Department of Computer Science
Series of Publications A
Report A-2009-10

Computing the Stochastic Complexity of Simple Probabilistic Graphical Models

Tommi Mononen

To be presented, with the permission of the Faculty of Science of the University of Helsinki, for public criticism in Auditorium XIV, University Main Building, on December 12th, 2009, at 10 o'clock before noon.

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

Copyright © 2009 Tommi Mononen
ISSN 1238-8645
ISBN 978-952-10-5898-1 (paperback)
ISBN 978-952-10-5899-8 (PDF)
Computing Reviews (1998) Classification: G.1.0, G.2.1, G.3, H.1.1
Helsinki 2009
Helsinki University Print

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@Helsinki.FI

PhD Thesis, Series of Publications A, Report A-2009-10
Helsinki, December 2009, 60 + 46 pages
ISSN 1238-8645
ISBN 978-952-10-5898-1 (paperback)
ISBN 978-952-10-5899-8 (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 difficulties, 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

Preface

“A fall into a ditch makes you wiser.”
Chinese proverb

Making a Ph.D. dissertation is a process. It is the process of knowing the inner-self better. During that process you feel sometimes happy, but mostly you feel miserable, stupid and lonely — or at least, I did. There are high moments, the moments of discovering something new, but nine times out of ten I found out I was wrong. Even in those cases where discoveries were valid and correct, the happiness rapidly decayed after a few days and imperfection filled my mind. However, no man is an island. Luckily there was and is other people in my life that were ready to listen to me and help me during the process. Without these people you would not be able to read this text, because this book would not exist.

It would be a lie to claim that this process of mine did not affect other people. There were several hard times when I was feeling no joy at all. I was just too focused and all those wonderful things that usually give me great joy, just irritated me. The journey of learning how to be a researcher is a two-bladed sword: it gives great pleasure to find new knowledge, but on the other hand it is very demanding to aim at a target infinitely far away — the dissertation.

The public discussion and indirect statements of the Ministry of Education that students are lazy and are using state funding inefficiently do not make the situation of a student or a Ph.D. student any better. As a student is thought to be an increment or decrement in some statistics coupled with financial figures, he or she can be handled via economic terms. We are not living persons with feelings, but just investments and we have some expected future gain. Bit by bit these ideas have reached the faculty and the department levels. I have not been treated like this, but it does not hinder me to have equal feelings. I feel like an inanimate object on an assembly line with a serial number — the student number.

As it took me almost ten years to reach this point, it must mean that at some point during the process I have been very inefficient. I luckily started my Ph.D. studies during the last days of the old and inefficient Finnish university — the very same that has produced most of current professors, but will not produce future ones. I truly believe that those Ph.D. students just now starting the process, do not have the luxury to spend ten years. They will be replaced by new, more eager students, if they are as inefficient as I have been.

This printed book represents an outcome of the process. Something that hopefully gives the Department of Computer Science possibility to collect the profit of its investment or at least minimize the loss. During the process I have learned many things and still managed to somehow live my life — my true life with my family. I want to express my special gratitude to those who were ready to listen to me and discussed also topics not related to research interests.

Acknowledgments: I would like to thank my advisor, Professor Petri Myllymäki, for helping me in formulation of the results and showing several incoherencies during editing. Without his invaluable help in writing, manuscripts might have never got accepted. I am also grateful to my previous advisor, Professor Henry Tirri, who let me freely wonder around and find the inspiration of my own. In those years I learned a lot and without those skills some of the results, essential to this thesis, would not exist.

It is very important to work in a research group, because new ideas are appearing too fast in active research areas. A single researcher, without a group, cannot keep track of results made by others. This limitation, however, does not seem to bother Professor Emeritus Jorma Rissanen, who has developed the MDL theory, on which this research is based on. I did this dissertation mostly while working in the CoSCo research group and I benefitted greatly from ideas of my fellow-workers Petri Kontkanen and Hannes Wettig. Petri's previous results and ideas presented in the discussions between me and him were the starting point of many papers in this dissertation. Preliminary work of Hannes gave inspiration for Paper 5. Sometimes only a few words can open new worlds: we travelled to Eindhoven to meet Alessandro Di Bucchianico and he gave us the idea of utilizing multivariate generating functions. Without his idea, although it was targeted to solve a different problem, the results presented in Paper 3 would not exist. I am grateful to him for the idea. I want to thank Professor Patric Östergård and Dr. Ciprian Doru Giurcaneanu for their useful comments that further improved this thesis.

For the financial support I would like to thank the Department of Computer Science, the Academy of Finland, the Finnish Funding Agency for Technology and Innovation (Tekes), the PASCAL network of excellence and Helsinki Graduate School in Computer Science and Engineering (Hecse). I want to express my deepest gratitude to Professor Ilkka Hanski from the Department of Ecology and Evolutionary Biology, who hired me after my leaving from the Department of Computer Science and let me finalize this dissertation during working hours. The Department of Computer Science and Helsinki Institute for Information Technology HIIT have provided me excellent working environment, and especially I want to thank Pekka Niklander for all the laptops and other equipments he has let me utilize during all these years.

I am grateful to those who spent time with me whenever I needed a break: the members of the CoSCo research group (including darts players) and people in the second floor coffee room. The persons (to mention some) with whom I have had many interesting conversations include Raul Hakli, Petteri Kaski, Mikko Koivisto, Jussi Lahtinen, Jussi Lindgren, Taina Nikko, Kimmo Valtonen and Virpi Ahola (in Viikki). And of course, I have not forgotten the four o'clock port in Ruoholahti (thank you, Henry).

Last but certainly not least is my family. I thank my wife Ilona and our daughters Erika and Varpu from the bottom of my hearth. If possible, the smiling and jumping children with the greeting 'Dad came home!' while I am still standing at the front door, will wipe away all the worries I have after a working day.

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, 30]. 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 [40]. One possible solution is of course to try to prove some theoretically valid method for determining the value of this parameter [42]. 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 [26]. 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) [35, 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 [32, 33, 34]. 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 [39]), which has been proven to be worst-case optimal [34, 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 [24, 25, 44]. In image denoising the best NML method is almost as good as the best methods [37]. 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 [41]. 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 forests. For practical purposes the algorithm is fast enough only for forests 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, 2, 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 that will be sent 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 that has the shortest code length for the observed data. The stochastic complexity then favors a simple good fitting model, thus it is obeying the Occam'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 Occam'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

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

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 (the terms inside brackets in (2.7)), 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*

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

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 (the terms inside brackets in (2.9)), 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*

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

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.

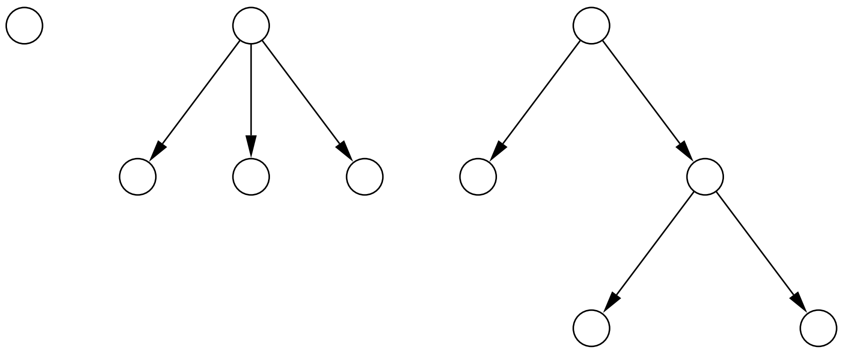

Figure 2.1: Multinomial (left), Naive Bayes (middle) and Bayesian tree (right) models. All graphs together can be interpreted as a forest with three trees.

2.3 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.

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 $(a_n) =$

$(a_0, a_1, \dots)$ and the function 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 well-known combinatorial trick that is presented for example in [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 function of k that gives coefficients is 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)[36]. 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. We can also do operations such as take a product between two multivariate polynomials. This kind of multiplication corresponds to a higher order convolution operation. In fact, 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 denote the (compositional) inverse of $B(t)$ by overline [36]. 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 [36, 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 [36]. 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 [36]: 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. We can denote the above using the standard notation

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

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

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

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

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

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 [31, 46]. 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* [27]. On the other hand, if we have the corresponding series, then there exists a *linear differential equation*

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

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. If they are not holonomic, then they are called *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 [27]. 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 $G(\mathbf{z})$ spans 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*

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

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*

$$p_0^{(j)}(i_j)a_{\mathbf{i}} + \sum_{l=1}^{r_j} p_l^{(j)}(i_1, \ldots, i_m)a_{i_1,\ldots,i_j-l,\ldots,i_m} = 0, \tag{2.36}$$

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*

$$\sum_{l=0}^{r_j} p_l^{(j)}(i_1, \ldots, i_m)a_{i_1,\ldots,i_j-l,\ldots,i_m} = 0. \tag{2.37}$$

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:

$$G^{1,\ldots,s}_{i_{s+1},\ldots,i_m}(z_1, \ldots, z_s) = \sum_{i_1,\ldots,i_s} a_{i_1,\ldots,i_m} z_1^{i_1} \cdots z_s^{i_s}. \tag{2.38}$$

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

$$a_{\mathbf{x}} z_1^{x_1} z_2^{x_2} \cdots z_k^{x_k}, \tag{2.39}$$

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

$$c_{\mathbf{y}} = \sum_{x_1=0}^{y_1} \cdots \sum_{x_k=0}^{y_k} a_{\mathbf{x}} \cdot b_{\mathbf{y}-\mathbf{x}}, \tag{2.40}$$

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 [47]. 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:

$$\{f_{1,i}y_1 + f_{2,i}y_2 + \cdots + f_{k,i}y_k \leq s \mid i = 1 \ldots r\}, \tag{2.41}$$

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.

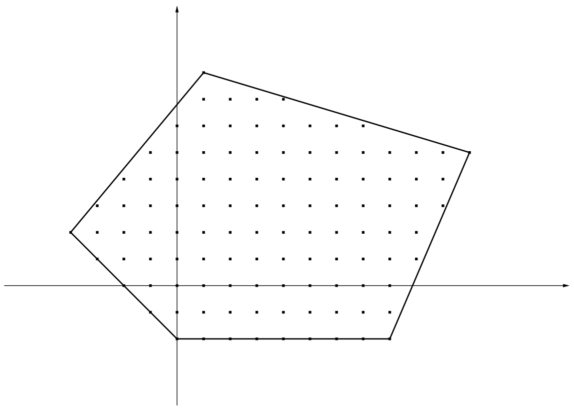

Figure 2.2: *A polytope with the integer lattice.*

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

The 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 Definitions

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, 43] 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: $T(z) = -W(-z)$. 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 [23]. 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. We have three different presentations: the bivariate generating function, the horizontal family and the vertical family. Any of these presentations can be used to compute the desired $\mathcal{C}_{MN}(L,n)$ and we want to use those that lead us to the most efficient solution or solutions. 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: $\mathcal{C}_{MN}(1,n) = 1$ 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. Although this kind of an algorithm does not seem to have any obvious practical use, it still reveals something that may be valuable in the future.

What about the recurrence equation over n? For the vertical family as a whole, not a single solution has been found so far. In fact, there has not been found a good solution for single members of the family either. Knuth and Pittel have presented one interesting recurrence formula in [19], but the formula needs all the previous normalizing terms (over n) to compute the next one, which makes it useless. The formula also changes parameter L to $L-2$ in the same time and therefore it does not correspond to the recurrence we are looking for. 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 the 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 algorithms 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 has not been found any 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*

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

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 the fixed integer parameters 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.

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 $L = 2$, then $(2-1)^{\overline{k}} = 1^{\overline{k}} = k!$ and the first part disappears (Paper 2). 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:

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

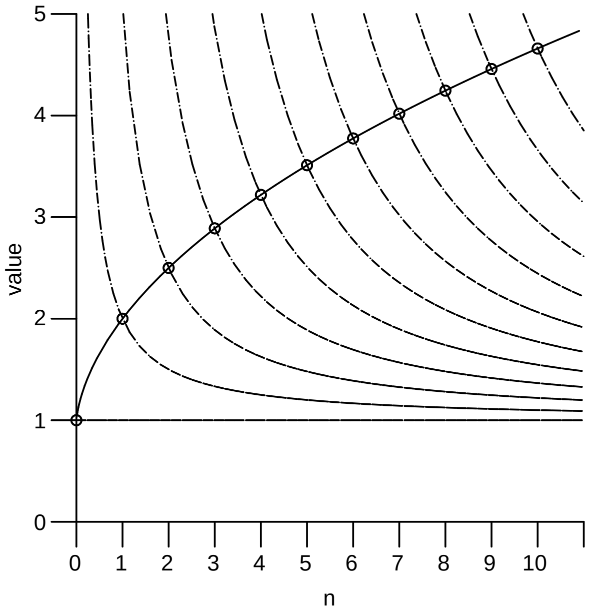

Figure 3.1: *The solid line gives the values of binomial sums and dotted lines are hypergeometric functions. The x-axis goes over n instead of $-\frac{1}{n}$ for achieving better separation between curves.*

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? We defined precision and the problem in a rigorous way in Paper 2, but here we only mention the main results. After some mathematical analysis, in which we did not use the Szpankowski approximation (explained in Section 3.4) due to analytical complexity, we got 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 with reasonable data sizes we always need only a sub-linear number of terms. However, if n is very small and d is large, n terms are still needed, because all the terms affect the result.

Next we take a closer look at a more complicated multinomial case. The

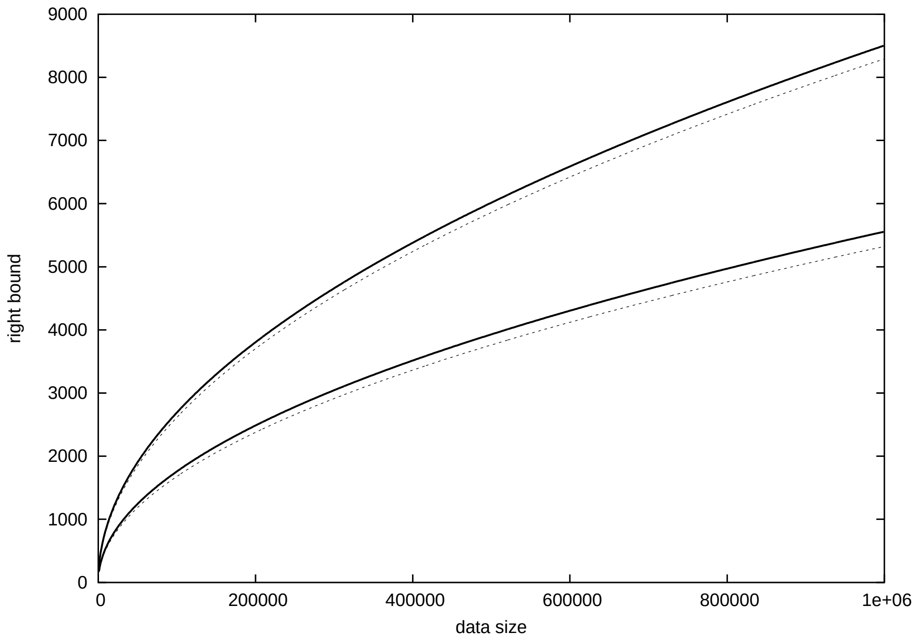

Figure 3.2: *Terms needed for 16 (above) and 7 digit precisions with given data size. Actual approximations are shown as a thick solid line. Thin dotted lines represent optimal index values.*

ratio of successive terms in the multinomial case is

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

and it looks 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 (Theorem 2 in Paper 2). 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 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 might produce nice proofs, we gain no increase in computational efficiency and therefore we did not try to prove these bounds.

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

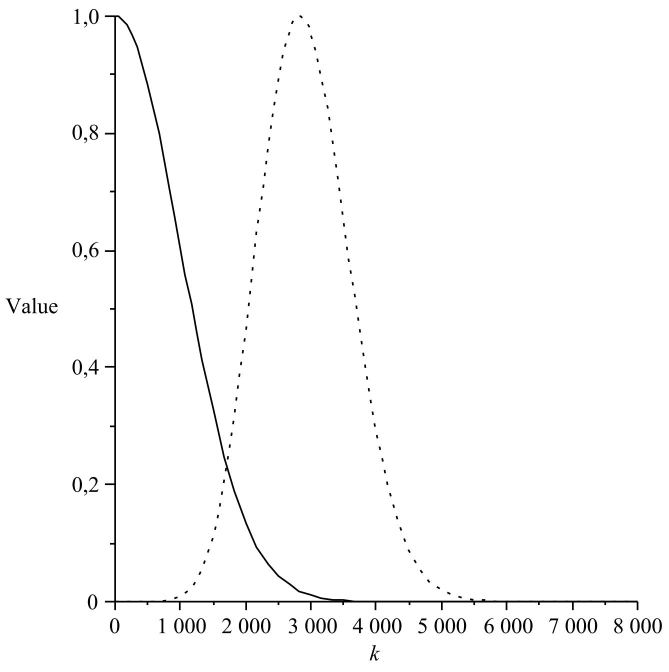

Figure 3.3: *The first 8000 terms of the trinomial (left) and the scaled 15-nomial (right) normalizing sums when data size (n) is one million.*

3.4 Efficient Computation and Algorithms

We will present two different type of algorithms that compute multinomial normalizing sums: efficient computation methods that give exact rational number answers and algorithms that give floating-point answers. The first type of methods are used for research purposes, because in the latter type of methods, we have to know the correct answers. The latter type is much faster and is used in 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 multinomial variable) 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 $\mathcal{O}(n^{3/2}\sqrt{d} + nL)$ 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 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 method to 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 [43]. 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 or hers 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 cryptographic attack method called birthday attack [28], which uses the mentioned property. The method can be used for a digital signature forgery. 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 (3.20) 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

Stochastic Complexity of 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 present the Naive Bayes generating function here without proving it. The first part of the proof can be found in [22] and the second part is in 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 seems to be computationally more complex than the multinomial case, and despite research efforts useful sub-quadratic time recurrence methods are still missing. It is not known, for example, whether there exists any recurrence formula for the 'horizontal' generating function family. 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

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

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):

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

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

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

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 suggests that the formulas cannot be applied directly to make the computation more efficient. However, the formulas suggest that there are some unknown properties that need to be examined more closely.

4.3 Efficient Computation and Algorithms

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

```
simplify(series((1+sum(hypergeom([3,-n],[],-1/n)
                       *hypergeom([4,-n],[],-1/n)*n**n/n!
                       *z**n,n=1..infinity))**2,z,101));
```

Here we have two predictor variables with 4 and 5 values and a binary class variable. The size of data is up to 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 idea presented by the previous Maple command line 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 with floating-point numbers the algorithm `ComputeC_MN` (or equivalent) with a modification that it saves during one pass the multinomial normalizing sums for all $L = 1, \dots, \max_i K_i$.

   With rational numbers the same algorithm can be applied, but `t` is obsolete and must be replaced with `n` in the loop.
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. One candidate algorithm for this phase is a method called the Miller formula [15]. If we use this formula, we need to do only $\mathcal{O}(n^2)$ multiplications when raising a truncated formal power series to an exponent (a positive real number). Thus the number of multiplications does not depend on an exponent. 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. This 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 in Section 2.3.1. This of course means that the total number of multiplications is $\mathcal{O}(n^2 \log_2 L)$.

Chapter 5

Stochastic Complexity of Bayesian Forests

In this chapter we will show how to compute the normalizing sums for Bayesian forests (the stochastic complexity was defined by (2.4) and the numerator of (2.3) for the Bayesian forests was introduced in Chapter 2.2). 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

We present the values of sufficient statistics as k*-compositions* and k*-partitions* (Paper 5)[45]. A k-composition is a partition of a positive integer n into k bins (k non-negative 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

$\mathbf{x} = (x_1, \dots, x_k)$, 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 $\mathbf{x}$ [29]. 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:

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

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

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

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 $c((x_1, \dots, x_k) \mid (n)) = c((x_1, \dots, x_k))$, which means that originally all data are in the same bin. The *conditional c-function* is

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

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

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

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).

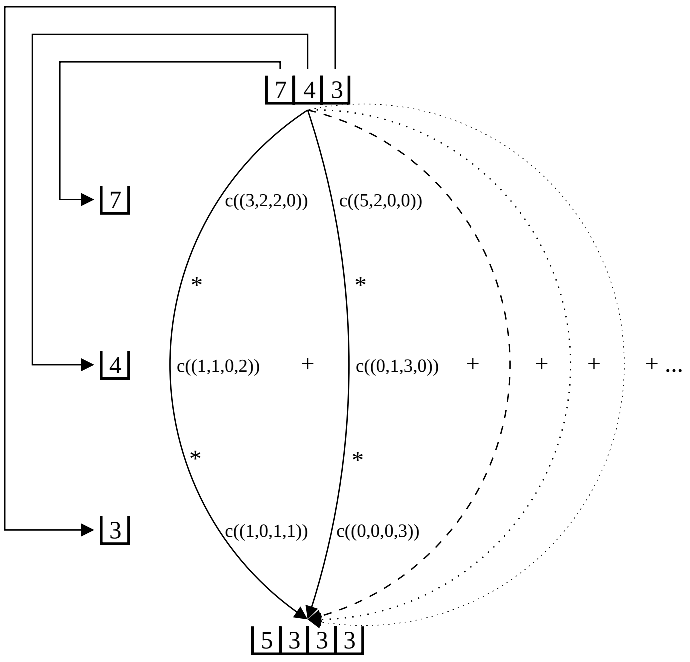

Figure 5.1: *Visualization of (5.4) with a couple of example paths. Data size is 14.*

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

$$R_X = \left[m(q_1^X) \cdot c(q_1^X) \quad \cdots \quad m(q_p^X) \cdot c(q_p^X)\right], \tag{5.10}$$

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,

```
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.

all the inner node matrices are actually sections of a bigger general matrix, 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

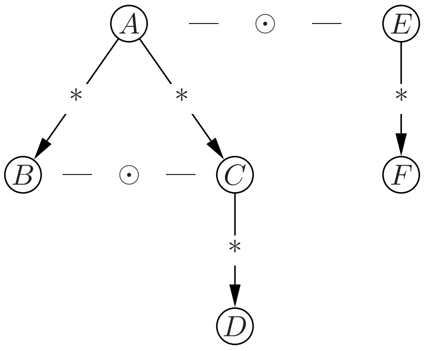

Figure 5.3: The operation between a parent and a child is the ordinary matrix multiplication (*) and between siblings the term-wise product ($\odot$).

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 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 $(y_1,\ldots,y_l)$:

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

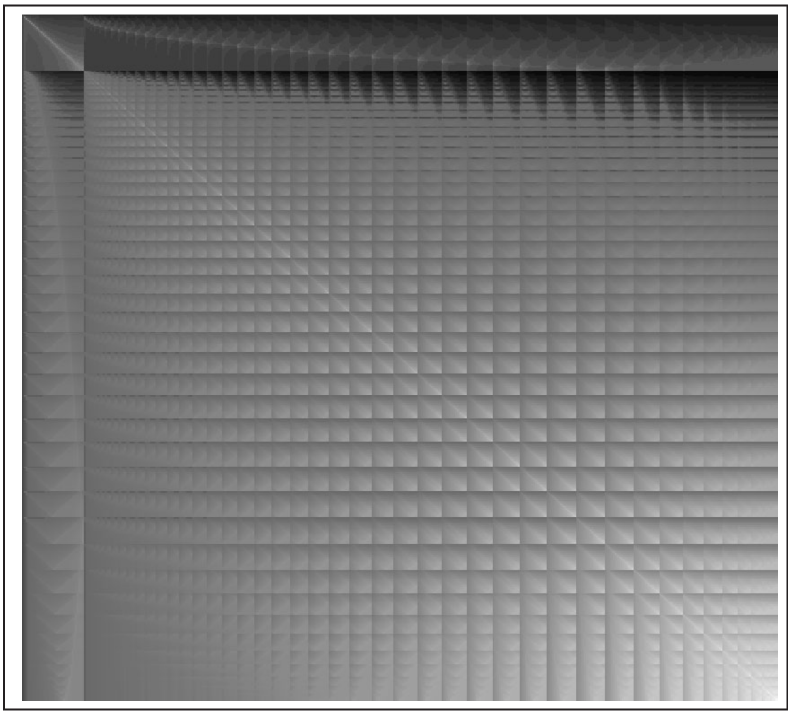

Figure 5.4: *Core matrix with values* $\max(Y) = \max(X) = 3$ *and* $n = 70$ *plotted. Brighter pixel means bigger value, but for structural visibility purposes the picture has been made brighter using image processing. Different regions are clearly visible: 2x2-partitions in the upper left corner, 2x3-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

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 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).

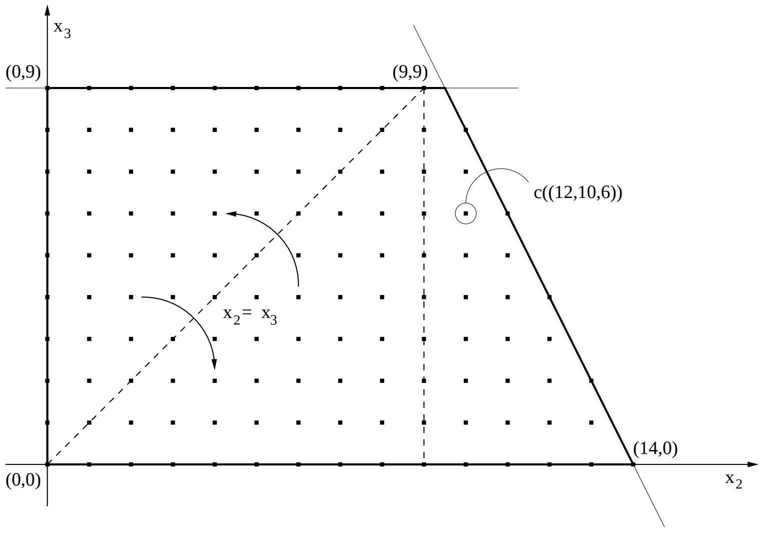

Figure 5.5: *An example of a symmetry inside the polytope, where we have three bins and 28 data vectors.*

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 [38]. 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 these series. 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 the actual 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 algoritms for tree-structured multinomial Bayesian networks. *EURASIP Journal on Bioinformatics and Systems Biology*, 2007.

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

[25] 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.

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

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

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

[29] 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.

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

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

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

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

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

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

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

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

[38] 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.

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

[40] 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.

[41] 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.

[42] 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.

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

[44] 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.

[45] J.H. Van Lint and R.M. Wilson. *A Course in Combinatorics.* Cambridge University Press, second edition, 2002.

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

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

Corrections

Paper I

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 III

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

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 homogeneous 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).