorithm for a given task. In this Thesis we use only the half-space description and call the body simply a polytope. An integer lattice is formed by all integer points inside the given polytope (Figure 2.2). As we already defined above, these or some set of these are only the points that are relevant to us.

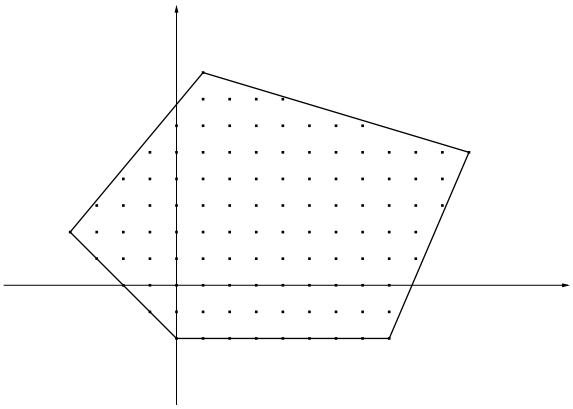

Figure 2.2: *A polytope with the integer lattice.*

We are only interested in some coefficients and therefore the general idea is to do a restricted multiplication of multivariate generating functions inside polytopes. This reduces the computational effort in our setting.

Chapter 3

Computing Multinomial Stochastic Complexity

In this chapter we show how to compute the stochastic complexity for a single multinomial variable. In our setting multinomial variables correspond with nodes of Bayesian network models. We will see later that we need the stochastic complexity of these basic components also with more complex models. First we define the multinomial stochastic complexity. Then we introduce a more general setting for the computation using bivariate generating functions instead of the previously used single variable generating function. After that, we will present new methods to compute the denominator of NML in the multinomial case.

3.1 Defininitions

As we saw earlier, stochastic complexity can be computed by taking a negative logarithm of the normalized maximum likelihood (Theorem 1). Thus computation reduces to computation of the NML. For those models, for which we can easily compute the maximum likelihood, also computation of the numerator is straightforward. We defined the NML numerator of a single multinomial variable already in (2.6). In the multinomial case it takes linear time with respect to data size to compute it. We have to go through the observed data once, because otherwise it is impossible to compute the relative frequencies exactly. On the other hand, if the relative frequencies are given, the task is trivial.

Later on, we use the term *sufficient statistics* [9], when we are referring to the relative frequencies. Thus, relative frequencies contain all the relevant information from the observed data that is needed to fix all free

parameters of a parametric model uniquely. Sufficient statistics can be seen as the original data packed losslessly with respect to a model family. In this Thesis we adopt the convention where a model structure (which defines the number of parameters and their meaning) is called (parametric) model, and for us a model family is a set of model structures — e.g. all Bayesian trees.

Now we are ready to define the NML denominator, which is much harder to compute than the previously presented numerator. The denominator (the normalizing constant or the *multinomial normalizing sum*) is

$$\mathcal{C}_{MN}(L,n) = \sum_{h_1+\cdots+h_L=n} \frac{n!}{h_1!\cdots h_L!} \prod_{k=1}^{L} \left(\frac{h_k}{n}\right)^{h_k}, \tag{3.1}$$

where L is the number of values of the variable and n is the size of observed data [22]. The multinomial model family is denoted by the subscript MN. Using the definition directly, we need to compute a sum of $\mathcal{O}(n^L)$ terms. This can be easily reduced to $\mathcal{O}(n^{L-1})$ using a simple parameter substitution trick that can be seen more easily from the most common definition of the *binomial normalizing sum*:

$$\mathcal{C}_{MN}(2,n) = \sum_{k=0}^{n} \binom{n}{k} \left(\frac{k}{n}\right)^{k} \left(\frac{n-k}{n}\right)^{n-k}, \tag{3.2}$$

where $h_1 = k$ and $h_2 = n - k$. One of the sums is in fact redundant, because of the requirement that all the counts h_i sum to n. Notice that the binomial normalizing sum is actually a somewhat misleading name, albeit a very convenient one: we should be talking about binary variable normalizing sums as we are computing the maximum likelihood for binary variables, not for the variables that define binomial distributions. However, the sum is exactly the same for both cases and the binomial normalizing sum is not as cumbersome to use as the binary variable normalizing sum, therefore we are using the word 'binomial'.

Next we are going to discuss recurrence formulas for computing the desired sum $\mathcal{C}_{MN}(L,n)$. After that we tackle the efficient presentation for the sum and show some useful properties of it. Finally we concretize the results by giving algorithms and computation methods.

3.2 Recurrence Formulas

We start by introducing generating functions that can be used for deriving new results for the computation of the denominator. This idea itself is

not new, as the best existing results [19, 20, 45] have been derived using a generating function that we define later. However, in this Thesis we define a more general setting — the bivariate generating function, which provides new results.

Definition 6 *The bivariate generating function for computing the multinomial normalizing sums is*

$$f(z,u) = \sum_{L=0}^{\infty}\sum_{n=0}^{\infty} \mathcal{C}_{MN}(L,n) n^n \frac{z^n}{n!} u^L = \frac{T(z)-1}{T(z)-1+u}, \tag{3.3}$$

where $T(z)$ is a tree function.

The right-hand side is only seemingly closed form, because the definition of the tree function [19] is

$$T(z) = \sum_{n=1}^{\infty} n^{n-1} \frac{z^n}{n!} \tag{3.4}$$

and it has no closed form. It also has a very simple connection to Lambert's W function [6] from physics: $T(z) = -W(-z)$, which may be more familiar to some readers. Using this bivariate generating function we can compute the previously presented horizontal and vertical generating function families.

The one-parametric vertical generating function family is previously known. Some of the previous authors simply use the name generating function for the whole family [24]. The family is defined by

$$f^{\langle L \rangle}(z) = \sum_{n=0}^{\infty} \mathcal{C}_{MN}(L,n) n^n \frac{z^n}{n!} = \left(\frac{1}{1-T(z)}\right)^L \tag{3.5}$$

with free parameter L. There are many highly useful results that can be derived using this closed form.

The one-parametric horizontal generating function family is previously unknown, although there is for example a simple recurrence formula over the coefficients.

Theorem 3.1 *The horizontal generating function family for the multino-*

mial normalizing sum with the free parameter n is of the form

$$f_0(u) = \frac{1}{1-u} \quad \textit{and} \tag{3.6}$$

$$f_n(u) = \sum_{L=0}^{\infty} \mathcal{C}_{MN}(L,n) n^n u^L \tag{3.7}$$

$$= n^n u \left(1 + \left(\frac{u}{1-u} \right) \sum_{L=0}^{n} \frac{n!}{n^L (1-u)^L (n-L)!} \right). \tag{3.8}$$

The (closed) form may look awkward, but if we fix n and expand, we get rational generating functions (Paper 3). This means that we have functions with a finite representation unlike in the vertical case.

Let us look at computational issues. Each generating function has the desired $\mathcal{C}_{MN}(L,n)$ attached to the terms and therefore we want to compute the coefficients as efficiently as possible. Furthermore, with more complex models, we usually start by computing the table of $\mathcal{C}_{MN}(L,n)$ all the way to some fixed L and n. Hence, with practical models it is not enough to compute just one normalizing sum. This kind of problems are commonly solved with dynamic programming. We need two recurrence formulas: one going over L (horizontal family) and the other going over n (vertical family). For this purpose, there must be the corresponding recurrence equations.

There is a well-known recurrence formula over L, and many experimental model selection applications are already using it. This formula [16, 20], in the form of a homogeneous linear recurrence equation, is

$$(L-2)\mathcal{C}_{MN}(L,n) + (2-L)\mathcal{C}_{MN}(L-1,n) + (-n)\mathcal{C}_{MN}(L-2,n) = 0 \tag{3.9}$$

and it is valid for all $L \geq 0$ and a for fixed data size $n > 0$. The equation also works for the case $n = 0$, because the second term goes to zero and $C_{MN}(L,0) = 1$ for all L. This same equation is valid for the whole horizontal family. When used as a recurrence formula, it needs two initial values at the beginning: $\mathcal{C}_{MN}(1,n)$ equals one for all n and the other value can be computed using for example (3.2). The formula can be proven using rather simple calculus of the one-parametric generating function family (vertical family) [20].

There exists also a standard way to construct a recurrence formula for a given rational generating function. The particular form of our rational generating functions gives another recurrence equation (Paper 3):

$$\sum_{j=0}^{n+1} \binom{n+1}{j} (-1)^j \mathcal{C}_{MN}(L-j,n) = 0. \tag{3.10}$$

We can write this using the backward difference operator ∇. Defining $\nabla_L \mathcal{C}_{MN}(L,n) = \mathcal{C}_{MN}(L,n) - \mathcal{C}_{MN}(L-1,n)$, the previous recurrence gets form

$$(\nabla_L)^{n+1} \mathcal{C}_{MN}(L,n) = 0, \tag{3.11}$$

where $(\nabla_L)^{n+1}$ means applying the operator $n+1$ times with respect to variable L.

Notice that the number of terms in this recurrence is depending on data size n, which means that the recurrence is not related to the family, but to single members of the horizontal family. However, the recurrence seems to have a very pleasant property: we can always leap over any constant number of terms and the recurrence is still valid. Thus, $\mathcal{C}_{MN}(L-j,n)$ can be replaced with $\mathcal{C}_{MN}(L-b\cdot j,n)$, where b is a positive non-zero integer value. Using this property, we can utilize an increasing leap size and this way compute the multinomial normalizing sum for arbitrary large values of L. This kind of algorithm has no value in our framework, but may be valuable for some other purposes.

What about the recurrence equation over n? Unfortunately, for the vertical family as a whole, nobody has managed to find one. In fact, there does not exist a good solution even for single members of the family. Knuth and Pittel have presented one in [19], but it takes all the previous coefficients to compute the next one. The formula also changes parameter L to $L-2$ in the same time. We showed in Paper 3 that the bivariate generating function is non-holonomic, because it has a non-holonomic section $f^{\langle 1\rangle}(z)$. Therefore, there cannot be a linear homogeneous recurrence formula for the vertical family. Otherwise the bivariate generating function would be holonomic as well as the non-holonomic section. We also argued that because we do not have a linear homogeneous recurrence for the sequence $\mathcal{C}_{MN}(L,n)n^n$, a recurrence cannot exist for the sequence of $C_{MN}(L,n)$ over n either. This argument, however, later appeared to be incorrect. Let us look at

$$\sum_{n=0}^{\infty}\sum_{L=0}^{\infty} \mathcal{A}(L,n) n^n \frac{z^n}{n!} u^L, \tag{3.12}$$

where $\mathcal{A}(L,n) = 1$ for all n and L. Now each vertical generating function is equal to $f^{\langle 1\rangle}(z)$. However, $\mathcal{A}(L,n)$ satisfies the homogeneous linear recurrence equation

$$\mathcal{A}(L,n) - \mathcal{A}(L,n-1) = 0. \tag{3.13}$$

Here we selected $\mathcal{A}(L,n)$ to be a constant, but for example the function $(L-1)n+1$ could have been used as well. For setting $L=1$, we find our

known non-holonomic section. However, this function has the second order homogeneous linear recurrences for both families.

The practical side of the non-holonomicity results is that automatic algorithms that are using the vertical generating function family, the bivariate generating function or the corresponding sequences of these, cannot find a homogeneous linear recurrence. However, such a recurrence for the sequence of $\mathcal{C}_{MN}(L,n)$ over n has not been found by any of the tested algoritms either. Notice also that discrete convolution is done over sequences that are known to be non-holonomic and nothing gets cancelled. These observations strongly suggest that there may not exist such a recurrence.

3.3 Properties of the Normalizing Sum

Our task is usually to compute a table of normalizing sums (as mentioned before), but there does not exist any known efficient recurrences over n. Let us start from a different view: how to compute each $\mathcal{C}_{MN}(L,n)$ as efficiently as possible without a recurrence. The whole table can be computed obviously in time $\mathcal{O}(n^2+nL)$, by computing the binomial normalizing sums first and then using the linear recurrence formula over L. If we want to compute just one normalizing sum, it takes time $\mathcal{O}(n+L)$. This is the quantity that we are trying to make as small as possible. Using an asymptotic approximation formula [23], we can achieve time complexity $\mathcal{O}(nL)$ for computing the whole table as each term $\mathcal{C}_{MN}(L,n)$ takes only a constant time to compute. This base line is our unreachable lower bound also for the exact computation methods.

The first representation for the multinomial normalizing sum can be derived using the vertical generating function family. The function family can be interpreted to be a composite function. For this composite function we apply the Lagrange inversion and binomial convolution formulas, which gives as a result the following theorem (Paper 1):

Theorem 3.2 *The multinomial normalizing sum can be written as*

$$\mathcal{C}_{MN}(L,n) = \sum_{k=0}^{n} \binom{n}{k} \frac{(L-1)^{\overline{k}}}{n^k}, \tag{3.14}$$

where $n \geq 1$, $L \geq 1$ *and* $n, L \in \mathbb{N}$.

This theorem practically says that as we are just interested in positive integer points of the normalizing sum, then the computation formula simplifies a great deal. This new form consists of only one sum and the rising factorial notation hides one product. An almost similar-looking, but less optimal

form for our purposes, can be derived using the previously mentioned umbral calculus. The idea is to notice that the vertical generating function family is a composition of two functions that are associated sequence form. Then the second form can be found using the umbral composition formula (Paper 1).

We can also represent the formula in Theorem 3.2 in another way using confluent hypergeometric functions (Paper 1):

Theorem 3.3 *A hypergeometric presentation for the multinomial normalizing sum is*

$$\mathcal{C}_{MN}(L,n) = {}_2F_0\left(\begin{matrix} L-1, -n \\ - \end{matrix}\middle| -\frac{1}{n}\right). \tag{3.15}$$

The hypergeometric form can be interpreted to be function ${}_2F_0$ with parameters $L-1$ and $-n$ evaluated at the point $-\frac{1}{n}$. Thus each function with a fixed positive integer parameter $L-1$ gives a value of the normalizing sum only in one point (Figure 3.1). Although hypergeometric functions are presented via infinite sums, the parameter $-n$ causes each sum to consist only of $n+1$ terms and all the other terms are equal to zero.

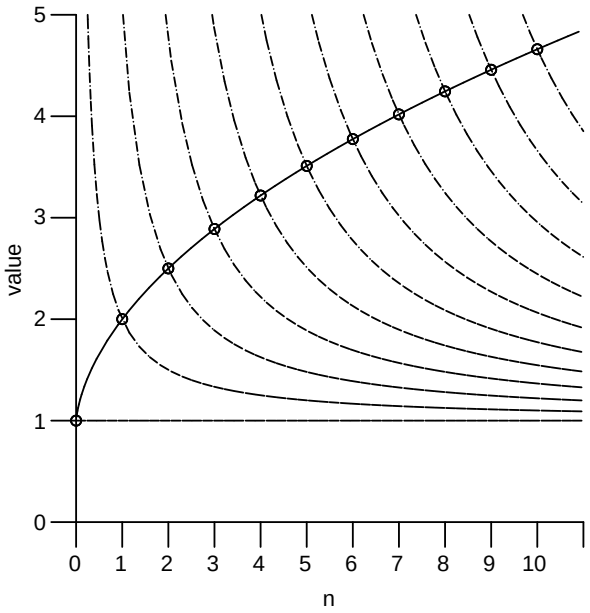

Figure 3.1: *The solid line gives the values of binomial sums and dotted lines are hypergeometric functions. The x-axis goes over n instead of* $-\frac{1}{n}$ *for achieving better separation between curves.*

The normalizing sum (3.15) has terms that can be written in the form

$$m_k = \frac{(L-1)^{\overline{k}}}{k!} \cdot \frac{n^{\underline{k}}}{n^k} \tag{3.16}$$

and if $L = 2$, the rising factorial is equal to the factorial (Paper 2), and the first part disappears. We denote the terms of this more simple case by b_k. A closer look at these terms reveals that in the two-valued case the sequence of the terms go rapidly to zero. This can be seen for example in the ratio of successive terms:

$$\frac{b_k}{b_{k-1}} = \frac{n-k+1}{n}. \tag{3.17}$$

As double precision floating-point numbers can only present arbitrary values by finite precision, a question arises: how many terms of the finite sum are needed to get a result with a given precision. After some mathematical analysis we get the answer:

Theorem 3.4 *Given precision in digits (d) and data size n, the index t of the last needed term for the binomial case is* $\left\lceil 2 + \sqrt{-2n \ln(2 \cdot 10^{-d} - 100^{-d})} \right\rceil$.

This is an upper-bound approximation, which means that if we sum $t+1$ first terms, we can be sure that we achieve the wanted precision. The approximation also seems to be reasonably tight as presented in Paper 2 (Figure 3.2). In fact, although minor changes to the proof would give even a tighter bound, these changes would not cause any noticeable effect in practice. The main consequence of this result is the observation that the required number of terms to achieve the precision d is $\mathcal{O}(\sqrt{dn})$. This means that in practice we always need only a sub-linear number of terms. Next we take a closer look at a more complicated multinomial case. The ratio of successive terms in the multinomial case is

$$\frac{m_k}{m_{k-1}} = \frac{(n-k+1)(k+L-2)}{nk}, \tag{3.18}$$

which is more complex than the binomial ratio. The terms m_k are first getting bigger instead of getting smaller as the terms b_k. However, as in Figure 3.3, we can plot the multinomial terms and it can be easily seen that they form a unimodal function. The peak is moving to the right, if L has bigger values. Thus we can interpret that in the binomial case, the peak is located at $k = 0$, because the first term is the biggest one. This behavior implicates that in the multinomial case there is an interval of indices of

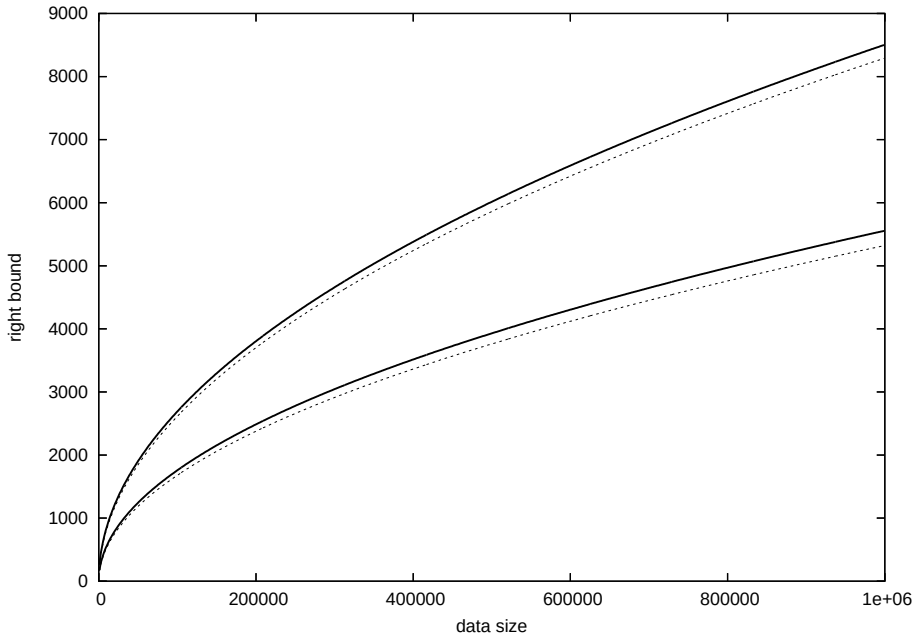

Figure 3.2: *Terms needed for 16 (above) and 7 digit precisions with given data size. Actual approximations are shown as a thick solid line. Thin dotted lines represent optimal index values.*

terms, which we have to compute in order to achieve certain precision d. However we have also the very efficient recurrence equation (3.9) that can be used for computing multinomial normalizing sums if the binomial sums are known. In fact, this recurrence method can be easily seen to be a more efficient way to compute multinomial normalizing sums than the direct ratio method using (3.18). Although deriving the left and right index bounds for the multinomial case holds some mathematical curiosity, we gain no increase in computational efficiency and therefore the derivation is left for someone else to do.

Now we are ready to collect all the observations and present efficient algorithms based on them.

3.4 Efficient Computation and Algorithms

There is a need for two different type of algorithms that compute multinomial normalizing sums: We are interested in efficient computation methods that give exact rational number answers as well as in algorithms that give floating-point answers. The first type of methods are used for research pur-

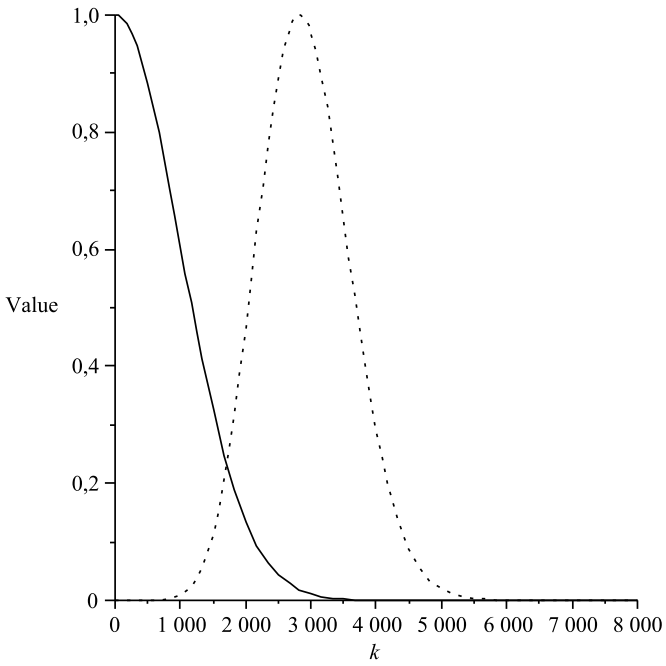

Figure 3.3: *The first 8000 terms of the trinomial (left) and the scaled 15-nomial (right) normalizing sums when data size (n) is one million.*

poses. To develop the latter type of methods, we have to know the correct answers. The latter type is of course much faster and is used in actual model selection tasks.

We can compute rational number solutions easily using standard mathematical software packages — for example Maple (Paper 1). Let the number of data points be 100 and the number of bins (number of the values of the multinomialvariable) be 4. The exact value of the multinomial normalizing sum can be computed in this case by writing

```
simplify(subs([L=4,n=100],hypergeom([L-1,-n],[],-1/n)));
```

and also the floating-point solution can be achieved by replacing the command `simplify` with the command `evalf`.

Usually the stochastic complexity criterion is coded using some programming language as a part of a model selection software. In this case the following sub-linear scheme for computing the denominator of the NML should be utilized:

```
ComputeC_MN(d,n,L){
  sum=1; b=1;

  t=2+ceiling(sqrt(2*n*(-(log(2)-d*log(10))
             -log(1-exp(-d*log(100)+d*log(10)-log(2))))));

  for k from 1 to t{
     b=(n-k+1)/n*b;
     sum=sum+b;
  }

  sum_old=1;
  for k from 3 to L{
     sum_new=sum+n/(L-2)*sum_old;
     sum_old=sum;
     sum=sum_new;
  }

  return sum;
}
```

Computation of the index t differs from the formula in Theorem 3.3. The reason is that direct usage of the formula causes some underflows and to avoid this we need to modify the formula using logarithmic tricks. The achieved time complexity for computing an $(n \times L)$-table of normalizing terms is now $\mathcal{O}(n \log n + nL)$ against previous $\mathcal{O}(n^2 + nL)$.

The same algorithmic ratio method can also be used for computing exact rational number solutions: we can just sum all $n + 1$ terms instead of $t + 1$ terms. Also the floating-point operations must be overloaded with rational number operations. The true time complexity of the algorithm rises quite high as these new operations are applied.

The simple algorithm does not fulfill requirements of scientific computing in the floating-point case, because the presented elementary operations make some floating-point errors and therefore the theoretical precision is not achieved. However, in Paper 2 we empirically showed that the total resulting error is not very significant and therefore in practice the simple code can be utilized. There is a very simple coding trick to truly achieve precision d: we should use higher precision floating-point numbers and cut the tail digits. Empirically it seems that even for the data size of 10^{12}, with

the double precision floating-point numbers it is enough to cut about the 6 last digits. The exact analysis confirming previous empirical results should be done in the future. A very interesting open question is how many digits we actually need in order to make the required difference between different models of some model family. The answer could be utilized to optimize the performance of a searching algorithm in a model selection task.

If the number of data points is very high and we need to compute a table of normalizing sums, even the sub-linear algorithm can be too slow. In this case we can use the previously known asymptotic approximation that was already mentioned in the beginning of the previous section. The asymptotic approximation is originally a result derived to compute the minimax redundancy of memoryless sources by Szpankowski [45]. A memoryless source generates a new data point each time without using information of previous generated points. However, this approximation happens to be the same as the logarithm of normalizing sums. The approximation is very good even with moderate data sizes, but still the requirements of scientific computing are not fulfilled. The given precision cannot be chosen or guaranteed even in theory. As we will see later, with more complicated models we have to use the discrete convolution formula over sequences of multinomial normalizing sums. This kind of operations will cause errors to cumulate and therefore usage of this approximation would give unpredictable results.

3.5 Connection to the Birthday Problem

The multinomial normalizing sum has a connection to the birthday problem. The birthday problem can be seen as a process, where we take a new person at each step and look at his birthday and compare birthdays of persons picked at preceding steps [10]. The process stops if two persons have the same birthday. Now we can set a question: How many people on average do we need so that at least two of them have the same birthday? Let n be a number of possible labels. In the classical setting n is set to 365 and each label corresponds to one day of the year. The probability that the process ends at step k is

$$P^{(n)}(X = k) = \frac{(k-1)n^{\underline{k-1}}}{n^k}. \tag{3.19}$$

An intuitive explanation of the numerator can be seen as choosing $k-1$ distinct birthdays and when you pick the kth person, you have $k-1$ possibilities to hit an already selected one. Now we get the answer to the

presented question by calculating the expectation

$$E^{(n)}(X) = \sum_{k=2}^{n+1} kP^{(n)}(X = k). \tag{3.20}$$

In the classical case we have $E^{(365)}(X) \approx 24.6166$. The answer is counter-intuitive and that is why this problem is also known as the birthday paradox. There is a method called birthday attack [29], which uses the mentioned property. The basic idea is to replace a legitimate message with a fraudulent message by doing minor modifications to both messages so that the digital signatures of both messages coincide. This way the legitimate message can be changed later to the fraudulent message. The birthday paradox causes matching signatures to be easier to find than what the intuition says.

We can compute the previous expectation also by using a binomial normalizing sum. In fact, there is one-to-one correspondence (Paper 1):

$$E^{(n)}(X) = \mathcal{C}_{MN}(2, n). \tag{3.21}$$

This immediately raises the question about an equivalent pair for the multinomial normalizing sum. We need to define a new concept first: The rising factorial moments. For the birthday problem these are

$$E^{(n)}(X^{\overline{m}}) = E^{(n)}(X(X+1)\cdots(X+m-1)) \tag{3.22}$$

and the sought relationship between the multinomial normalizing sums and these can be written as

$$E^{(n)}(X^{\overline{m}}) = m!\mathcal{C}_{MN}(m+1, n). \tag{3.23}$$

Using this equation and the results for normalizing sums, we can write trivial results for computing the rising factorial moments for the birthday problem (Paper 1). There might also be some methods or results developed for the birthday problem framework that can be useful in our side. An especially interesting question is whether there are connections between normalizing sums for trees and application areas of the birthday problem.

Chapter 4

Computing Stochastic Complexity for Naive Bayes models

In this chapter we propose a general framework for computing the normalizing sums for Naive Bayes models. We also present recurrence formulas that can be used for the computation.

4.1 Definitions

Naive Bayes models can be utilized in prediction, classification and clustering tasks. We already introduced this model class in Chapter 2.2 and also showed how to compute the maximum likelihood for it. Now we focus on the computation of the NML normalizing sum in this case. The original formula is not shown here, because it is relatively complicated and not easily interpretable. We only present the generating function here, and further details on how to derive this generating function, can be found in [22] and Paper 4.

Definition 7 *The* basic series *for the Naive Bayes model is of the form*

$$\mathcal{E} = \sum_{n=0}^{\infty} \mathcal{C}_{MN}(K_1, n) \cdots \mathcal{C}_{MN}(K_m, n) n^n \frac{z^n}{n!}, \tag{4.1}$$

where the terms $\mathcal{C}_{MN}(\cdot, \cdot)$ are the multinomial normalizing sums of the corresponding predictor variables and by K_i we denote the number of values (bins) of the ith predictor variable.

Raising the basic series to some power L, where L is the number of values of the class (root) variable, we get a power form of the sought exponential generating function. By expanding this form we have

$$\mathcal{E}^L = \left(\sum_{n=0}^{\infty} \mathcal{C}_{MN}(K_1, n) \cdots \mathcal{C}_{MN}(K_m, n) n^n \frac{z^n}{n!}\right)^L \tag{4.2}$$

$$= \sum_{n=0}^{\infty} \mathcal{C}_{NB}(L, K_1, \ldots, K_m, n) n^n \frac{z^n}{n!}, \tag{4.3}$$

where the Naive Bayes normalizing sum is denoted by $\mathcal{C}_{NB}()$. Thus in the basic series $L = 1$ and this corresponds to Naive Bayes models, where the class variable has only one value.

Let us take a closer look at Formulas (4.2) and (4.3). If we write the multinomial generating function in the same form, we have

$$\left(\sum_{n=0}^{\infty} n^n \frac{z^n}{n!}\right)^L = \sum_{n=0}^{\infty} \mathcal{C}_{MN}(L, n) n^n \frac{z^n}{n!}. \tag{4.4}$$

These two forms look quite similar. However, in the Naive Bayes case we do have a product of multinomial sums in the coefficients. The expansion in both cases can be made using convolution formulas the way we described in Section 2.3.1.

4.2 Recurrence Formulas

In the multinomial case there exist many efficient computation methods for computing normalizing sums. The Naive Bayes case is more complex, and despite research efforts truly effective recurrence methods are still missing. It is not known, for example, whether there exists any recurrence formula for the 'horizontal' generating function. Actually in the Naive Bayes case there is no natural horizontal generating function, because the model has more than two parameters. We still choose to call a generating function that goes over L a horizontal generating function, because it seems to have the same kind of qualities with the horizontal generating function in the multinomial case. However, there exist two recurrence formulas. In the following we first present a modification to the known recurrence method and derive a new recurrence that can be very effective in a certain case.

There is a known recurrence method for computing the normalizing sum for Naive Bayes models [22]. A similar formula also applies in the multinomial case. This suggests that in fact this method is just based on utilization of the exponential convolution formula. For the Naive Bayes the

recurrence is of form

$$\mathcal{C}_{NB}(L, K_1, \ldots, K_m, n) = \sum_{k=0}^{n} \binom{n}{k} \left(\frac{k}{n}\right)^k \left(\frac{n-k}{n}\right)^{n-k} \cdot \mathcal{C}_{NB}(L^*, K_1, \ldots, K_m, k)\, \mathcal{C}_{NB}(L - L^*, K_1, \ldots, K_m, n-k). \quad (4.5)$$

This is just a modified exponential convolution formula for two basic series raised to powers L^* and $L - L^*$. the two extra terms after the binomial multiplier are related to removal of the multiplying term n^n, which is present in the basic series coefficients. The recurrence above gives us a new normalizing sum when given two normalizing sums. However, if we use this formula, we unnecessarily compute extra terms that cancel n^n terms all the time. For the sake of efficiency, we should use the Naive Bayes basic series and the discrete convolution formula (2.14) for computation. When the needed term is computed, we can cancel the term $\frac{n^n}{n!}$ by multiplying and get the corresponding normalizing sum.

It appears that there may exist yet another recurrence (Paper 3) similar to (3.11):

$$(\nabla_L)^{n+1} \mathcal{C}_{NB}(L, K_1, \ldots, K_m, n) = 0. \quad (4.6)$$

This claim is however purely based on empirical tests, and we have not mathematically derived the horizontal generating functions for Naive Bayes models. If the claim is true, it means that the denominators of the generating functions (closed form) must be identical to the horizontal generating functions in the multinomial case. In fact these generating functions can be easily found using the Maple software: First compute the initial values for the recurrence and suppose that recurrence applies. Then use the Maple command `rectodiffeq` and after that solve the resulting equation.

What is most interesting is that a similar kind of recurrence is not valid only for the root variable, but it seems to work also for the leaf variables. In this case it takes the form

$$(\nabla_{K_i})^{n+1} \mathcal{C}_{NB}(L, K_1, \ldots, K_i, \ldots, K_m, n) = 0. \quad (4.7)$$

Both of these recurrences also allow us to jump over fixed and equal sized intervals. For example we can satisfy recurrence equations just by taking every third coefficient. The undesired fact is that when using these recurrence formulas, we need to compute initial values proportional to data size. However, usually in practical applications the number of data vectors is greater than the number values of a variable. This observation makes these recurrence formulas useless in our framework, but it does not mean that they are useless in all frameworks. In the future they may still directly or indirectly prove to be useful.

4.3 Efficient Computation and Algorithms

We start by first introducing methods for exact computation in the Naive Bayes case (Paper 1). Writing the power form of the generating function (formula (4.2)) using Maple notation, we get the first one hundred terms by writing

```
simplify(series((1+sum(hypergeom([-n,3],[],-1/n)
                       *hypergeom([-n,4],[],-1/n)*n**n/n!
                       *z**n,n=1..infinity))**2,z,100));
```

Here we have two predictor variables with 4 and 5 values and a binary class variable. The size of data is 100 data vectors. The first term is separated from the series and replaced according to the definition with the value 1, as otherwise we would be dividing with zero.

The computation can be done also by coding the previous method using some programming language. The following general scheme (Paper 4) can be used for computing the normalizing sum:

1. Compute a table of multinomial terms. The size of the table is $(n+1) \times (\max_i K_i)$. Use the proposed sub-linear algorithm when computing with floating-point numbers.
2. Compute the coefficients of a Naive Bayes basic series.
3. Use some algorithm to raise the basic series to the power of L.
4. Extract the normalizing sums from the formal power series coefficients by multiplying the kth coefficient by $\frac{k!}{k^k}$.

The most time-consuming part of this algorithm is phase 3. However, there exists a method called the Miller formula [15]. If we use this formula, we can raise a truncated formal power series with any real power and it always takes the same number of basic operations, thus we need to do only $\mathcal{O}(n^2)$ multiplications. This is quite an amazing result, since the operational cost is the same whether we take a product of a basic series by itself or raise the basic series to the power of 10^6. The Miller formula can be formalized in the form of the following proposition [18].

Proposition 4.1 (The Miller formula) *If two formal power series are* $V(z) = 1 + \sum_{k=1}^{\infty} v_k z^k$ *and* $W(z) = \sum_{k=0}^{\infty} w_k z^k$ *and* $W(z) = (V(z))^{\alpha}$, $\alpha \in \mathbb{R}$, *then* $w_0 = 1$ *and* $w_n = \sum_{k=1}^{n} \big((\frac{\alpha+1}{n})k - 1 \big) v_k w_{n-k}$.

This formula can be utilized only when computing exact solutions using rational numbers, because with floating-point numbers and almost with all Naive Bayes structures the Miller method seems to be unstable (Paper 4). There may exist some stable algorithm, that does phase 3 in $\mathcal{O}(n^2)$ multiplications also in the floating-point case, but we are not aware of such an algorithm. The fastest stable method that we know is the normal sequential multiplication method using the discrete convolution formula. Notice that instead of $\mathcal{O}(L)$ series multiplications we need only $\mathcal{O}(\log_2 L)$ series multiplications, if we use sub-results, as we already mentioned previously. For example when computing $\mathcal{E}^8$, the algorithm starts by computing $\mathcal{E} \cdot \mathcal{E}$. Then it computes $\mathcal{E}^2 \cdot \mathcal{E}^2$ and finally $\mathcal{E}^4 \cdot \mathcal{E}^4$, which gives the desired result. Using this type of a procedure we can achieve any integer power L. This of course means that the total number of multiplications is $\mathcal{O}(n^2 \log_2 L)$.

Chapter 5

Computing Stochastic Complexity for Bayesian Forests

In this chapter we will show how to compute the normalizing sums for Bayesian forests. The task is much harder than in the multinomial or Naive Bayes cases. In the previous cases the generating functions and power forms were known. Now we do not have the generating function, but we have to mainly do computation over all valid sufficient statistics and to use generating polynomials.

First we start by defining the problem, then we motivate and give insight on how to solve the computational problem and finally we present an algorithm for the task. There is also some discussion about accelerating the computation using various computational tricks.

5.1 Definitions

Sufficient statistics must be represented in some way. We use *k-compositions* [40] and *k-partitions* (Paper 5). A k-composition is a partition of a positive integer n into k bins (k positive integers that sum up to n). For example $(7,3,1)$, $(2,6,3)$ and $(0,0,11)$ are 3-compositions of 11. On the other hand, if the ordering of bins does not matter, we are actually talking about k-partitions. All 3-compositions of 2 are $(2,0,0)$, $(0,2,0)$, $(0,0,2)$, $(1,1,0)$, $(1,0,1)$ and $(0,1,1)$, but there are only two 3-partitions of 2: $(2,0,0)$ and $(1,1,0)$. Thus the number of k-compositions is a magnitude of k-factorial more than the number of k-partitions. For a given k-partition $\mathbf{x} = (x_1, \ldots, x_k)$, the corresponding number of k-compositions is given by

$$m(\mathbf{x}) = \frac{k!}{\prod_{w \in \mathbf{x}} \mu_w(\mathbf{x})!}, \tag{5.1}$$

where $\mu_w(\mathbf{x}) = |u : x_u = w|$ tells us how many times a value w appears in a k-partition $\mathbf{x}$ [30]. For example for a 5-partition $(3, 3, 2, 2, 0)$ there exists $\frac{5!}{2!2!1!} = 30$ 5-compositions, because there are 2 twos and 2 threes and 1 zero.

After these definitions we can define the problem-specific $c()$-function as in Paper 5. We rewrite the definition of the multinomial normalizing sum (3.1) in a new way:

$$\mathcal{C}_{MN}(k, n) = \sum_{x_1+\cdots+x_k=n} c((x_1, \ldots, x_k)), \tag{5.2}$$

where the sum goes over the set of all k-compositions of data size n. Notice that k is equal to the previously mentioned L, but as k-compositions is the generally used term, we shall from now on use k instead of L. An exact formula for the *c-function* is

$$c((x_1, \ldots, x_k)) = \frac{(\sum_{i=1}^k x_i)!}{\prod_{i=1}^k (x_i!)} \prod_{j=1}^k \left(\frac{x_j}{\sum_{i=1}^k x_i} \right)^{x_j}. \tag{5.3}$$

Next we define a conditional version of c-function and for a while we talk about splittings — without defining whether they are k-compositions or k-partitions. An intuition behind this function is that the data is already split in k bins and we want to compute a c-function value of new splitting given the present one. Hence, the unconditional c-function can be written as $c((x_1, \ldots, x_k) \mid (n)) = c((x_1, \ldots, x_k))$, which means that originally all data are in the same bin. The *conditional c-function* is

$$c((x_1, \ldots, x_k)|(y_1, \ldots, y_l)) = \sum_{Z \in \mathcal{Z}} \prod_{i=1}^{l} c(z_{1i}, \ldots, z_{ki}), \tag{5.4}$$

where Z is a matrix with marginals $(x_1, \ldots, x_k)$ and $(y_1, \ldots, y_l)$, and the terms z_{ij} are elements of a matrix Z. All elements are positive integer values. Set $\mathcal{Z}$ is the set of those matrices Z which satisfy the given marginals (row and column sums):

$$Z = \begin{matrix} \begin{bmatrix} z_{11} & \cdots & z_{1l} \\ \vdots & \ddots & \vdots \\ z_{k1} & \cdots & z_{kl} \end{bmatrix} & \begin{matrix} x_1 \\ \vdots \\ x_k \end{matrix} \\ \begin{matrix} y_1 & \cdots & y_l \end{matrix} & \end{matrix}$$

The product of c-functions in (5.4) corresponds to one valid path from the present composition to a new composition. Each c-function of the product

corresponds splitting the data from one of the present bins to new bins (bin by bin). The sum outside collects all possible independent paths (Figure 5.1).

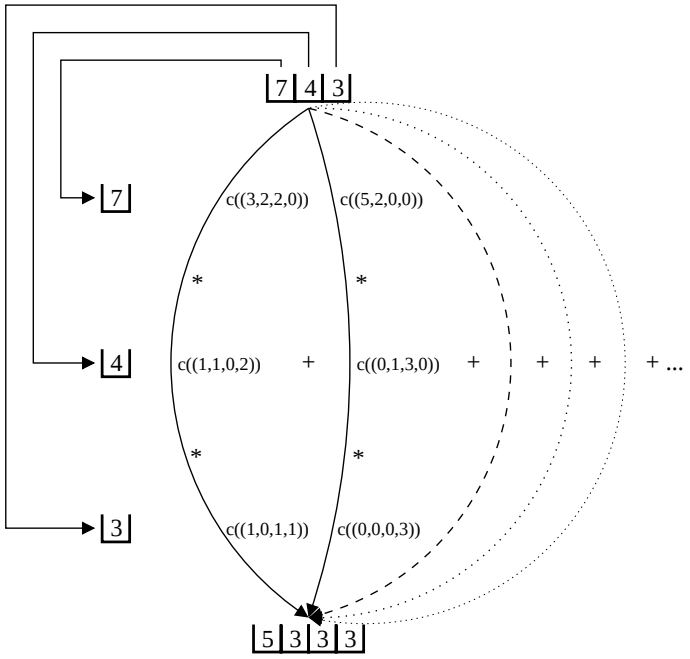

Figure 5.1: *Visualization of (5.4) with a couple of example paths. Data size is 14.*

5.2 Intuition behind the Algorithm

Let us start with an example tree T, whose structure is $(B \leftarrow A \rightarrow C \rightarrow D)$. We also consider first only k-compositions, because they are simpler to handle. The normalizing sum for the given tree T is

$$\mathcal{C}_F(T) = \sum_t \sum_s \sum_u \sum_v c(f_t^A) c(f_u^B | f_t^A) c(f_s^C | f_t^A) c(f_v^D | f_s^C), \qquad (5.5)$$

where f_i^X is the ith k-composition of variable X. Notice that the given notation hides the data size n and the number of bins k. The sums go over all possible sufficient data of each variable — all the k-compositions. The items of the formula compose like probabilities to unconditional and conditional terms. Conditional c-functions are taking the parent node's l-composition and turning it into the target node's k-composition.

We get more efficient computation by rearranging the sum formula above. First we can make the observation that

$$\sum_u c(f_u^B|f_t^A) = \prod_{i=1}^{l} \mathcal{C}_{MN}(k, y_i), \tag{5.6}$$

where y_i is the number of data points in the ith bin of the parent variable A, which has l bins. The result comes from the fact that we have l bins to split and we can do these independently, as we do not have any fixed target k-composition. Therefore the result corresponds to a product of multinomial normalizing sums. We use a shorthand notation $\bigvee f^X = (x_1, \ldots, x_k)_1 \vee \cdots \vee (x_1, \ldots, x_k)_b$, where b is a number of k-compositions of n. The symbol $\bigvee$ means that we accept any valid sufficient statistics for node X. Now (5.6) has the form

$$c(\bigvee f^B|f_t^A). \tag{5.7}$$

Using this we can write

$$\sum_t c(f_t^A)c(\bigvee f^B|f_t^A) \sum_s c(f_s^C|f_t^A)c(\bigvee f^D|f_s^C), \tag{5.8}$$

where s is the index over node A compositions and t is the index over node C compositions. So far we have only used k-compositions, but we can easily say that the sums go over indexes of k-partitions. In this case we have to multiply c-functions by the previously presented m-functions. For example

$$\mathcal{C}_{MN}(k, n) = \sum_i c(f_i^X) = \sum_j m(q_j^X)c(q_j^X), \tag{5.9}$$

where i goes over k-compositions of X and j goes over k-partitions of X. The latter sum of course has less terms. Now (5.8) can be written using the matrix form and the previous modification as $\mathcal{C}_F(T) = R_A(L_B^A \odot (M_C^A L_D^C))$, where $\odot$ is the *term-wise product* (Hadamard product) between matrix elements, R is a horizontal root node vector, M is an inner node matrix and L is a vertical leaf node vector (Paper 5). We shall present these components and operations in the next section.

5.3 Computation and Algorithm

Let X be the name of a node and Y be the name of its parent. Node X has k values and node Y has l values. We also assume that node X has p k-partitions and node Y has d l-partitions. The *root node component* is

$$R_X = \begin{bmatrix} m(q_1^X) \cdot c(q_1^X) & \cdots & m(q_p^X) \cdot c(q_p^X) \end{bmatrix}, \tag{5.10}$$

the *inner node component* is of the form

$$M_X^Y = \begin{bmatrix} m(q_1^X) \cdot c(q_1^X|q_1^Y) & \cdots & m(q_p^X) \cdot c(q_p^X|q_1^Y) \\ \vdots & \ddots & \vdots \\ m(q_1^X) \cdot c(q_1^X|q_d^Y) & \cdots & m(q_p^X) \cdot c(q_p^X|q_d^Y) \end{bmatrix} \tag{5.11}$$

and the *leaf node component* is

$$L_X^Y = \begin{bmatrix} \sum_{i=1}^p m(q_i^X) \cdot c(q_i^X|q_1^Y) \\ \vdots \\ \sum_{i=1}^p m(q_i^X) \cdot c(q_i^X|q_d^Y) \end{bmatrix}. \tag{5.12}$$

The root node component is trivial to compute using the definition. The leaf node component is easy to compute using observations (5.6) and (5.9). The hard part is the inner node component, but we leave further discussion on this topic until the next section.

After the components have been computed for nodes of the given forest, we can compute the normalizing sum of the forest doing simply a matrix computation. The computation is started from the leaf components and continues level-wise until we end up at the root nodes. The operation between siblings is the term-wise product and between a parent and a child the normal matrix multiplication. The operation between root nodes (trees) is also the term-wise product as we can interpret them to be siblings without a parent (Figure 5.3). For example the normalizing sum for the forest S=($B \leftarrow A \rightarrow C \rightarrow D$, $E \rightarrow F$) with two trees, can be written as $\mathcal{C}_F(S) = (R_A(L_B^A \odot (M_C^A L_D^C))) \odot (R_E L_F^E)$. A pseudo-code for the basic algorithm is given as Algorithm 5.2.

As the computation is done from leaves to roots, the heaviest operations are matrix-vector-multiplications, which can be performed in quadratic time. The result of this operation is a vector and therefore also the next operation is at most a matrix-vector-multiplication. Notice that the sizes of the matrices are determined by the number of k- and l-partitions. A real computational obstacle is still the computation time of the inner node matrices — the topic, which we will discuss next.

5.4 Computation of Inner Node Matrices

The definition of the inner node matrix does not really show us how to compute it efficiently. The first observation is that if we remove the m-functions, all the inner node matrices are actually sections of a bigger general matrix,

```
ComputeNormalizingTerm(bayesforest){
   ComputeRootVectors(bayesforest);
   ComputeLeafVectors(bayesforest);
   ComputeInnerMatrices(bayesforest);

   foreach(tree){
      go through all nodes level-by-level starting from leaves{
         case: the node X is a leaf node{
            set corresponding leaf vector L to the leaf node X;
         }
         case: the node X is an inner node{
            V <- take the termwise product of already computed
                 child vectors of the node X;
            W <- multiply corresponding inner matrix M with the product vector V;
            set the result vector W to the inner node X;
         }
         case: the node X is a root node{
            V <- take the termwise product of already computed
                 child vectors of the node X;
            w <- multiply corresponding root vector R with the product vector V;
            set value of the tree normalizing sum (w) to the root node X;
         }
      }
   }
   return(the product of tree normalizing sums (all w values) in root nodes);
}
```

Algorithm 5.2: A pseudo-code for computing the normalizing sum for a forest.

which we call the core inner node matrix (Paper 5). Later we simply say the core matrix.

First we have to fix the right ordering among partitions so that every matrix (and vector) has partitions in the same order. Otherwise sections consist only of arbitrary k-partitions and also multiplication operations compute arbitrary things. We choose the ordering to be the following: first, order k-partitions into blocks with respect to the number of zero bins (starting with the maximum number of zeros). It means that partitions in each block have the same number of zero bins. Then order each block according to the inverse lexicographic ordering.

The *core matrix* has to consist of all terms $c(q_i \mid q_j)$ that are needed for computation at any node in the corresponding forest. We define the general form of the core matrix to be

$$CM = \begin{bmatrix} c(q_1^{\max(X)}|q_1^{\max(Y)}) & \cdots & c(q_P^{\max(X)}|q_1^{\max(Y)}) \\ \vdots & \ddots & \vdots \\ c(q_1^{\max(X)}|q_D^{\max(Y)}) & \cdots & c(q_P^{\max(X)}|q_D^{\max(Y)}) \end{bmatrix}, \tag{5.13}$$

where $q_i^{\max(X)}$ and $q_j^{\max(Y)}$ are $\mathcal{K}$- and $\mathcal{L}$-partitions. Value D is the number

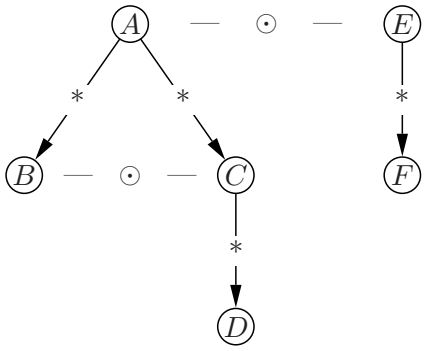

Figure 5.3: The operation between a parent and a child is the ordinary matrix multiplication (*) and between siblings the term-wise product ($\odot$).

of $\mathcal{L}$-partitions, where $\mathcal{L}$ is the maximum number of values that any inner node's parent has in the forest and P is the number of $\mathcal{K}$-partitions, where $\mathcal{K}$ is the maximum number of values any inner node has in the forest. The idea is now to take one by one all the needed sections of this matrix (Figure 5.4) and multiply every matrix element by its corresponding m-function value. In this way every inner node matrix can be achieved efficiently and there is no need to compute the same elements several times. There still remains one question: how to compute the core matrix itself efficiently? For that we need generating polynomials and polytopes as we proposed in Paper 5 and we revise the idea again here.

Generating polynomials that we use have c-functions as coefficients. We define them to be

$$P_k^0 = 1 \quad \text{and} \tag{5.14}$$

$$P_k^u = \sum_{x_1+x_2+\cdots+x_k=u} c((x_1,\ldots,x_k)) z_1^{x_1} z_2^{x_2} \cdots z_k^{x_k}, \tag{5.15}$$

where u is less or equal to the data size. We take the product of these polynomials with respect to some parent node's l-partition $(y_1,\ldots,y_l)$:

$$T_k^{(y_1,\ldots,y_l)} = P_k^{y_1} P_k^{y_2} \cdots P_k^{y_l} \tag{5.16}$$

and by extracting a coefficient with respect to the node's k-partition, we get the desired value of the conditional c-function:

$$c((x_1,\ldots,x_k)|(y_1,...,y_l)) = [z_1^{x_1} \cdots z_k^{x_k}] T_k^{(y_1,\ldots,y_l)}. \tag{5.17}$$

In fact we can read all values of a single core matrix row from the same product polynomial. However, the multiplication process also creates many

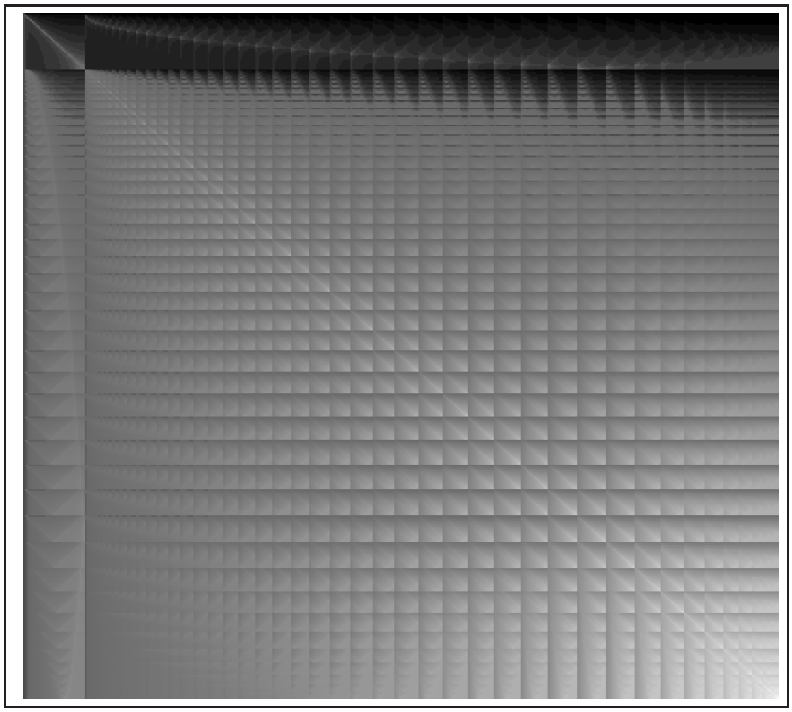

Figure 5.4: *Core matrix with values* $\max(Y) = \max(X) = 3$ *and* $n = 70$ *plotted. Brighter pixel means bigger value, but for structural visibility purposes the picture has been made brighter using image processing. Different regions are clearly visible: 2x2-partitions in the upper left corner, 2x3- and 3x2-partitions on up right and on the left and 3x3 is the big area int the bottom right corner. Areas 1x1, 1x2 and 2x1 are not visible in the picture, because they are only one pixel wide.*

terms that do not correspond to any desired value. We can reduce the number of nuisance terms by truncating the multiplication process by using restricting polytopes.

The first step in the polytope description is to observe that although we have k-parameters that are $(x_1, \ldots, x_k)$, we only need $k-1$ parameters, because every k-composition sums into the same value — namely n. One parameter is therefore irrelevant and we need only $k-1$ parameters to represent each of our k-compositions. After this we map $c(x_1, \ldots, x_k)$ to the coordinate $(x_2, \ldots, x_k)$. Thus we omit the biggest term x_1, in k-partition representation, to make our term space as small as possible during the multiplication process.

Restricting polytopes consists of two different kinds of inequalities. The

first type is

$$0 \leq x_i \leq \left\lfloor \frac{n}{i} \right\rfloor, \tag{5.18}$$

where i goes from 2 to k. This defines a restricting hyper-rectangle that consists of all required terms, because two-sided inequalities define the biggest possible number of data points that each bin can have in the k-partition representation.

The *second type* of inequalities describe additional restrictions, which are also caused by the fact that according to our representation, bin values are obeying the decreasing order. They describe situations where several bins have the same number of data points. All the inequalities are achieved by finding valid splittings between bins of k-partition and multiplying them by a number of x_i:s in each group. For example, 4-partitions have three different splittings $\{x_1x_2|x_3x_4, x_1x_2|x_3|x_4,\ x_1x_2x_3|x_4\}$, where vertical bars mean borders between different groups. Notice that as the count x_1 is redundant, there cannot be a split between x_1 and x_2, so x_1 and x_2 are always in the same group. We get the following three inequalities:

$$2x_2 + 2x_4 \leq n,\ 2x_2 + x_3 + x_4 \leq n,\ 3x_3 + x_4 \leq n. \tag{5.19}$$

Together all these inequalities define the wanted polytope. All the points of the integer lattice are needed and none of the points outside the lattice.

The only remaining question is how to do the multiplication. Let us do multiplication using two polytopes $\mathcal{P}_1$ and $\mathcal{P}_2$. The polytope formed in this process is a result polytope and denoted by $\mathcal{P}$. We get the value of the given result polytope lattice point $(v_1, \ldots, v_r)$, where $r = k - 1$, by computing

$$\mathcal{P}(v_1, \ldots, v_r) = \sum_{w_1=0}^{v_1} \cdots \sum_{w_r=0}^{v_r} \mathcal{P}_1(w_1, \ldots, w_r) \cdot \mathcal{P}_2(v_1 - w_1, \ldots, v_r - w_r), \tag{5.20}$$

which is naturally a higher order discrete convolution formula. The formula is used for all the points inside the corresponding polytopes. The terms outside are just ignored and therefore equal to zero.

The final big modification to make computation more efficient is to take advantage of symmetries. This part was not described in Paper 5 due to lack of space. Our integer lattice points are actually k-compositions, not only k-partitions. Now changing the order of bins gives us the same conditional c-function values. Thus inside the polytope there are many equal values on different sides of the symmetry axes. For example in the 3-partitions

case we have a symmetry axis $x_2 = x_3$ and for example lattice points $(2, 1)$ and $(1, 2)$ have the same c-function value (Figure 5.5). The algorithm can utilize these axes so that if some of these points are already computed, the algorithm does not compute the same result again using (5.20).

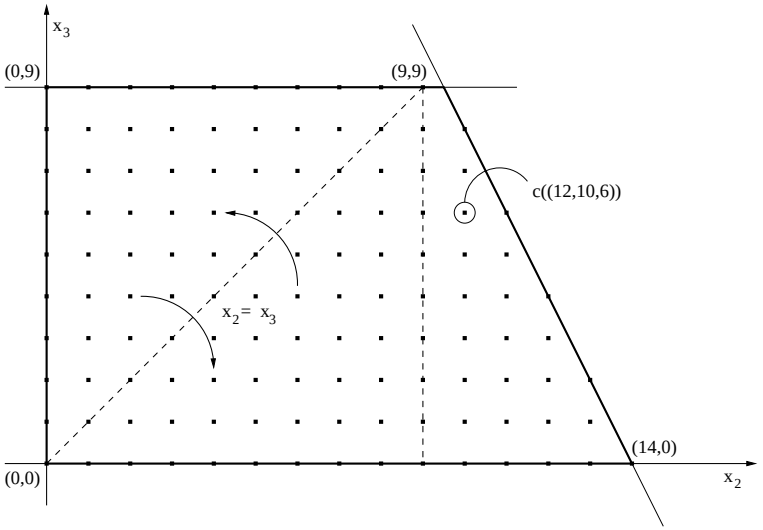

Figure 5.5: *An example of a symmetry inside the polytope, where we have three bins and 28 data vectors.*

Even with these enhancements the algorithm has no use in most real-life cases. A rough upper-bound approximation of efficiency says that the number of basic operations (ordinary sums and products) is $\mathcal{O}(n^{2\mathcal{K}+\mathcal{L}-3} + Hn^{\mathcal{K}+\mathcal{L}-2})$, where H is the number of inner nodes in the forest, $\mathcal{K}$ is the maximum number of values that any inner node has and $\mathcal{L}$ is the maximum number of values that any inner node's parent has. Probably a better way to compute the normalizing sum for more complicated structures is to use a Monte Carlo simulation as suggested in paper [39]. However, the proposed floating-point algorithm is useful in the binary case with time complexity $\mathcal{O}(n^3)$, because MC simulation cannot give results with full floating-point precision. Another possible use for this proposed algorithm is to verify the correctness of other simulation algorithms developed in the future.

Our final observation is that a recurrence analogous to (4.7) for the Naive Bayes normalizing sum based on rational generating functions seem to work also for the leaf variables. Our implementation does not allow us to test the recurrence in a case of root or inner node variables. This observation raises open questions and may eventually lead to the development of more useful algorithms.

Chapter 6

Conclusions

In this Thesis we have presented methods for computing the normalization sums of the normalized maximum likelihood (NML) in the case of simple Bayesian network models — single multinomial, Naive Bayes and Bayesian Forest models. Without efficient computation methods of normalizing sums, we cannot use NML-based model selection in practical applications. Several case studies have shown that the NML criterion chooses good models even with small data sets. If the fundamental information-theoretic base is accepted, the criterion can be seen to give an objective method, without any subjective parameters, for model selection.

We presented how to compute the normalizing sums for the multinomial and Naive Bayes models using Maple. For the multinomial normalizing sum we developed a fast fixed precision sub-linear algorithm for floating-point computation. For the Naive Bayes normalizing sums we defined a computational framework based on basic series and exponentiation of it. For the normalizing sums of Bayesian tree models we presented a method that uses matrix components. If the core matrix for a given data size is computed and stored, model search can be done relatively fast, because the problem of computing the normalizing sum factorizes efficiently to matrix components. The main task is then to do ordinary matrix computation, unfortunately, with huge matrices. We also developed an algorithm for core matrix computation that is using generating polynomials and polytopes.

This Thesis gives some new directions for future work. The most important subject of research is of course actual the performance of the stochastic complexity in model selection with real life-data sets. These tests can now be performed also with Bayesian trees. We also initiated the discussion on fixed precision computation in the most simple case. This framework can also be expanded for more complex models. Related to this, a very important question that we did not answer is, what is the optimal precision given

data size and a multinomial model? Optimal in this case means the smallest possible precision that gives the correct answer by the NML criterion. A very promising direction for efficient computation of normalizing sums is Monte Carlo simulation and the previous question about optimal precision is highly relevant also in this case. Another open issue is how to expand the matrix component framework for general Bayesian networks. This may not be interesting for application purposes, but it gives more information about the problem and also gives correct answers that can be compared with the results given by approximation and simulation methods. A more elaborate analysis could be done also for core matrices: is there some good approximation for the values of the elements or can they be computed even more efficiently? Finally, the initial questions that inspired also this research concern the problem of finding the generating functions and efficient recurrence formulas for more complex probabilistic graphical models.

References

[1] M. Abramowitz and I.A. Stegun, editors. *Handbook of Mathematical Functions*. Dover Publications, Inc., New York, 1970.

[2] G.B. Arfken and H.J. Weber. *Mathematical Methods for Physicists*. Academic Press, 4th edition, 1995.

[3] V. Balasubramanian. MDL, Bayesian inference, and the geometry of the space of probability distributions. In P. Grünwald, I.J. Myung, and M. Pitt, editors, *Advances in Minimum Description Length: Theory and Applications*, pages 81–98. The MIT Press, 2006.

[4] C. Banderier, M. Bousquet-Mélou, A. Denise, P. Flajolet, D. Gardy, and D. Gouyou-Beauchamps. Generating functions for generating trees. *Discrete Mathematics*, 246(1-3):29–55, March 2002.

[5] W. Buntine. Theory refinement on Bayesian networks. In B. D'Ambrosio, P. Smets, and P. Bonissone, editors, *Proceedings of the Seventh Conference on Uncertainty in Artificial Intelligence*, pages 52–60. Morgan Kaufmann Publishers, 1991.

[6] R.M. Corless, G.H. Gonnet, D.E.G. Hare, D.J. Jeffrey, and D.E. Knuth. On the Lambert W function. *Advances in Computational Mathematics*, 5:329–359, 1996.

[7] T. Cover and J. Thomas. *Elements of Information Theory*. John Wiley & Sons, New York, NY, 1991.

[8] A. Di Bucchianico. Introduction to umbral calculus. 1998. Lecture notes. Unpublished.

[9] E. Dudewicz and S. Mishra. *Modern Mathematical Statistics*. John Wiley & Sons, 1988.

[10] W. Feller. *An Introduction to Probability Theory and Its Applications*. John Wiley & Sons, 3rd edition, 1968.

[11] P. Flajolet and R. Sedgewick. *Analytic Combinatorics*. Cambridge University Press, 2009.

[12] R.L. Graham, D.E. Knuth, and O. Patashnik. *Concrete Mathematics (second edition)*. Addison-Wesley, 1994.

[13] P. Grünwald. *The Minimum Description Length Principle*. MIT Press, 2007.

[14] D. Heckerman, D. Geiger, and D.M. Chickering. Learning Bayesian networks: The combination of knowledge and statistical data. *Machine Learning*, 20(3):197–243, September 1995.

[15] P. Henrici. Automatic computations with power series. *Journal of the ACM*, 3(1):11–15, January 1956.

[16] S. Janson, D.E. Knuth, T. Uczak, and B. Pittel. The birth of the giant component. *Random Structures and Algorithms*, 4(3):233–358, 1993.

[17] F. Jensen. *An Introduction to Bayesian Networks*. UCL Press, London, 1996.

[18] D.E. Knuth. *The Art of Computer Programming, vol. 2 / Seminumerical Algorithms (third edition)*. Addison-Wesley, 1998.

[19] D.E. Knuth and B. Pittel. A recurrence related to trees. *Proceedings of the American Mathematical Society*, 105(2):335–349, 1989.

[20] P. Kontkanen and P. Myllymäki. A linear-time algorithm for computing the multinomial stochastic complexity. *Information Processing Letters*, 103(6):227–233, 2007.

[21] P. Kontkanen and P. Myllymäki. MDL histogram density estimation. In M. Meila and S. Shen, editors, *Proceedings of the Eleventh International Conference on Artificial Intelligence and Statistics*, March 2007.

[22] P. Kontkanen, P. Myllymäki, W. Buntine, J. Rissanen, and H. Tirri. An MDL framework for data clustering. In P. Grünwald, I.J. Myung, and M. Pitt, editors, *Advances in Minimum Description Length: Theory and Applications*. The MIT Press, 2006.

[23] P. Kontkanen, H. Wettig, and P. Myllymäki. NML computation algorithms for tree-structured multinomial Bayesian networks. *EURASIP Journal on Bioinformatics and Systems Biology*, 2007.

[24] P. Kontkanen, H. Wettig, and P. Myllymäki. Nml computation algoritms for tree-structured multinomial Bayesian networks. *EURASIP Journal on Bioinformatics and Systems Biology*, 2007.

[25] G. Korodi and I. Tabus. An efficient normalized maximum likelihood algorithm for DNA sequence compression. *ACM Trans. Inf. Syst.*, 23(1):3–34, 2005.

[26] G. Korodi and I. Tabus. Normalized maximum likelihood model of order-1 for the compression of dna sequences. In *Proceedings of the 2007 Data Compression Conference*, Snowbird, Utah, USA, March 2007.

[27] M. Li and P. Vitanyi. *An Introduction to Kolmogorov Complexity and Its Applications*. Springer Verlag, 1997.

[28] L. Lipshitz. D-finite power series. *Journal of Algebra*, 122:353–373, 1989.

[29] A. Menezes, P. van Oorschot, and S. Vanstone. *Handbook of Applied Cryptography*. CRC Press LLC, 1997.

[30] A. Orlitsky, N. Santhanam, K. Viswanathan, and J. Zhang. On modeling profiles instead of values. In *Proceedings of the 20th Conference in Uncertainty in Artificial Intelligence*, 2004.

[31] J. Pearl. *Causality: Models, Reasoning and Inference*. Cambridge University Press, 2000.

[32] M. Petkovsek, H. S. Wilf, and D. Zeilberger. *A=B*. AK Peters, Ltd., 1996.

[33] J. Rissanen. Modeling by shortest data description. *Automatica*, 14:445–471, 1978.

[34] J. Rissanen. Stochastic complexity. *Journal of the Royal Statistical Society*, 49(3):223–239 and 252–265, 1987.

[35] J. Rissanen. Fisher information and stochastic complexity. *IEEE Transactions on Information Theory*, 42(1):40–47, January 1996.

[36] J. Rissanen. *Information and Complexity in Statistical Modeling*. Springer, 2007.

[37] S. Roman. *The Umbral Calculus*. Dover, 2005.

[38] T. Roos. *Statistical and Information-Theoretic Methods for Data-Analysis*. PhD thesis, Report A-2007-4, Department of Computer Science, University of Helsinki, 2007.

[39] T. Roos. Monte carlo estimation of minimax regret with an application to mdl model selection. In *Proceedings of the IEEE Information Theory Workshop*, Porto, Portugal, May 2008.

[40] F. Ruskey. *Combinatorial Generation*. Unpublished.

[41] Yu.M. Shtarkov. Universal sequential coding of single messages. *Problems of Information Transmission*, 23:3–17, 1987.

[42] T. Silander, P. Kontkanen, and P. Myllymäki. On sensitivity of the MAP Bayesian network structure to the equivalent sample size parameter. In R. Parr and L. van der Gaag, editors, *Proceedings of the 23rd Conference on Uncertainty in Artificial Intelligence*, pages 360–367. AUAI Press, 2007.

[43] T. Silander, T. Roos, P. Kontkanen, and P. Myllymäki. Factorized normalized maximum likelihood criterion for learning bayesian network structures. In *Proceedings of the 4th European Workshop on Probabilistic Graphical Models (PGM-08)*, pages 257–264, Hirtshals, Denmark, 2008.

[44] H. Steck. Learning the bayesian network structure: Dirichlet prior vs. data. In *Proceedings of the 24th Conference on Uncertainty in Artificial Intelligence (UAI)*, July 2008.

[45] W. Szpankowski. *Average case analysis of algorithms on sequences*. John Wiley & Sons, 2001.

[46] I. Tabus and G. Korodi. Genome compression using normalized maximum likelihood models for constrained markov sources. In *Proceedings of the 2008 IEEE Information Theory Workshop*, Porto, Portugal, May 2008.

[47] J. Wimp. *Computation with Recurrence Relations*. Pitman Publishing Ltd., 1984.

[48] G. M. Ziegler. *Lectures on Polytopes*. Springer, 2007.

Corrections

Paper 1

Equation (3): $SC(\mathbf{x}^n \mid \mathcal{M}) = -\log\ldots$

Page 20, row 9: ... equivalent to $n^n\mathcal{C}(L,n)$ by...

Page 20, 2nd col, row 8: ... the solutions of these new confluent hypergeometric equations can be defined using *confluent* ...

Page 20, 2nd col, Proof of Theorem 3: ... all the extra sum ...

Paper 3

Equation (15): $p_0(i)a(i) + p_1(i)a(i+1) + \cdots + p_r(i)a(i+r) = 0$, $i, r \in \mathbb{N}$,

Page 285, 2nd col, row 3: $g(x_1,\ldots,x_m) = \sum_{i_1,\cdots,i_m} a(i_1,\ldots,i_m)x_1^{i_1}\cdots x_m^{i_m}$

Page 285, 2nd col, row 7: $g^{1,\ldots,s}_{i_{s+1},\ldots,i_m}(x_1,\ldots,x_s) = \sum_{i_1,\ldots,i_s} a(i_1,\ldots,i_m)x_1^{i_1}\cdots x_s^{i_s}$

Proof of Theorem 2: $f_1^1(z) = \sum_{n=0}^{\infty}\mathcal{C}(1,n)n^n\frac{z^n}{n!} = \cdots \qquad (x_1 = z,\ x_2 = u)$

Theorem 3: $\sum_{l=0}^{r_2} p_l(L,n)\ \mathcal{C}(L,n-l)\frac{(n-l)^{n-l}}{(n-l)!} = 0$

Proof of Theorem 3: ... sequence $\mathcal{C}(L,n)\frac{n^n}{n!}$ is ...

Correction of an consequence of Theorem 3: For the vertical family there cannot be a homogeneus linear recurrence equation, but this in fact does **not** prove that the same applies also for the sequence of $\mathcal{C}(L,n)$ over the variable n (The wrong consequence is mentioned in Abstract and Conclusions).

Paper 1

Tommi Mononen and Petri Myllymäki:

On the Multinomial Stochastic Complexity and its Connection to the Birthday Problem

In *Proceedings of the International Conference on Information Theory and Statistical Learning*, ITSL'08 (Las Vegas, Nevada, USA), pages 17-22, CSREA Press, 2008.

On the Multinomial Stochastic Complexity and its Connection to the Birthday Problem

Tommi Mononen
Helsinki Institute for Information Technology
Helsinki, Finland
tommi.mononen@hiit.fi

Petri Myllymäki
Helsinki Institute for Information Technology
Helsinki, Finland
petri.myllymaki@hiit.fi

***Abstract*—The *Minimum Description Length* (MDL) is an information-theoretic principle that can be used for model selection and other statistical inference tasks. A central concept in this framework is *stochastic complexity*, defined nowadays for a given parametric model class via the *Normalized Maximum Likelihood* (NML) distribution. In this paper we focus on the parametric model class of a single multinomial variable, as this case forms a very important building block for more complex models. We show that the computationally demanding normalization term of the multinomial NML can be written in a simple and effective form by using tools of umbral calculus. The time complexity of computing the exact form is $\mathcal{O}(n)$, where n is the number of data points. We also give two different descriptions for the normalization term using sets of confluent hypergeometric functions, show an interesting connection between the birthday problem and our problem, and demonstrate how the results can be exploited in practice.**

***Index Terms*—Minimum description length, normalized maximum likelihood, birthday problem, umbral calculus, confluent hypergeometric function.**

I. Introduction

Stochastic complexity (SC) is an information-theoretic model selection criterion, which can be seen as a theoretical instantiation of the minimum description length (MDL) principle [15], [7]. Intuitively speaking, the basic idea is that the best model for the data is the one which results in the shortest description for the data together with the model. There are many proposed ways to define the stochastic complexity formally; one theoretically solid way is to use the normalized maximum likelihood (NML) distribution.

In the following, let $\mathcal{M}$ denote a parametric model, $\hat{\theta}(\mathbf{x}^n)$ the maximum likelihood parameters of the model and $\mathbf{x}^n$ a matrix of observations (from a discrete alphabet). The NML distribution is defined as

$$P_{NML}(\mathbf{x}^n \mid \mathcal{M}) = \frac{P(\mathbf{x}^n \mid \hat{\theta}(\mathbf{x}^n), \mathcal{M})}{\sum_{\mathbf{y}^n} P(\mathbf{y}^n \mid \hat{\theta}(\mathbf{y}^n), \mathcal{M})} \tag{1}$$

$$= \frac{P(\mathbf{x}^n \mid \hat{\theta}(\mathbf{x}^n), \mathcal{M})}{\mathcal{C}(\mathcal{M}, n)}, \tag{2}$$

where in the numerator we have the maximum likelihood of our observed data and in the denominator we have the sum of maximum likelihoods over all the discrete data sets of the same size [19] (denoted in the sequel by $\mathcal{C}(\mathcal{M}, n)$).

One way to define the stochastic complexity is to take negative logarithm of (2). This gives us the formula

$$SC(\mathcal{M}|\mathbf{x}^n) = -\log \frac{P(\mathbf{x}^n \mid \hat{\theta}(\mathbf{x}^n), \mathcal{M})}{\mathcal{C}(\mathcal{M}, n)}. \tag{3}$$

The basic model selection task is to compute the value of this model selection criterion for parametric models of different complexity and choose the one for which this value is minimized given the observed data.

The single multinomial variable model is an important building block for building more complex models for discrete data (see e.g. [13], [17]). For computing the NML in this case, we have to be able to calculate the numerator and the denominator of (2) in the multinomial case. In the following, we simplify the notation and leave out $\mathcal{M}$: the model is implicitly defined by the number of values of the multinomial variable, denoted by L. The numerator is now

$$P(x^n \mid \hat{\theta}(x^n), L) = \prod_{k=1}^{L} \left(\frac{h_k}{n}\right)^{h_k}, \tag{4}$$

where h_k is a number of data points assigned to the kth value. The denominator is

$$\mathcal{C}(L, n) = \sum_{h_1+\cdots+h_L=n} \frac{n!}{h_1! \cdots h_L!} \prod_{k=1}^{L} \left(\frac{h_k}{n}\right)^{h_k}, \tag{5}$$

which is the sum of maximum likelihoods and summation goes over every possible data of length n. The time complexity of computing the denominator using the definition is $\mathcal{O}(n^{L-1})$, which is a very heavy operation. However, there exists a so called exponential generating function (EGF) whose coefficients represent every normalizing term given the number of outcomes L. The generating function is

$$\sum_{n=0}^{\infty} \mathcal{C}(L, n) n^n \frac{z^n}{n!} = \left(\sum_{n=0}^{\infty} n^n \frac{z^n}{n!}\right)^L = \left(\frac{1}{1 - T(z)}\right)^L, \tag{6}$$

where $T(z)$ is called a tree function [9], [8]. This function has a close relation to Lambert's W-function, given by $T(z) = -W(-z)$. The W-function has numerous applications in mathematics and physics [3]. The generating function has been used for proving an recurrence formula, which can be

used to compute the normalizing terms efficiently [12]:

$$\mathcal{C}(1,n) = 1, \tag{7}$$

$$\mathcal{C}(2,n) = \sum_{k=0}^{n} \binom{n}{k} \left(\frac{k}{n}\right)^k \left(\frac{n-k}{n}\right)^{n-k} \quad \text{and} \tag{8}$$

$$\mathcal{C}(L,n) = \mathcal{C}(L-1,n) + \left(\frac{n}{L-2}\right) \mathcal{C}(L-2,n). \tag{9}$$

The time complexity for computing one normalizing term is therefore $\mathcal{O}(n+L)$.

The main target of this paper is to show that for the multinomial normalizing term there is also a direct and compact representation which can be directly used with existing mathematical software packages. What is more, using this new form we can reduce in the floating point computations the time complexity to $\mathcal{O}(n)$. We also show that the classic *birthday problem* [4] has a very close connection to the NML, and therefore any method used in solving one of the two problems can be transferred easily to the other problem. The birthday problem itself has many application areas such as random mappings and population estimation [14].

Although our final form for the multinomial normalizing term is quite simple, we need quite heavy mathematical tools to prove the result. We start by introducing the needed basic tools of umbral calculus.

II. Umbral Calculus

Umbral calculus gives us a collection of efficient tools for solving certain kind of combinatorial problems. We are not giving a general introduction to umbral calculus, but we are just introducing tools, that are relevant in the derivation of our results. We also omit the mathematically more neater operator notation [18] and use Roman's notation [16], because we find it more accessible.

If we have polynomials of binomial or convolution type, we can derive seemingly complicated results using general tools of umbral calculus. For example Donald Knuth kind of rediscovered the umbral calculus in his hands-on article [8]. The next definition gives us the class of problems for which we can apply these tools:

Definition 1 A sequence $(s_n(x)) = (s_0(x), s_1(x), \dots)$ is called Sheffer sequence iff its generating function has the form

$\sum_{k=0}^{\infty} s_k(x) \frac{z^k}{k!} = A(z) e^{x\overline{B}(z)}$,

where
$A(z) = a_0 + a_1 z + a_2 z^2 + \cdots$,
$\overline{B}(z) = b_1 z + b_2 z^2 + \cdots$
are formal power series and coefficient $a_0 \neq 0$ and coefficient $b_1 \neq 0$.

Luckily our generating function is even simpler and therefore we can use a subclass of Sheffer sequences called associated sequences.

Definition 2 If $A(z) = 1$ we say that Sheffer sequence is associated sequence. Now $(s_n(x))$ is associated to the inverse function of $\overline{B}(z)$, which we call functional and denote by $B(t)$.

We show two examples to clarify our basic concepts. We present the rising factorial polynomials and Abel polynomials [16].

We denote the *rising factorial polynomial* by

$$\left(\frac{x}{a}\right)^{\overline{k}} = \left(\frac{x}{a}\right)\left(\frac{x}{a}+1\right)\cdots\left(\frac{x}{a}+k-1\right). \tag{10}$$

We will later need only the rising factorial polynomials with $a = 1$, so we restrict our example to this special case. The generating function for these rising factorial polynomials is

$$\sum_{k=0}^{\infty} p_k(x) \frac{z^k}{k!} = \sum_{k=0}^{\infty} x^{\overline{k}} \frac{z^k}{k!} = \left(\frac{1}{1-z}\right)^x \tag{11}$$

$$= e^{x(-\ln(1-z))} = e^{x\overline{R}(z)}, \tag{12}$$

where sequence of $p_n(x) = x^{\overline{n}}$ is called associated sequence, because the generating function is in that form. The sequence of $p_n(x)$ is associated to the backward difference functional $R(t) = 1 - e^{-t}$. We get this functional by computing the inverse function of $\overline{R}(z) = -\ln(1-z)$.

Our second polynomial family is the *Abel polynomials*. They are of the form

$$x(x-bn)^{n-1}, \tag{13}$$

where b is a non-zero constant. Again we make restriction $b = -1$, because we need this case later. The generating function of these restricted polynomials is

$$\sum_{k=0}^{\infty} q_k(x) \frac{z^k}{k!} = \sum_{k=0}^{\infty} x(x+k)^{k-1} \frac{z^k}{k!} = \tag{14}$$

$$e^{x(-W(-z))} = e^{x\overline{S}(z)}, \tag{15}$$

where the sequence of $q_n(x) = x(x+n)^{n-1}$ is the associated sequence type and $W(z)$ is previously mentioned Lambert's W-function. The associated sequence $(q_n(x))$ is associated to Abel functional $S(t) = te^{-t}$, which is the inverse function of $\overline{S}(z) = -W(-z) = T(z)$.

We will need both of these polynomial classes later in proving the theorem 1. The following proposition shows us what kind of sequence we get by taking a composition of two generating functions of associated sequence type.

Proposition 1 (Umbral composition)

$(p_n(x))$ is associated to $M(t)$ and
$(q_n(x))$ is associated to $N(t)$ then

$(\sum_{k=0}^{n} q_{n,k} p_k(x))$ is associated to $N(M(t))$,
where $q_n(x) = \sum_{k=0}^{n} q_{n,k} x^k$.

As the proposition says, we have to expand each coefficient of the outer series as a finite sum from 0 to n to achieve the composition series.

III. Multinomial Normalizing Terms

Now we are ready to prove two theorems. The first one is for connecting existing results to our problem and the second one gives a neater but equivalent form, which we prefer to use later on. The result of the first theorem has been mentioned (without a formal proof) also in [8], [9].

Theorem 1 The multinomial normalizing term has a formula

$$\mathcal{C}(L,n) = \sum_{k=0}^{n-1} \binom{n-1}{k} L^{\overline{k+1}} n^{n-1-k},$$

where $L^{\overline{k+1}} = L(L+1)\cdots(L+k)$ is a rising factorial and $n \geq 1$, $L \geq 1$ and $n, L \in \mathbb{N}$.

Proof: First we have to do some standard mathematical operations for the generating function to get it right form:

$$\left(\frac{1}{1-T(z)}\right)^L = e^{L\ln\left(\frac{1}{1-T(z)}\right)}.$$

From this form we can see that there must be an associated sequence, because $A(z) = 1$ and the generating function is otherwise in the sought form. We denote the inverse of the functional by $\overline{F}(z) = \ln\left(\frac{1}{1-T(z)}\right)$. Now we have to find out the functional by inverting this function:

$$\begin{aligned} t &= \ln\left(\frac{1}{1-T(z)}\right) \\ e^t &= \frac{1}{1-T(z)} \\ T(z) &= 1 - e^{-t} \\ z &= (1-e^{-t})e^{-(1-e^{-t})}. \end{aligned}$$

We notice that $F(t)$ is the composite functional:

$$F(t) = S(R(t)) = (1-e^{-t})e^{-(1-e^{-t})},$$

where

$$R(t) = 1 - e^{-t} \quad \text{and} \quad S(t) = te^{-t}.$$

These functionals we already discussed: $R(t)$ is the backward difference functional, which is associated to rising factorial polynomials $p_n(x) = x^{\overline{n}}$. Functional $S(t)$ is Abel functional, which is associated to Abel polynomials $q_n(x) = x(x+n)^{n-1}$. We can use the umbral composition to get the sequence associated to $F(t)$.

First we have to figure out the finite sum of $q_{n,k}$ terms, which produces $q_n(x)$ term. We can do this by relaying on the binomial theorem [6]:

$$\begin{aligned} q_n(x) = x(x+n)^{n-1} &= x\sum_{k=0}^{n-1}\binom{n-1}{k}x^k n^{n-1-k} \\ &= \sum_{k=0}^{n-1}\binom{n-1}{k}n^{n-1-k}x^{k+1} = \sum_{k=0}^{n-1} q_{n,k+1}x^{k+1}. \end{aligned}$$

As we now know the $q_{n,k}$-terms, we can use the umbral composition. However, first notice that $q_{n,0} = [x^0]q_n(x) = 0$, because there is no constant term in the sum. Let us use the umbral composition:

$$\begin{aligned} \sum_{k=0}^{n} q_{n,k}p_k(L) = \sum_{k=1}^{n} q_{n,k}p_k(L) &= \sum_{k=0}^{n-1} q_{n,k+1}p_{k+1}(L) \\ &= \sum_{k=0}^{n-1}\binom{n-1}{k}n^{n-1-k}L^{\overline{k+1}} = n^n\mathcal{C}(L,n). \end{aligned}$$

Finally we divide the both sides of last equality by n^n. ■

To prove our second theorem, we need first to introduce the Lagrange inversion formula. There are many different versions of this formula, but we prefer the third one presented in [16].

Proposition 2 The Lagrange inversion formula is

$$[z^n]G(\overline{H}(z)) = [z^n]G(z)H'(z)\left(\frac{H(z)}{z}\right)^{-n-1},$$

where G(z) is any formal power series and H(z) is any formal power series with the zero constant term.

By using this formula we can now easily achieve the desired nice form:

Theorem 2 The multinomial normalizing term can be written as

$$\mathcal{C}(L,n) = \sum_{k=0}^{n}\binom{n}{k}\frac{(L-1)^{\overline{k}}}{n^k},$$

where $n \geq 1$, $L \geq 1$ and $n, L \in \mathbb{N}$.

Proof: Although the two forms do not look so different, it might be the case that it is easier to prove also this second form starting from the generating function. And this we will do. We want to apply the Lagrange inversion formula, so we start by looking the generating function:

$$[z^n]\left(\frac{1}{1-T(z)}\right)^L = [z^n]\left(\frac{1}{1-\overline{H}(z)}\right)^L = [z^n]G(\overline{H}(z)),$$

thus we get

$$G(z) = \left(\frac{1}{1-z}\right)^L \quad \text{and} \quad H(z) = \frac{z}{e^z}.$$

The function $H(z)$ is the inverse of the tree function. Now we can apply the Lagrange inversion formula:

$$\begin{aligned} [z^n]\left(\frac{1}{1-T(z)}\right)^L &= [z^n]G(z)H'(z)\left(\frac{H(z)}{z}\right)^{-n-1} \\ &= [z^n]\left(\frac{1}{1-z}\right)^L e^{-z}(1-z)\left(\frac{z}{ze^z}\right)^{-n-1} \\ &= [z^n]\left(\frac{1}{1-z}\right)^{L-1} e^{nz}. \end{aligned}$$

We notice that now we have a product of two functions and we can separately look at their formal power series:

$$\left(\frac{1}{1-z}\right)^{L-1} = \sum_{k=0}^{\infty} (L-1)^{\overline{k}} \frac{z^k}{k!} = \sum_{k=0}^{\infty} p_k(L-1)\ \frac{z^k}{k!} \quad \text{and}$$

$$e^{nz} = \sum_{k=0}^{\infty} n^k \frac{z^k}{k!} = \sum_{k=0}^{\infty} v_k(n) \frac{z^k}{k!}.$$

The final step is just to apply the binomial convolution [16] to get the nth coefficient of (16):

$$\sum_{k=0}^{n} \binom{n}{k} p_k(L-1)\ v_{n-k}(n) = \sum_{k=0}^{n} \binom{n}{k} (L-1)^{\overline{k}} n^{n-k}$$

$$= n^n \sum_{k=0}^{n} \binom{n}{k} \frac{(L-1)^{\overline{k}}}{n^k},$$

which is of course equivalent to $n^n \mathcal{C}(L,k)$ by (6). ∎

When we use the falling factorial polynomials

$$x^{\underline{k}} = x(x-1)\cdots(x-k+1), \tag{16}$$

together with the result of the previous theorem, we get

$$\sum_{k=0}^{n} \left(\frac{n^{\underline{k}}}{n^k} \cdot \frac{(L-1)^{\overline{k}}}{k!} \right). \tag{17}$$

It is easy to see that we can meet the time complexity $\mathcal{O}(n)$ without any approximation steps. In the case $L = 2$, terms of this sum are in the Ramanujan Q-function form [11], [5].

In the next chapter we show a connection between these two forms and confluent hypergeometric functions. The description via hypergeometric functions also supports the claim that the form of Theorem 2 is neater.

IV. Normalizing Terms via Hypergeometric Functions

In the previous section we showed how to compute the normalizing term of a multinomial variable. We gave two different looking, but equal formulas. Now we show that these formulas can be presented via generalized hypergeometric functions. The motivation for doing this is that most mathematical software packages have built-in libraries for handling this kind of functions. These hypergeometric forms are also easier to remember and they are also more convenient when constructing more complex formulas.

A generalized hypergeometric function is of form

$$\sum_{k=0}^{\infty} c_k z^k = {}_pF_q\left(\begin{matrix} a_1, a_2, \cdots, a_p \\ b_1, b_2, \cdots, b_q \end{matrix} \middle| z \right) = \sum_{k=0}^{\infty} \frac{{a_1}^{\overline{k}} {a_2}^{\overline{k}} \cdots {a_p}^{\overline{k}}}{{b_1}^{\overline{k}} {b_2}^{\overline{k}} \cdots {b_q}^{\overline{k}}} \frac{z^k}{k!},$$

where a_i:s and b_j:s are complex numbers [6]. Constant p is the number of rising factorial terms in the numerator and q is the number of rising factorial terms in the denominator. The notation ${}_pF_q$ indicates the general structure of a given generalized hypergeometric function. These functions have a nice property that coefficients of successive terms of a hypergeometric series can be defined as a ratio. This ratio is

$$\frac{c_{k+1}}{c_k} = \frac{(k+a_1)(k+a_2)\cdots(k+a_p)}{(k+b_1)(k+b_2)\cdots(k+b_q)(k+1)}.$$

The solutions of the hypergeometric differential equation are hypergeometric functions ${}_2F_1$. If the hypergeometric differential equation is degenerated so that two singularities are merged together, the solutions of this new confluent hypergeometric equation are *confluent hypergeometric functions* of type ${}_1F_1$ and ${}_2F_0$ [2].

Our target is to show that each multinomial normalizing term can be written as a confluent hypergeometric function ${}_2F_0$ evaluated in a certain point, but first we need to look at two well known rising factorial identities.

Proposition 3 The rising factorial of a negative integer number $(-x)$ can be written as $(-x)^{\overline{k}} = (-1)^k (x-k+1)^{\overline{k}}$.

Proposition 4 The rising factorial of a positive integer number x can be written using factorials as $x^{\overline{k}} = \frac{(x+k-1)!}{(x-1)!}$.

We can now prove the connection between the NML normalizing term and confluent hypergeometric functions. First we convert the result in Theorem 1 into hypergeometric form by using the previous propositions and other basic operations.

Theorem 3 The first form represented via a confluent hypergeometric function is $\mathcal{C}(L,n) = \frac{L}{n} {}_2F_0\left(\begin{matrix} L+1, -n+1 \\ - \end{matrix} \middle| -\frac{1}{n} \right)$.

Proof:

$$\mathcal{C}(L,n) = \sum_{k=0}^{n-1} \binom{n-1}{k} L^{\overline{k+1}} n^{-1-k} \tag{18}$$

$$= \frac{1}{n} \sum_{k=0}^{n-1} \frac{(n-1)!}{k!(n-k-1)!} L^{\overline{k+1}} \left(\frac{1}{n}\right)^k \tag{19}$$

$$= \frac{1}{n} \sum_{k=0}^{n-1} (n-k)^{\overline{k}} L^{\overline{k+1}} \frac{(\frac{1}{n})^k}{k!} \tag{20}$$

$$= \frac{1}{n} \sum_{k=0}^{n-1} (-1)^k (-n+1)^{\overline{k}} L^{\overline{k+1}} \frac{(\frac{1}{n})^k}{k!} \tag{21}$$

$$= \frac{L}{n} \sum_{k=0}^{n-1} (-n+1)^{\overline{k}} (L+1)^{\overline{k}} \frac{(-\frac{1}{n})^k}{k!} \tag{22}$$

$$= \frac{L}{n} {}_2F_0\left(\begin{matrix} L+1, -n+1 \\ - \end{matrix} \middle| -\frac{1}{n} \right). \tag{23}$$

In step (20) we used Proposition 4 and in step (21) we used Proposition 3. In the last step we move from the finite sum to the infinite sum. As $(-n+1)^{\overline{k}}$ is zero for every integer k bigger than $n-1$, also all the external sum terms are. ∎

This result essentially says that we can compute the normalizing term $\mathcal{C}(L,n)$ by taking the confluent hypergeometric function ${}_2F_0(L+1, -n+1; -; z)$ and evaluating it in the point

$z = -\frac{1}{n}$. Unfortunately we have also the multiplier $\frac{L}{n}$ in front of our function.

We can now ask what happens if we do the same simplification with our second formula. It appears that it gives a very similar result except that the irritating multiplier disappears:

Theorem 4 The second form represented via a confluent hypergeometric function is $\mathcal{C}(L,n) = {}_2F_0\left(\begin{matrix} L-1, -n \\ - \end{matrix}\middle| -\frac{1}{n}\right)$.

Proof: Omitted (quite similar to the previous proof). ■

The two previous theorems give us an identity which seems to be quite hard to prove with straightforward manipulation as the two sets of confluent hypergeometric functions are different.

Theorem 5 Given $n \geq 1$, $L \geq 1$ and $n, L \in \mathbb{N}$, the following is true: ${}_2F_0\left(\begin{matrix} L-1, -n \\ - \end{matrix}\middle| -\frac{1}{n}\right) = \frac{L}{n}{}_2F_0\left(\begin{matrix} L+1, -n+1 \\ - \end{matrix}\middle| -\frac{1}{n}\right)$.

Proof: Nothing to prove anymore. ■

We will need the identity later with the birthday problem, and also we need the following definition [1]:

Definition 3 ${}_2F_0\left(\begin{matrix} a, 1+a-b \\ - \end{matrix}\middle| -\frac{1}{z}\right) = z^a U(a,b,z)$, where $U(a,b,z)$ is the hypergeometric (Kummer's) U-function.

Using this definition we can for example write the result of Theorem 4 also in the form $n^{L-1}U(L-1, L+n; n)$, but then we are again facing an annoying multiplier. This is the reason why we prefer the ${}_2F_0$-notation, although some may argue that the U-function notation would be a more natural choice.

V. Connection to the Birthday Problem

The *birthday problem* (paradox) is a well-known and classical problem of probability theory [4]. The basic idea is just to answer the following question: How many people on average we need so that at least two of them have the same birthday? (also other questions exist). In general we can compute the probability that if we already know $k-1$ distinct birthdays, what is the probability that the kth person's birthday is some of these. This probability is

$$P^{(n)}(X = k) = \frac{(k-1)n^{\underline{k-1}}}{n^k} = \frac{(k-1)n!}{n^k(n-k+1)!}, \qquad (24)$$

where n is the number of days. In the birthday problem n is usually 365. To answer the question, we must compute the expected value. Surprisingly the answer is as low as 25, because $E(X)$=24,6166. It is shown [14] that in general the expectation can be computed for example using formula

$$E(X) = \frac{n!}{n^n}\sum_{k=0}^{n}\frac{n^k}{k!} = \left(\frac{e}{n}\right)^n \Gamma(n+1, n), \qquad (25)$$

where $\Gamma(\alpha, x)$ is the incomplete gamma function. This same formula gives us also the value of the binomial normalizing term, which can be easily seen by starting from Theorem 2:

$$\begin{aligned}\mathcal{C}(2,n) &= \sum_{k=0}^{n}\binom{n}{k}\frac{1^{\overline{k}}}{n^k} = \sum_{k=0}^{n}\binom{n}{k}\frac{k!}{n^k} = n!\sum_{k=0}^{n}\frac{1}{(n-k)!n^k} \\ &= \frac{n!}{n^n}\sum_{k=0}^{n}\frac{n^{n-k}}{(n-k)!} = \frac{n!}{n^n}\sum_{k=0}^{n}\frac{n^k}{k!}.\end{aligned}$$

So the binomial normalizing term is identical to the expectation of the birthday problem, and it is natural to ask whether there is a similar counterpart in the birthday problem framework for the multinomial normalizing term as well. In the following we show that the counterpart is given by the rising factorial moments.

The rising factorial moments can be utilized in computation of the variance and other central moments as represented in [14]. These rising factorial moments for the birthday problem can be computed using formula

$$\begin{aligned}E(X^{\overline{m}}) &= E(X(X+1)\cdots(X+m-1)) \qquad (26)\\ &= (m+1)!n^{m+1}U(m+2, m+n+2; n), \qquad (27)\end{aligned}$$

where $U(a,b;z)$ is the hypergeometric U-function.

Proposition 5 The connection between the multinomial normalizing term and the rising factorial moments of the birthday problem is defined by the identity $E(X^{\overline{m}}) = m!\mathcal{C}(m+1,n)$.

Proof:

$$\begin{aligned}E(X^{\overline{m}}) &= \left(\frac{m!}{m!}\right)(m+1)!n^{m+1}U(m+2, n+m+2, n) \\ &= m!\left(\frac{m+1}{n}\right)n^{m+2}U(m+2, n+m+2, n) \\ &= m!\left(\frac{m+1}{n}\right){}_2F_0\left(\begin{matrix} m+2, -n+1 \\ - \end{matrix}\middle| -\frac{1}{n}\right) \qquad (28)\\ &= m!\mathcal{C}(m+1,n).\end{aligned}$$

We applied Theorem 3 to the form (28), to get the result. ■

We can now immediately notice two important implications. First, the recurrence formula (9) is suitable also in the computation of the rising factorial moments of the birthday problem. The modified recurrence formula is

$$E(X^{\overline{m}}) = mE(X^{\overline{m-1}}) + nmE(X^{\overline{m-2}}). \qquad (29)$$

The second implication is that by Theorems 2 and 5, the rising factorial moments of the birthday problem can be computed also using formula

$$E(X^{\overline{m}}) = m!\sum_{k=0}^{n}\binom{n}{k}\frac{m^{\overline{k}}}{n^k}. \qquad (30)$$

Other implications we leave to the reader to derive. However, we mention one approximation result, which may not be well-known in the area of the birthday problem and the related applications: Szpankowski has derived an approximation formula for the redundancy rate of memoryless sources [20] and this formula has been later noticed to be also an approximation of the logarithm of the multinomial normalizing term [10].

It is quite accurate for large n. One might consider to use this approximation formula in cases where even the presented $\mathcal{O}(n)$-formulas are too slow to use.

VI. Applications

Below we will show some examples illustrating how to easily compute the NML normalizing terms using Maple software. Our second hypergeometric form turns out to be particularly nice in this respect, as we can just write one function notation without any external multipliers. We give three examples that have practical significance.

In the first example let us consider a multinomial variable with four values and 100 data points. The exact value of the normalizing term can be computed by writing

```
simplify(subs([L=4,n=100],hypergeom([-n,L-1],[],-1/n)));
```

The second example is the so called Naive Bayes model, which is frequently applied in classification problems. The task is to predict the value of a target variable (class variable) using the observed features (values of predictor variables). The normalizing term generating function in the Naive Bayes case can be written in the power form as

$$\left(\sum_{n=0}^{\infty} \mathcal{C}(K_1, n) \cdots \mathcal{C}(K_m, n) n^n \frac{z^n}{n!} \right)^L , \qquad (31)$$

where n is number of data vectors, L is the number of values in the class variable, K_i is the number of values in the predictor variable i and m is the number of predictor variables [13]. If we for instance consider a model where we have a binary class variable and two predictor variables with four and five outcomes, we can write the generating function in Maple and expand it to series: for example the hundred first terms of the generating function of the normalizing term can be computed by

```
simplify(series((1+sum(hypergeom([-n,3],[],-1/n)
                      *hypergeom([-n,4],[],-1/n)*n**n/n!
                      *z**n,n=1..infinity))**2,z,100));
```

Each of these resulting coefficients must be multiplied by $\frac{n!}{n^n}$ to get the normalizing terms. If one has more than two predictor variables, then the product will contain more confluent hypergeometric functions.

The third example is Bayesian network model selection with the so called factorized NML (fNML) criterion [17]. The fNML criterion can be used for learning general Bayesian networks (DAGs). In this case we can directly utilize our hypergeometric form in the exact computations of the local NML distributions, using standard mathematical software packages as shown above.

VII. Conclusions

Using the tools of umbral calculus, we were able to derive a computationally simple form for the multinomial normalizing term of the NML distribution. We also showed its relation to hypergeometric functions, which allows us to efficiently utilize the built-in hypergeometric function libraries of standard mathematical software packages. Using these results, exact as well as floating-point values of the multinomial NML can now be computed fast and efficiently. This allows many non-trivial applications of NML model selection. We also found a complete relationship between our problem and the well-known birthday problem: this result has potentially a high impact on both problem domains and their application areas, as research results can be now transferred between the two domains.

Acknowledgment

This work was supported in part by the Academy of Finland under the project Civi and by the Finnish Funding Agency for Technology and Innovation under the projects Kukot and PMMA. In addition, this work was supported in part by the IST Programme of the European Community, under the PASCAL Network of Excellence.

References

[1] M. Abramowitz and I.A. Stegun, editors. *Handbook of Mathematical Functions*. Dover Publications, Inc., New York, 1970.

[2] G.B. Arfken and H.J. Weber. *Mathematical Methods for Physicists*. Academic Press, 4th edition, 1995.

[3] R.M. Corless, G.H. Gonnet, D.E.G. Hare, D.J. Jeffrey, and D.E. Knuth. On the Lambert W function. *Advances in Computational Mathematics*, 5:329–359, 1996.

[4] W. Feller. *An Introduction to Probability Theory and Its Applications*. John Wiley & Sons, 3rd edition, 1968.

[5] P. Flajolet, P.J. Grabner, P. Kirschenhofer, and H. Prodinger. On Ramanujan's Q-function. *Journal of Computational and Applied Mathematics*, 58:103–116, 1995.

[6] R.L. Graham, D.E. Knuth, and O. Patashnik. *Concrete Mathematics (second edition)*. Addison-Wesley, 1994.

[7] P. Grünwald. *The Minimum Description Length Principle*. MIT Press, 2007.

[8] D.E. Knuth. Convolution polynomials. *Mathematica Journal*, 2(4):67–78, Fall 1992.

[9] D.E. Knuth and B. Pittel. A recurrence related to trees. *Proceedings of the American Mathematical Society*, 105(2):335–349, 1989.

[10] P. Kontkanen, W. Buntine, P. Myllymäki, J. Rissanen, and H. Tirri. Efficient computation of stochastic complexity. In C. Bishop and B. Frey, editors, *Proceedings of the Ninth International Conference on Artificial Intelligence and Statistics*, pages 233–238. Society for Artificial Intelligence and Statistics, 2003.

[11] P. Kontkanen and P. Myllymäki. Analyzing the stochastic complexity via tree polynomials. Technical Report 2005-4, Helsinki Institute for Information Technology (HIIT), 2005.

[12] P. Kontkanen and P. Myllymäki. A linear-time algorithm for computing the multinomial stochastic complexity. *Information Processing Letters*, 103(6):227–233, 2007.

[13] T. Mononen and P. Myllymäki. Fast NML computation for naive Bayes models. In V. Corruble, M. Takeda, and E. Suzuki, editors, *Proceedings of the Tenth International Conference on Discovery Science*, October 2007.

[14] P.N. Rathie and P. Zörnig. On the birthday problem: Some generalization and applications. *International Journal of Mathematics and Mathematical Sciences*, 2003(60):3827–3840, 2003.

[15] J. Rissanen. *Information and Complexity in Statistical Modeling*. Springer, 2007.

[16] S. Roman. *The Umbral Calculus*. Dover, 2005.

[17] T. Roos, T. Silander, P. Kontkanen, and Myllymäki P. Bayesian network structure learning using factorized NML universal models. In *Information Theory and Applications Workshop*, San Diego, CA, January 2008.

[18] G.-C. Rota. *Finite Operator Calculus*. Academic Press, 1975.

[19] Yu.M. Shtarkov. Universal sequential coding of single messages. *Problems of Information Transmission*, 23:3–17, 1987.

[20] W. Szpankowski. *Average case analysis of algorithms on sequences*. John Wiley & Sons, 2001.

Paper 2

Tommi Mononen and Petri Myllymäki:

Computing the Multinomial Stochastic Complexity in Sub-Linear Time

In *Proceedings of the European Workshop on Probabilistic Graphical Models*, PGM'08 (Hirtshals, Denmark), pages 209-216, 2008.

Computing the Multinomial Stochastic Complexity in Sub-Linear Time

Tommi Mononen and Petri Myllymäki
Helsinki Institute for Information Technology (HIIT), Finland
{firstname}.{lastname}@hiit.fi

Abstract

Stochastic complexity is an objective, information-theoretic criterion for model selection. In this paper we study the stochastic complexity of multinomial variables, which forms an important building block for learning probabilistic graphical models in the discrete data setting. The fastest existing algorithms for computing the multinomial stochastic complexity have the time complexity of $\mathcal{O}(n)$, where n is the number of data points, but in this paper we derive sub-linear time algorithms for this task using a finite precision approach. The main idea here is that in practice we do not need exact numbers, but finite floating-point precision is sufficient for typical statistical applications of stochastic complexity. We prove that if we use only finite precision (e.g. double precision) and precomputed sufficient statistics, we can in fact do the computations in sub-linear time with respect to data size and have the overall time complexity of $\mathcal{O}(\sqrt{dn} + L)$, where d is precision in digits and L is the number of values of the multinomial variable. We present two fast algorithms based on our results and discuss how these results can be exploited in the task of learning the structure of a probabilistic graphical model.

1 Introduction

Stochastic complexity (SC) is an information-theoretic model selection criterion, which can be seen as a theoretical instantiation of the minimum description length (MDL) principle (Rissanen, 2007; Grünwald, 2007). Intuitively speaking, the basic idea is that the best model for the data is the one which results in the shortest description for the data together with the model. This principle gives us a non-informative, objective criterion for model selection, but there are many ways to define the stochastic complexity formally; one theoretically solid way is to use the *normalized maximum likelihood* (NML) distribution. Recent results suggest that this criterion performs very well in the task of learning Bayesian network structures (Roos et al., 2008).

In the following, let $\mathcal{M}$ denote a parametric probabilistic model, and $\hat{\theta}(\mathbf{x}^n)$ the maximum likelihood parameters of the model given a matrix of observations $\mathbf{x}^n$. The NML distribution is defined as

$$P_{NML}(\mathbf{x}^n \mid \mathcal{M}) = \frac{P(\mathbf{x}^n \mid \hat{\theta}(\mathbf{x}^n), \mathcal{M})}{\sum_{\mathbf{y}^n} P(\mathbf{y}^n \mid \hat{\theta}(\mathbf{y}^n), \mathcal{M})}, \quad (1)$$

where in the numerator we have the maximum likelihood of our observed data and in the denominator we have the sum of maximum likelihoods (denoted in the sequel by $\mathcal{C}(\mathcal{M}, n)$) over all the discrete data sets of size n (Shtarkov, 1987).

Let us define the stochastic complexity as the negative logarithm of (1):

$$SC(\mathbf{x}^n \mid \mathcal{M}) = -\log \frac{P(\mathbf{x}^n \mid \hat{\theta}(\mathbf{x}^n), \mathcal{M})}{\mathcal{C}(\mathcal{M}, n)}. \quad (2)$$

The basic model selection task is to compute the value of this model selection criterion for parametric models of different complexity and choose the one for which this value is minimized, given the observed data.

The single multinomial variable model is an important building block for building more complex probabilistic graphical models for discrete data. For this reason we want to able to compute the NML for multinomial variables as efficiently as possible. In the following, we simplify our notation and leave out $\mathcal{M}$: the model is implicitly defined by the number of values of the multinomial variable, denoted

by L. The numerator is now

$$P(x^n \mid \hat{\theta}(x^n), L) = \prod_{k=1}^{L} \left(\frac{h_k}{n}\right)^{h_k}, \qquad (3)$$

where h_k is a number of data points assigned to the kth value. We expect in this paper that sufficient statistics is known and computing (3) takes therefore only time $\mathcal{O}(L)$. The denominator is

$$\mathcal{C}(L,n) = \sum_{h_1+\cdots+h_L=n} \frac{n!}{h_1!\cdots h_L!} \prod_{k=1}^{L} \left(\frac{h_k}{n}\right)^{h_k},$$

which is a sum of maximum likelihoods so that the summation goes over every possible data of length n.

Although using the definition directly for computing the *multinomial normalizing sum* in the denominator is not computationally feasible, several algorithms for doing this in $\mathcal{O}(n)$ time have been recently developed (Kontkanen and Myllymäki, 2007; Mononen and Myllymäki, 2008b). In this paper we show that our earlier theoretical results presented in (Mononen and Myllymäki, 2008b) can be used for constructing algorithms that compute $\mathcal{C}(L,n)$ in sub-linear time with any desired (finite) precision. We start by briefly reviewing the relevant earlier results and then show how they can be exploited in deriving new ultra-fast algorithms.

2 Known Properties of the Normalizing Sums

In Mononen and Myllymäki (2008b) we proved that the multinomial normalizing sum can be described as a confluent hypergeometric function evaluated at a certain point. We showed that we can write the hypergeometric presentation also in another simple form using falling and rising factorial polynomials. The *falling factorial polynomials* are of the form

$$x^{\underline{k}} = x(x-1)\cdots(x-k+1), \qquad (4)$$

and the *rising factorial polynomials* are

$$x^{\overline{k}} = x(x+1)\cdots(x+k-1). \qquad (5)$$

The *binomial normalizing sum* is then

$$\mathcal{C}(2,n) = \sum_{k=0}^{n} b_k = \sum_{k=0}^{n} \frac{n^{\underline{k}}}{n^k}, \qquad (6)$$

and the general multinomial normalizing sum can be written as

$$\mathcal{C}(L,n) = \sum_{k=0}^{n} m_k = \sum_{k=0}^{n} \frac{n^{\underline{k}}\,(L-1)^{\overline{k}}}{n^k k!}. \qquad (7)$$

As these forms spin off from hypergeometric forms, and a hypergeometric series has the property that there exist a simple ratio of consecutive terms, we know that also (6) and (7) have this property. We will introduce these ratios later in sections 3.1 and 4.1 and use them in the computations.

It is known that the binomial normalizing sum equals to the expectation of the *birthday problem* with the mapping: data size is equal to the number of days (Mononen and Myllymäki, 2008b). We will use an approximation derived for this expectation later in our proof.

There is a recurrence formula for computing the multinomial normalizing sum as the value of the corresponding binomial normalizing sum is known (Kontkanen and Myllymäki, 2007):

$$\mathcal{C}(L,n) = \mathcal{C}(L-1,n) + \frac{n \cdot \mathcal{C}(L-2,n)}{L-2}, \qquad (8)$$

and $\mathcal{C}(1,n)$ is defined to be 1 for every n. This formula can be effectively used for linear time computation of multinomial normalizing sums.

3 The Binomial Normalizing Sum

3.1 Properties of the Sum Terms

Let us start by plotting the terms of (6). We can immediately observe that the first terms of the sum give the greatest impact and most of the terms are very small (see Figure 1). All the terms are positive and getting closer to the zero, because the ratio of successive terms of (6) is

$$\frac{b_k}{b_{k-1}} = \frac{n-k+1}{n}. \qquad (9)$$

Now the natural question is, how many terms do we need, if we want to compute the normalizing sum and use for example double precision. We study this question more closely in section 3.2, but in loose terms the number of needed terms is proportional to the square root on n, which promises sub-linear time performance.

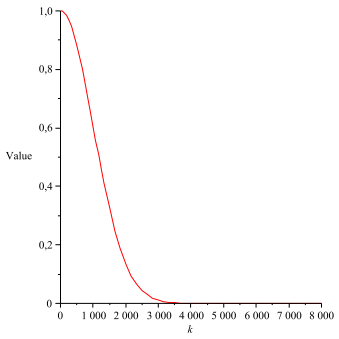

Figure 1: Magnitude of the first 8000 terms of the binomial normalizing sum when the data size (n) is one million.

However, first we have to quantify how to measure the precision. We measure in the standard way the error we make when pruning the sum: we compute the tail sum and compare it to the whole sum. Thus we compute a relative error. However, because setting the desired relative error directly is quite cumbersome, we rather compute it in digits: it much easier just to say that we need e.g. 7 digit precision. More precisely, we define that for positive real numbers p and q, where $p \leq q$ and $p, q \geq 1$, the precision in digits is

$$\left\lfloor -\log_{10}\left(\frac{q-p}{q}\right)\right\rfloor. \tag{10}$$

So if the target q is for example 1.100000 and the approximation p is 1.099999, the precision is $\lfloor 6.04139 \rfloor$. So although only the first digit is the same, the precision in digits according to the definition above is 6. But if we round p after the sixth digit, we get 1.10000. This means that although the precision in digits does not tell all the time how many correct digits we have, it is still very close to what we want. Next we do an upper bound approximation of the last required term of the binomial normalizing sum for achieving some fixed precision.

3.2 Proof of the Right Bound

There is no known closed form solution for (6). To compute the precision, we have to compute the sum starting from some b_r to all the way to the the term b_n. As said before, we are interested in this tail sum, because it tells us how big an error we make, if we stop computing the sum after the term b_{r-1}. Although the terms of the sum look simple, they are a bit tricky to handle, and therefore we perform several upper bound approximations for the terms. Upper bound approximations are of course required, because we want to be sure that whatever bound we get, it must give the promised result. We also move from the discrete sum to an integral presentation, because it makes things simpler in this case.

Next we give in a row four propositions needed for proving the index bound of the binomial sum. After the propositions we prove the mentioned bound, which we call the *right index bound* (in the multinomial case also the left bound exists).

Proposition 1. *We have the following upper bound approximation for the term b_k:*

$$\frac{n^{\underline{k}}}{n^k} \leq \left(1 - \frac{k-1}{2n}\right)^{k-1}, \textit{where} \quad 1 \leq k \leq n.$$

Proof. We prove that the ratio of both sides is bigger than one. Let us look at the ratio:

$$\frac{\left(1-\frac{k-1}{2n}\right)^{k-1}}{\frac{n^{\underline{k}}}{n^k}} = \frac{n \cdot n^{k-1}\left(\frac{n-\frac{1}{2}(k-1)}{n}\right)^{k-1}}{n^{\underline{k}}} \tag{11}$$

$$= \frac{(n-\frac{1}{2}(k-1))^{k-1}}{(n-1)\cdots(n-k+1)} \tag{12}$$

$$= \prod_{r=2}^{k} \frac{n-\frac{1}{2}(k-1)}{n-r+1} = \prod_{r=2}^{k} a_r. \tag{13}$$

Now we just look at the product of pairwise terms defined by

$$a_r a_{k-r+2} = \frac{(n-\frac{1}{2}(k-1))^2}{(n-r+1)(n-k+r-1)} = \frac{N_r}{D_r}.$$

We want to prove that each of these pairwise terms is bigger than 1. However, we can equivalently subtract the denominator from the numerator and require the result to be bigger than 0, because both are always positive in our range. The result is

$$N_r - D_r = n + \frac{1}{4}k^2 - rk + \frac{1}{2}k + r^2 - 2r + \frac{5}{4}. \tag{14}$$

We take the derivate of this difference with respect to k and get the minimum point $k = 2r - 1$. Substituting this in (14), we notice that the result is

$n-r+1$, which is always bigger than 0 in our range. This means that also the pairwise terms must be always bigger than 1, which implies that the proposition must be true for even number of terms a_r.

If we have an odd number of terms a_r, then we have to still prove that the median term is also bigger than 1. The median term is

$$a_{\frac{k}{2}+1} = \frac{2n-k+1}{2n-k}. \tag{15}$$

This cannot be smaller than 1, because $n \geq k \geq 0$. □

Proposition 2. *The following inequality is true for* $\frac{k}{2n} < 1$*:*

$$\left(1 - \frac{k}{2n}\right)^k \leq e^{-\frac{k^2}{2n}}.$$

Proof. Take the natural logarithm of both sides of the inequality and use the known logarithm inequality $\ln(1+x) \leq x$, which is valid for all $x > -1$. □

Proposition 3. *Because* b_k *is a continuous monotonically decreasing function inside the interval* $[0, n]$*, the discrete volume (the sum) corresponds to the upper Riemann sum with intervals of length one. Hence we can give the following inequality:*

$$\sum_{k=0}^{n} b_k \leq 1 + \int_{k=1}^{n} b_{k-1} \mathrm{d}k.$$

Proof. Our integral is always bigger than the given upper sum, because on the right hand side we shifted our function in such way that every discrete column is always entirely below the curve. The size of the first column is 1, which is the first term on the right hand side. □

Proposition 4. *The following inequality holds:*

$$\text{erf}\,(x) \geq \sqrt{1 - e^{-x^2}},$$

when $x \geq 0$ *and*

$$\text{erf}\,(x) = \frac{2}{\sqrt{\pi}} \int_{t=0}^{x} e^{-t^2} \mathrm{d}t.$$

Proof. First suppose $x \geq 0$. Let us now modify the inequality:

$$\text{erf}\,(x) \geq \sqrt{1 - e^{-x^2}} \tag{16}$$

$$(\text{erf}\,(x))^2 + e^{-x^2} - 1 \geq 0 \tag{17}$$

Denote the left hand side by $h(x)$ and take the derivative of it:

$$h'(x) = \frac{2e^{-x^2}(2\text{erf}\,(x) - x\sqrt{\pi})}{\sqrt{\pi}}. \tag{18}$$

We can see that all the roots in our range are solutions for the equation

$$2\text{erf}\,(x) - x\sqrt{\pi} = 0, \tag{19}$$

$$\text{erf}\,(x) = \frac{\sqrt{\pi}}{2} x. \tag{20}$$

As the error function is a convex function between 0 and infinity and the right hand side of (20) is a linear function, there can be only two solutions. The first one is $x = 0$, where $h(0) = 0$, and the other one is $x \approx 0.8982$. The second solution is a positive maximum point. As we have

$$\lim_{x \to \infty} h(x) = (1)^2 + 0 - 1 = 0, \tag{21}$$

this means that $h(x)$ must be always positive in the range, which completes our proof. □

Finally we are now ready to introduce our main theorem giving the right bound approximation:

Theorem 1. *Given precision in digits* (d) *and data size* n*, the right index bound* t *for the binomial normalizing sums is* $\left\lceil 2 + \sqrt{-2n \ln(2 \cdot 10^{-d} - 100^{-d})} \right\rceil$.

Proof. First we approximate the upper bound of the partial binomial normalizing sum from 0 to r:

$$\sum_{k=0}^{r} \frac{n^{\underline{k}}}{n^k} \leq 1 + \sum_{k=1}^{r} \left(1 - \frac{k-1}{2n}\right)^{k-1} \tag{22}$$

$$\leq 1 + \sum_{k=1}^{r} e^{-\frac{(k-1)^2}{2n}} \tag{23}$$

$$\leq 2 + \int_{k=2}^{r} e^{-\frac{(k-2)^2}{2n}} \tag{24}$$

$$= 2 + \sqrt{\frac{n\pi}{2}}\, \text{erf}\left(\frac{r-2}{\sqrt{2n}}\right) = F(r) \tag{25}$$

Previous inequality steps follow easily from propositions 1, 2 and 3. Now we can express the precision in digits (d) with the equation

$$-\log_{10}\left(\frac{F(n) - F(r)}{\sqrt{\frac{n\pi}{2}}}\right) = d, \tag{26}$$

where the denominator inside the logarithm is a lower bound approximation for the binomial normalizing sum. For example Laplace's method gives for (6) the approximation (Flajolet and Sedgewick, 2005):

$$\mathcal{C}(2,n) = \sqrt{\frac{n\pi}{2}} + \frac{2}{3} + \mathcal{O}\left(\frac{1}{\sqrt{n}}\right). \quad (27)$$

By omitting the constant term, we get our lower bound approximation for the denominator, which is valid for all data sizes (exact proof omitted). Thus we have

$$-\log_{10}\left(\frac{\sqrt{\frac{n\pi}{2}} - \sqrt{\frac{n\pi}{2}}\ \mathrm{erf}\left(\frac{r-2}{\sqrt{2n}}\right)}{\sqrt{\frac{n\pi}{2}}}\right) = d \quad (28)$$

$$-\log_{10}\left(1 - \mathrm{erf}\left(\frac{r-2}{\sqrt{2n}}\right)\right) = d. \quad (29)$$

We replaced the first error function in (26) with its supremum value 1 and got (28). If we solve r, we have

$$r = 2 + \sqrt{2n} \cdot R, \quad (30)$$

where

$$R = \mathrm{erf}^{-1}(1 - 10^{-d}). \quad (31)$$

The final task is now to approximate the inverse of the error function (Winitzky, 2008). We need this for approximating R to get a nice, clean and computable bound. First we compute the Taylor approximation:

$$g(x) = \ln(1 - \mathrm{erf}(x)^2) \approx -\frac{4}{\pi}x^2 + \mathcal{O}(x^4). \quad (32)$$

We need only the first non-zero term for our purpose. The approximation is then

$$\mathrm{erf}(x) = \sqrt{1 - e^{g(x)}} \approx \sqrt{1 - e^{-\frac{4x^2}{\pi}}}, \quad (33)$$

and it is good enough as our tests later show. This is not a lower bound approximation, but we need one, because we invert the approximating function. A little dirty trick solves our problem. We just change the multiplier 4 to π (Proposition 4). This new function can be easily inverted and the result therefore is

$$\mathrm{erf}^{-1}(u) = \sqrt{-\ln(1-u^2)}. \quad (34)$$

The final step is to set $u = 1 - 10^{-d}$ and we have the result

$$R \approx \sqrt{-\ln(1 - (1-10^{-d})^2)} \quad (35)$$

$$= \sqrt{-\ln(2 \cdot 10^{-d} - 100^{-d})}. \quad (36)$$

□

We continue approximating R, because for the time complexity reasons, we need a simplified form to see the magnitude of this term. We easily see that

$$\sqrt{-\ln(2 \cdot 10^{-d} - 100^{-d})} \leq \sqrt{-\ln(10^{-d})} \quad (37)$$

$$= \sqrt{d\ln(10)}, \quad (38)$$

and therefore the required number of terms is $\mathcal{O}(\sqrt{dn})$.

The index bound seems to be quite good. In Figure 2 we have plotted optimal indexes and indexes given our bound with respect to data size n. We chose precisions so that they correspond approximately to single and double precision floating-point numbers. If n is one million, the index error is about $+250$ in both cases. The single precision error is a bit larger than the double precision error, because the index bound is getting tighter as precision increases.

4 The Multinomial Normalizing Sum

4.1 Properties of the Sum Terms

As we already saw, the ratio of the terms of the binomial normalizing sum is a simple rational function. In the multinomial case the ratio is the function

$$\frac{m_k}{m_{k-1}} = \frac{(n-k+1)(k+L-2)}{nk}. \quad (39)$$

Let us look at the terms of the multinomial normalizing sum. Figure 3 suggest that there is the biggest term and if we look at the term function, we see that it is unimodal. The next theorem and its proof give formal justifications for these claims.

Theorem 2. *The index of the biggest term of the multinomial normalizing sum is* $\left\lfloor \frac{1}{2}\left(3 - L + \sqrt{L^2 + (4n-2)L - 8n + 1}\right)\right\rfloor$.

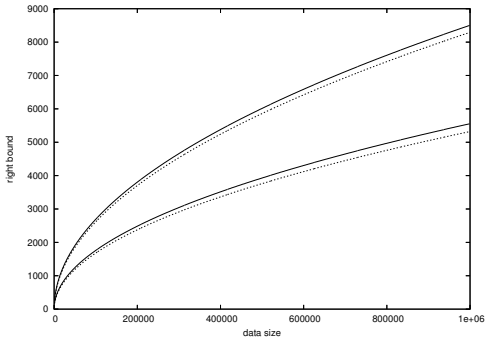

Figure 2: Terms needed for 16 (above) and 7 digit precisions with given data size. Actual approximations are shown as thick solid line. Thin dotted lines represent optimal index values.

Proof. We have to solve the equation

$$\frac{m_k}{m_{k-1}} = 1 \tag{40}$$

with respect to k. In other words, we are interested in values of real-valued k, where two consecutive sum terms have the same value. This requires solving roots of second order polynomial with respect to k. The other root is always negative and therefore is not in our range; let us denote the positive root by r. We are allowed to take the floor, because we know that the peak is between continuous index values $r-1$ and r. Outside this range all the sum term values are smaller than inside the range. Inside this range there are one or two integer points. If only one, then it is $\lfloor r \rfloor$. If there are two integer indexes, then $r-1$ and r must be these and they both give the same maximum value and thus $\lfloor r \rfloor$ gives the other of the two maximum values. □

This proved at the same time that the sum terms are getting bigger until they reach the peak, after which they start getting smaller. This unimodality gives us a great opportunity to construct simple and efficient algorithms.

4.2 About the Index Bounds

We can guess from the 15-nomial in Figure 3 that if we want to compute the sum in a fixed precision, we actually have to compute the sum terms from some $s \geq 0$ to some $t \leq n$. This means that the index of the first required term can be bigger than 0.

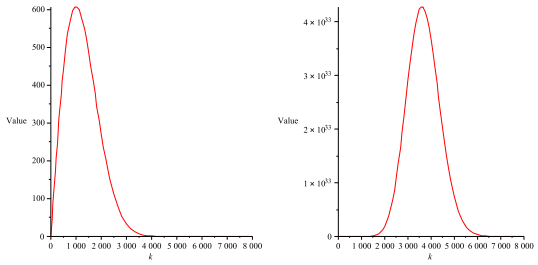

Figure 3: Magnitude of the first 8000 terms of the trinomial (left) and the 15-nomial (right) normalizing sums when data size (n) is one million.

However, we have to compute the factorials starting from 0 (to avoid any approximation), so we still have to start from index 0. Our empirical tests also show that we have to compute a lot more terms than in the binomial case. The effect is visible also by comparing figures 1 and 3. However, we can use (8) for computing the multinomial normalizing sum when the value of the binomial normalizing sum is known. This recurrence formula method seems to be much more efficient and therefore it is unnecessary to prove bounds in the multinomial case.

5 Sub-Linear Algorithms

First we present a simple algorithm (Algorithm 1), which is validated directly by unimodality. The basic idea is that the algorithm can sum the terms of a multinomial sum until the sum does not change anymore. Terms of the left slope always change the sum because each term is bigger than the previous one. After the peak, terms are monotonically getting smaller, and we know that the terms are never getting bigger anymore. Therefore the algorithm can stop after the sum has converged. All terms in the tail are so small and decaying so rapidly, that they cannot have a significant effect on the finite sum (we will return to this subject after the second algorithm). A precision of the result is determined by the precision of the used floating-point numbers.

Notice that the algorithm is using the recurrence

$$m_k = \frac{(n-k+1)(k+L-2)}{nk} \cdot m_{k-1} \tag{41}$$

while computing the sum terms. This way we can avoid huge factorial values and also floating-point

```
Compute_Multinomial(L,n){
  double sum = 1, previous_sum = -1, m = 1;
  int k = 1;

  for k from 1 to n by 1{
    m = (k+L-2) * (n-k+1) / (n*k) * m;
    sum = sum + m;

    if(sum is same as previous_sum){
      return sum;
    }

    previous_sum = sum;
  }

  return sum;
}
```

Algorithm 1: A simple algorithm for computing the multinomial normalizing sum.

```
Compute_Multinomial_With_Recurrence(L,n){
  double sum = 1, b = 1, old_sum, new_sum;
  int k,j;
  int bound = ceil( 2 + sqrt( -2*n*ln( 2*10^(-d)
                   -100^(-d) ) ) );

  for k from 1 to bound by 1{
    b = (n-k+1) / n * b;
    sum = sum + b;
  }

  old_sum = 1;

  for j from 3 to L by 1{
    new_sum = sum + (n * old_sum) / (j-2);
    old_sum = sum;
    sum = new_sum;
  }

  return sum;
}
```

Algorithm 2: A faster algorithm for computing the multinomial normalizing sum.

errors seem to be much lower. However the time complexity is not proved for this algorithm, as we did not compute the index bounds in the multinomial case. In addition, we cannot set the digit precision freely. For these reasons we present a second algorithm, which also seems to be twice as fast as the first one.

The intuitive idea of Algorithm 2 is to first compute the index bound, and then compute the binomial normalizing sum using the ratio of successive terms. After this, the algorithm uses the recurrence formula to compute the wanted multinomial normalizing sum. Variable `bound` is not directly computable as presented in the pseudocode, but we can use some standard logarithmic manipulation to avoid underflow. The time complexity of this second algorithm is $\mathcal{O}(\sqrt{dn} + L)$.

Now we can revise the question about the tail sum. Terms below the index bound do not affect the sum, but the first algorithm actually stops in the binomial case before the right bound is reached, because single terms do not change the sum. But if we take a sum starting from the stopping point all the way to the index bound, we can notice that this tail sum can affect the original sum. In our empirical tests we found out that the effect seems to be only on the few last digits, which is of the same magnitude as the floating point errors of our algorithms.

In Figure 4 we can see the average decay of the last digits with respect to data size. The variance is small between different numbers of same magnitude, so the figure gives a realistic impression. In the end we decided to keep the algorithms simple, because such small errors in the criterion hardly have any effect on the actual model selection task.

There are two operations that in fact increase precision. First we found out that the recurrence formula in Algorithm 2 tends to increase precision of those terms with an odd number of outcomes. Second we still have to take a logarithm of the normalizing sum, when computing actual stochastic complexity. The effect of the latter operation seem to be about one digit.

Precise values can be achieved using a simple trick: use a floating-point precision that is higher than the precision d, and crop the tail digits (e.g. quad precision for triple precision).

6 Conclusions

Stochastic complexity is an elegant, information-theoretic criterion for learning probabilistic graphical model structures. Although probabilistic in nature, it is fully objective and does not involve any hyperparameters which may be introduce problems in learning (Silander et al., 2007).

The fastest previously known algorithms for computing the multinomial stochastic complexity are $\mathcal{O}(n)$-algorithms. In this paper we showed that our previous hypergeometric representation of multinomial normalizing sums can be used for deriving sub-linear algorithms.

As the multinomial variable is an important building block in many more complex model classes, the results are directly applicable to model selec-

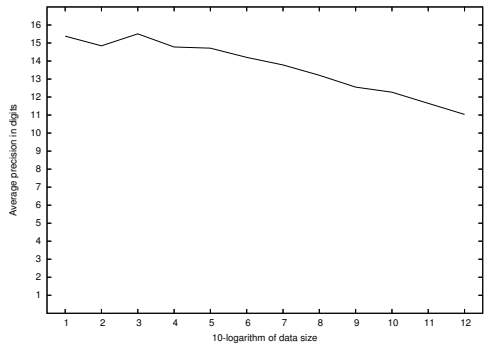

Figure 4: Average precision in the binomial normalizing sum with respect to data size using double precision floating-point numbers and Algorithm 2.

tion tasks in these cases as well. However, it should be noted that if the learning criterion is defined via the standard normalized maximum likelihood, the savings in the overall computational complexity are not necessarily very significant: for example, in the Naive Bayes case (classification or clustering tasks if the root node is hidden), the learning criterion involves a product of multinomial normalizing sums corresponding to the predictor variables, and these can be now computed in sub-linear time using the results above. Nevertheless, the real bottleneck of the computation process is a convolution operation, which still takes at least $\mathcal{O}(n^2 \log L)$ floating-point operations to compute (Mononen and Myllymäki, 2007). Similarly, when learning tree-structured Bayesian networks (Mononen and Myllymäki, 2008a), we can speed-up some parts of the computation, but not all.

On the other hand, there now exists a slightly different way to define the stochastic complexity, based on the factorized NML (fNML) criterion (Roos et al., 2008). The fNML criterion can be used in the Naive Bayes or in the Bayesian tree case, or even for learning Bayesian networks (DAGs) in general. In this case, the learning criterion factorizes into a product of multinomials, which means that the speed-up offered by our sub-linear algorithm is more apparent.

Acknowledgments

This work was supported in part by the Academy of Finland under the project Civi and by the Finnish Funding Agency for Technology and Innovation under the projects Kukot and PMMA. In addition, this work was supported in part by the IST Programme of the European Community, under the PASCAL Network of Excellence.

References

P. Flajolet and R. Sedgewick. 2005. *Analytic Combinatorics*. Unpublished.

P. Grünwald. 2007. *The Minimum Description Length Principle*. MIT Press.

P. Kontkanen and P. Myllymäki. 2007. A linear-time algorithm for computing the multinomial stochastic complexity. *Information Processing Letters*, 103(6):227–233.

T. Mononen and P. Myllymäki. 2007. Fast NML computation for Naive Bayes models. In V. Corruble, M. Takeda, and E. Suzuki, editors, *Proceedings of the 10th International Conference on Discovery Science*, October.

T. Mononen and P. Myllymäki. 2008a. Computing the NML for Bayesian forests via matrices and generating polynomials. In *Proceedings of the IEEE Information Theory Workshop*, Porto, Portugal, May.

T. Mononen and P. Myllymäki. 2008b. On the multinomial stochastic complexity and its connection to the birthday problem. In *Proceedings of the International Conference on Information Theory and Statistical Learning*, Las Vegas, NV, July.

J. Rissanen. 2007. *Information and Complexity in Statistical Modeling*. Springer.

T. Roos, T. Silander, P. Kontkanen, and Myllymäki P. 2008. Bayesian network structure learning using factorized NML universal models. In *Proceedings of the Information Theory and Applications Workshop*, San Diego, CA, January.

Yu.M. Shtarkov. 1987. Universal sequential coding of single messages. *Problems of Information Transmission*, 23:3–17.

T. Silander, P. Kontkanen, and P. Myllymäki. 2007. On sensitivity of the MAP Bayesian network structure to the equivalent sample size parameter. In R. Parr and L. van der Gaag, editors, *Proceedings of the 23rd Conference on Uncertainty in Artificial Intelligence*, pages 360–367. AUAI Press.

S. Winitzky. 2008. A handy approximation for the error function and its inverse. Unpublished.

Paper 3

Tommi Mononen and Petri Myllymäki:

On Recurrence Formulas for Computing the Stochastic Complexity

In *Proceedings of the International Symposium on Information Theory and its Applications*, ISITA'08 (Auckland, New Zealand), pages 281-286, IEEE, 2008.

International Symposium on Information Theory and its Applications, ISITA2008
Auckland, New Zealand, 7-10, December, 2008

On Recurrence Formulas for Computing the Stochastic Complexity

Tommi Mononen and Petri Myllymäki

Helsinki Institute for Information Technology (HIIT), University of Helsinki, Finland
E-mail: <firstname>.<lastname>@hiit.fi

Abstract

Stochastic complexity is a criterion that can be used for model selection and other statistical inference tasks. Many model families, like Bayesian networks, use multinomial variables as their basic components. There now exists new efficient computation methods, based on generating functions, for computing the stochastic complexity in the multinomial case. However, the theoretical background behind these methods has not been been extensively formalized before. In this paper we define a bivariate generating function framework, which makes the problem setting more comprehensible. Utilizing this framework, we derive a new recurrence relation over the values of a multinomial variable, and show how to apply the recurrence for computing the stochastic complexity. Furthermore, we show that there cannot be a generic homogeneous linear recurrence over data size. We also suggest that the presented form of the marginal generating function, which is valid in the multinomial case, may also generalize to more complex cases.

1. Introduction

Minimum Description Length (MDL) is an information-theoretic principle for statistical inference [12]. A central concept in this framework is *stochastic complexity*. Given a parametric model $\mathcal{M}$, the stochastic complexity of a discrete observed data matrix $\mathbf{x}^n$ with n data vectors is

$$SC(\mathbf{x}^n \mid \mathcal{M}) = -\log P_{NML}(\mathbf{x}^n \mid \mathcal{M}) \qquad (1)$$

where

$$P_{NML}(\mathbf{x}^n \mid \mathcal{M}) = \frac{P(\mathbf{x}^n \mid \hat{\theta}(\mathbf{x}^n), \mathcal{M})}{\sum_{\mathbf{y}^n} P(\mathbf{y}^n \mid \hat{\theta}(\mathbf{y}^n), \mathcal{M})}. \qquad (2)$$

The above probability, where $\hat{\theta}(\mathbf{x}^n)$ denotes the maximum likelihood parameters of the model, defines the normalized maximum likelihood (NML) distribution [14]. The model selection task is to search from a fixed model family (a set of parametric models with varying complexity) the model minimizing the stochastic complexity (1).

Many probabilistic models, like for example Bayesian networks, use the multinomial variable as an important building block. However, even in this simple case computing the stochastic complexity is difficult, as the denominator of (2) (denoted in the sequel by $\mathcal{C}(L, n)$), requires summation over all the data tables $\mathbf{y}^n$ that are of the same size as our observed data. For a multinomial variable the normalizing sum is

$$\mathcal{C}(L, n) = \sum_{h_1+\cdots+h_L=n} \frac{n!}{h_1! \cdots h_L!} \prod_{k=1}^{L} \left(\frac{h_k}{n}\right)^{h_k}, \qquad (3)$$

where h_k is a number of data points assigned to the kth value and L is the number of values of the multinomial variable. Using this definition directly is obviously not feasible, but there now exist several efficient linear time algorithms for this task [6, 11] and also a very good asymptotic approximation [4] and for fixed precision there is even a sub-linear time computation method [9]. Derivation of efficient computation methods for the multinomial normalizing sum utilize the one-parameter generating function family:

$$\mathcal{B}^L(z) = \left(\frac{1}{1 - T(z)}\right)^L = \sum_{n=0}^{\infty} \mathcal{C}(L, n) n^n \frac{z^n}{n!}, \qquad (4)$$

where $T(z)$ is the tree function [5, 4]. For example, there exists an efficient recurrence formula

$$\mathcal{C}(L, n) = \mathcal{C}(L-1, n) + \left(\frac{n}{L-2}\right) \mathcal{C}(L-2, n), \qquad (5)$$

which can be used for computing multinomial normalizing sums, if the corresponding binomial normalizing sum ($L = 2$) is computed first, which can be done in $\mathcal{O}(n)$ time using the definition with a trivial parameter substitution trick [6].

However, the generating function and the recurrence formula seem to be mismatching pairs: the coefficient sequence in (4) goes over the variable n, while (5) goes over the variable L. Next we redefine the problem so that this ambiguity disappears.

2. Bivariate Generating Function

The multinomial normalizing sum has two parameters L and n, hence we are looking for a bivariate generating function [2]. First we however define the tree function properly [5]:

$$T(z) = \sum_{n=1}^{\infty} n^{n-1} \frac{z^n}{n!} \tag{6}$$

and it satisfies

$$T(z) = ze^{T(z)}. \tag{7}$$

Now we can return to the original problem and write the bivariate generating function in the form

$$\sum_{L=0}^{\infty} \mathcal{B}^L(z) u^L = \sum_{L=0}^{\infty} \left(\frac{1}{1-T(z)} \right)^L u^L \tag{8}$$

$$= \sum_{L=0}^{\infty} \sum_{n=0}^{\infty} \mathcal{C}(L,n) n^n \frac{z^n}{n!} u^L. \tag{9}$$

We want to have (9) in closed form. Since we know that

$$\frac{1}{1-au} = 1 + au + a^2u^2 + a^3u^3 + \cdots,$$

we can just replace a by $\mathcal{B}(z)$ and get

$$f(z,u) = \sum_{L=0}^{\infty} \sum_{n=0}^{\infty} \mathcal{C}(L,n) n^n \frac{z^n}{n!} u^L \tag{10}$$

$$= \frac{1}{1-\frac{u}{1-T(z)}} = \frac{T(z)-1}{T(z)-1+u}. \tag{11}$$

This function is the bivariate exponential generating function. Now we have the bivariate formal power series with two variable coefficients. If we arrange coefficients $\mathcal{C}(L,n)n^n$ into the form of a table, we can ask, which generating function generates coefficients of some row or column? We adopt terms and notation used in [2] and call these marginal generating functions *vertical* and *horizontal generating functions*. Now we start looking at how to compute the marginal generating functions of (11). The vertical generating function family is now our original function $\mathcal{B}^L(z)$, which we denote from now on by

$$f^{\langle L \rangle}(z) = \sum_{n=0}^{\infty} \mathcal{C}(L,n) n^n \frac{z^n}{n!} = \left(\frac{1}{1-T(z)} \right)^L. \tag{12}$$

The horizontal generating function family is a previously unknown function, although (5) applies to its sequence of coefficients. However, to derive this function, we need first the so called Lagrange inversion formula [13].

Proposition 1 *The Lagrange inversion formula is*

$$[z^n] G(\overline{H}(z)) = [z^n] G(z) H'(z) \left(\frac{H(z)}{z} \right)^{-n-1},$$

where $G(z)$ is any formal power series and $H(z)$ is any formal power series with the zero constant term and $\overline{H}(z)$ is the compositional inverse of $H(z)$.

Above we denoted the coefficient extraction by standard notation $[z^n]$. Now we are ready to present the previously missing marginal generating function family in a form of theorem.

Theorem 1 *The horizontal generating function family is of the form*

$$f_n(u) = \sum_{L=0}^{\infty} \mathcal{C}(L,n) n^n u^L$$

$$= n^n u \left(1 + \left(\frac{u}{1-u} \right) \sum_{L=0}^{n} \frac{n! n^{-L}}{(n-L)!} \cdot (1-u)^{-L} \right).$$

Proof 1 *We will utilize the Lagrange inversion, but first we have to find two functions that form a composite function:*

$$\frac{1}{1-\frac{u}{1-T(z)}} = \frac{1}{1-\frac{u}{1-\overline{H}(z)}} = G(\overline{H}(z), u),$$

thus the functions are

$$G(z,u) = \frac{1}{1-\frac{u}{1-z}} = \frac{z-1}{z-1+u} \quad \textit{and}$$

$$\overline{H}(z) = T(z) \quad \textit{and} \quad H(z) = \frac{z}{e^z}.$$

Now we are ready to use the Lagrange inversion formula. We do the Lagrange inversion only with respect to variable z. The dummy variable u is only in the outer function, so we can write in this case

$$[z^n] \frac{1}{1-\frac{u}{1-T(z)}} = [z^n] G(z,u) H'(z) \left(\frac{H(z)}{z} \right)^{-n-1}$$

$$= [z^n] \frac{z-1}{z-1+u} e^{-z} (1-z) \left(\frac{1}{e^z} \right)^{-n-1}$$

$$= [z^n] \frac{(z-1)^2 e^{nz}}{1-z-u}.$$

Next we will do series expansion for the last form. We split the form in two parts:

$$S(z,u) = \frac{(z-1)^2}{1-z-u} = \frac{z^2-2z+1}{1-z-u} \quad \textit{and}$$

$$R(z) = e^{nz}.$$

The series expansions with respect to variable z are

$$S(z,u) = (u+1) - z + \sum_{k=0}^{\infty} \frac{u^2}{(1-u)^{k+1}} z^k \quad \textit{and}$$

$$R(z) = \sum_{k=0}^{\infty} n^k \frac{z^k}{k!}.$$

The final step is just to use the discrete convolution formula between $S(z,u)$ and $R(z)$:

$$\begin{aligned}[z^n]S(z,u)R(z) &= \frac{n^n}{n!}(u+1) - \frac{n^{n-1}}{(n-1)!} \\ &\quad + \frac{u^2}{1-u} \sum_{L=0}^{n} \frac{n^{n-L}}{(n-L)!(1-u)^L} \\ &= \frac{n^n u}{n!} + \frac{u^2 n^n}{1-u} \sum_{L=0}^{n} \frac{n^{-L}}{(n-L)!}(1-u)^{-L},\end{aligned}$$

which is identical to the claim when multiplied by $n!$ (because the vertical generating function family is of exponential type). □

This result can be represented also in another form using confluent hypergeometric functions:

$$f_n(u) = n^n u + \left(\frac{n^n u^2}{1-u}\right) {}_2F_0\left(\begin{matrix}1, -n \\ — \end{matrix}\middle| -\frac{1}{n(1-u)}\right).$$

For further information on ${}_2F_0$ functions, see [11].

This one-parameter family does not look very nice at first sight, but if we enter some values, for example $n = 1, 2, 3$, we observe that the horizontal generating functions in the simplified form are

$$f_1(u) = \sum_{L=0}^{\infty} \mathcal{C}(L,1) 1^1 u^L = \frac{u}{(1-u)^2},$$

$$f_2(u) = \sum_{L=0}^{\infty} \mathcal{C}(L,2) 2^2 u^L = \frac{-2u(u-2)}{(1-u)^3} \quad \text{and}$$

$$f_3(u) = \sum_{L=0}^{\infty} \mathcal{C}(L,3) 3^3 u^L = \frac{3u(3u^2-10u+9)}{(1-u)^4}.$$

The formal power series coefficients of these functions give all the normalizing sums for a fixed amount of data. These example functions, and the horizontal generating functions in general, are so called rational generating functions. This observation enables us to use the huge mathematical toolbox designed for rational generating functions, and may be a reason why the multinomial normalizing sum has such nice properties with respect to variable L. Next we present direct implications of the redefinition.

2.1. Recurrence Relations over L

There are two known recurrence relations over variable L: the first one is (5) and the second one is actually a simple sequential convolution. The second method is based on the observation that if we have a formal power series raised to some positive integer power, we get expanded coefficients by computing many convolutions sequentially. This result applies to $f^{\langle L \rangle}(z)$.

However, our new horizontal generating function family gives the third recurrence relation. There is also a standard way to construct a recurrence relation for rational functions using Taylor series. One can say loosely that the numerator of a rational function gives initial conditions and the denominator gives the recurrence equation. Hence, in this case we get for $f_n(u)$ the recurrence equation

$$\sum_{j=0}^{n+1} \binom{n+1}{j} (-1)^j \mathcal{C}(L-j,n) = 0, \tag{13}$$

which we can write in a simpler operator form

$$(\nabla_L)^{n+1} \mathcal{C}(L,n) = 0, \tag{14}$$

where ∇_L is the backward difference operator with respect to variable L. We can write $\nabla_L = I - E_L^{-1}$, where I is the identity operator and E is the shift operator. If we apply the operator to $\mathcal{C}(L,n)$, the identity operator gives the same $\mathcal{C}(L,n)$, and the shift operator E_L^{-1} gives $\mathcal{C}(L-1,n)$. For example, if $n = 2$, we have

$$\begin{aligned}(\nabla_L)^3 \mathcal{C}(L,2) &= (I - E_L^{-1})^3 \mathcal{C}(L,2) \\ &= (I - 3E_L^{-1} + 3E_L^{-2} - E_L^{-3})\, \mathcal{C}(L,2) \\ &= \mathcal{C}(L,2) - 3\mathcal{C}(L-1,2) \\ &\quad + 3\mathcal{C}(L-2,2) - \mathcal{C}(L-3,2) \\ &= 0.\end{aligned}$$

The recurrence is valid for $f_2(u)$. In fact, also the general form of the backward difference operator is working. This means that we can skip coefficients and for example take every third one. So the general form of the operator is $\nabla_L^b = I - E_L^{-b}$. This leads to a fast algorithm for computing normalizing sums with huge values of L and a small fixed value of n. For example, for $L = n \cdot 2^m$, where $m \in \mathbb{N}$, we can compute the normalizing sums in following way: First compute the initial n values, then use the above recurrence with $b = 1$. After $2n$ values, use the recurrence with $b = 2$ and after $4n$ values use the recurrence with $b = 3$ etc. So we always make bigger and bigger leaps until we reach the desired target L. Of course we can modify the scheme so that we can reach any desired value of L using a similar approach.

This recurrence is not very useful in our framework: it starts to work after the number of values in a variable exceeds the number of data points. So it actually tells us how the value of the normalizing sum changes, if we add excess bins (values). What is more important is that the same recurrence applies also to more complex cases, like the Naive Bayes model discussed below.

2.2. Recurrence Relations over n

There are many different recurrence relations over L. For single horizontal generating functions we have the convolution recurrence and (13). For the whole horizontal family we have (5). However, there does not exist a single efficient recurrence formula over n. The existence of a simple recurrence formula over n would lead to efficient dynamic programming methods. Unfortunately we are not going to present any recurrence formulas over n; actually, in the following we prove that in this case, it is impossible to have a homogeneous linear recurrence relation for the family of vertical generating functions.

We start by definitions. We define the basic concepts first in the single variable case and later expand definitions to the multivariate case. A sequence is called *holonomic* if it satisfies a *homogeneous linear recurrence equation*

$$p_0(i)a(i) + p_1(i)a(i+1) + \cdots + p_d(i)a(i+r) = 0 \qquad n \geq 0, \quad r \in \mathbb{N}, \tag{15}$$

where the terms $p_k(i)$ are polynomials, which are not all zero. Respectively, a formal power series is holonomic (D-finite) if and only if its coefficient sequence is holonomic (P-recursive). The following proposition defines holonomicity condition between ordinary and exponential generating functions [1].

Proposition 2 *The ordinary generating function* $\sum_{n=0}^{\infty} g_n z^n$ *is holonomic if and only if the exponential generating function* $\sum_{n=0}^{\infty} g_n \frac{z^n}{n!}$ *is holonomic.*

Now we are ready to give the proposition that is connected to our specific problem [3]:

Proposition 3 *Generating function* $\sum_{n=0}^{\infty} n^n z^n$ *is not holonomic.*

The two previous propositions together imply that also $f^{\langle 1 \rangle}(z)$ is non-holonomic. So there is no homogeneous linear recurrence relation, which gives the next coefficient given a fixed number of previous coefficients.

Until this point, we had a formal power series only with one variable. Now we need bivariate power series, because we want to prove that the bivariate generating function is non-holonomic. In general we have a formal multivariate power series

$$g(x_1, \ldots, x_2) = \sum_{i_1, \cdots, i_m} a(i_1, \ldots, i_m) x_1^{i_1} \cdots x_m^{i_m}.$$

The *section* of $g(x_1, \ldots, x_m)$ is a lower dimensional formal power series, where some variables have been assigned to certain values:

$$g^{1,\ldots,s}_{i_{s+1},\ldots,i_m}(x_1, \ldots, x_m) = \sum_{i_1,\ldots,i_s} a(i_1, \ldots, i_m) x_1^{i_1} \cdots x_s^{i_s},$$

where $(i_{s+1}, \ldots, i_m)$ are the assigned values. Now the definition of holonomicity for multivariate formal power series says that if $g(x_1, \ldots, x_m)$ is holonomic then all the sections must be holonomic [7]. This leads to the following theorem:

Theorem 2 *The bivariate generation function f(z,u) is non-holonomic.*

Proof 2 *All sections of a holonomic function must be holonomic. Now take the section*

$$f_1^n(z,u) = \sum_{n=0}^{\infty} \mathcal{C}(1,n) n^n \frac{z^n}{n!} = \sum_{n=0}^{\infty} n^n \frac{z^n}{n!} = f^{\langle 1 \rangle}(z),$$

which is non-holonomic. Therefore the bivariate generating function is also non-holonomic. □

If the bivariate generating function would have been holonomic, it would have implied that there is also a homogeneous linear recurrence over n. However now the situation is a bit more complicated. We cannot infer converse, meaning that there is no linear homogeneous recurrence, but we need a proposition that says how to end up to multivariate holonomic sequences [7]:

Proposition 4 *Let the sequence* $a(i_1, \ldots, i_m)$ *satisfy a system of recurrences, one for each* $j = 1 \ldots m$*, of the form*

$$p_0^{(j)}(i_j)a(i_1, \ldots, i_m) + \sum_{l=1}^{r_j} p_l^{(j)}(i_1, \ldots, i_m)a(i_1, \ldots, i_{j-1}, i_j - l, i_{j+1}, \ldots, i_m) = 0,$$

where $p_0^{(j)}$ *are nonzero polynomials of one variable. Then the sequence* $a(i_1, \ldots, i_m)$ *is holonomic.*

Intuitively this means that if there exists the above recurrence equations with respect to each variable, then we have a multivariate holonomic function. This leads to the following theorem:

Theorem 3 *For the family of vertical generating functions of $f(z,u)$ there is no recurrence equation of the form*

$$\sum_{l=0}^{r_2} p_l(L,n)\mathcal{C}(L,n-l)=0,$$

where r_2 is some non-negative integer and $p_0(L,n)$ is non-zero.

Proof 3 *We can use Proposition 4. First we notice that (5) can be written in form*

$$(L-2)\mathcal{C}(L,n)+(2-L)\mathcal{C}(L-1,n)\\ +(-n)\mathcal{C}(L-2,n)=0,$$

which means that the first recurrence equation exists. Now suppose that also the second recurrence exists. This implies that the sequence $\mathcal{C}(L,n)n^n$ is holonomic. However, this is a contradiction because we know that the sequence is not holonomic. So, the second recurrence must be false.

Our only concern is now that by Proposition 4 we have $p_0(n)$ instead of $p_0(L,n)$. However, if there would be such a recurrence, then by setting $L=1$ in the recurrence equation we would get a homogeneous linear recurrence for the non-holonomic $f^{\langle 1\rangle}(z)$, which is of course a contradiction. □

This result could have been proved without the bivariate holonomicity, but we wanted to bring insight to the problem setting. There still may be some other type of recurrence formulas, but finding one can be a rather difficult task as a general methodological foundation is mostly missing.

3. Recurrences in the Naive Bayes Case

The Naive Bayes classifier is a probabilistic model used widely in classification and clustering tasks. The model has m predictor variables with K_i outcomes for the ith variable, and one class variable with L outcomes. The predictor variables are assumed to be independent given the value of the class variable.

We already know that the convolution type recurrence works for the normalizing sums also in the Naive Bayes case [8]. Validity is easy to see from the fact that we can write the 'vertical' generating function family in the form

$$\left(\sum_{n=0}^{\infty}\mathcal{C}(K_1,n)\mathcal{C}(K_2,n)\cdots\mathcal{C}(K_m,n)n^n\frac{z^n}{n!}\right)^L$$

$$=\sum_{n=0}^{\infty}\mathcal{C}_{NB}(L,K_1,\ldots,K_m,n)n^n\frac{z^n}{n!}. \qquad (16)$$

We just expand the power L sequentially and compute convolutions as we did in the multinomial case. A far more interesting fact is that our new recurrence formula seems to be valid also in the Naive Bayes case. In the following, we list some consequences of this conjecture.

It is hardly surprising that the operator equation works also over L. In this case the equation is

$$(\nabla_L)^{n+1}\mathcal{C}_{NB}(L,K_1,\ldots,K_m,n)=0. \qquad (17)$$

An astonishing fact is that the same equation works also over the number of outcomes of the predictor variables. This is the first recurrence equation of this kind. The equation in this case is

$$(\nabla_{K_i})^{n+1}\mathcal{C}_{NB}(L,K_1,\ldots,K_i,\ldots,K_m,n)=0. \qquad (18)$$

Using the conjecture we can construct 'horizontal' generating functions also for Naive Bayes models. We just need to compute some initial values and expect validity of recurrence equations to get the sought generating functions. We used Maple command `rectodiffeq` and solved the generating function from the result. In empirical tests we did not found any flaws. The generating functions seem to be the correct ones.

We give an example. Let us take a Naive Bayes model whose root node has 5 values and the three leaf nodes 3, 3 and K_3 values. For values $n=1,2,3$, the generating functions are

$$f_1(u)=45\frac{u}{(1-u)^2},$$
$$f_2(u)=\frac{405}{2}\frac{u(7u+10)}{(1-u)^3} \quad \text{and}$$
$$f_3(u)=\frac{5}{27}\frac{u(126525u^2+1152890u+497673)}{(1-u)^4}.$$

We can see that the denominators are of course same as in the multinomial case for these few instances. Moreover, also the horizontal generating functions are quite similar, only the coefficients of the numerators are different.

4. Using Horizontal Generating Functions

Let us suppose that we have precomputed the expanded form of a horizontal generating function for some model and fixed n. Now we can compute the normalizing values using convolution. The series expansion of the denominators of horizontal generating functions is

$$\frac{1}{(1-u)^{n+1}}=\sum_{j=0}^{\infty}\binom{j+n}{j}u^j=\sum_{j=0}^{\infty}h_j u^j. \qquad (19)$$

and the coefficients of the series can be computed efficiently using the ratio

$$\frac{h_j}{h_{j-1}} = \frac{j+n}{j}. \tag{20}$$

We just have to do the discrete convolution for coefficient sequences of (19) and the numerator. This way we can test different number of values in a single leaf (predictor) variable of a Naive Bayes model very fast. We have also noticed that leaf nodes of Bayesian trees have similar kind of horizontal generating functions. This may generalize also to inner and root nodes, but our present tree computation algorithm does not allow us to test this hypothesis.

However, as the computation has been previously shown to involve higher level convolutions with respect to sequences of length n [8, 10], we expect there to be hidden costs. We are still hoping that there could be efficient shortcuts with respect to a tree structure, which would allow us to construct the horizontal generating functions efficiently. However, it is most likely that bivariate generating functions are not descriptive enough for more complex models, but in this case we need multivariate generating functions.

5. Conclusions

We redefined the generating function for the multinomial normalizing sum of NML. The previously unknown marginal generating function family has interesting properties and it may eventually lead to development of efficient algorithms for computing the stochastic complexity of Bayesian trees and more general Bayesian models. We also proved that for computing the multinomial normalizing sum, there does not exist a generic homogeneous linear recurrence formula over the data size.

Acknowledgments

Authors thanks Alessandro Di Bucchianico and Petri Kontkanen for fruitful discussions. This work was supported in part by the Academy of Finland under the project Civi and by the Finnish Funding Agency for Technology and Innovation under the projects Kukot and PMMA. In addition, this work was supported in part by the IST Programme of the European Community, under the PASCAL Network of Excellence.

References

[1] C. Banderier, M. Bousquet-Mélou, A. Denise, P. Flajolet, D. Gardy, and D. Gouyou-Beauchamps. Generating functions for generating trees. *Discrete Mathematics*, 246(1-3):29–55, March 2002.

[2] P. Flajolet and R. Sedgewick. *Analytic Combinatorics*. Unpublished, 2005.

[3] S. Gerhold. *Combinatorial Sequences: Non-Holonomicity and Inequalities*. PhD thesis, Johannes Kepler University, Linz, 2005.

[4] P. Jacquet and W. Szpankowski. Markov types and minimax redundancy for Markov sources. *IEEE Transactions on Information Theory*, 50(7):1393–1402, 2004.

[5] D.E. Knuth. Convolution polynomials. *Mathematica Journal*, 2(4):67–78, Fall 1992.

[6] P. Kontkanen and P. Myllymäki. A linear-time algorithm for computing the multinomial stochastic complexity. *Information Processing Letters*, 103(6):227–233, 2007.

[7] L. Lipshitz. D-finite power series. *Journal of Algebra*, 122:353–373, 1989.

[8] T. Mononen and P. Myllymäki. Fast NML computation for Naive Bayes models. In V. Corruble, M. Takeda, and E. Suzuki, editors, *Proceedings of the 10th International Conference on Discovery Science*, October 2007.

[9] T. Mononen and P. Myllymäki. Computing the multinomial stochastic complexity in sub-linear time. In *Proceedings of the 4th European Workshop on Probabilistic Graphical Models*, 2008.

[10] T. Mononen and P. Myllymäki. Computing the NML for Bayesian forests via matrices and generating polynomials. In *Proceedings of the IEEE Information Theory Workshop*, Porto, Portugal, May 2008.

[11] T. Mononen and P. Myllymäki. On the multinomial stochastic complexity and its connection to the birthday problem. In *Proceedings of the International Conference on Information Theory and Statistical Learning*, Las Vegas, NV, July 2008.

[12] J. Rissanen. *Information and Complexity in Statistical Modeling*. Springer, 2007.

[13] S. Roman. *The Umbral Calculus*. Dover, 2005.

[14] Yu.M. Shtarkov. Universal sequential coding of single messages. *Problems of Information Transmission*, 23:3–17, 1987.

Paper 4

Tommi Mononen and Petri Myllymäki:

Fast NML Computation for Naive Bayes Models

In *Proceedings of the 10th International Conference on Discovery Science*, DS'07 (Sendai, Japan), pages 151-160, Springer, 2007.

Fast NML Computation for Naive Bayes Models

Tommi Mononen and Petri Myllymäki

Complex Systems Computation Group (CoSCo)
Helsinki Institute for Information Technology (HIIT)
University of Helsinki & Helsinki University of Technology
P.O.Box 68 (Department of Computer Science)
FIN-00014 University of Helsinki, Finland
{Firstname.Lastname}@hiit.fi

Abstract. The *Minimum Description Length* (MDL) is an information-theoretic principle that can be used for model selection and other statistical inference tasks. One way to implement this principle in practice is to compute the *Normalized Maximum Likelihood* (NML) distribution for a given parametric model class. Unfortunately this is a computationally infeasible task for many model classes of practical importance. In this paper we present a fast algorithm for computing the NML for the Naive Bayes model class, which is frequently used in classification and clustering tasks. The algorithm is based on a relationship between powers of generating functions and discrete convolution. The resulting algorithm has the time complexity of $\mathcal{O}(n^2)$, where n is the size of the data.

1 Introduction

The information-theoretical *Minimum Description Length* (MDL) principle [15, 4, 17, 3] for model selection is based on the conceptually simple idea that given a data set, the best model for the data is the one which results in the shortest description for the data together with the model. Hence, we wish to select a model representing a balance between too simple models (in which case the code length for the data is large) and too complex models (in which case the code length for the data is small, but for the model itself large).

Consider a parametric probabilistic model class, i.e., a set of models each defining a probability distribution over all possible data sets. Let us call the shortest possible code length obtainable with the given set of models *stochastic complexity*. Consequently, given a data set, we can choose between alternative parametric models (model classes) with different number of parameters by comparing the corresponding stochastic complexities for the given data.

However, there remains the question of how to formally define the stochastic complexity for a model class. As each of the probability distributions in a probabilistic model class corresponds to a code length, it is obvious that no code can be shorter than all the other codes in the model class for all data sets, because no probability distribution can dominate another probability distribution over all data sets. A *universal model* is a model (code) which can imitate any model

V. Corruble, M. Takeda, and E. Suzuki (Eds.): DS 2007, LNAI 4755, pp. 151–160, 2007.

in a given parametric model class. The *normalized maximum likelihood (NML)* distribution [16, 18] is the *worst-case optimal* universal model giving a desired formal definition for the stochastic complexity (see the next Section).

Unfortunately, computing the NML is very difficult for many model classes of practical interest. In this paper we consider Bayesian networks, probabilistic model classes defined by acyclic directed graphs [14, 5]. The Naive Bayes model is a simple Bayesian network, which is continuously used with success in areas such as clustering and classification. It has been earlier shown [11] how to compute the NML for the Naive Bayes model family in $\mathcal{O}(n^2 \log L)$ time, where L denotes the number of values of the class variable of the Naive Bayes model. In this paper we introduce a faster $\mathcal{O}(n^2)$ algorithm for this task, based on generating functions.

2 The Problem

The normalized maximum likelihood (NML) distribution [16, 18] is defined by

$$P_{NML}(\mathbf{x}^n|\mathcal{M}) = \frac{P(\mathbf{x}^n|\hat{\theta}(\mathbf{x}^n, \mathcal{M}))}{\sum_{\mathbf{y}^n} P(\mathbf{y}^n|\hat{\theta}(\mathbf{y}^n, \mathcal{M}))}, \tag{1}$$

where the numerator is the maximum likelihood for the observed data $\mathbf{x}^n$ within the model class $\mathcal{M}$. The *normalizing term* in the denominator is the sum over maximum likelihoods of all possible data sets of size n, with respect to the model class. As shown in [18], this yields the worst case universal distribution with respect to the model class $\mathcal{M}$.

Although NML was defined as the worst-case optimal universal model, without considering model complexity regularization, it is interesting to note how it behaves as a model class selection criterion. Namely, if the model class is very complex, then the maximum likelihood for the given data (the numerator in (1)) is large, but so is also the denominator as a complex model gives a high maximum likelihood for many data sets. For simple model classes the sum in the denominator is small, but so is the numerator. Consequently, the denominator behaves a a regularization term, and the model class optimizing the stochastic complexity $-\log(P_{NML}(\mathbf{x}^n|\mathcal{M}))$ has to balance between model complexity and fit to the given data.

Let us now consider Naive Bayes models. The *Naive Bayes* is a Bayesian network with one root node and m leaf nodes attached to the root node. Variables related to nodes are multinomially distributed. The joint distribution corresponding to the Naive Bayes is defined by

$$P(\mathbf{x}) = P(x_0) \prod_{i=1}^{m} P(x_i|x_0), \tag{2}$$

where $\mathbf{x} = (x_0, x_1, \ldots, x_m)$ is a vector of variable value assignments and x_0 is the value in the root.

For Naive Bayes models, computing the numerator of (1) is trivial, but this is not the case with the denominator. In this paper we derive an efficient algorithm

for computing the normalizing term for this model family and call it the *Naive Bayes normalizing term*.

3 Generating Functions and the Naive Bayes

The normalizing term for a single multinomial variable is called *multinomial normalizing term*. We can compute the multinomial normalizing term efficiently: the most efficient known method is proved using generating functions [9, 10]. We now use this same methodology in the Naive Bayes case. First we have to define the needed operations, which we use with generating functions, and then take a closer look at the Naive Bayes normalizing term.

3.1 Generating Functions

An *ordinary generating function* (OGF) of a sequence a_n is

$$F(z) = \sum_{n=0}^{\infty} a_n z^n = a_0 + a_1 z + a_2 z^2 + \cdots , \tag{3}$$

where $z \in \mathbb{C}$ [2]. We are only interested in coefficients a_n, not the value of the function $F(z)$ itself. The function $F(z)$ is only used for computation of some a_n coefficients or in derivation of recurrence formulas. With a *recurrence formula* we can compute the coefficient a_{n+1} with the help of the fixed and finite set of previous coefficients. A generating function may have a closed form, in which case manipulation is easier.

As a generating function is also a formal power series, all general formal power series operations are applicable. In the case of the multinomial normalizing term, however, we need the *exponential generating function* (EGF), which is of form

$$G(z) = \sum_{n=0}^{\infty} b_n \frac{z^n}{n!}. \tag{4}$$

We need to define for later use also two operations: a *coefficient extraction* from a formal power series and taking the power of the exponential generating function. The first operation is defined by

$$[z^n]G(z) = \frac{b_n}{n!}, \tag{5}$$

which means that $[z^n]$ gives us the coefficient of term z^n. The second operation defines what happens to coefficients when we exponentiate the generating function. Rising the generating function $G(z)$ to the power of two, denoted by

$$G^2(z) = \left(\sum_{n=0}^{\infty} b_n \frac{z^n}{n!} \right)^2 = \sum_{n=0}^{\infty} c_n \frac{z^n}{n!}, \tag{6}$$

corresponds to the binomial convolution

$$c_n = \sum_{h=0}^{n} \binom{n}{h} b_h b_{n-h} \tag{7}$$

in the level of coefficients. Similarly, the power of L gives

$$d_n = \sum_{h_1+\cdots+h_L=n} \left(\frac{n!}{h_1!h_2!\cdots h_L!} \right) b_{h_1} b_{h_2} \cdots b_{h_L}, \tag{8}$$

which is the multinomial convolution [2, 8]. This relation between the *expanded form* and the *power form* is the key feature for achieving a new, more efficient algorithm for computing the Naive Bayes normalizing term.

3.2 Naive Bayes Generating Function in the Power Form

First we have to define the generating function for the multinomial normalizing term. We do not give this function in the expanded form, but use a more compact notation: Lth power of a generating function [9, 10]. The power form is

$$\mathcal{B}^L(z) = \left(\sum_{n=0}^{\infty} n^n \frac{z^n}{n!} \right)^L. \tag{9}$$

We call the series inside the parentheses a *basic series*. The basic series here is of exponential type and formal power series coefficients are now $\frac{n^n}{n!}$. Coefficients of the exponential generating function are n^n. When we expand power L, we get an exponential generating function

$$\mathcal{B}^L(z) = \sum_{n=0}^{\infty} \mathcal{C}_{MN}(L, n) n^n \frac{z^n}{n!}, \tag{10}$$

where $\mathcal{C}_{MN}(L, n)$ is the multinomial normalizing term with L values and n data vectors [9, 10]. By a strict definition of generating functions this is not such a function, as it is not explicitly defined. However, we misuse the definition here slightly and in same way also later in the Naive Bayes case, because the implicit form is sufficient for our purposes. There are efficient ways to compute the term $\mathcal{C}_{MN}(L, n)$, and we will show one of them later.

Now we focus on the Naive Bayes normalizing term. The normalizing term is represented in the previous papers only using the expanded form. We denote the Naive Bayes normalizing term by $\mathcal{C}_{NB}(L, K_1, \ldots, K_m, n)$, where L is the number of values of the root variable and K_i is the number of values in leaf variable i. The following theorem shows the simple power form of the normalizing term.

Theorem 1

$$\frac{n^n}{n!} \mathcal{C}_{NB}(L, K_1, \ldots, K_m, n) = [z^n] \left(\sum_{n=0}^{\infty} n^n \left(\prod_{i=1}^{m} \mathcal{C}_{MN}(K_i, n) \right) \frac{z^n}{n!} \right)^L.$$

Proof. A vector $(h_1, \ldots, h_L)$ is a sufficient statistics i.e. data counts of the root variable. The used formula for the Naive Bayes normalizing term is from the paper [11]. With standard manipulation we get

$$\frac{n^n}{n!}\mathcal{C}_{NB}(L, K_1, \ldots, K_m, n) \tag{11}$$

$$= \frac{n^n}{n!} \sum_{h_1+\cdots+h_L=n} \frac{n!}{h_1! \cdots h_L!} \left(\prod_{k=1}^{L} \left(\frac{h_k}{n} \right)^{h_k} \right) \prod_{i=1}^{m} \prod_{k=1}^{L} \mathcal{C}_{MN}(K_i, h_k) \tag{12}$$

$$= \frac{n^n}{n!} \sum_{h_1+\cdots+h_L=n} \frac{n!}{h_1! \cdots h_L!} \left(\frac{1}{n^n} \prod_{k=1}^{L} h_k^{h_k} \right) \prod_{k=1}^{L} \left(\prod_{i=1}^{m} \mathcal{C}_{MN}(K_i, h_k) \right) \tag{13}$$

$$= \frac{1}{n!} \sum_{h_1+\cdots+h_L=n} \frac{n!}{h_1! \cdots h_L!} \prod_{k=1}^{L} \left(h_k^{h_k} \prod_{i=1}^{m} \mathcal{C}_{MN}(K_i, h_k) \right) \tag{14}$$

$$= [z^n] \left(\sum_{n=0}^{\infty} n^n \left(\prod_{i=1}^{m} \mathcal{C}_{MN}(K_i, n) \right) \frac{z^n}{n!} \right)^L . \tag{15}$$

The last form is the power form, from where we can easily extract the basic series. We started from the expanded form and ended up with the power form. □

Let us compare the generating functions of the multinomial and the Naive Bayes normalizing terms. In the multinomial case we have

$$\left(\sum_{n=0}^{\infty} n^n \frac{z^n}{n!} \right)^L \tag{16}$$

and in the Naive Bayes case we have

$$\mathcal{E}^L = \left(\sum_{n=0}^{\infty} \mathcal{C}_{MN}(K_1, n) \cdots \mathcal{C}_{MN}(K_m, n) n^n \frac{z^n}{n!} \right)^L . \tag{17}$$

The two forms seem to be quite similar, except that in the Naive Bayes case we have additional multinomial normalizing terms inside the basic series terms. These extra multinomial normalizing terms makes the expanded form look quite ugly. However, despite of the complex terms, there exists an $\mathcal{O}(n^2 \log L)$ algorithm for computing the Naive Bayes normalizing term [11]. The basic idea is very simple: we can split the exponent L into two parts. Let's call these parts L^* and $L - L^*$. Then we get $\mathcal{E}^L = \mathcal{E}^{L^*} \mathcal{E}^{L-L^*}$. Now we can simply take the normal discrete convolution in the right hand side to get one term of the series in the left hand side. If we require that the result is also a normalizing term, we get the known recurrence formula

$$\mathcal{C}_{NB}(L, K_1, \ldots, K_m, n) = \sum_{k=0}^{n} \binom{n}{k} \left(\frac{k}{n} \right)^k \left(\frac{n-k}{n} \right)^{n-k} \cdot \mathcal{C}_{NB}(L^*, K_1, \ldots, K_m, k)\, \mathcal{C}_{NB}(L - L^*, K_1, \ldots, K_m, n-k). \tag{18}$$

So two lower exponents produce a higher one. To achieve the $\log L$ -term in the time complexity, we have to merge exponents wisely, so that we do not make any unnecessary steps. For example, if we want to compute $\mathcal{E}^L$ with $L = 16$, we first compute $\mathcal{E}^2$ and then compute $\mathcal{E}^2\mathcal{E}^2$ to get $\mathcal{E}^4$. In the same way we get the series $\mathcal{E}^8$ and finally the series $\mathcal{E}^{16}$. If the target value is not two to some power, then we have to do more complicated multiplications based on same idea. However, in the next section we present a novel, even more efficient way for computing the Naive Bayes normalizing term.

4 Powers of Formal Power Series

As basic series are formal power series, we can use some known powers of formal power series formula. One of these formulas is the *Miller formula* [6]. It is originally a result of Euler and it has time complexity of $\mathcal{O}(n^2)$ for any real number exponent, but of course only natural numbers are meaningful in our case.

The proof of the Miller formula has been sketched many times in history [7, 13, 6], but we were not able to find a detailed proof in the literature. The detailed proof is relatively straightforward and uses only standard manipulation. We complete below the missing parts of Knuth's proof for sake of clarity.

Theorem 2 (The Miller formula). *If two formal power series are* $V(z) = 1 + \sum_{k=1}^{\infty} v_k z^k$ *and* $W(z) = \sum_{k=0}^{\infty} w_k z^k$ *and* $W(z) = (V(z))^\alpha$, $\alpha \in \mathbb{R}$, *then* $w_0 = 1$ *and* $w_n = \sum_{k=1}^{n} \big((\frac{\alpha+1}{n})k - 1\big) v_k w_{n-k}$.

Proof. It is evident that $w_0 = 1$, since $v_0 = 1$ and $1 = 1^\alpha$. Next we derivate the basic equation of the theorem and get

$$W'(z) = \alpha V(z)^{\alpha-1} V'(z).$$

Then we multiply both sides with $V(z)$ and substitute $V(z)^\alpha = W(z)$, which gives us the equation

$$W'(z)V(z) = \alpha W(z)V'(z).$$

Let us now look at the z^{n-1}-coefficients of both sides:

$$[z^{n-1}]W'(z)V(z) = \alpha[z^{n-1}]W(z)V'(z) \tag{19}$$

$$\sum_{k=0}^{n} k w_k v_{n-k} = \alpha \sum_{k=0}^{n} (n-k) w_k v_{n-k} \tag{20}$$

$$\sum_{k=0}^{n-1} k w_k v_{n-k} + n w_n v_0 = \alpha \sum_{k=0}^{n-1} (n-k) w_k v_{n-k} + 0 \tag{21}$$

$$n w_n v_0 = \sum_{k=0}^{n-1} \big(\alpha(n-k) w_k v_{n-k} - k w_k v_{n-k}\big) \tag{22}$$

$$w_n = \frac{1}{n v_0} \sum_{k=0}^{n-1} \big((\alpha(n-k) - k) w_k v_{n-k}\big) \tag{23}$$

$$w_n = \sum_{k=0}^{n-1} \left(\frac{\alpha(n-k)-k}{n} \right) w_k v_{n-k} \tag{24}$$

$$w_n = \sum_{k=1}^{n} \left(\frac{\alpha(n-n+k)-n+k}{n} \right) w_{n-k} v_{n-n+k} \tag{25}$$

$$w_n = \sum_{k=1}^{n} \left(\left(\frac{\alpha+1}{n} \right) k - 1 \right) w_{n-k} v_k. \tag{26}$$

After some straightforward manipulation we get the result. □

We can obviously use this method for computing the normalizing term of the Naive Bayes model. It should be noted that while this result is elegant, if we use the discrete Fourier transform, we can achieve the time complexity $\mathcal{O}(n \log n)$ by using the basic identity $(V(z))^\alpha = \exp(\alpha \log(V(z)))$ and the *fast Fourier transform* (FFT). The FFT method involves utilization of Newton's method and is explained in the paper [1]. However, the usefulness of this approach is unclear as some earlier tests with the multinomial normalizing term [12] show that the used floating point numbers must have very high precision in practical cases. This is due to the fact that the values of the normalizing terms can be quite large, and consequently, as the data size increases, the precision of the floating point numbers must also increase. This means that increasing the precision will affect the efficiency of the algorithm, although the number of operations remains in principle the same.

5 Computation of the Naive Bayes Normalizing Term

The computation of the Naive Bayes normalizing term is quite straightforward given the results derived above. Now we collect these results in a form of an algorithm. As we did not describe earlier how to compute efficiently the multinomial normalizing term, we start by defining that.

5.1 Recurrence Formula for the Multinomial Normalizing Term

The multinomial normalizing term can be computed by using a recurrence formula [9, 10]. Initial values for this formula are

$$\mathcal{C}_{MN}(1,n) = 1 \quad \text{and} \tag{27}$$

$$\mathcal{C}_{MN}(2,n) = \sum_{k=0}^{n} \binom{n}{k} \left(\frac{k}{n} \right)^k \left(\frac{n-k}{n} \right)^{n-k}, \tag{28}$$

where $\mathcal{C}_{MN}(2,n)$ is a *binomial normalizing term*. After this we use the recurrence formula

$$\mathcal{C}_{MN}(L+2,n) = \mathcal{C}_{MN}(L+1,n) + \frac{n}{L} \mathcal{C}_{MN}(L,n) \tag{29}$$

to get the normalizing term with the wanted number of values in a multinomial variable. Time complexity of this whole method is $\mathcal{O}(n)$, as the number of values in a variable is usually much smaller than the number of the data points n. If we have to compute all normalizing terms between $[0, n]$ and we choose to use FFT, then binomial normalizing terms should be computed using (16). For the multiplication we can apply FFT.

5.2 The Algorithm

Now we have all the components we need for our algorithm. As said before, our main theorem is quite obvious given the earlier results (Theorems 1 and 2) and needs no proof.

Theorem 3. *The Naive Bayes normalizing term can be efficiently calculated in following way:*

1. *Compute first $n + 1$ binomial normalizing terms.*
2. *Use the recurrence formula to get the needed multinomial normalizing terms.*
3. *Compute the basic series $\sum_{k=0}^{n} \mathcal{C}_{MN}(K_1, k) \cdots \mathcal{C}_{MN}(K_m, k) k^k \frac{z^k}{k!}$.*
4. *Use the Miller formula to compute a new series, which is the basic series to the power of L.*
5. *Extract the Naive Bayes normalizing terms from the computed series by extracting coefficients and multiplying every coefficient so that the kth coefficient is multiplied by $\frac{k!}{k^k}$.*

Time complexity is $\mathcal{O}(n^2)$ for any exponent, because complexities of the steps are $\mathcal{O}(n^2)$, $\mathcal{O}(n \cdot \max(K_i))$, $\mathcal{O}(n \cdot m)$, $\mathcal{O}(n^2)$ and $\mathcal{O}(n)$, respectively. This way we get all the Naive Bayes normalizing terms between $[0, n]$ in the given time, not just the nth of them. Notice that if the FFT approach could be used, time complexities of the first (explained in the Sect. 5.1) and the fourth steps would become $\mathcal{O}(n \log n)$. In this case the Miller formula in the fourth step is replaced with the algorithm mentioned in Section 4. Theorem 3 gives us actually a general framework for designing this kind of algorithms, as step 4 can be replaced with any exponentiation algorithm.

The method given in Theorem 3 is more efficient than the $\mathcal{O}(n^2 \log L)$-algorithm presented in [11]. This is easy to see, as the previous algorithm essentially performs in the fourth step at minimum $\log L$- power series multiplications instead of something which corresponds just one power series multiplication. In fact even when using the previous method, in some case it can be wise to compute series coefficients and not to require that all sub-results has to be normalizing terms. This way we can replace (18) just with normal convolution and achieve some more efficiency by omitting unnecessary multipliers present in the old formula. Furthermore, the old formula is applicable for values L greater than 2, but we can use normal convolution for values L greater than 1. In the fifth step we then convert wanted series coefficients into normalizing terms.

The new Miller method algorithm works perfectly fine with exact rational numbers. However our preliminary implementations show that in practice this is

not necessarily the case with fixed precision floating point numbers and all formal power series: for some tested basic series, small errors in elementary operations tend to corrupt the normalizing terms very fast as n grows (because the algorithm uses iteratively previous values). Therefore with finite precision floating point numbers, using the previous, slower algorithm may be more advisable.

We have derived an efficient algorithm for computing the Naive Bayes normalizing term exactly. The computational complexity of computing the NML criterion for a Naive Bayes model is the same as for this algorithm, as the numerator of (1) is trivial to compute. Further information on computing the stochastic complexity for Naive Bayes models can be found in papers [11, 12].

6 Concluding Remarks

We presented an $\mathcal{O}(n^2)$ time algorithm for computing the normalizing term of the NML distribution exactly in the case of the Naive Bayes model. As the remaining term of the NML distribution is trivial to compute in this case, this result leads to a computationally efficient algorithm for computing the NML exactly for Naive Bayes models. We also defined a general framework for developing efficient algorithms for the NML computation in the Naive Bayes case and showed how the old $\mathcal{O}(n^2 \log L)$-algorithm can be seen as an special case of the framework, and how to make the algorithm more efficient.

We believe that it is not possible to do formal power series exponentiation in this case faster than $\mathcal{O}(n^2)$ without resorting to the Fast Fourier transform, which would easily lead to numerical problems, as discussed earlier. So unless the basic series for the Naive Bayes model reveals new hidden regularities with respect to exponentiation, our algorithm meets the lower limit of the time complexity for computing the NML exactly for Naive Bayes models.

Acknowledgements

The authors would like thank the anonymous reviewers for constructive comments. This work was supported in part by the Academy of Finland under the project Civi and by the Finnish Funding Agency for Technology and Innovation under the projects Kukot and PMMA. In addition, this work was supported in part by the IST Programme of the European Community, under the PASCAL Network of Excellence, IST-2002-506778. This publication only reflects the authors' views.

References

[1] Brent, R.P.: Multiple-precision zero-finding methods and the complexity of elementary function evaluation. In: Traub, J.F. (ed.) Analytic Computational Complexity, Academic Press, New York (1976)
[2] Flajolet, P., Sedgewick, R.: Analytic Combinatorics (in preparation)

[3] Grünwald, P.: The Minimum Description Length Principle. MIT Press, Cambridge (2007)
[4] Grünwald, P., Myung, J., Pitt, M. (eds.): Advances in Minimum Description Length: Theory and Applications. MIT Press, Cambridge (2005)
[5] Heckerman, D., Geiger, D., Chickering, D.M.: Learning Bayesian networks: The combination of knowledge and statistical data. Machine Learning 20(3), 197–243 (1995)
[6] Henrici, P.: Automatic computations with power series. Journal of the ACM 3(1), 11–15 (1956)
[7] Knuth, D.E.: The Art of Computer Programming, volume. 2: Seminumerical Algorithms, 3rd edn. Addison-Wesley, Reading (1998), ISBN: 0201896842
[8] Knuth, D.E., Pittel, B.: A recurrence related to trees. Proceedings of the American Mathematical Society 105(2), 335–349 (1989)
[9] Kontkanen, P., Myllymäki, P.: Analyzing the stochastic complexity via tree polynomials. Technical Report 2005-4, Helsinki Institute for Information Technology (HIIT) (2005)
[10] Kontkanen, P., Myllymäki, P.: A linear-time algorithm for computing the multinomial stochastic complexity. Information Processing Letters 103(6), 227–233 (2007)
[11] Kontkanen, P., Myllymäki, P., Buntine, W., Rissanen, J., Tirri, H.: An MDL framework for data clustering. In: Grünwald, P., Myung, I.J., Pitt, M. (eds.) Advances in Minimum Description Length: Theory and Applications, MIT Press, Cambridge (2006)
[12] Kontkanen, P., Wettig, H., Myllymäki, P.: NML computation algorithms for tree-structured multinomial Bayesian networks. EURASIP Journal on Bioinformatics and Systems Biology,(to appear)
[13] Nakos, G.: Expansions of powers of multivariate formal power series. Mathematica Journal 3(1), 45–47 (1993)
[14] Pearl, J.: Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference. Morgan Kaufmann, San Mateo (1988)
[15] Rissanen, J.: Stochastic Complexity in Statistical Inquiry. World Scientific, New Jersey (1989)
[16] Rissanen, J.: Fisher information and stochastic complexity. IEEE Transactions on Information Theory 42(1), 40–47 (1996)
[17] Rissanen, J.: Information and Complexity in Statistical Modeling. Springer, Heidelberg (2007)
[18] Shtarkov, Y.M.: Universal sequential coding of single messages. Problems of Information Transmission 23, 3–17 (1987)

Paper 5

Tommi Mononen and Petri Myllymäki:

Computing the NML for Bayesian Forests via Matrices and Generating Polynomials

In *Proceedings of the 2008 IEEE Information Theory Workshop*, ITW'08 (Porto, Portugal), pages 276-280, IEEE, 2008.

Computing the NML for Bayesian Forests via Matrices and Generating Polynomials

Tommi Mononen, Petri Myllymäki
Helsinki Institute for Information Technology, Finland
Email: {firstname.lastname}@hiit.fi

***Abstract*— The *Minimum Description Length* (MDL) is an information-theoretic principle that can be used for model selection and other statistical inference tasks. One way to implement this principle in practice is to compute the *Normalized Maximum Likelihood* (NML) distribution for a given parametric model class. Unfortunately this is a computationally infeasible task for many model classes of practical importance. In this paper we present a fast algorithm for computing the NML for the model class of Bayesian forests, which are graphical dependency models for multi-dimensional domains with the constraint that each node (variable) has at most one predecessor. The resulting algorithm has the time complexity of $\mathcal{O}(n^{2\mathcal{K}+\mathcal{L}-3})$, where n is the number of data vectors, and $\mathcal{K}$ and $\mathcal{L}$ are the maximal number of values (alphabet sizes) of different types of variables in the model.**

I. Introduction

Let us consider an i.i.d. sample of n vectors $\mathbf{x}^n = \mathbf{x}_1, \ldots, \mathbf{x}_n$, where each vector $\mathbf{x}_i$ consists of s discrete symbols taken from some alphabet (with each of the s components having its own alphabet). Given a parametric model $\mathcal{M}$, the normalized maximum likelihood (NML) distribution (or code) [7], [9] is defined by

$$P_{NML}(\mathbf{x}^n|\mathcal{M}) = \frac{P(\mathbf{x}^n|\hat{\theta}(\mathbf{x}^n, \mathcal{M}))}{\sum_{\mathbf{y}^n} P(\mathbf{y}^n|\hat{\theta}(\mathbf{y}^n, \mathcal{M}))}, \qquad (1)$$

where the numerator is the maximum likelihood for the observed data table $\mathbf{x}^n$ within $\mathcal{M}$. The *normalizing term* in the denominator is the sum over maximum likelihoods of all possible data sets of size n, with respect to the model class. The NML offers a theoretically appealing, minmax optimal criterion for model selection and other statistical inference tasks, but is typically hard to compute.

Bayesian forests are tree-structured graphical Bayesian network models [6], [3]: a tree is a connected directed acyclic graph where each node has at most one parent, and a forest is a set of trees. In a Bayesian tree, each node corresponds to a random variable (a column in $\mathbf{x}^n$), and the joint probability distribution of a vector $\mathbf{v} = (v_1, \ldots, v_s)$ factorizes as

$$P(\mathbf{v}) = \prod_{i=1}^{s} P(v_i|v_{f(i)}), \qquad (2)$$

where $f(i)$ is the index of the parent node of node i. So each node has a local probability distribution conditioned on its parent node and the joint distribution is a product of these local distributions. Trees are independent of each other, so the joint probability of a forest is a product of joint probabilities of trees. In this paper we use multinomial local distributions, and hence the joint distribution is a product of multinomials.

For Bayesian forests, computing the numerator of (1) is trivial [10], while computing the denominator is computationally very demanding. In the following we represent a novel algorithm for computing the normalizing term of (1) in the case of Bayesian forests. The efficiency of the algorithm depends on the number of values of the variables (the sizes of the column alphabets).

II. Matrix representation

Computing the NML normalizing term for a Bayesian forest is quite complicated: the basic task is to take a sum over all possible data sets of size n and compute the maximum likelihood for each data. However, note that this is equivalent to summing over all possible sufficient statistics related to the model and computing each maximum likelihood multiplied by a factor which counts the number of data sets producing the same sufficient statistics. As we saw earlier, Bayesian multinomial forests have a nice property that the global distribution decomposes into local probability distributions. Now the main idea is to claim that we can use a similar decomposition for computing our NML normalizing term more efficiently. For accomplishing this, we define three different type of local *components*: a root node component, an inner node component and a leaf node component. We can represent these components as vectors and matrices. The normalizing term corresponding to the whole forest can be then computed using simple matrix operations between the precomputed components. As we compute over all possible local sufficient statistics in every component, we get the wanted result: the value of the normalizing term for an forest. For further investigations of the sufficient statistics related to this problem, see [10].

A. General algorithm

Let us first have a look at the two-step matrix algorithm on a general level without defining exactly the components. The algorithm needs three different type of components: horizontal vectors for root nodes, matrices for inner nodes and vertical vectors for leaf nodes. In the first step the algorithm computes required components. Identical components need to be computed just once. Only the number of values of a variable and its parent is essential: if there are identical numbers of

978-1-4244-2271-5/08/$25.00 ©2008 IEEE

outcomes in several locations in the forest, components are identical.

Let us denote the *root node component* with R_X, where X is the name of the node. The *inner node component* is denoted with M_X^Y, where X is the name of the inner node and Y is the name of the parent node, and finally L_X^Y is the *leaf node component*. The operation between siblings (or between trees) is the entry-wise product (Hadamard product) and denoted by symbol $'\odot'$. The operation between a parent and a child is the ordinary matrix multiplication. In the second step the algorithm executes these matrix computations. Notice that there is no need to compute matrix-matrix-operations. Computation proceeds always from leaves to root and the most demanding operations are therefore matrix-vector-multiplications. We can write computations using the matrix notation: take for example the forest ($B \leftarrow A \rightarrow C \rightarrow D$, $E \rightarrow F$) with six nodes. Now it is easy to write the computation using the matrix notation as $(R_A(L_B^A \odot (M_C^A L_D^C))) \odot (R_E L_F^E)$.

Next we proceed by specifying in subsections C, D and E the individual components in such a manner that the above algorithm provably computes the desired normalization term, but first we have to shortly discuss k-compositions, k-partitions and their relations to each other.

B. k-compositions and k-partitions

A single multinomially distributed variable X forms the *multinomial model*. The sufficient statistics for this model is a k-composition, where k is the number of values of X. The *k-composition* of n can be represented as a k-tuple $\mathbf{x} = (x_1, \ldots, x_k)$, where $x_1 + \cdots + x_k = n$, $x_i \in \mathbb{N}$ and n is the number of data points [8]. For brevity, let us in the sequel denote the multinomial probability with maximum likelihood parameters for the observed counts by $c((x_1, \ldots, x_k))$:

$$c((x_1, \ldots, x_k)) = \frac{(\sum_{i=1}^k x_i)!}{\prod_{i=1}^k (x_i!)} \prod_{j=1}^k \left(\frac{x_j}{\sum_{i=1}^k x_i} \right)^{x_j}, \quad (3)$$

and hence the NML normalizing term for the multinomial model class is

$$\mathcal{C}_{MN}(k, n) = \sum_{x_1 + \cdots + x_k = n} c((x_1, \ldots, x_k)), \quad (4)$$

where the sum goes over the set of all k-compositions of the data size n [5]. But note that we can reduce the number of terms in the previous sum by using k-partitions instead of k-compositions: A *k-partition* is an unordered set of counts defined in a similar way as k-compositions. However, for simplicity, we represent each k-partition as a k-tuple with a standard decreasing order of counts.

We denote all k-partitions of a variable X by $q^X = [q_1^X, \ldots, q_p^X]$, where k is the number of values in variable X and p is the number of k-partitions. The number of data points is omitted from this notation, because we can determine it from k and p. To be sure that we multiply right elements with each other in the second step of the algorithm, we have to select also an fixed ordering among the set of k-partitions. We use the following ordering (the rationale of which becomes apparent in section III-D): First order all k-partitions so that the first one is the one with most zero counts. Then comes the ones with one zero count less, and so on. After this, order each of these sub-blocks with a same number of zero counts using inverse lexicographic ordering.

When two k-compositions $\mathbf{x}$ and $\mathbf{y}$ have the same ordered k-partition, then the maximum likelihoods are also the same, i.e., $c(\mathbf{x}) = c(\mathbf{y})$. Consequently, if we use k-partitions instead of k-compositions, we have to use an extra multiplier in the computations to take into account the fact that many k-compositions map to the same k-partition. Formally, the *multiplier function* is given by

$$m(\mathbf{x}) = \frac{k!}{\prod_{w \in \mathbf{x}} \mu_w(\mathbf{x})!}, \quad (5)$$

where $\mu_w(\mathbf{x}) = |u : x_u = w|$ tells how many times a value w appears in a k-partition $\mathbf{x}$.

C. Root node component

Root nodes have no parents, so the sufficient statistics of each root node goes simply through all k-partitions, where k is the number of values of the root node variable. The root node component is defined as a horizontal vector of multinomial probabilities with maximum likelihood parameters for all sufficient statistics of given data size n:

$$R_X = \begin{bmatrix} m(q_1^X) \cdot c(q_1^X) & \cdots & m(q_p^X) \cdot c(q_p^X) \end{bmatrix}, \quad (6)$$

where p is the number of k-partitions in the root node X. The multipliers collect all identical values into the same vector element. We naturally could have used k-compositions instead of k-partitions, but then the length of the vector would have increased radically: when the number of data vectors is large, there are $k!$-times more k-compositions than k-partitions.

D. Inner node component

The inner node component is a matrix. We define the matrix here, but all computational issues will be handled in section III.

Unlike in the case of root nodes, with an inner node we have to take into account the sufficient statistics of the parent node, and for that we introduce the *conditional term* $c(q_i^X | q_j^Y)$, where q_i^X is the sufficient statistics of a node and q_j^Y is the sufficient statistics of the parent node. In the root node case we can think that the missing parent node is a one valued node (a constant). The conditional term has in this case the form $c(q_i^X | (n)) = c(q_i^X)$. Hence we can split the data between different outcomes of the root node and get the correct result.

In the inner node case we can to think that the data is originally in l separate bins (we use the ball analogy here), where l is a number of possible outcomes of the parent node. We spread the data from each parent bin to the inner node's bins, but unfortunately we cannot do this independently, because we have to achieve the given k-partition, where k is the number of the inner node's possible values. As there are many different valid paths to get the right k-partition, our

conditional term gets the form

$$c((x_1,\ldots,x_k)|(y_1,\ldots,y_l)) = \sum_{Z\in\mathcal{Z}}\prod_{i=1}^{l} c(z_{1i},\ldots,z_{ki}), \quad (7)$$

where Z is a matrix with marginals $(x_1,\ldots,x_k)$ and $(y_1,\ldots,y_l)$, and the terms z_{ij} are elements of the matrix Z. All elements are positive integer values. Set $\mathcal{Z}$ is the set of those matrices Z which satisfy the given marginals:

$$\mathcal{Z} = \{Z | z_{11}+\cdots+z_{k1} = y_1,\ldots,z_{1l}+\cdots+z_{kl} = y_l,$$
$$z_{11}+\cdots+z_{1l} = x_1,\ldots,z_{k1}+\cdots+z_{kl} = x_k\}.$$

To compute the conditional term, we have to form all the matrices which satisfy the given marginals. This is a tremendously heavy operation. Given the defined conditional terms, the general form of the inner node matrix is

$$M_X^Y = \begin{bmatrix} m(q_1^X)\cdot c(q_1^X|q_1^Y) & \cdots & m(q_p^X)\cdot c(q_p^X|q_1^Y) \\ \vdots & \ddots & \vdots \\ m(q_1^X)\cdot c(q_1^X|q_d^Y) & \cdots & m(q_p^X)\cdot c(q_p^X|q_d^Y) \end{bmatrix},$$

where X is a node and Y is its parent node. Dimensions of this matrix are defined by the number of l-partitions and the number of k-partitions ($d \times p$). As before, multipliers exist as we are using k-partitions, not k-compositions.

E. Leaf node component

Leaf nodes are easier to handle than inner nodes, although they also have parent nodes. In the inner node case we have the target k-partition, but in the case of leaf nodes, any k-partition is valid. As we do not have a fixed target partition, we can just marginalize over all individual targets and the result is a product of ordinary multinomial normalizing terms defined in the formula (4). So we can separately compute each parent node bin, without any restrictions:

$$\begin{aligned}\sum_{i=1}^{p} & m(q_i^X)\cdot c(q_i^X|q_j^Y) \\ &= \sum_{i=1}^{p} m((x_1,\ldots,x_k)_i)\cdot c((x_1,\ldots,x_k)_i|(y_1,\ldots,y_l)_j) \\ &= \mathcal{C}_{MN}(k,y_1)\cdots\mathcal{C}_{MN}(k,y_l), \end{aligned} \quad (8)$$

where $(x_1,\ldots,x_l)_i$ is the ith k-partition and $(y_1,\ldots,y_l)_j$ is some l-partition. Efficient computation of the term $\mathcal{C}_{MN}(\cdot,\cdot)$ is not trivial, but fortunately there is a recent result showing how to use a simple recurrence formula to compute the term in linear time with respect to data size [4].

The leaf node component is a vertical vector

$$L_X^Y = \begin{bmatrix} \sum_{i=1}^{p} m(q_i^X)\cdot c(q_i^X|q_1^Y) \\ \vdots \\ \sum_{i=1}^{p} m(q_i^X)\cdot c(q_i^X|q_d^Y) \end{bmatrix}. \quad (9)$$

The vector has d elements, where d is the number of l-partitions. Although the leaf vector can also be seen as an inner node matrix where we take a sum over the other dimension, luckily it is much easier to compute than the inner node matrix.

III. Efficient computation of inner node matrices

In the previous section we defined the inner node component, but we did not yet give an algorithm for computing it. One could of course make the obvious brute-force algorithm for finding the set of all matrices Z, and then compute each of the $c(q_i^X|q_j^Y)$ terms. However, this approach is clearly infeasible, as the number of matrices Z grows very fast when the number of data vectors increases. Also the size of the inner node component is growing at the same time.

Fortunately it is possible to use generating functions [2] and other computational simplifications for calculating the inner node components, as we will shortly see. These modifications make the computations feasible for small data sets, if the maximum number of values of the variables is relatively small. First we introduce generating polynomials, which form the very basis of our efficient computation.

A. Generating polynomials

A *multivariate generating polynomial* is a finite multivariate series of the form

$$\sum_{\mathbf{x}\in\mathcal{X}} a(x_1,\ldots,x_k) z_1^{x_1} z_2^{x_2}\cdots z_k^{x_k}, \quad (10)$$

where $\mathbf{x} = (x_1,\ldots,x_k)$ are vectors in a set of positive integer-valued vectors $\mathcal{X}$ and each variable $z_i \in \mathbb{C}$. The coefficients $a(x_1,\ldots,x_k)$ encode some relevant information. When we take a product of these generating polynomials, the coefficients of the resulting polynomial then correspond to higher level convolution operations. In the following we define our generating polynomials in such a manner that the result corresponds to the convolution operations we need.

Let us now have a polynomial of the form (10) with the sum going over all k-compositions of data size u. A natural choice for the coefficients is the function $c((x_1,\ldots,x_k))$, in which case the resulting generating polynomial is

$$P_k^0 = 1, \quad \text{and} \quad (11)$$

$$P_k^u = \sum_{x_1+x_2+\cdots+x_k=u} c((x_1,\ldots,x_k)) z_1^{x_1} z_2^{x_2}\cdots z_k^{x_k}. \quad (12)$$

The coefficients of this polynomial describe the value of $c((x_1,\ldots,x_k)|(u))$, which is a root vector element without the multiplier. Now the trick is just to multiply these generating polynomials with respect to a parent node l-partition:

$$T_k^{(y_1,\ldots,y_l)} = P_k^{y_1} P_k^{y_2}\cdots P_k^{y_l}. \quad (13)$$

The coefficients of this polynomial now correspond to the needed conditional terms $c((x_1,\ldots,x_k)|(y_1,...,y_l))$. In fact one can read all the k-partitions $(x_1,\ldots,x_k)$ for a given parent node l-partition from this new generating polynomial. So each product polynomial gives us a whole row of an inner node matrix. Denoting the coefficient extraction by standard notation, we can write this as

$$c((x_1,\ldots,x_k)|(y_1,...,y_l)) = [z_1^{x_1}\cdots z_k^{x_k}] T_k^{(y_1,\ldots,y_l)}. \quad (14)$$

The above scheme gives us a concrete way to compute the conditional terms. The only remaining question is how to do

the multiplication of these multivariate polynomials efficiently. For this we will not use normal multiplication methods for computing all the terms of the polynomial $T_k^{(y_1,\ldots,y_l)}$: as we only need the coefficients of those terms which correspond to k-partitions, we can use the concept of a polytope to bound the set of computationally relevant terms.

B. Convex polytopes

As polygon is the name for a figure on a plane bounded by a finite number of line segments that form a closed path, a *polytope* is the name for a similar object in any dimension. The bounding objects are called *facets*: facets of a k-dimensional polytope are $(k-1)$-dimensional and are itself polytopes. We can represent a *convex polytope* as an intersection of half spaces. *Integer lattice points* of a convex polytope are all the points $(x_1,\ldots,x_k)$ which belong to the polytope and have $x_i \in \mathbb{Z}$ for all i [1].

Now we define the set of all terms that we need for computing $T_k^{(y_1,\ldots,y_l)}$. First we need all those terms that correspond to our k-partitions of n. Second, because we are taking a product of polynomials, we need also all those terms that can produce a k-partition of n -terms via multiplication process. For example, $x^2y^2z^0$ times $x^2y^0x^2$ gives us the term $x^4y^2z^2$ which corresponds to a 3-partition $(4,2,2)$ of 8. These two sets of terms are all we need. We could map these points to a k-dimensional space, but we can in fact map these points also into a $(k-1)$-dimensional space, because one of the parameters is redundant. The redundancy is caused by the fact that the total degree of all the terms is the same in our case, as the polynomials we multiply are homogeneous. When we map our terms into a $(k-1)$-dimensional space, we drop the first count and say that each k-composition $(x_1,x_2,\ldots,x_k)$ corresponds to the point $(x_2,\ldots,x_k)$. It is most useful to drop the first count with the biggest value, because then we need to compute the least number of terms during the polynomial multiplications.

As we have now defined the set of all relevant terms, we want a compact representation for the set. This set happens to be a $(k-1)$-dimensional convex polytope, which we call *k-multiplication polytope*. We define the convex k-multiplication polytope using half spaces. There are two different kind of inequalities which define our polytope. First inequalities are of type

$$0 \le x_i \le \left\lfloor \frac{n}{i} \right\rfloor, \tag{15}$$

where x_i is value in the ith bin. There is one inequality for every bin except the first one, because it is redundant. Inequalities give the lowest and the highest count for every bin. These inequalities form a hyper rectangle. But there are still unnecessary terms inside this polytope, corresponding to invalid counts that are already too big to produce any valid k-partition of n. Therefore we need a second type of inequalities in addition.

To achieve the second type of inequalities, we make splits between the bins of a k-partition while preserving the order of bins. Notice that as the count x_1 is redundant, there cannot be a split between x_1 and x_2, so x_1 and x_2 are in the same group. For example, 4-partitions have three different splittings $\{x_1x_2|x_3x_4, x_1x_2|x_3|x_4,\ x_1x_2x_3|x_4\}$, where the vertical bar means the border of two groups. Now the new set of inequalities can be written using these formed groups. We name each group using the name of the last count in that group, and a group is then multiplied by the number of members in that group. We get three inequalities:

$$2x_2 + 2x_4 \le n,\ 2x_2 + x_3 + x_4 \le n,\ 3x_3 + x_4 \le n. \tag{16}$$

These inequalities describe situations where there are several bins with a same value. In general we get the second type of inequalities by finding all possible splittings and writing the inequalities in a similar manner.

Our polytope is now kind of a cage: we must compute all the terms that are inside the cage, but none of the outside ones. Next we will define how to do the polynomial multiplications.

C. Restricted multiplication of multivariate polynomials

We start by computing all those terms of a polynomial P_k^u that correspond to lattice points of a k-multiplication polytope. Notice that the polytope is defined by the data size n, not by the number of data points u in some parent bin. We assign coefficients $c((x_1,\ldots,x_k))$ to lattice point $(x_2,\ldots,x_k)$. This means that there will be many lattice points which will remain zero, as the polynomial P_k^u does not have the corresponding terms.

As we take a product of several multivariate polynomials, it is wise to multiply first the smaller ones, because then the number of resulting polynomial terms is minimized, as well as is the number of multiplications. This means that the multiplication order must be $P_k^{y_l}P_k^{y_{l-1}}\cdots P_k^{y_1}$ when computing the $T_k^{(y_1,\ldots,y_l)}$ polynomial.

Now we can define the actual operation between the values of lattice points of polytopes, so that the operation corresponds to the multiplication of multivariate polynomials. The value of the resulting polytope lattice point $(v_1,\ldots,v_r)$ is computed by

$$\mathcal{P}(v_1,\ldots,v_r) = \sum_{w_1=0}^{v_1}\cdots\sum_{w_r=0}^{v_r} \mathcal{P}_1(w_1,\ldots,w_r)\cdot\mathcal{P}_2(v_1-w_1,\ldots,v_r-w_r),$$

where $\mathcal{P}_1$ and $\mathcal{P}_2$ are the polytopes to be multiplied and $r = k-1$. We set previously the coefficients of a multivariate polynomial to lattice points of the polytope. Therefore, when we do the above operation for lattice points of the first two polytopes, we see that the computed value of a lattice point is

$$c((h_1,\ldots,h_k)|(y_1,y_2)) = \sum_{Z\in\mathcal{W}} c(z_{11},\ldots,z_{1k})\cdot c(z_{21},\ldots,z_{2k}), \tag{17}$$

where $\mathcal{W}$ is the set of all matrices Z with marginals $(h_1,\ldots,h_k)$ and (y_1,y_2). From this we can see directly that when we do several multiplications, we get the term $c((x_1,\ldots,x_k)|(y_1,\ldots,y_l))$ defined in formula (7).

We explained how to compute the relevant terms of the polynomial $T_k^{(y_1,\ldots,y_l)}$ efficiently. Next we show that there is no need to compute the inner node matrices separately, as they all have common terms, which is a property we can utilize to make the computation even more efficient.

D. Core inner node matrix

Computation of the inner node matrices is still a very slow operation, but we can avoid unnecessary computational work by exploiting the particular order of partitions we earlier chose. Namely, if we use the given order, we can first compute a matrix, which we will name a *core inner node matrix*, as follows:

$$CM = \begin{bmatrix} c(q_1^{\max(X)}|q_1^{\max(Y)}) & \cdots & c(q_P^{\max(X)}|q_1^{\max(Y)}) \\ \vdots & \ddots & \vdots \\ c(q_1^{\max(X)}|q_D^{\max(Y)}) & \cdots & c(q_P^{\max(X)}|q_D^{\max(Y)}) \end{bmatrix},$$

where $q_i^{\max(X)}$ and $q_j^{\max(Y)}$ are $\mathcal{K}$- and $\mathcal{L}$-partitions. Value D is the number of $\mathcal{L}$-partitions, where $\mathcal{L}$ is the maximum number of values that any inner node's parent has in the forest and P is the number of $\mathcal{K}$-partitions, where $\mathcal{K}$ is the maximum number of values any inner node has in the forest. Ignoring the multipliers, we notice that every inner node matrix is just a section from the core matrix. As we ordered the partitions so that we first have embedded 1-partitions, then embedded 2-partitions and so on, we can just make the following operation to the core matrix to get an inner matrix: If the number of values of a node is less than $\mathcal{K}$, we drop the last invalid columns from the right, and if the number of values of a parent is less than $\mathcal{L}$, we drop the last invalid rows from the bottom of the core matrix.

The reason why all inner node matrices have the same $c(q_i^X|q_j^Y)$ elements is easy to see: the bins with zero counts do not affect the value of the conditional term, so we can add as many zero counts as we want and the terms still remain same.

E. Efficiency and the time complexity

The time complexity of the whole algorithm reduces to one question: how much time does it take to compute the core matrix? We get a rough approximation for the time complexity in the following way: A size of a $\mathcal{K}$-multiplication polytope with given n is $\mathcal{O}(n^{\mathcal{K}-1})$. The maximum time that a polytope element multiplication takes is also $\mathcal{O}(n^{\mathcal{K}-1})$. We have to compute D-times these polytopes, where D is the number of $\mathcal{L}$-partitions of n. The complexity of this term is $\mathcal{O}(n^{\mathcal{L}-1})$. Therefore the time complexity of the whole algorithm is $\mathcal{O}(n^{2\mathcal{K}+\mathcal{L}-3})$. However, if we have precomputed the core matrix, then the time complexity of computing the NML for any forest (compatible to the core matrix) is $\mathcal{O}(Hn^{\mathcal{K}+\mathcal{L}-2})$, where H is the number of inner nodes in the forest. This means that for example with two-valued nodes, the time complexity is $O(n^3)$, but if we have precomputed the core matrix, we can compute the NML in time $O(Hn^2)$ for any forest structure. Note that this time complexity applies for all structures with any number of values in the leaf nodes, because the number of values in a leaf node does not affect the inner node computation.

We have run some tests for examining how large n is still computationally feasible in practice. The algorithm is coded using Perl and it utilizes some additional tricks. With 3-valued nodes, computing the core matrix for $n = 200$ took about 6 hours using a 2.13GHz Intel® Core™2 Duo -processor with only one core, while with 4-valued nodes it took 16 hours to compute the core matrix for $n = 75$. Note also that the core matrix can be efficiently computed in parallel using multiple processors (cores), as the rows of the core inner node matrix can be computed separately using a different processor for each row. Core matrices can also be stored for later use.

IV. Conclusion

We presented an algorithm for computing the normalized maximum likelihood (NML) for tree structured Bayesian network models. The efficiency of the algorithm depends on the sizes of the alphabets used, and it is significantly more efficient than the only existing alternative reported in [10]. The algorithm offers us an opportunity to empirically compare the behavior of the NML approach to other graphical model selection methods in many non-trivial cases. Furthermore, as the algorithm computes exact results, this gives us also an opportunity to empirically validate approximative methods for computing the NML.

Acknowledgment

This work was supported in part by the Academy of Finland under the project Civi and by the Finnish Funding Agency for Technology and Innovation under the projects Kukot and PMMA. In addition, this work was supported in part by the IST Programme of the European Community, under the PASCAL Network of Excellence, IST-2002-506778. This publication only reflects the authors' views.

References

[1] M. Beck and S. Robins. *Computing the Continuous Discretely: Integer-point Enumeration in Polyhedra.* Springer, 2007.

[2] P. Flajolet and R. Sedgewick. *Analytic Combinatorics*. Unpublished.

[3] D. Heckerman, D. Geiger, and D.M. Chickering. Learning Bayesian networks: The combination of knowledge and statistical data. *Machine Learning*, 20(3):197–243, September 1995.

[4] P. Kontkanen and P. Myllymäki. A linear-time algorithm for computing the multinomial stochastic complexity. *Information Processing Letters*, 103(6):227–233, 2007.

[5] P. Kontkanen, P. Myllymäki, W. Buntine, J. Rissanen, and H. Tirri. An MDL framework for data clustering. In P. Grünwald, I.J. Myung, and M. Pitt, editors, *Advances in Minimum Description Length: Theory and Applications*. The MIT Press, 2006.

[6] J. Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann Publishers, San Mateo, CA, 1988.

[7] J. Rissanen. Fisher information and stochastic complexity. *IEEE Transactions on Information Theory*, 42(1):40–47, January 1996.

[8] F. Ruskey. *Combinatorial Generation*. Unpublished.

[9] Yu.M. Shtarkov. Universal sequential coding of single messages. *Problems of Information Transmission*, 23:3–17, 1987.

[10] H. Wettig, P. Kontkanen, and P. Myllymäki. Calculating the normalized maximum likelihood distribution for Bayesian forests. In *Proc. IADIS International Conference on Intelligent Systems and Agents*, Lisbon, Portugal, July 2007.