Probabilistic, Information-Theoretic Models for Etymological Alignment

Hannes Wettig

To be presented, with permission of the Faculty of Science of the University of Helsinki, for public criticism in Auditorium XIV of the University Main Building, on February 9th 2013 at 12 o’clock noon.
Probabilistic, Information-Theoretic Models for Etymological Alignment

Hannes Wettig

Department of Computer Science
P.O. Box 68, FI-00014 University of Helsinki, Finland
wettig@hiit.fi

Helsinki, January 2013, 143+59 pages
ISSN 1238-8645
ISBN 978-952-10-8588-8 (paperback)
ISBN 978-952-10-8589-5 (PDF)

Abstract

This thesis starts out by reviewing Bayesian reasoning and Bayesian network models. We present results related to discriminative learning of Bayesian network parameters. Along the way, we explicitly identify a number of problems arising in Bayesian model class selection. This leads us to information theory and, more specifically, the minimum description length (MDL) principle. We look at its theoretic foundations and practical implications. The MDL approach provides elegant solutions for the problem of model class selection and enables us to objectively compare any set of models, regardless of their parametric structure. Finally, we apply these methods to problems arising in computational etymology. We develop model families for the task of sound-by-sound alignment across kindred languages. Fed with linguistic data in the form of cognate sets, our methods provide information about the correspondence of sounds, as well as the history and ancestral structure of a language family. As a running example we take the family of Uralic languages.

Computing Reviews (1998) Categories and Subject Descriptors:
H.1.1 [Information Systems] Models and Principles
   — Systems and Information Theory.
I.2.6 [Computing Methodologies] Artificial Intelligence
   — Learning.
I.2.7  [Computing Methodologies] Artificial Intelligence
— Natural Language Processing.

**General Terms:**
Data Analysis, Probabilistic Modeling, Information Theory, Natural Language Processing.

**Additional Key Words and Phrases:**
Foreword

“ℵ₀ + 1”
—Peace and unity.

It is customary to acknowledge those that have contributed to the emergence of a dissertation, directly or indirectly. I find this a good tradition and want to make no exception.

Most directly involved in this work have been my supervisors Petri Myllymäki and Roman Yangarber, under whose wings I have been able to do the underlying work, as well as the co-authors of the included publications. I further want to thank the pre-examiners for a long list of valuable suggestions. In particular, Timo Honkela has provided me with a lot of good references to related work in linguistics and Steven de Rooij’s comments have forced me to clarify my thinking and writing about Bayesian Reasoning. While I had time to include only some of the additional references, Bayesians have been saved from some undue criticism. Most points that were made I have taken without a murmur, only on very few I still beg to differ. I do still believe that the assumption of a universal distribution to govern the behaviour of this world is very reasonable. As a result of the work of the reviewers, my thesis has become better in content, easier to access and less polemic. Complaints about shortcomings still present in this book are to be addressed to me alone.

The next batch of thanks go out to the CoSCo research group at the University of Helsinki, especially that of the early days dating back to 1996 when I had the privilege to join. Back then, the border between science and the real world was a thin line, creativity was blooming and nothing seemed impossible. You have shaped my way of scientific thinking more than anyone else. I want to especially thank Tomi Silander, who always seemed to have time for me and never took long to understand what I was talking about. Even when I did not, which is why I frequently sought his advice.
But above all, I need to thank a number of people less directly involved, while being no less vital for the completion of this book. To explain, I need to take a slightly wider turn.

Writing this dissertation has not been easy. It may not be supposed to be easy either, but for me it was much harder than that. In 2008, I was diagnosed with depression. It turned out to be the real deal, not just burn-out as in low batteries that you can refill in a couple of weeks on Bali. It was depression, the kind that kills. Statistically, about 20% of all patients diagnosed with severe depression end their misery by ending their days altogether.

The doctor prescribed me some pills and gradually, over a period of many months, things were starting to look up. But then I had another breakdown. And another. And while most of the time I seemed to recover, every breakdown was worse than the preceding and gradually my depression had indeed become severe. Other doctors prescribed other drugs and, eventually, intended to have me hospitalized, my brains fried with electric shock ‘therapy’. Somehow—I do not remember very well—I managed to escape that. These doctors have not been helpful and, on numerous occasions, gave me the feeling that they were not even trying to be. Fortunately, I was not being left to my own devices entirely. I wish to thank Riitta Niskanen and Leena ‘Lohtu ja Leipä’ Laine for professional, as well as Jouni Järvinen for semi-professional, voluntary help. You are among the whom-without.

In 2010 I stopped taking medication when I realized I could no longer think straight. I left university and went on to become a car mechanic. I take pride in the words “Ich bin ein Mechaniker”. I started to feel better, and rejoined university in 2011. But shortly after I was having another worst-yet breakdown. Finally, I managed to pull myself out of this ditch, too.

As of today, I am still among the surviving 80%. And I have actually written a doctoral dissertation, who would have thought? It was not easy and it has not been fun, but somehow it did come together.

This is very much due to some very special people. Thank you, not for enabling me to write this book, but for saving my life. Most importantly, I thank my wonderful wife Esther. For not leaving me when I gave her all reason to do so. For loving me while I could not love her back. For being incredibly patient and for believing in me. I also thank my daughters Vanamo, Judita and Pinja for welcoming me back, whenever I was capable of being their father. You give me reason to live. I know you never had a choice, please forgive me.
I have also been lucky to have some very good friends. The kind of friends that you can rely on in any given situation. Piia Karjalainen always had an open ear of me, as well as a mug of fresh coffee. Thank you for solace and true understanding. Special thanks also go to Jukka Palomurto for ‘Torvoila therapy’ and to Arto Löyttyniemi for lodging me when I needed to escape. It was at his place where I wrote chapters one to three of this book. Thank you for being there for me when I needed you, and for leaving me alone when I needed to be left alone. Thanks to all of you for listening, for asking the right questions and—most importantly—for not asking the wrong ones.

Finally, my deepest gratitude to my animal guide, Alfons. For unconditional love. For holistic understanding. For travelling through time and space to make me understand. Thank you.

Kulloo, January 2013
Acknowledgements

This work has been supported, to varying degree, by the following institutions:

- The Department of Computer Science at the University of Helsinki
- The Helsinki Institute for Information Technology (HIIT)
- The Helsinki Doctoral Programme in Computer Science (HeCSE)
- The Academy of Finland under the Finnish Center of Excellence in Computational Inference Research (COIN) and the Uralink project (Grant 129185)
- Projects Modest, Prose and CIVI
- The Finnish Funding Agency for Technology and Innovation under projects PMMA, KUKOT and SIB
- The IST Programme of the European Community under the PASCAL Network of Excellence
Contents

Foreword v

Acknowledgements viii

List of Included Publications 1

List of Abbreviations 3

1 Introduction 5

2 Bayesian Reasoning 13
   2.1 Bayes’ Rule . . . . . . . . . . . . . . . . . . . . . . . . . . . 13
   2.2 Marginal Likelihood . . . . . . . . . . . . . . . . . . . . . 17
   2.3 Bayesian Network Models . . . . . . . . . . . . . . . . . . 20
   2.4 Bayesian Model Class Selection . . . . . . . . . . . . . . . 26
   2.5 Supervised Learning Tasks . . . . . . . . . . . . . . . . . . 33
   2.6 Discriminative Parameter Learning . . . . . . . . . . . . . 36
   2.7 Empirical Evaluation . . . . . . . . . . . . . . . . . . . . . 43
   2.8 Summary . . . . . . . . . . . . . . . . . . . . . . . . . . . . 47

3 Information Theory 49
   3.1 The Minimum Description Length Principle . . . . . . . 49
   3.2 Two-Part Codes . . . . . . . . . . . . . . . . . . . . . . . . 55
   3.3 Kolmogorov Complexity . . . . . . . . . . . . . . . . . . . . 61
   3.4 Normalized Maximum Likelihood . . . . . . . . . . . . . . 66
   3.5 More Properties of the NML Distribution . . . . . . . . . 70
   3.6 Computing the NML Distribution . . . . . . . . . . . . . . 73
   3.7 Summary . . . . . . . . . . . . . . . . . . . . . . . . . . . . 76

4 Etymology 77
   4.1 Motivation . . . . . . . . . . . . . . . . . . . . . . . . . . . 77

ix
4.2 The Data ................................. 81
4.3 The Alignment Problem ..................... 84
4.4 Baseline Model and Extensions ............... 87
  4.4.1 Baseline Model ......................... 87
  4.4.2 Learning Procedure ................. 88
  4.4.3 Sanity Checking ..................... 90
  4.4.4 NML .................................. 91
  4.4.5 Codebook ............................. 91
  4.4.6 Distinguishing Between Kinds of Events 93
  4.4.7 Multiple Sound Alignment .......... 93
  4.4.8 Separate Encoding of Affixes .......... 95
  4.4.9 Multilingual Alignment ............. 97
4.5 Featurewise Context Modeling ............... 99
  4.5.1 Larger Context ....................... 99
  4.5.2 Phonetic Features ................. 99
  4.5.3 Context Trees ...................... 100
  4.5.4 Codelength .......................... 102
  4.5.5 Learning ......................... 103
  4.5.6 Evaluation .......................... 104
  4.5.7 Exploiting Monolingual Rules ....... 105
4.6 Imputation ................................ 107
4.7 Phylogenetic Language Trees ................ 109

5 Summary and Current Work .................... 113

References .................................. 119

Original Publications ........................ 131
  Summary and Contributions ................. 131
List of Included Publications


For detailed references, summaries and contributions of the current author see pages 131–133.
## List of Abbreviations

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AIC</td>
<td>Akaike Information Criterion</td>
</tr>
<tr>
<td>BIC</td>
<td>Bayesian Information Criterion</td>
</tr>
<tr>
<td>DAG</td>
<td>Directed Acyclic Graph</td>
</tr>
<tr>
<td>ESS</td>
<td>Equivalent Sample Size; <em>the parameter prior for Bayesian network models, which spreads pseudo-counts evenly across the data space</em></td>
</tr>
<tr>
<td>FAN</td>
<td>Forest-Augmented Naive Bayes</td>
</tr>
<tr>
<td>fNML</td>
<td>factorized Normalized Maximum Likelihood</td>
</tr>
<tr>
<td>i.i.d.</td>
<td>independent and identically distributed; <em>the assumption, that data come from a stationary distribution, from which we repeatedly draw independent samples</em></td>
</tr>
<tr>
<td>KL</td>
<td>KL-divergence: Kullback-Leibler divergence</td>
</tr>
<tr>
<td>LR</td>
<td>Logistic Regression</td>
</tr>
<tr>
<td>MAP</td>
<td>Maximum A Posteriori; <em>the model of maximal posterior probability</em></td>
</tr>
<tr>
<td>MB</td>
<td>Markov Blanket</td>
</tr>
<tr>
<td>MDL</td>
<td>Minimum Description Length</td>
</tr>
<tr>
<td>MarLi</td>
<td>Marginal Likelihood</td>
</tr>
<tr>
<td>Abbreviation</td>
<td>Description</td>
</tr>
<tr>
<td>--------------</td>
<td>-------------</td>
</tr>
<tr>
<td>ML</td>
<td>Maximum Likelihood</td>
</tr>
<tr>
<td>NB</td>
<td>Naive Bayes</td>
</tr>
<tr>
<td>NCD</td>
<td>Normalized Compression Distance</td>
</tr>
<tr>
<td>NED</td>
<td>Normalized Edit Distance</td>
</tr>
<tr>
<td>nfp(C)</td>
<td>Number of Free Parameters (of a model class C)</td>
</tr>
<tr>
<td>NML</td>
<td>Normalized Maximum Likelihood</td>
</tr>
<tr>
<td>PCB</td>
<td>Prefix CodeBook</td>
</tr>
<tr>
<td>SCB</td>
<td>Suffix CodeBook</td>
</tr>
<tr>
<td>sNML</td>
<td>sequential Normalized Maximum Likelihood</td>
</tr>
<tr>
<td>TAN</td>
<td>Tree-Augmented Naive Bayes</td>
</tr>
<tr>
<td>UPGMA</td>
<td>Unweighted Pair Group Method with Arithmetic Mean</td>
</tr>
<tr>
<td>UTM</td>
<td>Universal Turing Machine</td>
</tr>
<tr>
<td>XIC</td>
<td>Information Criteria <em>such as AIC, BIC and so on</em></td>
</tr>
</tbody>
</table>
Chapter 1
Introduction

“Essentially, the world is probabilistic.”

‘Data modeling’ is a term which, in all its generality, applies to the subject of this thesis. It means that, given some data, we learn a machine that describes the data in a generalizing manner. This machine, or model, may then reveal properties of the data not immediately evident and can be used for inference—prediction of entities that are not given, but are assumed to come from the same generating process.

Another—even more general—term frequently used in this context is ‘data mining’. Nowadays, data can be collected automatically, cheaply and therefore in large amounts, hoping it will be useful in some way, some day. This calls for methods to process this data. Wikipedia states that “the overall goal of the data mining process is to extract information from a data set and transform it into an understandable structure for further use”. This is exactly what we will be doing with etymological data in Chapter 4 of this work, but of course this requires specification. Data mining as such simply means “to do something interesting with data”.

Data modeling, as opposed to data mining, includes the notion of a model, a machine to describe the data. Hence, in data modeling, we need to have an idea of what it is that we expect—or hope—to find in the data. Instead of simply scanning for conspicuous statistics, we have a model to describe the entire data, a simulated process to encode or generate it. The choice of an appropriate model is naturally guided by the data at hand, but also requires some initial understanding of the process that has generated the data. For any given data, multiple such choices may seem reasonable, which calls for an objective means of model (class) selection.

All interesting modeling problems involve uncertainty. It is of no prac-
tical relevance, whether the data truly come from a random process, or whether the model itself introduces uncertainty through generalization. In fact, it is a philosophical question whether true randomness exists in the first place. But whether a process is random by nature or whether we are simply unaware of the underlying determinism, in either case we need to deal with uncertainty in modeling it.

Mathematical probability is a natural means to describe—and process—this uncertainty. Our approach will be, still very generally speaking, that of probabilistic data modeling. Although the term 'probability' is being used in many different ways, all of them describe a measure of belief and uncertainty. Subjectivists, for instance, use probabilities simply to describe their personal, subjective view on things. Frequentists in turn regard probabilities as relative frequencies of repeatable random events. Bayesians combine the two by first defining a prior, a subjective view on what to expect before observing any data, and then augmenting it with the observed data frequencies to form the posterior. The posterior is the (Bayesian) belief of what the data-generating distribution most probably looks like, after having seen the data. This distribution can then be used to predict unobserved data entities, such as future data. The posterior depends on both the initial belief as described by the prior, and the given training data. In information theory, the probability of some data is considered to be two-to-the-minus number of bits needed to describe that data unambiguously, possibly subject to normalization.

We consider classes of parametric models. A model class $C$ is thus a set of models (probability distributions or densities) sharing the same parametric form, a model $M = C(\Theta)$ is defined by the model class $C$ it belongs to, together with a parameter vector $\Theta$ specifying a distribution (or density) from this class.

In model class selection problems we also speak of a model family $F$. By this we mean a collection of model classes, from which we want to choose a suitable one, guided by the given data. For example, we may start out with the model family of all polynomials of finite degree. A model class $C_n$ would then include all polynomials of given degree $n$ and a model would become a polynomial $a_n x^n + a_{n-1} x^{n-1} + \ldots + a_1 x + a_0$ with given coefficients $a_i$. But a model family need not be nested like this. It may also, for instance, be the set of possible Bayesian networks—directed acyclic graphs on a given set of variables that encode independence assumptions of the corresponding models. More generally, a model family may be any collection of model classes, which can be of numerous types and structures.

How is the model in the above example probabilistic? In general, to fit
data of size $n$ exactly we need a polynomial of degree $n - 1$, therefore there will be one parameter (coefficient) for each data item. This leads to sharply oscillating functions which are not well-suited to make predictions about areas inbetween data points. This can easily be seen by removing some data point and fitting a polynomial of degree $n - 2$. Typically, the value of this polynomial at the position of the removed data will greatly differ from the held-out value. Similarly, when fitting a polynomial of degree $n - 1$, the coefficient values are highly sensitive to noise, such as measurement inaccuracy. Since we have not actually simplified the data, but merely rewrote the $n$ data points as $n$ coefficients, the model does not generalize.

For this reason we hardly ever aim to model the data exactly, overly complex models generalize poorly. We need to choose a model class that truly simplifies the data, which typically means fitting it imperfectly. We then determine the model parameters (here: the coefficients of the polynomial) according to some objective function, e.g. goodness of fit. For polynomial fit, we often minimize the sum of squares of the residuals. Implicitly, we thereby assume the data to be normally distributed around the point indicated by the polynomial. Explicitly or implicitly, the model $M = C_n(Θ = (a, σ^2))$ includes a variance parameter $σ^2$, to define a conditional probability distribution $P(y|x, M)$, a normal distribution of variance $σ^2$ with its mean given by the polynomial with coefficients $a$ evaluated at $x$.

Comfortably, all data appearing in the course of this work are of discrete nature, all distributions will be multinomial. This simplifies many things. The parameters of the Bayesian classifiers of Chapter 2 are specified by local counts, priors take on a similar form, and the parametric complexity of a class of Bayesian forest models in Chapter 3 is guaranteed to be finite. Probabilities sum—rather than integrate—to one and form a distribution directly, without the need to specify the precision to which the data is being modeled. The only probability densities that appear are those that apply to (continuous) model parameters.

The course of this work roughly follows the topics of the six included publications, referred to as Publications I–VI. However, we draw a larger picture, filling in gaps between the publications and giving motivation step-by-step. This thesis tells a story, a thread of the author’s work in research, of which the attached publications only represent some aspects. The body of this thesis is meant to be as intuitive—easy to read—as possible, while still making all the points that seem essential and explaining all concepts that are relevant from the point of view of understanding the contents of this work. There is a tradeoff in writing, between complexity and being
comprehensive, much like the tradeoff between complexity and data fit in the modeling and coding problems appearing in this work. But in writing, complexity is not measured in number of free parameters, nor in number of bits or words, but more subjectively by how well the author’s wife manages to follow.

Chapter 2 introduces the reader to Bayesian reasoning and the related topic of Bayesian network models. This probabilistic framework can be seen as the foundation of modern probability theory and is appealing in many ways. As it separates subjective beliefs—the prior—from learned probability ratios in an explicit way, the data analyst is forced to formulate his or her expectations clearly.

There are also some drawbacks to this approach, which we explicitly identify. For one, Bayesian network models are not well-suited for supervised, discriminative learning tasks such as classification. Publication I gives theoretical evidence, and Section 2.7 provides empirical backup for this claim.

Secondly, the formulation of a reasonable prior can be quite challenging indeed, as subjective belief hardly ever translates into a prior (on the model parameters) directly. When we fix a model class that is simple with respect to the data size, then in many cases this prior has little influence on the posterior, as the sheer amount of data causes the learned probability ratio to outweigh it. However, the situation becomes very different when we want to perform model class selection. As the complexity of the most suitable model class can be seen to be a function of the data size, there seems to never be ‘a sufficient amount’ of data available. It has been shown that in many cases the choice of the most suitable model class by Bayesian means greatly depends on the chosen prior [Silander 2007].

Finally, model class selection not only depends on the prior chosen for the model parameters, but we also need to formulate a prior distribution over the model classes under consideration. As rule of thumb, more complex models should be less likely a priori, in order to prevent the learning algorithm from overfitting the training data, which would result in poor generalization capability.

The standard Bayesian approach to model class selection is the use of Bayes factors [Berger 1985], which enable us to compare model classes according to the probability they assign to the given data. In hypothesis testing these are simply likelihood ratios, the Bayes factor tells how much more likely a hypothesis has become by influence of the observed data. This does not solve the problem of defining a prior distribution over the hypotheses to choose from, but it does separate this issue from the in-
ference problem. But for the model class selection problem it gets more complicated. The Bayes factor is now the average probability assigned to the data by all models in the class, a weighted sum or integral over the respective parameter spaces. Averaging requires a (parameter) prior to average with respect to, which has strong influence on the selection that is made. There is some sort of automatic complexity penalty built in to this approach, as a larger (e.g. higher-dimensional) parameter space decreases the likelihood average, given that its mass is concentrated in a small area. In practice, however, this alone cannot entirely prevent overfitting (poor generalization).

Another way of dealing with the situation in which we choose among model classes of differing complexity is to add a complexity penalty term to the objective. This leads to the so-called information criteria, such as the Akaike Information criterion (AIC, [Akaike 1974]), the Bayesian Information Criterion (BIC, [Schwarz 1978]) and so on. These penalty terms however, typically amount to counting the free parameters of a model, preceded by some weight. This is not necessarily a good measure of complexity, especially when comparing model classes of different nature.

An entirely different solution to these problems offers the Minimum description Length (MDL) principle, which brings us to Chapter 3. We employ Kraft’s inequality to regard the probability of any data as the two-to-the-minus number of bits needed to decodably encode it—the shorter the code, the higher the probability. This information theoretic paradigm has its roots in Kolmogorov theory [Kolmogorov 1965, Li 1997]. Kolmogorov Complexity—the length of the shortest input to a universal Turing machine to produce a given string and halt—is, in a very specific theoretic way, the optimal way to define this codelength, which then translates into probabilities. Unfortunately, it has been proven to be incomputable [Li 1997].

MDL methods computably approximate the Kolmogorov complexity. The simplest way of achieving this are the so-called two-part codes. We first encode the model we want to use, and subsequently the data by means of this model. Encoding in this context simply means describing in a compact manner such that the original—model or data—can be recovered from this description. While the length of the first part of this description can also be seen as a complexity penalty term much like that of the information criteria mentioned above, in most cases this ‘prior’ can be formulated in an easy, natural and intuitive way. At the same time, the decodability requirement acts as a constant sanity check, making the researcher’s life much easier.

But often we can do better than to use two-part codes. Normalized Maximum Likelihood (NML, [Shitarkov 1987, Rissanen 1987]) is a one-part
code, defining a distribution over all possible data of given structure and size, with respect to a chosen model class directly. It does not involve any prior or model codelength and can—where it exists and computation is feasible—be used to compare model classes of utterly different structures. NML minimizes the worst-case regret, the additional description length as compared to the best model in the class that we could have chosen only with hindsight. Under the assumption that the data come from a universal distribution in the Kolmogorov sense as discussed in Section 3.3, this implies average-case minimal description length. Of course, such assumption may well be questioned. But in most cases, assuming a universal distribution is closer to the truth than any prior distribution we are able to guess (and formulate).

The fundamental problems with NML are issues of its efficient computability. While two-part codes are—in many cases—easy to compute, and Kolmogorov Complexity cannot be computed at all, NML lies somewhere inbetween. It is superior to, but harder to compute than two-part codes, and inferior to Kolmogorov Complexity but in exchange computable. For some simple model classes it is also efficiently computable. Publication II investigates efficient computation of the NML distribution for a family of Bayesian network models, the so-called Bayesian forests. Since the NML distribution has many desirable properties, we want to be able to employ it in as many cases as possible, which makes NML computability issues an interesting topic for the scientific community.

MDL, both in the form of a two-part code and using NML, has been successfully applied to a wide range of practical applications. These include—but are not limited to—histogram density estimation [Kontkanen 2007b], image denoising [Roos 2005a], clustering [Kontkanen 2006] and DNA sequence compression [Korodi 2005].

In the work presented in this thesis, we have added etymology to this list, the study of the history of words. Specifically, we have developed models for the problem of etymological alignment, which we use to investigate processes of phonetic sound change in natural languages. Probabilistic and information-theoretic methods have previously been applied in many areas of linguistics, including morphology [Creutz 2007], syntax [Stolcke 1994] and topic modeling [Blei 2003]. In recent years we have also seen a new interest in computational historical linguistics, as phylogenetic methods have been transferred from evolutionary biology onto this field [Greenhill 2009]. Our MDL approach models etymological data on the level of single sounds, but may also be used to infer language phylogenies.

Chapter 4, as well as the attached Publications III–VI, present MDL
methods for the analysis of etymological data, which comes to us in the form of cognate sets, collections of related words from a family of kindred natural languages. Our methods can be applied to any such data. As a running example in this work we use the StarLing Data on the Uralic language family. While Chapters 2 and 3 give us theoretic foundation for these methods, Chapter 4 is the application part of this thesis.

Being interested in the changing of sounds across languages, we must align words from different languages to obtain information on which symbols or sounds do in fact correspond. Our models provide such alignment in an automated fashion and generate rules of phonetic change that are valid throughout a set of languages. We also use the models and induced alignments to build phylogenetic trees, depicting the inferred history of language separation within the family. This approach is not only an interesting, novel application of MDL, but also offers new insight to linguists working in the field. The corresponding software is currently being made publicly available for anyone to use on their favourite language family and will appear on the project homepage at etymon.cs.helsinki.fi.

Chapter 5 provides a summary of this work, draws conclusions and discusses directions of current and future research. Following the references, there is a brief summary of the included publications, indicating the contributions of the current author. Reprints of these publications are to be found at the very end of this book.
1 Introduction
Chapter 2

Bayesian Reasoning

“All models are wrong, but some are Bayesian.”

Bayesian reasoning has got its name from the English mathematician (and Presbyterian minister) Thomas Bayes (1701–1761), who formulated what was to become known as Bayes’ Rule as a solution to a problem of ‘inverse probability’. Bayes’ Theorem in turn, a generalized version of this result, is being accounted to Pierre-Simon Laplace (1749–1827). But it was not until about 1950 that the term Bayesian has come into use, and only in the 1980s have Bayesian methods begun to spread. There are a number of good books on the subject, such as [Berger 1985] and [Bernardo 1994], to name only a couple.

2.1 Bayes’ Rule

Bayes’ Rule is the central foundation of Bayesian reasoning. It is a direct consequence of the chain rule of probabilities which states that the joint probability of a set of random variables can be written as a chain of conditional probabilities.

Rule 1 (Chain Rule) For a set \( A = \{A_1, \ldots, A_M\} \) of random variables \( A_i \), their joint probability can be rewritten as

\[
P(A) = P(A_1)P(A_2|A_1) \cdots P(A_M|A_1, \ldots, A_{M-1})
= \prod_{i=1}^{M} P(A_i|A_1, \ldots, A_{i-1}). \quad (2.1)
\]
The same is true for any set $A = \{A_1, \ldots, A_M\}$ of events with corresponding probabilities $P(A_i)$.

Throughout this thesis we will use boldfaced symbols to denote sets, vectors or matrices and thereby visually separate them from single numbers or entities.

Since $A$ is an unordered set, we are free to renumber the $A_i$, change the ordering, and thus for random variables $A$ and $B$ we can write

$$P(A, B) = P(A)P(B|A) = P(B)P(A|B).$$

(2.2)

In MDL, where we encode events (or random variables) $A$ and $B$—or a larger set $A$—this simply means that we may do so in any order we choose. Bayesians use the above to invert probabilities. This is Bayes’ Rule, which is most frequently encountered in the following shape.

**Rule 2 (Bayes’ Rule)** For any event $A$, and an event $B$ with non-zero probability $P(B)$, we have

$$P(A|B) = \frac{P(B|A)}{P(B)}P(A).$$

(2.3)

This observation can be utilized in various ways. For one, it couples the probability of a model $M$ given data $D$ to its reverse—the likelihood of $D$ under $M$:

$$P(M|D) = \frac{P(D|M)}{P(D)}p(M),$$

(2.4)

and further, as $P(D)$ is independent of $M$,

$$P(M|D) \propto P(D|M)p(M).$$

(2.5)

That is, the conditional probability of model $M$ given data $D$ is proportional to the likelihood of $D$ under $M$ times the prior probability of $M$, which we choose to visually separate from all other probabilities by using a lower case $p$. An easy—but problematic—choice for this prior is the so-called uniform prior $p(M) \propto 1$. In case there either are no known preferences on the set of models $M \in \mathcal{C}$, or these preferences cannot be formulated as a prior distribution, this is sometimes seen as the non-informative
choice, as it means ‘to me any model is—a priori—as good as any other’. Under the uniform prior (2.5) simplifies to

\[ P(M|D) \propto P(D|M), \]  

(2.6)

that is, the probability of a model \( M \) given data \( D \) is directly proportional to the probability it assigns to that data.

But the models of a class are indexed by a set of parameters, and thus the chosen prior is also a distribution over the parameters of a model. Therefore, ‘the uniform prior’ can be uniform only with respect to a chosen parametrization. A solution to this problem was offered by [Jeffreys 1946], and Jeffreys’ prior is often regarded to be truly non-informative. In any case, in Bayesian reasoning there is no way around the formulation of a prior, for without it there can be no posterior, the probability of a model \( M \) given data \( D \).

It is important to note that the prior has to be given before any data has been observed. The frequently used term ‘data prior’ is a contradiction in terms; using the same data to formulate a prior and to calculate the posterior from it is circular reasoning.

With a uniform prior, the posterior is proportional to the likelihood of \( D \) under \( M \):

\[ P(M|D) \propto P(D|M). \]  

(2.7)

So what does this mean? Under the assumption that our data come from a distribution which lies within our chosen model class \( C \), when we have specified a prior distribution over all models \( M \in C \), this gives us a posterior distribution over all models, consisting of probabilities for each \( M \) to be the one that generated \( D \). This posterior combines our prior belief with the evidence that the data provide. We play the so-called prior-to-posterior game, where the data likelihood transforms the prior into a posterior distribution over the models in the class under consideration.

Under the frequentist assumption of independent, identically distributed (i.i.d.) data from a random source, it can be shown that in this way—given that our prior assigns non-zero probability to the correct model—we will, with increasing data size, eventually find it, or approximate it to arbitrary precision. Moreover, if the generating distribution lies outside of the model class under consideration, we still minimize the Kullback-Leibler Divergence between the true and the estimated distributions in the limit [Gelman 1995]. In both cases, the actual prior chosen is irrelevant, as the data likelihood overrides it when we throw an unlimited amount of data at the model.
However, it is questionable whether a 'true distribution' needs to exist at all. On one occasion the author had the pleasure to witness, Jorma Rissanen illustrated this point by drawing a number of chalk dots on the blackboard, then asking the audience which distribution they assessed these had come from. Furthermore, in-the-limit results are of little use in practical applications with limited data availability. They can act as a sanity check, as any method should at least consistent, i.e., find the optimal model in case data availability is not an issue. But for data of limited size the prior can play an important role, and its formulation is not an easy task.

In this context, one often hears George E. P. Box’s famous quote ‘all models are wrong, but some are useful’ [Box 1979], which is of course hard to disagree with. In the following, we will keep a close eye on the aspect of usefulness.
2.2 Marginal Likelihood

It has been noted, that the most probable model in a model class, the **maximum a posteriori** (MAP) model, is not the most suitable—or useful—for the task of **prediction**. Take as an example the data to be a single toss of a biased coin. The corresponding model class \( \mathcal{C} \) is the set of **Bernoulli** models, indexed by the single free parameter \( \Theta \), defining the probability of heads facing up. Assuming a uniform prior over the values of \( \Theta \in [0, 1] \), the MAP model will then be either \( M = \mathcal{C}(\Theta = 0) \) or \( M = \mathcal{C}(\Theta = 1) \), depending on whether we observe tails or heads. Using this model for prediction of the second toss would mean giving probability one to the outcome being the same as the first. Clearly, this does not reflect any belief we could reasonably argue.

Instead, when predicting, we can use all models in the class, integrating over all parameters/models. For an i.i.d. sample \( D = \{d_1, \ldots, d_n\} \), the **marginal likelihood** (MarLi, see, e.g. [MacKay 2002]) is given by

\[
P(D|\mathcal{C}) = \int_\Theta P(D|M = \mathcal{C}(\Theta))p(M = \mathcal{C}(\Theta))d\Theta. \tag{2.8}
\]

Whenever \( \mathcal{C} \) is fixed, we may drop it from the notation and simply write

\[
P(D) = \int_\Theta P(D|\Theta)p(\Theta)d\Theta. \tag{2.9}
\]

For the binomial distribution, **conjugate priors** are given by a **Beta distribution**, and, for multinomial distributions over more than two values, by its generalized form, the **Dirichlet distribution**, see [MacKay 2002]. By conjugate we mean that the prior and the posterior distributions are of the same form. Therefore, we can play the prior-to-posterior game any number of times and include more data into the posterior as it arrives. By the chain rule (Rule 1), the posterior is invariant to the split and ordering by which we add data:

\[
P(\Theta, D = \{D_1, D_2\}) = P(D_1, D_2|\Theta)p(\Theta) \overset{i.i.d.}{=} P(D_1|\Theta)P(D_2|\Theta)p(\Theta)
= P(D_2|\Theta)P(\Theta, D_1) = P(D_1|\Theta)P(\Theta, D_2). \tag{2.10}
\]

Also, for the posterior we have

\[
P(\Theta|D_i) \propto P(\Theta|D_i)P(D_i) = P(\Theta, D_i) \tag{2.11}
\]

since the data probability \( P(D_i) \) is constant wrt. the chosen model.
For conjugate priors, where the posterior \( P(\Theta|D_1) \) is of the same form as the prior \( p(\Theta) \), we can therefore regard it as a new prior before observing the second batch \( D_2 \). For multinomial distributions, when we choose a Dirichlet prior, we always remain within the world of Dirichlet distributions.

The prior \( p \), which is uniform with respect to the standard parametrization of a multinomial distribution, is conjugate and given by

\[
p \sim \text{Dir}(1, \ldots, 1).
\]

In the above coin tossing example, the uniform (flat) prior therefore is

\[
p(\Theta, 1 - \Theta) \sim \text{Dir}(\alpha_{\text{heads}}, \alpha_{\text{tails}}) = \text{Dir}(1, 1)
\]

and the marginal likelihood solves to

\[
P(D) = \frac{c(\text{heads})!c(\text{tails})!}{(c(\text{heads}) + c(\text{tails}) + 1)!},
\]

where \( c(\text{heads}) \) and \( c(\text{tails}) \) are the counts observed in \( D \).

Prediction of the next sample \( d_{n+1} \) turning up heads becomes (with \( \alpha_{\text{heads}} = \alpha_{\text{tails}} = 1 \))

\[
P(d_{n+1} = \text{heads}|D) = \frac{P(D, d_{n+1} = \text{heads})}{P(D)}
\]

\[
= \frac{(c(\text{heads}) + c(\text{tails}) + 1)! c(\text{heads} + 1)! c(\text{tails})!}{(c(\text{heads}) + c(\text{tails}) + 2)! c(\text{heads})! c(\text{tails})!}
\]

\[
= \frac{c(\text{heads}) + 1}{c(\text{heads}) + c(\text{tails}) + 2},
\]

and thereby the probability of seeing the same outcome a second time, after having observed just one toss of the biased coin, under the uniform prior becomes \( \frac{2}{5} \).

In general for a \( K \)-valued multinomial, taking on values in \{1, \ldots, \( K \)\}, assuming any Dirichlet prior \( \alpha \sim \text{Dir}(\alpha_1, \ldots, \alpha_K) \) the marginal likelihood has the following form [Heckerman 1995].

\[
P(D) = \int_{\Theta} P(D|\Theta)p(\Theta)d\Theta = \frac{\prod_k \Gamma(c(k) + \alpha_k)}{\Gamma(\sum_k \alpha_k)} \frac{\Gamma(\sum_k (c(k) + \alpha_k))}{\prod_k \Gamma(\alpha_k)}
\]

Here, the \( c(k) \) are the observed counts of each value \( k \) in \( D \), and \( \Gamma \) is the gamma function.

Prediction of an unobserved value \( d \) takes on the following simple form:
2.2 Marginal Likelihood

\[ P(d = k|D) = P(d = k|c) = \frac{c(k) + \alpha_k}{\sum_k(c(k) + \alpha_k)}. \]  

(2.17)

This takes on the same form as the MAP model or, in fact, any model in the class of multinomials. Therefore, we can speak of the **marginal likelihood model**, which emulates the use of all models in the class. While the parameters of the MAP model are just the observed relative frequencies, the parameters of the marginal likelihood (MarLi) model are the relative augmented frequencies, to which we have added the pseudo-counts given by the Dirichlet prior. The count vector \( c \) completely (and minimally) determines both MAP and MarLi models.

But marginal likelihood is not only a tool for prediction. Without it, the data likelihood \( P(D|C) \)—given just the model class with no parameters—is undefined and meaningless. The Bayesian answer is to integrate over all possible parameters, weighted by their prior probability. If these prior probabilities are conjugate, i.e. in the case of multinomials follow a Dirichlet distribution, then this can be done elegantly and efficiently. Now we can, by Bayes’ rule, also calculate the reverse—\( P(C|D) \)—the probability that the data has been generated by a model \( M \in C \), as we shall further explore in Section 2.4.

**Remark 1 (on prequential probability)**

The marginal likelihood for multinomial distributions as given by Equation 2.16 can also be seen as the **prequential** (for ‘predictive sequential’ [Dawid 1984]) probability. The rationale goes as follows. We interpret the Dirichlet prior as pseudo-counts, e.g. for the uniform prior we pretend that we have seen one of each kind in advance. For a data set of size \( n \) we then play the prior-to-posterior game \( n \) times. For example, for a series of \( n \) coin tosses \( d_1, \ldots, d_n \) consisting of \( c(\text{heads}) \) heads and \( c(\text{tails}) \) tails, the first outcome will always get probability \( \frac{1}{2} \), the second either \( \frac{1}{3} \) or \( \frac{2}{3} \), as we have added the first toss to the pseudo-counts, and so on.

If we order the series heads first, tails last, the sequence of probabilities \( P(d_j|d_1, \ldots, d_{j-1}) \) becomes:

\[
\begin{align*}
1 & \quad 2 & \quad c(\text{heads}) \\
2 & \quad 3 & \quad c(\text{heads}) + 1 & \quad 1 & \quad c(\text{tails}) \\
& \quad \ldots & \quad c(\text{heads}) + 1 & \quad c(\text{heads}) + 2 & \quad \ldots & \quad c(\text{tails}) + c(\text{heads}) + 1
\end{align*}
\]

and the probability of the complete sequence—the product of the above terms—is given by Equation 2.14. Reordering the sequence only permutes the enumerators, and thus leaves the product unchanged. Therefore the marginal likelihood can be interpreted as the product of the sequence of predictive probabilities \( P(d_j|d_1, \ldots, d_{j-1}) \).
2.3 Bayesian Network Models

So far we have dealt only with a single multinomial variable. The situation gets more interesting, when there are multiple, potentially interdependent variables involved. Suppose that data $D$ consists of $m$ multinomial variables $X = \{X_1, \ldots, X_m\}$, where each $X_i$ can take on values in $\{1, \ldots, K_i\}$. Then by the chain rule their joint probability distribution is given by a product of conditional probability distributions

$$P(X) = \prod_{i=1}^{m} P(X_i|X_1, \ldots, X_{i-1}),$$

where of course we are free to choose the ordering of the $X_i$.

Each such conditional probability distribution $P(X_i|X_1, \ldots, X_{i-1})$ is represented by a table of size $\prod_{i'=1}^{i} K_{i'}$ listing the single probabilities $P(X_i = x_i|X_1 = x_1, \ldots, X_{i-1} = x_{i-1})$, which are the parameters of the corresponding model. This easily becomes a large number of parameters. Counting the free parameters, for the joint distribution (2.18) we get an overall number of $\prod_{i=1}^{m} K_i - 1$. When data availability is limited—as it tends to be—already for a very moderate amount of variables there clearly is too little support for each parameter; the full model is overly complex and we need to simplify it.

Bayesians do so by introducing independence assumptions. Each conditional probability distribution is approximated by a distribution conditioned only on a subset $Pa(i) \subseteq \{X_1, \ldots, X_{i-1}\}$ of the preceding variables, such that we have

$$P_B(X) = \prod_{i=1}^{m} P(X_i|Pa_i^B).$$

These independence assumptions can then be depicted in a Bayesian Network $B$—which we have included in the above equation as an index—such that $Pa_i^B$ is the parent set of $X_i$ in $B$. As the network $B$ completely defines the parametric structure of the associated model class $C_B$ we can identify the two. To simplify notation, in the following we may write $B$ in place of $C_B$—e.g., $B \in \mathcal{F}$ in model class selection—$B(\Theta)$ instead of $C_B(\Theta)$ for the instantiated model, and so on.

More formally, a Bayesian Network is a directed acyclic graph (DAG) on the relevant variables of a problem which is interpreted as in Equation 2.19. Acyclicity ensures that—after suitably reordering the variables $X_i$—we indeed have $Pa_i \subseteq \{X_1, \ldots, X_{i-1}\}$ for all $i$ [Pearl 1988], such that there can be no circular reasoning.
To make this more clear, let us look at an example—the **burglar alarm** problem introduced in [Pearl 1988]—which is one of the most commonly used in tutorials and educational material.

**Example 1 (Burglar alarm)** Suppose you are at work and your neighbour gives you a call (\(C\)) to inform you that the burglar alarm (\(A\)) at your home has gone off. So, thinking there has been a burglary (\(B\)), you jump into your car and drive home. On the way, you listen to the news (\(N\)) reporting an earthquake (\(E\)) in the area where you live. This changes your degree of belief about a burglary having actually occurred.

There are five relevant variables involved: \(C, A, B, N\) and \(E\). They are all binary (boolean), taking on values in \(\{t, f\}\), short for \(\{true, false\}\). A corresponding Bayesian Network \(B\) that we might want to use is given in Figure 2.1.

![Figure 2.1: A Bayesian Network \(B\) for the burglar alarm problem.](image)

The joint probability distribution then simplifies to

\[
P_B(E, B, N, A, C) = P(E)P(B)P(N|E)P(A|E, B)P(C|A). \tag{2.20}
\]

\(P(E)\) depends on the area where you live and so does \(P(B)\), \(P(A|E, B)\) depends on the quality of your alarm system, and so on. Let us now define the model, by writing down the probability tables for each of these conditional distributions in Table 2.1. Note that this is a significant simplification to arbitrary joint distributions represented by a fully connected Bayesian network and specified by a probability table of size \(2^5 = 32\). The full table lists 31 free parameters (the 32nd given by the preceding 31 as...
Table 2.1: Subjective conditional probabilities for the burglar alarm problem.

We can now calculate the probability you would give to the event of a burglary at your home after your neighbour has called by summing over the unobserved values \( e \) of \( E \), \( n \) of \( N \) and \( a \) of \( A \) as well as normalizing over the values \( b \) of \( B \).

\[
P_B(B = t|C = t) = \frac{\sum_{e,n,a} P(e)P(B = t)P(n|e)P(a|e,B = t)P(C = t|a)}{\sum_{e,b,n,a} P(e)P(b)P(n|e)P(a|e,b)P(C = t|a)}
\]

\[
= \frac{\sum_{e,a} P(e)P(B = t)P(a|e,B = t)P(C = t|a)}{\sum_{e,b,a} P(e)P(b)P(a|e,b)P(C = t|a)} \\
\approx 13.6\% \quad (2.21)
\]
and similarly the belief, after the earthquake news has been broadcast changes to

\[
P_B(B = t|N = t, C = t) \approx 5.4\%.
\] (2.22)

The news about the earthquake has influenced our belief about there having been a burglary. If there had not been a call from the neighbour about the alarm sounding, this would not have been the case, as earthquake (E) and burglary (B) are modeled as being independent events in \( \mathcal{B} \). Let us now rewind and introduce some definitions.

**Definition 1 (d-separatedness)** Let \( \mathcal{B} \) be a Bayesian network on variables \( X = \{X_1, \ldots, X_m\} \) and \( Z \subseteq X \) be a subset of these variables. Then any two variables \( X_i, X_j \notin Z \) are said to be d-separated by \( Z \) in \( \mathcal{B} \), if every trail (path in the underlying undirected network) \( T \) from \( X_i \) to \( X_j \) is blocked by \( Z \), that is, we encounter at least one of the following situations:

- \( T \) contains a partial trail \( X_u \rightarrow X_v \leftarrow X_w \) such that either \( X_v \) or a descendant of \( X_v \) is an element of \( Z \)

- \( T \) contains a partial trail \( X_u \rightarrow X_v \rightarrow X_w \) such that at least one of the two arcs points away from \( X_v \), and \( X_v \notin Z \)

Otherwise, \( X_i \) and \( X_j \) are said to be d-connected in \( \mathcal{B} \) given \( Z \).

Equivalently, \( X_i \) and \( X_j \) are d-separated (d for 'dependence') by \( Z \) in \( \mathcal{B} \) if, and only if, for any \( \mathcal{B} \)-model they are conditionally independent given \( Z \). In other words, for any set of parameters \( \Theta \) of \( \mathcal{B} \) we have

\[
P(X_i, X_j|Z, \mathcal{B}(\Theta)) = P(X_i|Z, \mathcal{B}(\Theta))P(X_j|Z, \mathcal{B}(\Theta)).
\] (2.23)

Therefore, d-separatedness is a means to read off conditional independence assumptions from a network \( \mathcal{B} \).

In the burglar alarm example, \( N \) is d-separated from \( B \) given \( Z = \emptyset \) (burglary and earthquake—or the news about one—are initially independent events), but d-connected given \( \{C\} \), the call providing evidence for the alarm \( A \). Therefore, after the phone call has been received, the radio news can influence our belief in the event of a burglary.

**Definition 2 (collider)** A substructure \( X_u \rightarrow X_v \leftarrow X_w \) of a Bayesian network \( \mathcal{B} \) is called a collider (inverted fork).
A collider thus blocks a trail, if no evidence for its middle node \( X_v \), i.e. for none of its descendants nor \( X_v \) itself, is given (in \( Z \)). In all other situations, a path is blocked by a variable \( X_v \) which is given.

**Definition 3 (v-structure)** A substructure \( X_u \to X_v \leftarrow X_w \) is called a v-structure (unshielded collider) if there is no direct arc between \( X_u \) and \( X_w \) in \( \mathcal{B} \).

**Lemma 1** When checking for conditional independence, it suffices to consider v-structures instead of colliders.

**Proof** Let \( T \) be a trail between \( X_i \) and \( X_j \) which is not blocked by \( Z \), and \( X_u, X_v, X_w \) form a shielded collider along \( T \) with tip \( X_v \). It follows that \( X_u \notin Z, X_w \notin Z \) and there is evidence about \( X_v \) in \( Z \) (\( X_v \) or at least one of its descendants is in \( Z \)).

Then also the trail \( T \setminus \{X_v\} \), which short cuts from \( X_u \) to \( X_w \), is not blocked, since both \( X_u \) and \( X_w \) lie outside of \( Z \), but have a descendant which lies inside of \( Z \). Therefore neither of the situations of Definition 1 applies to a partial trail of \( T \setminus \{X_v\} \) consisting of three nodes with middle node \( X_u \) or \( X_w \).  

**Definition 4 (network equivalence)** Bayesian networks \( \mathcal{B} \) and \( \mathcal{B}' \) are said to be equivalent, if they encode the same conditional independence assumptions.

**Lemma 2** Bayesian networks \( \mathcal{B} \) and \( \mathcal{B}' \) are equivalent, if and only if their underlying undirected graphs are identical and both have the same v-structures.

This is a direct consequence of Lemma 1.

**Example 2** The networks depicted in Figures 2.1 and 2.2 are equivalent. The arc between variables 'earthquake' and 'news' has been reversed, but this changes no v-structures. Although the latter seems false on an intuitive level, it encodes the exact same conditional independence assumptions. Bayesian networks do not encode causal dependencies.

**Definition 5 (Markov blanket)** The Markov Blanket \( MB_\mathcal{B}(X) \) of a node \( X \) in \( \mathcal{B} \) is the set of all its parents, all its children, and all parents of any of its children in \( \mathcal{B} \).

The Markov Blanket of \( X \) consists of exactly those nodes on which \( X \) is dependent—not d-separated—in the fully instantiated case. That is, for \( X \neq Y \in X \) and \( Z := X \setminus \{X,Y\} \), it holds that
Figure 2.2: An alternative Bayesian network for the burglar alarm problem, equivalent to the network of Figure 2.1

\[ Y \notin MB_B(X) \iff [P(X|Y, Z, B(\Theta)) = P(X|Z, B(\Theta)) \text{ for all } \Theta]. \quad (2.24) \]

In equivalent networks all nodes have identical Markov blankets.

Using network \( B \) from Figure 2.1 with parameters \( \Theta \) given by Table 2.1 we can calculate the joint probability of any instantiation \((e, b, n, a, c)\) of the variables \( E, B, N, A, C \). Using Bayes’ rule we can also—by marginalizing over unobserved values—calculate all conditional probabilities. After receiving the phone call we had \( P(B = t|C = t, M = B(\Theta)) \approx 13.6\% \), and after listening to the news about the earthquake, the probability of a burglary had dropped to \( P(B = t|C = t, N = t, M = B(\Theta)) \approx 5.4\% \). This phenomenon, in which the branches of a v-structure become dependent in the case where we have evidence for the tip of the v-structure, is called explaining away. We have explained away the burglary by finding another reasonable cause—the earthquake—for the behaviour of their common child, the alarm.
2.4 Bayesian Model Class Selection

Model class selection is the problem of, given some data \( D \), finding a suitable model class \( C \) to describe it. In the Bayesian approach, this means calculating the probability \( P(C|D) \), which by Bayes’ rule is

\[
P(C|D) = \frac{P(D|C)P(C)}{P(D)}. \tag{2.25}
\]

The marginal data likelihood \( P(D|C) \) is defined in Equation 2.8, \( P(C) \) is another prior distribution, which we need to define over all model classes \( C \in \mathcal{F} \) in the model family under consideration, and the term \( P(D) \) is independent of \( C \) and thus serves as a normalizing constant.

One may then interpret \( P(C|D) \) as a degree of belief that the distribution \( \mathcal{M} \) that has generated \( D \) is a member of \( C \), and conclude that the maximizing \( C \) is the most probable class.

This approach, simple as it sounds, is problematic in a number of ways, as we argue in the following. To make our criticism more explicit, we take as an example the model family consisting of all Bayesian networks on \( m \) multinomial variables, out of which we are to choose a suitable structure \( \mathcal{B} \) for a given data set \( D \) of size \( n \). The Bayesian approach to model class selection as such does not limit us to Bayesian network models, and it is more of a coincidence that both are being labeled ‘Bayesian’. However, considering Bayesian network models is a good means to identify the problems arising in Bayesian model class selection.

Let \( X = \{X_1, \ldots, X_m\} \) be the data space consisting of \( m \) multinomial variables \( X_i \) of corresponding cardinalities \( K_i \) and \( D \) be an \( n \)-fold i.i.d. sample, i.e. an \( n \times m \)-matrix, where each entry \( d_{ji} \in \{1, \ldots, K_i\} \) is the value that sample \( d_j \) (\( j^{th} \) row) takes on at variable \( X_i \) (\( i^{th} \) column). We search for the most probable Bayesian network \( \mathcal{B} \), the one that maximizes Equation 2.25.

Each Bayesian network \( \mathcal{B} \) is parametrized by a vector \( \Theta^{\mathcal{B}} \) with components of the form \( \Theta^{\mathcal{B}}_{x_i|\text{pa}_i} = P(X_i = x_i|\text{pa}_i = \text{pa}_i, \mathcal{M} = \mathcal{B}(\Theta^{\mathcal{B}})) \), which are the entries of the conditional probability tables such as the ones listed in Table 2.1. The joint data likelihood of data set \( D \) given network structure \( \mathcal{B} \) becomes

\[
P(D|\mathcal{B}(\Theta^{\mathcal{B}})) \overset{i.i.d.}{=} \prod_{j=1}^{n} P(d_j|\mathcal{B}(\Theta^{\mathcal{B}})) = \prod_{j=1}^{n} \prod_{i=1}^{m} \Theta^{\mathcal{B}}_{d_{ji}|\text{pa}_i(d_j)}, \tag{2.26}
\]

where \( \text{pa}_i(d_j) \) is the instantiation of the parent set \( \text{pa}_i \) of variable \( X_i \) in \( \mathcal{B} \) which is given by the \( j^{th} \) sample \( d_j \).
If we now define a Dirichlet parameter prior for each conditional distribution \( P(X_i|Pa_i = pa_i, B(\Theta^B)) = \Theta^B_{pa_i} \) such that

\[
\Theta^B_{pa_i} \sim \text{Dir}(\alpha_{1|pa_i}, \ldots, \alpha_{K_i|pa_i})
\]  

we also have the marginal data likelihood

\[
P(D|B) = \int_{\Theta^B} P(D|B(\Theta^B))p(\Theta^B|B)d\Theta^B
\]

\[
= \prod_{i=1}^{m} \prod_{pa_i \in Pa_i} \left( \frac{\prod_{k=1}^{K_i} \Gamma(c(k|pa_i) + \alpha_{k|pa_i}) \Gamma(\sum_{k=1}^{K_i} \alpha_{k|pa_i})}{\Gamma(\sum_{k=1}^{K_i} (c(k|pa_i) + \alpha_{k|pa_i})) \prod_{k=1}^{K_i} \Gamma(\alpha_{k|pa_i})} \right). 
\]  

(2.28)

Whereas in Equation 2.16 we only had one multinomial distribution, we now take the product over all conditional distributions appearing in \( B \). Again, \( c(k|pa_i) \) are the observed data counts and \( \Gamma \) is the Gamma function.

Let us now look at the problems arising, when we seek the 'most probable' Bayesian network for given data.

**Problem 1 (the generating distribution assumption)**

In order to define a distribution over the model classes \( C \in F \), consisting of the probabilities that a model \( M \in C \) has generated the data \( D \), we need to be sure that in fact such model exists.

It is a philosophical question, whether or not any generating distribution exists. But let us assume so. We still need to be sure that it is part of a model class in \( F \).

In our structure learning example, any multivariate multinomial distribution lies in \( B_{\text{full}} \), the class corresponding to any fully connected network (all of which are equivalent). But we have also made the i.i.d. assumption. If it does not hold—the data source was not completely stationary after all—it follows that the generating distribution lies outside of \( F \), which renders the search for the most probable model or model class meaningless.

Unless we are certain of our assumptions, i.e. we have generated the data ourselves, all we can hope for is to find a useful network structure, one that will give us good predictive performance.

**Problem 2 (overlapping model classes)**

The model classes in \( F \) may be overlapping, i.e. some of the models may be contained in multiple classes. In this case, \( P(C|D) \) should not be a distribution, but a superdistribution—sum to more than one—since models in the areas of overlap contribute their corresponding probability mass to more than just one class.
This is true for the structure learning problem, not only because of network equivalence. Observe, that adding arcs to a network $B_1$ to arrive at a network $B_2$ only makes the model class larger (reduces the conditional independence assumptions), such that we have $B_1 \subset B_2$ (as sets of distributions). More severely

**Problem 3 (undefined data probability)**

*The term $P(D)$ in Equation 2.25 is undefined as such.*

We can, however, interpret it as

$$P(D|\mathcal{F}) = \sum_{B \in \mathcal{F}} P(D|B)P(B), \tag{2.29}$$

that is, as a normalizing constant. But usually the model classes in $\mathcal{F}$ are not disjoint, as is the case for the family of Bayesian network structures.

**Problem 4 (large number of model classes)**

*Under the assumption that $P(D|\mathcal{F})$ is meaningfully defined by Equation 2.29, we may still not be able to compute it, if the number of model classes in $\mathcal{F}$ is large.*

The number of Bayesian network structures is huge already for a reasonably small number of variables. There is only one structure for a single variable, for two variables there are three networks (two of which are equivalent), and for five variables—the situation of the burglar alarm example—there are already 29,281 different structures. The number of networks grows superexponentially with the number of variables, which can be seen by only considering chains of the form $X_1 \rightarrow X_2 \rightarrow \cdots \rightarrow X_m$. For each ordering of the variables there is one such chain, which means that there are already $m!$ chains for $m$ variables. Restricting to equivalence classes of structures does not change the situation either, since for each $m$ there are only two equivalent chains. For 10 variables we have more than $10^{19}$ network structures, and for 20 variables this number grows to over $10^{73}$ [Robinson 1977]. The number of equivalence classes is of the same order [Gillispie 2001].

This means that, in general, $P(D|\mathcal{F})$ is beyond computational capacity. We can, however, still compute the relative probability for any two networks. In other words, we can still search for the best structure $\mathcal{B}$. Unfortunately, the absolute probabilities $P(\mathcal{B}|D,\mathcal{F})$ will be very small. By Bayesian reasoning, the most probable network will therefore almost certainly be wrong.
Summarizing over Problems 1–4, we find that it makes no sense to talk about the most probable model class. But can we still search for the most useful class, the one that predicts best? The answer to this question—naturally—greatly depends on the priors we choose.

**Problem 5 (the parameter prior)**

*For each model class (Bayesian network structure $B$) we need to define a suitable (Dirichlet) parameter prior.*

As discussed in Sections 2.1 and 2.2, defining a reasonable parameter prior $\alpha$ is not an easy task. Since usually we do not have expert domain knowledge available, or are unable to transform it into Dirichlet distributions, a frequently made choice is the so-called 'uniform' prior $\alpha_{\text{pa}_i} \sim \text{Dir}(1, \ldots, 1)$, which assigns equal probability to all parameter vectors $\Theta^B$. As the uniform prior also assigns equal probability to all data sets of size 1, that is, to the first sample in the prequential game, it is often thought to be 'non-informative', see, e.g. [Hill 1997].

However, we need to remember that 'uniformity' is a property only defined with respect to a given parametrization. In fact it turns out that the 'uniform' prior, in general, yields different marginal likelihood for a data set $D$ when using different, but equivalent Bayesian networks (Definition 4), which encode the exact same independence assumptions and consist of the exact same models (joint probability distributions). Equivalent network structures differ in the way they are parametrized and therefore the 'uniform' prior can have a different meaning depending on the actual representative one chooses for a given equivalence class, see Figure 2.3.

Jeffreys’ prior [Jeffreys 1946] is invariant to reparametrization, yet being used less frequently, largely for technical reasons. Its main drawback is that it does not take on the Dirichlet form and hence is not conjugate. Both

![Figure 2.3: Two equivalent Bayesian network structures $B_1$ (left) and $B_2$ (right). For the 'uniform' parameter prior the associated marginal likelihood distributions are different, iff the cardinalities of the multinomials are: $K_1 \neq K_2 \Leftrightarrow P(\cdot|B_1) \neq P(\cdot|B_2)$. This can be seen most easily by interpreting the Dirichlet prior as pseudo-counts.](image-url)
Bayesian Reasoning

prior and posterior take on forms which cannot typically be formulated in closed form and need to be approximated.

It can be shown, that the only Dirichlet priors which are invariant to this type of parameter transformation are those that, when viewed as pseudo-counts, correspond to actual data samples, non-integer counts allowed. Therefore the only 'non-informative' Dirichlet priors, in the forementioned sense, are the so-called equivalent sample size (ESS) priors given by

$$\alpha^{ESS}(S) : \Theta_{\{\mathrm{pa}_i\}} \sim \text{Dir} \left( \frac{S}{K_i}, \ldots, \frac{S}{K_i} \right) \quad \text{with} \quad K_i := K_i \prod_{i' \in \text{Pa}_i} K_{i'}.$$ (2.30)

Using this type of parameter prior, we are left to choose only a single parameter, the size $S$ of the assumed pseudo-data, which is then being spread evenly across the data space $X$.

Unfortunately, there is no natural choice for $S$, nor is the resulting prior non-informative. As it turns out, the choice of the most probable network is extremely sensitive to this parameter [Silander 2007].

**Problem 6 (the class prior)**

*We need to define a suitable prior distribution $P(C)$ over all classes in the family $\mathcal{F}$.*

In structure learning, once more for lack of better knowledge, a uniform prior over the network structures is often assumed. Due to the large number of structures, all of them are therefore assigned very low prior probability. Sometimes, also a uniform prior over the equivalence classes is being used, which leads to similar results.

If we search for the *most probable* class, then we should assign probability one to the equivalence class of fully connected networks, as it contains all multivariate multinomial distributions. If, however, we look for a *useful* network, we hope for something much simpler, as it will have larger support for its parameters and generalize to unseen cases better.

**Problem 7 (overfitting)**

*A too complex model class will overfit the observed data, generalize poorly and not be useful at all.*

The principle of **Ockham’s razor** tells us to choose the *simplest* hypothesis explaining the observations, see, e.g. [Angluin 1983].
Bayesian model class comparison is classically based on Bayes Factors, see [Berger 1985], the ratio of marginal likelihoods. However, the choice of model class according to this ratio may strongly depend on the parameter prior being used, as demonstrated in [Silander 2007]. Also observe that the marginal likelihood does not a priori favour simple model classes. In fact, for the empty data $D = \{\emptyset\}$ it is equal to one for any model class, which can be observed by plugging zero counts $c \equiv 0$ into Equation 2.16. Therefore, complexity is often being penalized in a more explicit way.

So how can we measure complexity or simplicity in order to apply Ockham’s razor? With Bayesian network we do have some idea of what 'simple' means. At least we know that adding arcs makes a network more complex, as it will represent a larger set of distributions. But what does 'explain' mean in this context? Each network $B$ assigns a probability $P(D|B)$ to the given data, which is strictly positive, but usually quite small. While in the sense of Ockham’s razor a hypothesis either explains the data or does not, for us ‘explanation’ is measured continuously. The higher the assigned probability, the better the explanation.

Clearly, there is a tradeoff between data fit and generalization capability. To this end, the so-called information criteria have been developed, the most prominent ones being the Akaike Information Criterion (AIC, [Akaike 1974]) and the Bayesian Information Criterion (BIC, [Schwarz 1978]). They make this tradeoff explicit by optimizing—instead of the model class probability $P(C|D)$—a score of the form

$$XIC(C|D) = \text{data fit}(D|C) - \text{penalty}_{XIC}(C),$$

(2.31)

$XIC$ standing for ‘any information criterion’.

The data fit term typically—and theoretically correctly—is the minus-logarithm of the maximum likelihood (ML) $\hat{P}(D|C)$, defined as

$$\hat{P}(D|C) = P(D|C(\hat{\Theta}(D))),$$

(2.32)

where

$$\hat{\Theta}(D) = \underset{\Theta}{\text{argmax}} P(D|C(\Theta))$$

(2.33)

is the set of parameters which, when we instantiate $C$ with them, assigns the maximum probability to $D$. The maximum likelihood parameters $\hat{\Theta}(D)$ need not be unique for $\hat{P}(D|C)$ to be defined. XIC scores are purely model class selection criteria and using the maximum likelihood to measure data fit does not imply we should also use a maximum likelihood model $C(\hat{P}(D))$ for prediction.
The *penalty* term differs for the various information criteria. Its purpose is to measure expected generalization capability. For AIC this penalty equals the number of free parameters $\text{nfp}(C)$ in the model class,

$$\text{penalty}_{AIC}(C) = \text{nfp}(C),$$

(2.34)

while for BIC also the data size $n$ is relevant,

$$\text{penalty}_{BIC}(C, n) = \frac{\log n}{2} \text{nfp}(C).$$

(2.35)

Seemingly, Equation 2.31 couples two entirely different things, the number of free parameters of a model class and the negative logarithm of a probability. But in fact the different penalty terms are approximations of quantities of the same form. Both AIC and BIC (among others) have been derived from asymptotic behaviour where the data size $n$ goes to infinity and have some desirable properties *in the limit*.

While asymptotics are good as a sanity check, in the real-world situation of limited data availability we do not know how well the information criteria will work, that is, how useful a model class chosen in this way will turn out to be.
2.5 Supervised Learning Tasks

Let us now return to the task of parameter learning in Bayesian network models. We have already seen that defining a suitable parameter prior is problematic. Another problem arises, when we are presented with a supervised (discriminative) learning task [Vapnik 1998]. This means that we are not interested in the joint likelihood of a data sample, but only in some aspects of it. In the multivariate multinomial data domain the simplest—and most frequently encountered—supervised learning task is that of classification. For clarity of notation we add a class variable \( X_0 \) of cardinality \( K_0 \) to the set of predictors \( \{X_1, \ldots, X_m\} \) to be defined as in the preceding.

A classifier then is a conditional distribution \( P(X_0|X_1, \ldots, X_m) \).

Traditionally, classifiers are seen as a functions \( f : X_1, \ldots, X_m \rightarrow X_0 \), but here we deviate from this definition, since we consider probabilistic models. We therefore require a classifier to return a distribution instead of a single class. Of course, we can do that as well: return the class that gets the highest probability (see Section 2.7). More generally, a supervised learning task may also be to model a larger subset of the domain variables, or some other aspects of the data space. For simplicity, we only consider classification tasks here. The class variable (or, in general, the objective of the supervised learning task) supervises parameter learning, by telling us which aspects of the data are important to us.

It has been recognized, that for supervised prediction tasks such as classification, we should also use a supervised (discriminative) learning algorithm, such as conditional likelihood maximization [Greiner 2001, Ng 2001, Greiner 1997, Kontkanen 2001, Friedman 1997]. Nevertheless, in most related applications, model parameters are still determined using unsupervised methods, such as joint likelihood maximization and ordinary Bayesian methods.

Remember, that Bayesian networks model joint probability distributions over the data space \( X = \{X_0, \ldots, X_m\} \) by decomposing it into \( m+1 \) local, conditional distributions. Their parameters, regardless of whether we choose to use marginal likelihood or maximum a posteriori, are determined with respect to the joint distribution. Of course we can calculate the conditional distribution of the class given the predictor variables from
the joint
\[
P(x_0|x_1,\ldots,x_m) = \frac{P(x_0,x_1,\ldots,x_m)}{P(x_1,\ldots,x_m)} = \frac{P(x_0,x_1,\ldots,x_m)}{\sum_{x_0'=1}^{K_0} P(x_0',x_1,\ldots,x_m)}, \quad (2.36)
\]
and this is exactly what is typically done, leading to Bayesian network classifiers, see [Friedman 1997]. But this type of use does not correspond to the design of these models, since the joint data likelihood when used for parameter selection does not reflect performance in classification. This is like driving your tractor to work. It can be done, but tractors were designed for something else and there are better vehicles to get you to work.

Then why is it, that joint models, such as Bayesian classifiers, are still being used for supervised tasks? The main reason may be the (most often erroneous) assumption, that the chosen model class is 'correct'. Employing asymptotics of unlimited data availability as a sanity check, we pass the test if in fact data \( D \) is an i.i.d. sample drawn from a distribution in \( \mathcal{B} \). This type of situation is visualized in Figure 2.4.

Figure 2.4: If the generating distribution (solid small circle) lies within the model class (large circle), then the learned joint model (open small circle) will approach it with growing data size to arbitrary precision with probability one. The conditional distributions (projection to the \( x \)-axis) will behave in the same way.

But if the generating distribution (assuming it exists) lies outside of \( \mathcal{B} \)
the situation becomes different. The conditional distribution obtained from
the joint distribution in $\mathcal{B}$ which is closest to the true distribution (in terms
of KL-divergence) need not be the best classifier in the class. Therefore,
even with unlimited data availability, we may never get close to the most
useful model, i.e. the one that minimizes the conditional KL-divergence
from the true model. Intuition for this is given in Figure 2.5.

Figure 2.5: If the generating distribution (solid small circle) lies outside of
the model class (large circle) then, with growing data size, the learned joint
model (open small circle) will approach (under fairly weak assumptions,
see [Cover 1991]) the distribution closest to it within the model class (gray
circle). Conditioning this distribution (projecting to the $x$-axis) may not
mean approaching the best conditional model.

The second reason why Bayesian network classifiers often determine
their parameters by learning the joint distribution instead of the condi-
tional, is the difficulty in finding the global maximum of the conditional
likelihood. Publication I investigates the situations in which the model
parameters can be efficiently learned in a discriminative fashion and the
situations in which this is hard to accomplish. Section 2.6 summarizes
these results as well as closes a gap which had been left by the original
paper. While Publication I provides a sufficient condition under which a
Bayesian network classifier is equivalent to a logistic regression model, here
we are able to prove that this condition is also necessary.
2.6 Discriminative Parameter Learning

Given a network structure $\mathcal{B}$, we now investigate whether we can learn its parameters in a discriminative fashion, that is, with respect to the conditional distribution that is our objective. We do so by mapping $\mathcal{B}$ to a class of logistic regression models.

A model $\mathcal{B}(\Theta)$ with $\Theta = \{\Theta_{x_i|\text{pa}_i}\}$ defines the conditional likelihood of the class $X_0$ as

$$P^B(x_0|x_1, \ldots, x_m) = \frac{P^B(x_0, x_1, \ldots, x_m)}{\sum_{x'_0} P^B(x'_0, x_1, \ldots, x_m)} = \prod_{i=0}^m \frac{\Theta_{x_i|\text{pa}_i}(x)}{\sum_{x'_0} \prod_{i:X_0\in\text{Pa}_i} \Theta_{x_i|\text{pa}_i}(x')}, \quad (2.37)$$

where we set $x' = (x'_0, x_1, \ldots, x_m)$, and $\text{pa}_i(x)$ is the instantiation of $\text{Pa}_i$ given by $x$. Note that the variables appearing in the rightmost expression are only $X_0$ and its Markov blanket. This is due to the fact that all other terms—the ones not involving $x_0$ (resp. $x'_0$)—cancel. Equation 2.37 defines the $\mathcal{B}$-classifier. We denote the set of conditional distributions obtained using $\mathcal{B}$ in this way as $\mathcal{B}_{\text{cond}} = \{\mathcal{B}_{\text{cond}}(\Theta)\}$.

Logistic regression models, e.g., [McLachlan 1992, p.255], are of similar shape. Let $X_0 = \{1, \ldots, K_0\}$ and let $Y_1, \ldots, Y_K$ be real-valued random variables. The multiple logistic regression model with dependent variable $X_0$ and covariates $Y_1, \ldots, Y_K$ is defined as the set of conditional distributions

$$P^{LR}(x_0 | y_1, \ldots, y_K, \beta) := \frac{\exp \sum_{k=1}^K \beta_{x_0|k} y_k}{\sum_{x'_0=1}^{K_0} \exp \sum_{k=1}^K \beta_{x'_0|k} y_k} \quad (2.38)$$

where the model parameters $\beta_{x_0|k}$ are allowed to take on any value in $\mathbb{R}$. This defines a conditional model parametrized in $\mathbb{R}^{K_0 \cdot K}$.

Now, for $i \in \{0\} \cup \{i : X_0 \in \text{Pa}_i\}$ (the class and its children in $\mathcal{B}$), $x_i \in \{1, \ldots, K_i\}$ and $\text{pa}_i$ in the set of parent configurations of $X_i$, let

$$Y_{x_i|\text{pa}_i} := \begin{cases} 1 & \text{if } X_i = x_i \text{ and } \text{pa}_i = \text{pa}_i \\ 0 & \text{otherwise} \end{cases} \quad (2.39)$$

Subsequently, for $\mathcal{B}$-parameters $\Theta = \{\Theta_{x_i|\text{pa}_i}\}$ we set

$$\beta_{x_i|\text{pa}_i} = \log \Theta_{x_i|\text{pa}_i}. \quad (2.40)$$
The indicator variables $Y_{x_i|pa_i}$ can be lexicographically ordered and renamed $1,\ldots,K$. The corresponding parameters $\beta_{x_i|pa_i}$ are of suitable form—as $x_0$ is either $x_i$ itself or a member of $pa_i$—but need to be renamed accordingly. This shows that we have transformed the Bayesian network model $\mathcal{B}_{\text{cond}}(\Theta)$ into a logistic regression model, which we denote $LR_\beta(\beta)$. Moreover, both models encode the same conditional distribution, as can easily be verified.

This proves Theorem 1 of Publication I, namely

$$\mathcal{B}_{\text{cond}} \subseteq LR_\beta.$$ (2.41)

At first sight, it would seem that we would even have equality here. Can we not transform any model $LR_\beta(\beta)$ back into a model $\mathcal{B}_{\text{cond}}(\Theta)$ by setting $\Theta_{x_i|pa_i} = \exp \beta_{x_i|pa_i}$? Unfortunately, this is not the case in general. If of course, the $\beta$’s come from $\Theta$’s, transformed by taking their logarithms as above, this can be done. However, for a general set $\beta \in \mathbb{R}^{K_0 \cdot K}$ (for suitable $K$), the $\Theta$’s will violate the sum-to-one constraints they are bound to by the fact that they form local, conditional probability distributions. That is, in order for $\Theta$ to be a set of parameters defining a model $\mathcal{B}_{\text{cond}}(\Theta)$ we must have

For all $i \in \{0\} \cup \{i : X_0 \in Pa_i\}$ and all $pa_i \in Pa_i : \sum_{x_i=1}^{K_i} \Theta_{x_i|pa_i} = 1$. (2.42)

But what Theorem 1 does give us, is the fact that any class of conditional Bayesian network models $\mathcal{B}_{\text{cond}}$ is contained in a (possibly larger) class of logistic regression models $LR_\beta$. In this model class, learning parameters that maximize the conditional likelihood

$$\prod_{j=1}^{n} P^{LR}(d_{jo}|d_{j1},\ldots,d_{jm},\beta)$$ (2.43)

for given data $D = (d_{ji})_{j=1..n}$ is relatively easy. By Theorem 2 of Publication I, the conditional log-likelihood

$$\sum_{j=1}^{n} \log P^{LR}(d_{jo}|d_{j1},\ldots,d_{jm},\beta)$$ (2.44)

is a concave function of the parameters $\beta$. Together with the fact that the parameter space $\mathbb{R}^{K_0 \cdot K}$ is convex we know that there can be no local
maxima that are not at the same time also global maxima. Choosing a strictly log-concave prior \( p(\beta) \) on \( \beta \), and maximizing

\[
\sum_{j=1}^{n} \log P^{LR}(d_{j0}|d_{j1}, \ldots, d_{jm}, \beta) + \log p(\beta) \tag{2.45}
\]

instead of Equation 2.44, we get a strictly concave objective, which can be optimized locally, e.g. by hillclimbing methods. However, the model \( LR_B(\beta) \) found in this way may not correspond to a model \( B_{\text{cond}}(\Theta) \) for any \( \Theta \) satisfying (2.42).

**Theorem 3** of Publication I identifies the situations in which this does not happen. If \( B \) satisfies the following condition, then for each model \( LR_B(\beta) \) in \( LR_B \) there exists a parameter set \( \Theta \), such that \( B_{\text{cond}}(\Theta) \) encodes the same conditional distribution.

**Condition 1.** For all \( i \) such that \( X_0 \in Pa_i \), there exists \( X_i' \in Pa_i \) such that \( Pa_i \subseteq Pa_i' \cup \{X_i'\} \).

The proof of Theorem 3 can be found in Publication 1. An alternative, equivalent definition of Condition 1 is given in [Roos 2005b]. It states that, when we restrict \( B \) to the Markov blanket of \( X_0 \) and connect all parents of \( X_0 \) to arrive at a network \( B^* \) (which can always be done without introducing cycles, see [Lauritzen 1996]), then any two nodes having a common child in must be connected in \( B^* \).

In simpler terms, Condition 1 demands that any two parents of any child \( X_i \) of the class \( X_0 \) must be connected, unless they are both also parents of \( X_0 \) itself. The converse therefore is

**Converse of Condition 1.** There exists a child \( X_i \) of \( X_0 \) with parents \( X_j \) and \( X_k \), such that

- \( X_j \) and \( X_k \) are not connected in \( B \), and
- \( X_j \) is not a parent of \( X_0 \)

Figure 2.6 depicts examples of four classes of Bayesian networks, for which Condition 1 is satisfied. Naive Bayes (NB) models assume that all predictors are independent once the class \( x_0 \) is known, see [Rish 2001]. In the corresponding network all children of the class have only one parent, the class itself. A diagnostic classifier [Kontkanen 2001] is defined by a network structure in which the class does not have children at all. The network
2.6 Discriminative Parameter Learning

Figure 2.6: Examples of four types of Bayesian network satisfying Condition 1: Naive Bayes (NB, top left), a diagnostic classifier (top right), tree-augmented naive Bayes (TAN, bottom left) and forest-augmented naive Bayes (FAN, bottom right).

specifying a tree-augmented naive Bayes (TAN) classifier [Friedman 1997] is that of naive Bayes, to which arcs have been added that form an out-tree on the predictors. Similarly, forest-augmented naive Bayes (FAN) is NB augmented by a forest on the predictors, i.e. a collection of out-trees. TAN and FAN have in common, that any predictor $X_i$ has the class $X_0$ as a parent and either no other parent, in which case $\text{Pa}_i \subseteq \text{Pa}_0 \cup \{X_0\} = \{X_0\}$, or exactly one other parent $X_{i'}$, in which case $\text{Pa}_i \subseteq \text{Pa}_{i'} \cup \{X_{i'}\}$. Therefore, by Theorem 3, the conditional versions of all these model classes are equivalent to a class of logistic regression models with freely varying parameters.

Theorem 3 has proven, that Condition 1 is sufficient for model class equivalence of the Bayesian network classifier to a suitably defined class of logistic regression models. But is it also necessary? Figure 2.7 depicts three types of networks, which do not satisfy the condition. Publication I proves in its Theorem 4 that for the network marked 'Type I' there in fact exists a data set $D$ for which the conditional log-likelihood (Equation 2.46) does exhibit local, non-global maxima. This not only suggests that it cannot be optimized locally, but moreover that the classifier is not equivalent to any LR model, see below.
Here, we prove a stronger result, which Publication I had left for 'future work'. We claim that in fact Condition 1 is necessary and therefore we have

**Theorem 4 (previously unpublished)** For a Bayesian network \( \mathcal{B} \), there exists a class of logistic models \( LR_\mathcal{B} \) that consists of the same conditional distributions \( \mathcal{B}_{\text{cond}} = LR_\mathcal{B} \) if and only if Condition 1 holds.

**Proof (sketch).** The 'if' part is Theorem 3. It remains to show 'and only if'. We prove that, if \( \mathcal{B} \) does not satisfy Condition 1, i.e. the Converse holds, then there exist data \( \mathbf{D} \) for which the conditional log-likelihood

\[
\sum_{j=1}^{n} \log P^{\mathcal{B}}(d_{jo}|d_{j1}, \ldots, d_{jm}, \Theta)
\]

exhibits multiple peaks. This not only suggests that we cannot find the global maximum by local search methods such as hillclimbing, but also that the classifier is not equivalent to any LR model. For if it were, then the log-transform (Equation 2.40) would preserve the peaks of the conditional log-likelihood in contradiction to Theorem 2.

Let \( \mathcal{B} \) be a Bayesian network for which the 'Converse of Condition 1' holds, \( X_0 \) being the class, \( X_i \) a child of \( X_0 \), \( X_j \) and \( X_k \) parents of \( X_i \) such that \( X_j \notin Pa_0 \) and \( X_j \) and \( X_k \) are not connected in \( \mathcal{B} \).

If \( X_0 = X_j \) or \( X_0 = X_k \), or either of them is not connected to the class, then there is a subgraph in \( \mathcal{B} \) of Type I in Figure 2.7. If in turn this is not the case, then \( X_0 \), \( X_j \) and \( X_k \) are three distinct nodes, and both \( X_j \) and \( X_k \) are connected to \( X_0 \). By assumption, \( X_j \) is not a parent of \( X_0 \), which means it must be a child of the class. If \( X_k \) is likewise, then we have a subgraph of Type II, otherwise we find a subgraph of Type III in Figure 2.7. In fact, it is easy to see that Types II and III are equivalent. It suffices to restrict attention to these subgraphs, since any data for the variables of a
subgraph displaying multiple log-likelihood peaks can be extended to data for all of \( B \)'s variables with the same property, e.g. by setting all values of any vector for variables not appearing in the subgraph to the same value. Also, we restrict to considering binary variables, without loss of generality. Note, that for larger variable cardinalities the additional values need not appear in \( D \), as the counter example data is of our own construction.

For Type I our case has been proven in Publication I. We defined a data set

\[
D_I = \begin{bmatrix}
1 & 1 & 1 \\
1 & 1 & 2 \\
2 & 2 & 1 \\
2 & 2 & 2 \\
\end{bmatrix}
\] (2.47)

consisting of four data vectors, the columns corresponding to binary variables \( X_0 \), \( X_1 \) and \( X_2 \) in this order. We then could show, that there are four local, non-connected suprema of the conditional log-likelihood of \( D_I \). With a little trick, we could also make these suprema maxima, each having a different value. For the technical details please refer to Publication I, let us here look at what happens on a more intuitive level.

Observe that the only dependency this data displays is the equality of values in \( X_0 \) and \( X_1 \) for all four vectors, while all combinations of values in \( X_1 \) and \( X_2 \) appear. But there is no edge in \( B_{\text{Type I}} \) to directly model this dependency. On the other hand, \( X_2 \) is arbitrary. For each vector with \( X_2 = 1 \) we also have its counterpart, differing from it only in that it has \( X_2 = 2 \). The Bayesian classifier can now exploit its ability to explain away (cf. Sec. 2.3) and use the conditional probability table at \( X_2 \) to introduce the observed dependency between \( X_0 \) and \( X_1 \) into the conditional model. While doing so, the joint likelihood decreases, but the conditional goes up. We 'sacrifice' probability at \( X_2 \), which plays no part in the objective of the classification task.

For Types II and III we take a similar approach. We define

\[
D_{II/III} = \begin{bmatrix}
1 & 1 & 1 & 1 \\
2 & 1 & 1 & 2 \\
1 & 1 & 2 & 1 \\
2 & 1 & 2 & 2 \\
2 & 2 & 1 & 1 \\
1 & 2 & 1 & 2 \\
2 & 2 & 2 & 1 \\
1 & 2 & 2 & 2 \\
\end{bmatrix};
\] (2.48)

columns corresponding to variables \( X_0 \), \( X_1 \), \( X_2 \) and \( X_3 \). In \( D_{II/III} \) we find
all combinations of values for $X_1$, $X_2$ and $X_3$ but the class takes on value

$$X_0 = \begin{cases} 
1 & \text{if } X_1 = X_3 \\
2 & \text{if } X_1 \neq X_3 
\end{cases}.$$  

(2.49)

Therefore the class depends on none of the predictors directly, but we can explain away the class based on the value of ‘$X_1 = X_3$?’ through $X_2$. In effect, we have reduced Type II/III to Type I, where now ‘$X_1 = X_3$?’ plays the role of $X_1$ in Type I. The rest of the proof is analogue to the proof of (the weaker) Theorem 4 in Publication I. We omit the technicalities. ■

We have shown, that learning the parameters of a Bayesian network classifier via the conditional likelihood, rather than the joint, can be done efficiently whenever the corresponding network structure satisfies Condition 1. If it does not, then there can be multiple local, non-global optima, which potentially makes finding the global optimum a hard task. It remains to show that indeed, the resulting classifier is to be preferred over the vanilla solution. We investigate this empirically in the following section.
2.7 Empirical Evaluation

We study the usefulness of discriminative parameter learning using the naive Bayes classifier as an example. The corresponding network structure (an example is given in Figure 2.6, top left) satisfies Condition 1, and therefore the classifier is equivalent to logistic regression with suitably defined covariates. The following results have been published in a technical report accompanying Publication I, [Wettig 2002a].

Note, that the conditional log-likelihood as a function of the standard Bayesian network parameters $\Theta$ cannot have local, non-global maxima, since the log-transform (Equation 2.40) would preserve them and we know that there are no such maxima in the conditional log-likelihood viewed as a function of the LR parameters $\beta$. However, in the original parametrization the conditional log-likelihood is not concave and, if we restrict the parameter space to a convex subset, we can no longer be sure that there will be no unconnected local maxima. The log-transform mends this situation by concavifying the objective, see [Wettig 2002a]. Also, the sum-to-one constraints seem inconvenient in optimization, and therefore we choose to maximize the conditional log-likelihood in the LR parameter space. Empirical evidence, giving additional reason to prefer the LR parametrization over standard Bayesian, has been reported in [Greiner 2002].

We maximize the objective by iteratively optimizing one parameter $\beta_{x_0|k}$ at a time with growing precision, and terminate the search when the norm of the gradient at the current solution has dropped below a predefined threshold. This was sufficiently fast for our purposes, for a comparison of more sophisticated methods see, e.g., [Minka 2001].

The vanilla NB classifier uses the uniform prior wrt. its standard parametrization. To be fair, we tried to define the prior $p(\beta)$ on the LR parameters to be as close as possible to this. To that end, we transform the $\beta$’s back into the standard parameter space—by taking their exponentials and normalizing suitably—and then take their product

$$p(\beta) := \prod_{k=1}^{K} \prod_{x_0=1}^{K_0} \frac{\exp \beta_{x_0|k}}{\sum_{x_0'} \exp \beta_{x_0'|k}}.$$  \hspace{1cm} (2.50)

This is what we obtain when we interpret the uniform NB prior as pseudo-counts and look at the predictors one-at-a-time. Of course, this is not actually the same thing as a Bayesian uniform prior on the joint distributions, but since the LR model does not define any joint distribution, this seemed to be the best we can do.

We compared the NB and LR models using 32 real-world data sets from
<table>
<thead>
<tr>
<th>Data</th>
<th>n</th>
<th>m</th>
<th>K</th>
<th>0/1-loss</th>
<th>log-loss</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>NB</td>
<td>LR</td>
<td>NB</td>
<td>LR</td>
<td></td>
</tr>
<tr>
<td>Mushrooms</td>
<td>8124</td>
<td>21</td>
<td>234</td>
<td>95.57</td>
<td>100.00</td>
</tr>
<tr>
<td>Page Blocks</td>
<td>5473</td>
<td>10</td>
<td>420</td>
<td>94.74</td>
<td>96.29</td>
</tr>
<tr>
<td>Abalone</td>
<td>4177</td>
<td>8</td>
<td>1008</td>
<td>23.49</td>
<td>25.95</td>
</tr>
<tr>
<td>Segmentation</td>
<td>2310</td>
<td>19</td>
<td>917</td>
<td>94.20</td>
<td>97.01</td>
</tr>
<tr>
<td>Yeast</td>
<td>1484</td>
<td>8</td>
<td>1060</td>
<td>55.59</td>
<td>57.75</td>
</tr>
<tr>
<td>German Credit</td>
<td>1000</td>
<td>20</td>
<td>244</td>
<td>75.20</td>
<td>74.30</td>
</tr>
<tr>
<td>TicTacToe</td>
<td>958</td>
<td>9</td>
<td>56</td>
<td>69.42</td>
<td>98.33</td>
</tr>
<tr>
<td>Vehicle Silhouettes</td>
<td>846</td>
<td>18</td>
<td>1116</td>
<td>63.95</td>
<td>72.22</td>
</tr>
<tr>
<td>Annealing</td>
<td>798</td>
<td>31</td>
<td>740</td>
<td>93.11</td>
<td>99.00</td>
</tr>
<tr>
<td>Diabetes</td>
<td>768</td>
<td>8</td>
<td>178</td>
<td>76.30</td>
<td>75.78</td>
</tr>
<tr>
<td>Breast C. (Wisc.)</td>
<td>699</td>
<td>10</td>
<td>222</td>
<td>97.42</td>
<td>96.42</td>
</tr>
<tr>
<td>Australian Credit</td>
<td>690</td>
<td>14</td>
<td>232</td>
<td>86.52</td>
<td>85.94</td>
</tr>
<tr>
<td>Balance Scale</td>
<td>625</td>
<td>4</td>
<td>63</td>
<td>92.16</td>
<td>93.60</td>
</tr>
<tr>
<td>Congr. Voting</td>
<td>435</td>
<td>16</td>
<td>104</td>
<td>90.11</td>
<td>96.32</td>
</tr>
<tr>
<td>Mole Fever</td>
<td>425</td>
<td>32</td>
<td>408</td>
<td>90.35</td>
<td>88.71</td>
</tr>
<tr>
<td>Dermatology</td>
<td>366</td>
<td>34</td>
<td>804</td>
<td>97.81</td>
<td>97.81</td>
</tr>
<tr>
<td>Ionosphere</td>
<td>351</td>
<td>33</td>
<td>402</td>
<td>92.31</td>
<td>92.59</td>
</tr>
<tr>
<td>Liver</td>
<td>345</td>
<td>6</td>
<td>236</td>
<td>64.06</td>
<td>68.70</td>
</tr>
<tr>
<td>Primary Tumor</td>
<td>339</td>
<td>17</td>
<td>903</td>
<td>48.97</td>
<td>49.26</td>
</tr>
<tr>
<td>Ecoli</td>
<td>336</td>
<td>7</td>
<td>760</td>
<td>80.36</td>
<td>81.85</td>
</tr>
<tr>
<td>Soybean</td>
<td>307</td>
<td>35</td>
<td>2527</td>
<td>85.02</td>
<td>90.23</td>
</tr>
<tr>
<td>Heart D. (Clevel.)</td>
<td>303</td>
<td>13</td>
<td>510</td>
<td>58.09</td>
<td>55.78</td>
</tr>
<tr>
<td>H.D. (Hungarian)</td>
<td>294</td>
<td>13</td>
<td>210</td>
<td>83.33</td>
<td>82.99</td>
</tr>
<tr>
<td>Heart D. (Statlog)</td>
<td>270</td>
<td>13</td>
<td>162</td>
<td>85.19</td>
<td>83.33</td>
</tr>
<tr>
<td>Thyroid Disease</td>
<td>215</td>
<td>5</td>
<td>198</td>
<td>98.60</td>
<td>94.88</td>
</tr>
<tr>
<td>Glass Identification</td>
<td>214</td>
<td>10</td>
<td>804</td>
<td>70.09</td>
<td>69.63</td>
</tr>
<tr>
<td>Wine Recognition</td>
<td>178</td>
<td>13</td>
<td>573</td>
<td>97.19</td>
<td>96.63</td>
</tr>
<tr>
<td>Hepatitis</td>
<td>155</td>
<td>19</td>
<td>274</td>
<td>79.35</td>
<td>82.58</td>
</tr>
<tr>
<td>Iris Plant</td>
<td>150</td>
<td>4</td>
<td>255</td>
<td>94.00</td>
<td>94.67</td>
</tr>
<tr>
<td>Lymphography</td>
<td>148</td>
<td>18</td>
<td>240</td>
<td>85.81</td>
<td>86.49</td>
</tr>
<tr>
<td>Postoperative</td>
<td>90</td>
<td>8</td>
<td>75</td>
<td>67.78</td>
<td>66.67</td>
</tr>
</tbody>
</table>

Table 2.2: Leave-one-out cross-validation results

the UC Irvine Machine Learning Repository (archive/ics.uci.edu/ml/). Continuous data were discretized, the exact method and discretized data sets can be found at www.cs.Helsinki.FI/u/pkontkan/Data/. We performed leave-one-out cross-validation, n times holding out one data sample at a time and training on the remaining n – 1. We report, for both model classes, the 0/1-loss as percentage of correct classifications (when we return
the most probable class) and the log-loss, i.e. the average natural logarithm of the probability assigned to the correct class. Apart from the prior, the latter is exactly what we are optimizing on the training data with the LR model. The former is correlated, and commonly reported for classification tasks, but—also due to its discrete nature—we do expect it to behave less decisively.

The results we have obtained in this way are listed in Table 2.2, where \( n \) denotes data size, \( m \) the number of predictors and \( K \) the number of parameters—which is equal for both model classes—as we had before. Winning scores are boldfaced.

We observe, that in 26 out of 32 cases the discriminative method has produced lower log-loss. On all larger data sets, it consistently outperformed the NB classifier, in several cases by a large margin. On six of the smaller data LR actually lost to NB, but by much smaller margin. As expected, the situation is less clear-cut for the 0/1-loss, but LR still wins 18:13 (with one draw), again larger data working in favour of LR. Similar observations have been reported in [Ng 2001].

We presume that this behaviour means that LR, as it better fits the training data, is also more prone to overfitting. But this is not to say that on small data sets optimizing the joint likelihood is to be preferred. In fact, unsupervised—joint—learning is no automatism to prevent overfitting. Instead, we propose a different approach. In logistic regression, it is easy to prune parameters from the model, effectively just fixing them to zero. In Bayesian network models, we can remove arcs from the network structure to obtain a simpler, better generalizing model class, e.g. predictor selection in naive Bayes classification. This amounts to the removal of a whole set of parameters. But logistic regression is more flexible than that. We can remove any set of parameters—not just sets that correspond to arc removal—even just a single parameter. One possible criterion for parameter selection is the size of its support, as suggested in [Wettig 2002b]. Here we remove a parameter \( \beta_{x_i|pa_i} \) (set it to zero), whenever its support \(|\{1 \leq j \leq n : d_{ji} = x_i \text{ and } pa_i(d_j) = pa_i\}| \leq T \) in \( D \) does not exceed a given threshold \( T \).

An even more elegant solution is the use of the so-called \( \alpha \)-prior (Laplace prior, \( L_1 \)-prior) defined by

\[
\log p_\alpha(\beta) := \alpha \sum_{k=1}^{K} \sum_{x_0=1}^{K_0} |\beta_{x_0|k}|.
\]

(2.51)

An example of its behaviour as compared to that of the 'transformed uniform' prior we have used in our experiments is plotted in Figure 2.8.
Figure 2.8: The logarithms of the prior probabilities $p_\alpha(\beta_1|k, \beta_2|k)$ with $\alpha = 1$, and the 'transformed uniform' (2.50) of a pair of parameters predicting a binary class evaluated at $\beta_1|k = -\beta_2|k$.

We see that for large parameter values the two priors essentially agree, but the $\alpha$-prior much more clearly prefers values near zero. In fact, test runs we have performed on the same data used here (plus some others) have shown that a model trained using the $\alpha$-prior usually becomes quite sparse. In typical cases with hundreds of parameters and data size ranging from the hundreds to the thousands, most parameter values will be zero exactly, and thus have been pruned away. Exactly how sparse a model will be obviously depends on the value chosen for $\alpha$. This way of automatic parameter selection clearly improves prediction, especially with small data size. We have not published these results, as others have beat us to it, e.g. [Cawley 2007].
2.8 Summary

This chapter has given an introduction to Bayesian reasoning and, more specifically, to Bayesian network models. We have seen that Bayesian network models can be a very neat way of expressing our subjective beliefs, as long as we have a good understanding of the problem domain. That is, we need to be able to explicitly encode reasonable independence assumptions as a network structure and instantiate its parameters locally as conditional probability tables. Then we can calculate the resulting joint (global) probability distribution and—by conditioning—any conditional distributions we might be interested in, which need no longer be local within the network.

Learning from data, however, is much harder. First of all, the conditional distributions that have been calculated from the joint need not be optimal with respect to the training data. Learning the joint distribution is always a compromise, as we optimize all conditional distributions simultaneously. Any specific such conditional may thus be far from optimal. It is hardly the fault of Bayesian network models, that they are frequently being used for discriminative learning tasks such as classification. We have shown, that in many cases there is no need to do so, either. This is where the author’s contributions of this chapter lie. In Section 2.6 we have explicitly identified the situations in which we can—and those in which we cannot—(easily) learn the parameters of a Bayesian network classifier optimally in a discriminative way instead of suboptimally from the joint. We did so by transforming the classifier into a class of logistic regression models. This can always be done, and this transformation is always one-to-one, but only under 'Condition 1' on the network structure is it also onto. The condition holds, intuitively speaking, whenever the Bayesian classifier does not explain away the class. As we would have guessed, logistic regression models cannot explain away. But any other type dependencies a Bayesian classifier is able capture can also be represented by a logistic regression model, which is easier to handle and more flexible.

Another problem associated with parameter learning in Bayesian networks is the need for a prior distribution. We need this parameter prior, in order to obtain a posterior distribution; the training data transform the prior into a posterior. And while it can be seen as an asset that Bayesian network models force us to make our prior beliefs explicit, in practice it is typically unclear how to transform these beliefs into a prior distribution over the parameters. In fact, the only computationally convenient prior is Dirichlet, which is far from being intuitive. We are—more or less—safe, as long as we have a fixed, simple network structure and loads of data. But when we want to learn the network structure from data the situation
becomes awkward, as in structure learning the parameter prior plays a decisive role. The main reason for this being that, with respect to the model complexity, we never seem to have a sufficient amount of data available, as in model class selection complexity is a function of the data size. Therefore, the amount of data serving as support for any given parameter will remain strictly limited.

In many important areas, such as admission of pharmaceutic products and courts of law, classic hypothesis testing using $t$-tests with $p$-values is still the norm. Over the last decades, however, it has become increasingly accepted that Bayesian model class selection should be preferred, as it provides more objective, less biased answers [Kruschke 2012]. But also this approach faces its problems, as listed and discussed in Section 2.4. One of them being the ’X-Files assumption’ (“the truth is out there”), the conjecture that the data generating distribution exists, and lies inside the considered model family. And while we do not claim that there exist no Bayesian solutions to these problems, it often is beneficial to take an utterly different approach.

In the following chapter, we introduce the **Minimum Description Length (MDL)** Principle, which offers an elegant framework for the task of model class selection. Many of the problems that the Bayesian approach has to deal with simply will not arise, to others there will be simple and straightforward solutions.
Chapter 3

Information Theory

“The truth is in there.”

3.1 The Minimum Description Length Principle

Information theory quantifies information, measuring it in bits. Classic information theory, developed by the American mathematician Claude E. Shannon (1916–2001) [Shannon 1948], measures the information content of a data generating source as the expected number of bits needed to communicate a sample from it over a noiseless channel. This entity is called the (Shannon) entropy (denoted $H$ after Boltzmann’s $H$-theorem) and is, for a multinomial random variable $X$ with cardinality $K$, defined by

$$H(X) = - \sum_{k=1}^{K} P(X = k) \log P(X = k).$$

Figure 3.1 shows a plot of the binary entropy $H_2(X)$ as a function of $P(X = 1) = 1 - P(X = 2)$. Entropy can also be viewed as a measure of uncertainty. It is maximized for the uniform distribution with value $\log K$ and minimized for the completely specified case (probabilities equal zero or one), when there is no uncertainty about the outcome and the entropy is zero. For example, tossing a fair coin contains one bit of information per throw, and a coin toss with known outcome delivers no additional information. The information content of the throw of a biased coin lies inbetween these extrema and is measured by the entropy.
Note, that the entropy is a function of a distribution $Q$, so sometimes it is more intuitive to write $H(Q)$ in place of $H(X)$. The following theorem links the entropy to codelength.

**Theorem 5 (Shannon’s source coding theorem, [Shannon 1948])**

Let $Q$ be a distribution over a finite set $\Sigma$, and $L$ a uniquely decodable code (e.g. a prefix code) that for $\sigma \in \Sigma$ assigns a code word of length $|L(\sigma)|$. If $L$ is optimal in that it minimizes the expected codelength $\mathbb{E}_Q[|L(\sigma)|]$ then we have

$$H(Q) \leq \mathbb{E}_Q[|L(\sigma)|] \leq H(Q) + 1.$$  \hspace{1cm} (3.2)

In practice, we will disregard the '+1'. When encoding $n$ entities $\sigma_1, \ldots, \sigma_n$, we can achieve

$$nH(Q) \leq \mathbb{E}_Q \left[ \sum_{j=1}^{n} |L(\sigma_j)| \right] \leq nH(Q) + 1,$$

and therefore rounding up to an integer number of bits has no significance in our applications. Or, as Peter Grünwald puts it, 'Non-integer Codelengths
Are Harmless’ [Grünwald 2007, p.95]. We crudely allow real-valued code-lengths.

In the year following the publication of Shannon’s theorem, Leon G. Kraft—referring to Shannon’s work—proved a stronger result, which has become known as Kraft’s inequality.

**Theorem 6 (Kraft’s inequality, [Kraft 1949])**

Let each symbol from an alphabet $\Sigma = \{\sigma_1, \ldots, \sigma_K\}$ be encoded by a uniquely decodable code with corresponding codeword lengths $l_1, \ldots, l_K$ then

$$\sum_{i=1}^{K} 2^{-l_i} \leq 1. \quad (3.4)$$

Conversely, for a set of integers (read: numbers) $l_1, \ldots, l_K$ satisfying this inequality, there exists a uniquely decodable code for $\Sigma$ with these code-lengths.

**Remark 2** This result also applies to countably infinite alphabets.

**Remark 3** Kraft originally proved the theorem only for prefix codes. The generalization to any uniquely decodable code is due to Brockway McMillan [McMillan 1956]. For this reason, the above is sometimes also referred to as the Kraft-McMillan inequality.

**Remark 4** We use a capital $L$ to denote the actual encoding $L(\sigma)$ of an entity $\sigma$. Its length—a real-valued number of bits—will be denoted by a lower case $l$: $l(\sigma) = |L(\sigma)|$.

We can now, for any probability distribution $P = (p_1, \ldots, p_K)$ over the alphabet $\Sigma$ define a uniquely decodable code with corresponding codeword lengths $l_i = -\log p_i$, satisfying Kraft’s inequality. In fact, there is no need to explicitly define it, it suffices to know that such code exists. Of course, the expected codelength per symbol will be $H(P)$, the entropy of the distribution.

But Kraft’s inequality gives us more than that. We can also turn any uniquely decodable code into a probability distribution, by setting $p_i = C^{-1} \cdot 2^{l_i}$, where $C \leq 1$ is a normalizing constant needed in case (3.4) is strict, i.e.

$$\sum_{i=1}^{K} 2^{-l_i} = C < 1. \quad (3.5)$$

\(^1A\) rumour goes that, for the second edition of the book, this will be changed into ‘Mostly Harmless’.
If equality does hold, then $C = 1$ plays no role.

Therefore, codelengths and probabilities are essentially the same. Coding and modeling are equivalent. The shorter the codelength, the higher the probability the code assigns to a data. With this observation, learning from data becomes finding regularities in it. These regularities help us to compress the data, and also to predict future data.

This is the **Minimum Description Length** (MDL) Principle, introduced by Jorma Rissanen [Rissanen 1978]. When the (two-to-the-minus) size of a compressed file is being viewed as a probability, then minimum description length means maximum probability.

But 'probability' takes on a slightly different meaning now. The probability distribution defined by any model does convert into an encoding scheme. And if the model is 'correct'—i.e., it equals the data generating distribution—then, by Shannon's theorem, the corresponding code is optimal in that it minimizes the expected codelength of data coming from this source. This expectation is the entropy of the source. But, when we learn a model from data, i.e. choose a model from a model class which best fits the data, then the resulting code is *not* uniquely decodable without knowledge of that choice.

A good way to describe the situation is the way Shannon looked at it. Picture a sender, a (noiseless) channel, and a receiver. The sender encodes data $D$, the encoded message $L(D)$ is transmitted over a channel (which is expensive and there is no flat-rate; we want to keep the message short), and the receiver needs to be able to recover the original data from this message. Figure 3.2 visualizes this concept.

![Figure 3.2: Schematic representation of the channel coding game.](image)

This scheme forces us to be very explicit about three things:

1. **Is the message decodable?** This is a very good sanity check, which keeps us from cheating. If we cannot recover $D$ from $L(D)$, then we have forgotten to transmit some vital information. As a result, Kraft's inequality may not hold and therefore the code may not be transformable into a probability (sub-) distribution.

   In standard modeling, there is no such sanity check. Of course, every
model is a probability distribution and thereby forced to sum to one. But
the actual distribution used is often implicit, e.g., we can hardly recover
a model from a file of size equal to an information criterion penalty term
such as (2.34) or (2.35).

2. Is this the shortest we can do? Of course, we really have no way
of knowing, see Section 3.3. But we can always scan the message \( L(D) \)
for remaining regularities and, if we find any, include them in our code
to make the message shorter. Observe that also any standard compressor
such as \texttt{gzip}, \texttt{bzip2} and the like defines a code, as the compressed files
are uniquely decodable. Therefore, we can always compare our code length
\( |L(D)| \) against, say, the length of the file ‘\texttt{D.gz}’. We can also use standard
compressors on the message \( L(D) \). If the message further compresses then
we must have overlooked some regularity.
There is no such ‘remainder’ to scan for additional regularity in prob-
abilistic modeling, nor are other methods, such as standard compressors,
directly comparable.

3. What have we agreed upon? In order for the receiver to be able
to interpret the message, she and the sender must have agreed upon an
encoding scheme. They have to meet at least once, e.g. on an open channel
in the internet, and talk about what it is exactly that will be sent, in which
order, and how it is going to be encoded. In other words, we need to
be very clear about what is given and what is assumed. For example,
sender and receiver may agree upon an i.i.d. assumption and that the data
may arrive in any order. Of course, this is not enough. There has to be
common knowledge about what the data structure is, which model family
is being used, any side information and so on. Everything that is needed to
interpret the message must have been agreed upon in advance. When we
are being this explicit, it is easy to check whether this agreement matches
the problem, whether we are actually sending the information we want to
find regularities in.

Extreme cases of this are the null agreement (e.g., ‘I will send you a
selfextracting file. Run it and it will produce everything you need to know’) and
full specification (e.g., ‘I will send you data of size \( n \) for the burglar
alarm problem, using the network from Figure 2.1 with probabilities from
Table 2.1’). Of course, the latter does not make much sense, we have not
actually learned anything or found any regularities in the data. Learning
is always a problem of selection, we need to find a model class and/or
a model, that describes the data well or, equivalently, a coding scheme
that compresses the data well. Usually, the nature of the sender-receiver agreement will lie somewhere inbetween the two extremes. In any case, we are fine as long as we remember to never agree on anything, which may depend on data that is yet to be sent.

The 'fully specified' example illustrates, that any model whatsoever is also a code. But the main area of application for the MDL principle lies in model class selection. Observe that, at this point, we have already solved the first three problems related to Bayesian model class selection discussed in Section 2.4.

**Solution to Problem 1 (no generating distribution assumption)**

*We do not assume that data $D$ have been generated by any distribution. We are simply compressing it, wherever it might have come from. If we do a good job, the resulting code will be useful, but there is no such thing as a correct code.*

**Solution to Problem 2 (overlapping model classes)**

*Since we only want to find a useful code, we do not need to worry about this. Of course, there may be many ways to encode the same data $D$ and therefore, using the shortest encoding we can find, usually renders an incomplete code, i.e. Kraft’s inequality will be strict. However, the normalizing constant

\[
C = \sum_{D' \sim D} 2^{-l(D')} < 1, \quad (3.6)
\]

is a value associated only with the code $L$. So if $L$ allows the use of alternative model classes, minimizing $l(D)$ will still pick the best, most compressing one. By $D' \sim D$ we mean that $D'$ and $D$ are of the same (previously agreed upon) format, e.g. matrices of the same dimensions with values from the same alphabets.*

**Solution to Problem 3 (defined data codelength)**

* $P(D) = C^{-1}2^{-l(D)}$ is always defined, even though $C$ may not be known.*

The simplest way to implement the MDL Principle are the so-called **two-part codes**. The following section explains this concept and gives illustrating examples.
3.2 Two-Part Codes

The basic idea of two-part coding, given a model class \( C \), is to first encode the model parameters \( \Theta \) and subsequently the actual data \( D \) using distribution \( C(\Theta) \):

\[
l^2_p(D) = l(\Theta) + l(D; C(\Theta)).
\] (3.7)

We may then choose the best model class as the one that minimizes this combined codelength.

Taking the negative of Equation 3.7 (now to be maximized) makes it look a lot like the information criteria (2.31),

\[
-l^2_p(D) = \log P(D|C(\hat{\Theta})) - l(\Theta).
\] (3.8)

Therefore we can regard \( l(\Theta) \) as a complexity penalty term. If we choose to encode the parameters in a given (previously agreed upon) range to given precision, then we arrive at a penalty term that only depends on the number of (free) parameters, as we have for AIC. If range and/or precision depend on the data size, then so does the penalty term, as it does for BIC. But where the XIC were just criteria that have fallen from the skies of asymptotics, we now have the decodability requirement to meet.

At this point, it becomes clear why it makes no sense to simply select the model that assigns the highest probability \( P(D|C(\hat{\Theta})) \) to the data. This would be disregarding the term \( l(\Theta) \), the length of the encoding of the used model, which is a measure of its complexity. We cannot recover \( D \) from a message of length \(-\log P(D|C(\hat{\Theta}))\), as—summed over all possible data—these code lengths violate Kraft’s inequality.

**Remark 5** We use the term 'two-part code' whenever we encode the model parameters separately, following the notation of [Grünwald 2007]. Otherwise we speak of 'one-part codes'. This terminology may be somewhat confusing, as both types of encoding may involve multiple parts, a one-part code may consist of two or more parts and a two-part code of, say, five. In the literature (including attached Publications III–VI), there is no unanimous way of using 'two-part coding' to denote 'separate parameter encoding'. For obvious reasons, this is sometimes referred to as 'naive MDL', e.g. [Djurić 1998].

Let us now look at a few examples.

**Two-part encoding using a Bayesian network \( B \)**

Let, as in Section 2.4, \( X = \{X_1, \ldots, X_m\} \) be the data space consisting of \( m \) multinomial variables \( X_i \) of corresponding cardinalities \( K_i \) and
\( D = (d_{ji})_{j=1..n} \) consist of \( n \) samples \( d_j \), with \( m \) entries \( d_{ji} \in \{1, \ldots, K_i\} \) each. Given a Bayesian network \( B \) on the variables \( X_i \in X \), we order the \( X_i \) such that for all \( i \) we have \( Pa_i \subseteq \{X_1, \ldots, X_{i-1}\} \). We want to first encode the model parameters \( \Theta = (\Theta_{x_i|pa_i}) \) and subsequently \( D \) using \( B(\Theta) \).

Since the parameters are continuous, we can only encode them to finite precision. However, the MAP parameters are relative frequencies, so for our purposes it is enough to encode these frequencies, the counts of \( D \) with respect to \( B \). Slightly deviating from standard two-part coding, we encode \( D \) columnwise, for each variable \( X_i \) first encoding the corresponding parameters, and then its \( n \) values \( d_{1i}, \ldots, d_{ni} \).

Assuming that the data size \( n \) is known to the receiver, the counts at \( X_1 \) can be communicated using

\[
\log \left( \frac{n + K_1 - 1}{K_1 - 1} \right) \tag{3.9}
\]

bits, since \( \binom{n+K_1-1}{K_1-1} \) is the number of weak compositions of \( n \) into \( K_1 \) parts, see [Andrews 1976], or the number of different count vectors at \( X_1 \) for data of size \( n \). The first column of \( D \) can then be transmitted in

\[
- \sum_{k=1}^{K_1} c^1_k \log \Theta^1_k = - \sum_{k=1}^{K_1} c^1_k \log \frac{c^1_k}{n} = - \sum_{k=1}^{K_1} c^1_k \log c^1_k + n \log n \tag{3.10}
\]

bits, where \( c^1 = (c^1_1, \ldots, c^1_{K_1}) \) is the vector of counts at \( X_1 \) encoded above, which transforms into MAP parameters \( \Theta^1_k = \frac{c^1_k}{n} \). For zero counts, we take on the convenient convention that \( 0 \log 0 = 0 \).

For the remaining variables \( X_2, \ldots, X_M \), which may have parents in \( B \), we need to encode \( K_i = \prod_{i':X_i \in Pa_{i'}} K_{i'} \) sets of parameters (counts) \( (\Theta^i_{k|pa_i})_{k=1..K_i} \), which takes

\[
\log \left( \frac{c(pa_i) + K_i - 1}{K_i - 1} \right) \tag{3.11}
\]

bits for each instantiation \( pa_i \in Pa_i \). Note, that the parent counts \( c(pa_i) \) are known to the receiver at this point. Coding the \( i^{th} \) column of \( D \) using these MAP parameters then takes

\[
- \sum_{pa_i \in Pa_i} \sum_{k=1}^{K_i} c^i_{k|pa_i} \log \frac{c^i_{k|pa_i}}{c(pa_i)} = \sum_{pa_i \in Pa_i} \left( - \sum_{k=1}^{K_i} c^i_{k|pa_i} \log c^i_{k|pa_i} + c(pa_i) \log c(pa_i) \right) \tag{3.12}
\]
3.2 Two-Part Codes

bits. The overall codelength becomes

\[ l^2p(D; n, \pi, B) = \sum_{i=1}^{m} \sum_{p(a_i) \in Pa_i} \log \left( \frac{c(p(a_i)) + K_i - 1}{K_i - 1} \right) \]

\[ + \sum_{i=1}^{m} \sum_{p(a_i) \in Pa_i} \left( -\sum_{k=1}^{K_i} c^i_{k|p(a_i)} \log c^i_{k|p(a_i)} + c(p(a_i)) \log c(p(a_i)) \right), \] (3.13)

where for any \( X_i \) with \( Pa_i = \emptyset \) we assume there is a single instantiation \( p(a_i) \) with \( c(p(a_i)) = n \). Sender and receiver have previously agreed upon data size \( n \), the ordering of the variables \( \pi \) and the network structure \( B \). For this reason we have included them in the codelength term \( l^2p(D; n, \pi, B) \), separated from the data to be encoded by a semicolon. Read: ‘Two-part codelength of \( D \) using \( n, \pi \) and \( B \).

Note that the ‘penalty term’ in the upper line of Equation 3.13 depends on the actual data, not only on its size. This is not dangerous. In MDL, any clever way of encoding is permitted. Which is not to say that the above is very clever, it merely serves as an example of what one could do.

The parameter prior we have implicitly used is uniform in that any count vector \( (c^i_{k|p(a_i)})_{k=1..K_i, p(a_i) \in Pa_i} \) for each variable \( X_i \) with parent instantiation \( p(a_i) \) is encoded with constant length. On the other hand, it gives point mass only to parameters corresponding to data counts, while assigning zero probability to any other parameters.

Two-part encoding using any Bayesian network

Let us now assume, that sender and receiver have only agreed on the fact that they will use a Bayesian network to encode data from domain \( X \), but not on its actual structure. The sender wants to be free to, after having seen the data \( D \), choose some network \( B \) that yields short codelength.

We first encode the data size \( n \) in a selfdelimiting way, [Li 1997, p.79], which can be done in \( 2 \log n + 1 \) bits by first sending \( \log n \) ones, then a single zero followed by \( \log n \) bits to encode \( n \) itself. In general, there is a shorter selfdelimiting description of an integer \( n \). With the iterated logarithm, we can achieve \( l(n) = \log(n) + O(\log \log n) \), see [Li 1997, p.80]. Next, we encode the network structure. As there are superexponentially many such structures, but for our limited data size we expect to be using a relatively

\[ k \left( \frac{\log \log n}{\log(2^k - 1)} + 1 \right) + \log n. \]
sparse network, it seems beneficial not to use a uniform distribution. Instead, we may simply list all arcs, first giving the number $0 \leq |Pa_i| \leq m-1$ of parents using $\log m$ bits per node and then specifying which parents are present using $\log (m-1)$ bits for node $X_i$. Of course, in this way we can also encode networks which are not DAGs, and therefore we could define a tighter code if we bothered. A suitable ordering $\pi$ of the variables can be retrieved by the receiver from the structure. We get an overall codelength of

$$l^{2p}(D) = 2 \log n + 1 + m \log m + \sum_i \log \left( \frac{m-1}{|Pa_i|} \right) + l^{2p}(D; n, \pi, \mathcal{B}).$$

(3.14)

One-part encoding using a Bayesian network
Of course, we could have also employed the marginal likelihood, instead of encoding the parameters explicitly. Here, sender and receiver agree on a set of pseudo-counts $\alpha$, for example an ESS prior as in Equation 2.30, and then play the prequential game of Remark 1 in Section 2.2. That is, the data samples are transmitted one at a time, and after each transmission both sender and receiver update their counts (including the pseudo-counts $\alpha$) and always use the relative counts observed so far as the probability distribution to encode/decode the next sample. The resulting codelength is the negative of the marginal log-likelihood

$$l^1p(D; n, \pi, \mathcal{B}, \alpha) = \sum_i \sum_{Pa_i \in Pa_i} \left( - \sum_k \log \Gamma(c(k|Pa_i) + \alpha_k|Pa_i) \right) + \log \Gamma(\sum_k \alpha_k|Pa_i) + \sum_k \log \Gamma(\alpha_k|Pa_i),$$

(3.15)

cf. Equation 2.28.

Two-part encoding using logistic regression
For classification, it makes sense to only transmit the class labels of given data $D$ from a domain $X = \{X_0, \ldots, X_m\}$, where $X_0$ is the class variable. We therefore search for a code of length $l^{2p}_{LR}(d^0, d^1, \ldots, d^m)$, where $d^i$ is the $i$th column of matrix $D$. From the predictor data, the receiver also knows the number $n$ of class labels to be transmitted. We further assume, that we have agreed on the candidate set $Y = Y_1, \ldots, Y_K$ of covariates that may be included in the logistic regression model.
In order to minimize the total length of the transmitted code, we want to choose a sparse model, avoiding to encode unnecessarily many parameters, cf. Section 2.7. If we are using $K'$ parameters out of the given $K$ candidates, we can encode our choice in

$$l(K') = \log(K + 1) + \log\left(\frac{K}{K'}\right)$$  \hspace{1cm} (3.16)$$

bits, first encoding $K'$ according to a uniform distribution over $\{0, \ldots, K\}$ and then giving the actual indices of the non-zero parameters. For convenience let us—without loss of generality—assume that the chosen indices are $\{0, \ldots, K'\}$.

How to optimally encode continuous parameters is discussed in, e.g., [Gao 2000]. As we have had some good experience with the $\alpha$-prior, we choose to encode the $\beta$’s with respect to a discretized Laplace (double exponential) distribution located around zero with scale one. We have

$$P(\beta|\varepsilon) = \frac{1}{1 - \frac{1}{2}(\exp \frac{\varepsilon}{2} - \exp -\frac{\varepsilon}{2})} \frac{1}{2} \left(\exp \frac{\varepsilon}{2} - \exp -\frac{\varepsilon}{2}\right) \exp(-|\beta|),$$  \hspace{1cm} (3.17)$$

where $\varepsilon$ is the precision we use, the first factor comes from the fact that the receiver knows that a parameter encoded in this way will be non-zero, and the rest is the integral

$$\int_{x=\beta-\frac{\varepsilon}{2}}^{\beta+\frac{\varepsilon}{2}} \frac{1}{2} \exp(-|x|)dx = \frac{1}{2} \left(\exp \frac{\varepsilon}{2} - \exp -\frac{\varepsilon}{2}\right) \exp(-|\beta|)$$  \hspace{1cm} (3.18)$$

of the chosen distribution over an $\varepsilon$-interval around the value of $\beta$. The corresponding codelength $l(\beta; \varepsilon)$ is the minus logarithm of (3.17). We further choose $\varepsilon$ to be the inverse of an integer, which needs to be encoded selfdelimitingly in $l(\varepsilon) = 2 \log \frac{1}{\varepsilon} + 1$ bits.

The overall codelength becomes

$$l^{2p}_{LR}(d^0; d^1, \ldots, d^m)$$

$$= l(K') + l(\varepsilon) + \sum_{k=1}^{K'} l(\beta_k; \varepsilon) - \sum_{j=1}^{n} \log P^{LR}(d_{j0} | Y_1(d_j), \ldots, Y_{K'}(d_j), \beta)$$  \hspace{1cm} (3.19)$$

where all $\beta_k$ are multiples of $\varepsilon$ and the final term is the conditional log-likelihood under the logistic regression model given by Equation 2.38. It is now up to the sender to optimize the model with respect to this cost function.
For all the examples mentioned here, we have made arbitrary choices. For instance, for the $\beta$’s of the logistic regression model we could have chosen a different distribution, such as a Laplace distribution with variable, separately encoded scale. More examples will be given in Chapter 4.

We now look at the remaining open problems of model class selection raised in Section 2.4.

Solution to Problems 4, 6 and 7 (class encoding, overfitting)
As we have seen, even when the number of classes in $\mathcal{F}$ is large, there are ways to encode what we expect to be using in considerably fewer bits than $\log |\mathcal{F}|$, e.g. encoding Bayesian network structures by explicitly listing its arcs. Note, that also for sparse logistic regression models, $K' \ll K$, we have $\log(K+1) + \log \binom{K}{K'} \ll K$, where $K$ is the codelength we need to choose any of the $2^K$ subsets of the parameters using a uniform distribution. The ‘complexity’ of the class is now measured in bits, as is the data log-likelihood, and therefore the two parts of the code are directly comparable. The combined codelength trades off data fit against complexity automatically, quantifying Ockham’s razor and preventing overfitting.

Solution to Problem 5 (the parameter prior, postponed)
We have made some progress by encoding the parameters to specified precision. The larger the data, the higher the precision, as the relative weight of the parameter codelength decreases. However, we are still encoding the parameters with respect to an assumed distribution, effectively a prior. And while we can now compare different such distributions by the resulting total codelength, in many cases we can still do much better. Section 3.4 introduces the Normalized Maximum Likelihood (NML) distribution, which no longer needs a prior at all.

There are always many ways to encode data of any sort, the best being the one that yields the shortest codelength. It makes sense to ask oneself, which is the optimal codelength. The length of the shortest decodable description of a given data is known as the Kolmogorov Complexity and it comes with extensive theory. In the following section, we briefly review its main concepts and the implications relevant to this work.
3.3 Kolmogorov Complexity

A good introduction to the theory of Kolmogorov Complexity is provided by [Li 1997], which we use as the standard reference throughout this section. In the following, references to this book will be limited to page numbers. For an introduction, the book is rather lengthy (790 pages!), due to the fact that the theory of Kolmogorov complexity is very extensive. For this reason, this section is going to be very brief, hand-wavy at times. For instance, we will not go into the definition of prefix complexity, the self-delimiting version of the Kolmogorov Complexity, even though we are using it in the definition of the universal distribution. Nonetheless, we include all definitions and results we will need for motivation of the remainder of this work.

**Definition 6 (Identification of strings and natural numbers, p.12)**

We identify strings and natural numbers by the following bijection

\[
\mathbb{B}^* \rightarrow \mathbb{N}
\]

\[
s_n \ldots s_0 \mapsto \sum_{i=0}^{n} (s_i + 1)2^i
\]

This definition differs from standard binary representation (which is not a bijection) in that leading zeroes do add numerical value. It provides a natural way of enumerating strings of any length, where the empty string \(\epsilon\) corresponds to the number zero. It also enables us to define the Kolmogorov complexity (Definition 8) for natural numbers as well as for strings.

**Definition 7 (Universal Turing Machine, p.30)**

A **Turing Machine** is a device that manipulates symbols, which can be zero, one or blank, on a one-dimensional tape according to a finite program—i.e. set of rules—with a read/write head. At the beginning of a run, the tape is filled with blanks, except for a finite interval to the right of the read/write head of the machine. The string in this area, consisting of zeroes and ones, is called the **input**. If the machine halts, then the (finite) string found on the tape is the **output**. A **Universal Turing Machine (UTM)** is a Turing machine that can emulate the behaviour of any other Turing machine.

**Example 3** The author’s favourite programming language ANSI C (as most other programming languages) can be viewed as a UTM, since any Turing machine can be encoded in it. It clarifies thinking to regard universal Turing machines to simply be programming languages.
Definition 8 (Kolmogorov Complexity, p.106)

The Kolmogorov Complexity $C_U(s)$ of a string $s$ with respect to a universal Turing machine $U$ is defined as the length of the shortest input—or program—$\pi$ to $U$ that prints $s$ and then halts. We write

$$C_U(s) = \min_{\pi:U(\pi)=s} |\pi|.$$  \hspace{1cm} (3.21)

Kolmogorov complexity is defined only with respect to a UTM $U$. However, its dependence on $U$ is bounded by an additive constant, regardless of the length $\ell(s)$ of a string $s$.

Lemma 3 (Uniqueness of the $C_U(s)$ up to a constant, p.111)

For any two UTMs $U$ and $U'$ and all strings $s \in \mathbb{B}^*$ we have

$$C_{U'}(s) - c_{U',U} \leq C_U(s) \leq C_{U'}(s) + c_{U',U}.$$  \hspace{1cm} (3.22)

The constants $c_{U',U}$ and $c_{U,U'}$ only depend on the universal Turing machines $U$ and $U'$ and are given by the length of a program in language $U$ to interpret $U'$ and vice versa. For this reason, we can speak of the Kolmogorov complexity $C(s)$, remembering that it is defined only up to an additive constant.

Example 4 The string $s = \{01\}^{1000}$ of length $\ell(s) = 2000$ has low Kolmogorov complexity $C(s) \ll 2000$ since there is a short program to produce it:

```
for(i=0;i<1000;i++)printf("01");
```

For the same reason the number $n = 2^{1000}$ has low complexity, i.e. contains little information.

Lemma 4 (Upper bound to $C(s)$, p.108)

There is a constant $c$ such that for all strings $s$ of length $\ell(s)$

$$C(s) \leq \ell(s) + c.$$  \hspace{1cm} (3.23)

This constant obviously depends on the chosen UTM, but can be assumed to be small. The above is easy to see, as the program

```
printf("%s",s);
```

provides the required.

Lemma 5 (Almost all strings are incompressible, p.117)

The fraction of strings which are compressible by at least $k$ bits is smaller than $2^{-(k-1)}$.  \hspace{1cm}
Consider the set of strings of length \( n \). There are \( 2^n \) such strings, but only at most \( 2^n - (k-1) - 1 \) descriptions of length \( \leq n - k \). This proves the point for any \( n \), and therefore also for the set of all strings.

This means that, if we generate a random string, e.g. by tossing a fair coin, it will almost certainly be incompressible. In contrast to that, our experience tells us that many data we come across in real life are highly regular and therefore compressible by a large amount. The world is far from random. Certainly, this holds for natural languages, which are highly structured, as well as contain a lot of redundancy, as discussed in [Shannon 1948]. We will exploit this fact in Chapter 4 of this work.

**Remark 6** For any string \( s \), its shortest description \( L_{min}(s) \) (of length \( C(s) \)) is incompressible. Otherwise, there would be a shorter description of length \( C(L_{min}(s)) \). Therefore, any compressible description of \( s \) is suboptimal. Shortest descriptions look random.

**Lemma 6 (\( C(s) \) is incomputable, p.127)**

The Kolmogorov complexity \( C(s) \) of a given string \( s \) cannot be computed.

Consider the expression

The smallest integer which cannot be described in ten words.

Seemingly, this is a contradiction. Have we not just described this number in ten words? The proof deduces, that the above cannot be a description. Hence, we cannot compute the number from it. More formal proof is provided by the referenced book.

But how can this be? Given a UTM \( U \), can we not simply run it on all inputs in parallel, until the shortest program to produce a given string \( s \) has halted? In fact, in this way we can compute a decreasing sequence of upper bounds for \( C_U(s) \).

**Lemma 7 (Semicomputability of \( C_U(s) \))**

The Kolmogorov Complexity \( C_U(s) \) can be approximated from above.

This can be seen as follows. Let a UTM \( U \) and a string \( s \) be given. Run \( U(0) \) for one time step. Then run \( U(0) \) and \( U(1) \) for one time step each. Continue to add the next input \( n \) and run each program \( U(0), \ldots, U(n) \) for another time step, until one of them halts with output \( U(i) = s \). Any such program provides an upper bound for \( C_U(s) \) and we can abort all runs of \( U(n) \) with \( n > i \). We only need to execute the programs that remain running to see whether an even shorter description—a new upper bound
for $C_U(s)$—can be found. Eventually, this bound will be tight and we will have found the shortest description of $s$.

But does this not contradict Lemma 6? Have we not just described a way to compute $C_U(s)$? Unfortunately, this is not the case. Yes, eventually we will find the shortest description, but we will not know that we did. The problem is, that the above procedure never terminates, as some of the programs never stop running. In effect, they get stuck in infinite loops.

**Lemma 8 (The Halting Problem, p.34)**
*Given a Turing machine $T$, its halting set is defined as the set of inputs $n$ for which $T(n)$ eventually halts, $H_U = \{ n : T(n) \text{ halts} \}$. There is no program to decide for a given number $n$ whether $n \in H_U$.***

Some of the programs of length $< C_U(s)$ will therefore run indefinitely. And since we cannot know whether any of them will halt (with output $s$), we never know, whether an even shorter description exists.

This is very much the dilemma in which an MDL researcher finds himself every day. We have found a description of a given data, but are uncertain of its actual quality. It is soothing to know that we are not alone, we simply cannot know whether a shorter description exists. But at least we have an objective measure of quality, namely the codelength. So we can always compare and happily shout out: ’mine is shorter than yours!’

**Definition 9 (The Universal Distribution, p.273)**
*We can define the Universal Distribution with respect to a UTM $U$ to be

$$P_U(s) \propto 2^{-K_U(s)},$$

where $K_U(s)$ is the prefix complexity [p.202], the length of the shortest selfdelimiting program for $U$ that prints $s$ and halts.*

Of course, this distribution cannot be computed. But it has some remarkable properties. First of all, most random strings coming from it are in fact compressible. Remember that by Lemma 5 almost all strings are incompressible, yet a universal distribution assigns high probability exactly to those strings that can be compressed. Therefore it reflects our experience that many data we encounter are highly regular.

Also, the universal distribution dominates all computable distributions up to a multiplicative constant, which depends on the length of the program $\pi$ for $U$ that generates that distribution.
We can imagine the universe as a universal Turing machine $U$ which runs on random inputs $\pi$ from a distribution\(^3\) defined by $P(\pi) \propto 2^{-\ell(\pi)}$. What we observe will then be the outputs $U(\pi)$ for those inputs $\pi \in H_U$ on which the machine halts.

There is another remarkable property of the\(^4\) universal distribution: it equalizes worst- and average-case.

**Lemma 9 (Worst-case and Average-case are identical, p.290)**

*With respect to a universal distribution, the average-case performance of any algorithm with respect to any objective is the same as its worst-case performance.*

The reason for this is, loosely speaking, that being the worst-case input to the algorithm is almost a description—of constant length—of a string $s$. To specify such $s$, we must only provide its length. For any given length, the universal distribution reserves a constant fraction of probability mass for the worst-case, dragging the average-case down to the same level.

**Example 5 (Quicksort, p.291)**

*The sorting algorithm quicksort is known to have a worst-case running time of $O(n^2)$, quadratic in the size $n$ of its input. Its average-case performance (with respect to a uniform distribution over inputs of size $n$) is of complexity $O(n \log n)$. It has been noticed though, that the worst-case behaviour does surface, and more often than one would expect. Under the assumption that real-world data come from a universal distribution, Lemma 9 explains this fact.*

\(^3\)In order to define this probability distribution, $\ell(\pi)$ needs to be such that $\sum_{\pi=0}^{\infty} 2^{-\ell(\pi)} < \infty$, e.g., a selfdelimiting (naive) description.

\(^4\)we can speak of the universal distribution, defined up to a multiplicative constant, in the same way that we can speak of the Kolmogorov complexity, cf. Lemma 3.
3.4 Normalized Maximum Likelihood

As we have seen in the previous section, there are two major drawbacks to Kolmogorov Complexity. Firstly, it is only defined up to an additive constant, which depends on the chosen universal Turing machine. This constant may be of substantial weight, when—instead of asymptotics—we are interested in the complexity of actual, real-world data of limited size. Secondly, Kolmogorov Complexity is incomputable. And while it can be approximated from above, Lemma 7 supplies no way of doing so in practice.

Therefore, instead of using the theoretically optimal minimum description length $C(s)$, which uses a universal Turing machine, we need to retract to something less universal—but more practical. In the following, we introduce the Stochastic Complexity, which is defined with respect to a model class of our choice. It can be used to objectively choose among model classes. Once the model class has been fixed, it also defines a probability distribution, the Normalized Maximum Likelihood (NML).

In a sense yet to be specified, this is the optimal way of describing the data, discovering its regularities and predicting unobserved entities. The quality of this approach will depend—up to the usual constant we deal with in the theory of Kolmogorov complexity—only on the suitability of the model family under consideration.

In 1987, Yuri Shtarkov has proposed the following minimax problem:

$$\min_{Q \in C} \max_{|D|=n} \log \frac{\hat{P}(D|C)}{Q(D)} = \min_{Q \in C} \max_{|D|=n} \left( -\log Q(D) - \left( -\log \hat{P}(D|C) \right) \right),$$

(3.24)

[Shtarkov 1987], where $\hat{P}(D|C)$ is the maximum likelihood (2.32).

**Definition 10** Given a model class $C$, data $D$ and a distribution $Q$ we call the quantity

$$\log \frac{\hat{P}(D|C)}{Q(D)} = -\log Q(D) - \min_{Q' \in C} \left( -\log Q'(D|C) \right)$$

(3.25)

the regret of $Q$ for $D$ relative to $C$.

The regret is the number of excess bits we need to encode $D$ using $Q$, instead of the best possible distribution $\hat{P}(D|C)$ in $C$, which we cannot know before seeing the data.

Let us stare at (3.24) for a while. Given model class $C$, we play the minimax game by picking a distribution $Q$, against an opponent who picks a data set $D$ of size $n$ and a distribution $\hat{P} \in C$. We move first, and pick
any distribution $Q$ (which need not be a member of $\mathcal{C}$). Then our opponent picks data $D$ and encodes it with hindsight, using the best compressing model $\hat{P}(D|\mathcal{C}) \in \mathcal{C}$ there is in the class. We, in turn, use $Q$ to encode $D$, a distribution we had to pick before we knew what we were going to be using it for. We try to minimize the regret—the difference of the two codelengths—, i.e. the number of bits we need more than the max-player. Our opponent tries to maximize the regret. When picking $Q$, we therefore minimize the worst-case regret.

Shtarkov also provided the unique solution to (3.24), the Normalized Maximum Likelihood (NML) distribution (a.k.a. the Shtarkov distribution) given by

$$P_{\text{NML}}^C(D) = \frac{\hat{P}(D|\mathcal{C})}{R(\mathcal{C}, n)},$$  \hspace{1cm} (3.26)

where the normalizing constant $R(\mathcal{C}, n)$ depends only on model class $\mathcal{C}$ and data size $n$. The regret $\log R(\mathcal{C}, n)$ is therefore the same for any data of given size, and is also being called the parametric complexity of $\mathcal{C}$ for data size $n$. It is easy to see that this in fact does solve the minimax problem. Any distribution $Q$ differing from $P_{\text{NML}}^C$ is bound to have larger regret for some data. Since both $Q$ and $P_{\text{NML}}^C$ are distributions—i.e., sum to unity—there must be some data $D$ such that $Q(D) < P_{\text{NML}}^C(D)$ and therefore $Q$’s worst-case regret is larger than that of $P_{\text{NML}}^C$.

On the other hand, there are also data $D'$ for which the opposite is true and $Q$ achieves lower regret. Hence, $P_{\text{NML}}^C$ is not only optimal in the worst-case, but also the worst in the best case. The NML distribution spreads the regret evenly across all data. It is the only distribution that has constant regret, while any other distribution has larger regret on some, and smaller regret on other data.

The denominator—normalizing the maximum likelihood—is given by

$$R(\mathcal{C}, n) = \sum_{|D'|=n} \hat{P}(D'|\mathcal{C}).$$  \hspace{1cm} (3.27)

The codelength

$$- \log P_{\text{NML}}^C(D) = - \log \hat{P}(D|\mathcal{C}) + \log R(\mathcal{C}, n)$$  \hspace{1cm} (3.28)

the NML distribution assigns to data $D$ is called the stochastic complexity of $D$ relative to $\mathcal{C}$, [Rissanen 1987].

**Remark 7** This definition of the NML distribution is sufficient for the multivariate multinomial data domain $X$ we consider in this thesis. In general, $P_{\text{NML}}(D|\mathcal{C})$ can also be a probability density, in which case $R(\mathcal{C}, n)$
is given by an integral over all appropriate data. In many such cases—as well as in many cases of countably infinite data range—$R(\mathcal{C}, n)$ can be infinite and we have to restrict the data domain in order to define the stochastic complexity. But where it is defined, it still solves the minimax problem (which may turn into an inf-sup problem), see [Rissanen 2007]. For the scope of this work, the above is well-defined.

**Remark 8** The NML distribution, like the marginal likelihood distribution for Bayesian networks, mimics the behaviour of all distributions $\mathcal{M} \in \mathcal{C}$. But not with respect to an assumed prior distribution, but worst-case optimally relative to the model class. Unlike the marginal likelihood, the NML distribution does not share the parametric structure of $\mathcal{C}$, and therefore it lies outside of the model class it mimics. In fact, it is a non-parametric distribution altogether. Nonetheless we can use it for prediction, by using the plug-in predictor

$$P_{NML}^{\mathcal{C}}(d_{n+1}|D) = \frac{P_{NML}^{\mathcal{C}}(D \cup d_{n+1})}{\sum_{d'_{n+1}} P_{NML}^{\mathcal{C}}(D \cup d'_{n+1})} = \frac{\hat{P}(D \cup d_{n+1}|\mathcal{C})}{\sum_{d'_{n+1}} \hat{P}(D \cup d'_{n+1}|\mathcal{C})}. \quad (3.29)$$

**Solution to Problem 5 (no parameter prior)**

Being non-parametric, the NML distribution very elegantly avoids the problem of finding a suitable parameter prior: it does not need any.

In many cases, the NML distribution is the best we can do when we are using model class $\mathcal{C}$. Not knowing the data generating process, we aim to minimize the expected codelength. But with respect to which distribution? If there were a generating distribution and further it would be known to us, then there would be nothing to do. By Shannon’s theorem, this distribution itself is the one we should use to encode the data, the expected codelength being its entropy. Instead, we may want assume that data come from a universal distribution which, of course, cannot be computed. For this reason, we retract to the use of a model class $\mathcal{C}$, of which we hope that it is able to capture the regularities appearing in $D$, or at least a large portion of them. Worst-case optimality assures, that we have taken into account all regularities that $\mathcal{C}$ can capture. Remember that, with respect to a universal distribution, the worst-case is the average-case.

Worst-case optimality with respect to the regret therefore means average-case minimal codelength with respect to the universal distribution. While
the assumption that data come from a universal distribution is debatable, it still is a natural choice when there is no better candidate available. It is also a safe choice, as the universal distribution dominates any (computable) prior distribution we might have chosen. This, of course, only up to a constant, which depends on both the prior distribution it is compared against and the UTM with respect to which it is defined. Successful applications of the NML distribution in histogram density estimation [Kontkanen 2007b], image denoising [Roos 2005a], clustering [Kontkanen 2006], DNA sequence compression [Korodi 2005] and other areas have given evidence for this.

Naturally, there will also be regularities that \(C\) cannot capture, and a bad choice of model class yields a bad NML distribution. But NML can also be used for model class selection, by choosing the class within a family \(F\) which minimizes

\[
l^C_{\text{NML}}(D|n) = l(C) - \log P^C_{\text{NML}}(D).
\]

(3.30)

As in Section 3.2, we need to first encode the model class itself, before the receiver knows the NML distribution we will be using.

The model class \(C\) chosen in this way is the one which best suits \(D\) and therefore, also the corresponding NML distribution \(P^C_{\text{NML}}\) is the most meaningful among all NML distributions for \(F\).
3.5 More Properties of the NML Distribution

Invariance to parameter transformation
By definition—which does not employ the parametric form of model class \( C \) — the NML distribution is invariant to any sort of parameter transformation. For instance, it is automatically identical for all Bayesian network structures belonging to the same equivalence class.

The regret as a penalty term
Let us unfold Equation 3.30 to

\[
I_{\text{NML}}^C(D|n) = l(C) - \log \hat{P}(D|C) + \log \sum_{|D'|=n} \hat{P}(D'|C) \tag{3.31}
\]

and observe that it looks—apart from the sign, as now we are minimizing—a lot like Equation 2.31, the Bayesian penalized model class selection criterion. The NML cost (i.e. the negative penalty) then is

\[
\text{cost}_{\text{NML}}(C|n) = l(C) + \log \sum_{|D'|=n} \hat{P}(D'|C). \tag{3.32}
\]

The first term—\( l(C) \)—is not actually related to the NML distribution, it simply comes from the encoding of the model class we have chosen. The second term—the parametric complexity (regret) of \( C \)—measures the complexity of \( C \) in a very sensible way, as the sum over the maximum likelihoods assigned to any data (of given size) we might have observed. The NML model selection criterion (3.31) therefore chooses a model class \( C \) that allows for good fit of \( D \) relative to the fit of any data \( D' \).

The regret, as does the penalty term of the Bayesian information criterion, depends on the data size \( n \). In fact, BIC has been viewed as an approximation to the stochastic complexity, see [Kontkanen 2003], but found to be inferior to other approximations. Also, BIC disregards the class encoding term \( l(C) \), which can make a crucial difference [Roos 2009].

Conditional NML
For supervised learning tasks, where the receiver is assumed to be aware of the predictor data \( d^1, \ldots, d^m \) (cf. Eq. 3.19) we can define the conditional NML distribution

\[
P^C_{\text{NML}}(d^0|d^1, \ldots, d^m) = \frac{\hat{P}(d^0|d^1, \ldots, d^m, C)}{\sum_{|d^0|=n} \hat{P}(d^0|d^1, \ldots, d^m, C)}, \tag{3.33}
\]
3.5 More Properties of the NML Distribution

where \( \hat{P}(d^0|d^1,\ldots,d^m) \) is now the conditional maximum likelihood. In this way, we get a model selection criterion

\[
l_NML(d^0|d^1,\ldots,d^m) = l(C) - \log \hat{P}(d^0|d^1,\ldots,d^m,C) + \log \sum_{|d^0|=n} \hat{P}(d^0|d^1,\ldots,d^m,C) \tag{3.34}
\]

where the sum of the regret only goes over that part of the data in which we are interested.

NML to deal with sampling bias

The situation in which the data itself has an influence on the fact whether or not it will be observed by us, is known as the sampling bias. In such case, we often still want to model the underlying process that generated the data in the first place, excluding the sampling bias. Then we should—when we want to minimize the expected MDL codelength—weight the data \( D \) by their probability \( w(D) \) of being observed, to arrive at an unbiased (weighted) NML distribution

\[
P_{NML}^{C,w}(D) = \frac{w(D)\hat{P}(D|C)}{\sum_{D'} w(D')\hat{P}(D'|C)}. \tag{3.35}
\]

This, of course, requires that we have a fairly good understanding of the nature of the involved sampling bias. [Grünwald 2009] investigates such problem, in which statistical data is to be used as evidence in a court of law. Part of this same data had led to the trial in the first place, introducing sampling bias: had the data been different, the case might have never been raised. Grünwald studies several different approaches, all leading to the conclusion that the evidence that had led to police investigation should have lesser weight in court than the evidence gathered during the investigation.

The basic assumption has been, that the police got involved only when the observed data \( D \) had become sufficiently extreme, that is, a function \( f(D) \) had exceeded some threshold \( T \). The point of time when that had happened gives us a fairly good estimate of \( T \). The biasing weights are therefore binary,

\[
w(D) = \begin{cases} 1 & \text{if } f(D) \geq T \\ 0 & \text{else} \end{cases}. \tag{3.36}
\]

The unbiased NML distribution is then similar to the standard NML, only differing from it by a restriction in the sum of the regret

\[
P_{NML}^{C,T}(D) = \frac{\hat{P}(D|C)}{\sum_{D': f(D') \geq T} \hat{P}(D'|C)}. \tag{3.37}
\]
The larger the threshold $T$, the smaller the $T$-restricted regret. The $T$-restricted NML distribution assigns higher probability to $\mathbf{D}$ with increasing $T$, as it is a distribution over fewer possible data the higher the threshold becomes. Higher probability in turn means less decisive evidence in a court. This is in accordance with Grünwald’s findings.

**Computability**

Unlike the universal distribution, NML is computable or, in case of continuous data, can be approximated to arbitrary precision. But its computation is demanding, in many cases forbiddingly so. First, we need to be able to efficiently compute the maximum likelihood $\hat{P}(\mathbf{D}|\mathcal{C})$ and then, for the regret, an exponential sum (resp. multidimensional integral) over all data we might have observed. This means that straightforward computation takes exponential time. In some cases, however, we can calculate the NML distribution in polynomial time, in a less brute-force way.

We have seen that we can hardly expect to ever get closer to the universal distribution than with NML. But whether we can compute the NML distribution efficiently, still depends on the model class under consideration. Section 3.6 reviews recent advances in NML computation.
3.6 Computing the NML Distribution

Computation through the sufficient statistics

We cannot efficiently compute the parametric complexity straightforwardly, as it involves an exponential sum or a multidimensional integral. However, the maximum likelihood under model class $C$ depends on the data $D$ only through its counts, see Section 2.2. Grouping the possible data $D'$ of size $n$ into sets corresponding to the same count vectors can therefore enable us to calculate the regret more efficiently.

In this way, [Rissanen 2000] has developed an NML criterion for linear regression and applied it to denoising problems. While the (worst-case) regret itself is infinite in this case, multiple levels of bounding the involved integral and renormalization lead to a criterion which is independent of the chosen bounds, as their influence on the resulting codelength is constant with respect to the model class. Calculation of this criterion can be done in linear time. Moreover, for orthonormal basis functions, it can be shown that the class which optimizes this criterion retains, out of $n$ coefficients, the largest $k$ for some $0 \leq k \leq n$. Therefore we can find the NML-optimal model in time $O(n \log n)$, the computational complexity of ordering the coefficients. This has been exploited in speech signal [Rissanen 2000] and image [Roos 2005a] denoising.

For a single multinomial variable of cardinality $K$ (cf. Section 2.2), [Kontkanen 2003] has proposed an algorithm that computes the NML distribution with time complexity $O(n^{K-1})$. The same paper discovered a recursive formula, which improves this result to $O(n^2 \log K)$.

For Bayesian networks of multiple multinomial variables, the situation becomes more complicated. Unlike the marginal likelihood (Eq. 2.28), the NML distribution does not factorize into local terms for each variable. However, for the naive Bayes network structure (e.g. top left network in Figure 2.6), [Kontkanen 2005] introduces an algorithm of time complexity $O(n^2 \cdot \log K_0)$, where $K_0$ is the cardinality of the root node, again using a recursion formula. Once more, the same paper immediately improves upon its own result, with a fast Fourier transform to $O(n \log n \cdot K_0)$.

Publication II of this thesis developed an algorithm to compute the normalized maximum likelihood for Bayesian forests, Bayesian networks in which any node can have at most one parent (cf. Section 2.6). Its time complexity is $O(n^{K^*-1})$, where $K^* = \max_{i: \exists (j: i=P_a_j)} (K_i \cdot K_{P_a_i})$. Therefore the algorithm is polynomial in the data size $n$, but the degree of this polynomial depends on the maximal product of the cardinalities of an inner node (one that has a parent and at least one child) and its parent. This clearly is tolerable only when these cardinalities are small, e.g. for binary
variables. The reason for this behaviour lies in the non-factorizing nature of the regret. The bottleneck of the algorithm is at the inner nodes of the forest. Here, the problem of summing over all possible data $D'$ of size $n$—weighted by their contribution to the regret—is equivalent to the problem of counting non-negative integer matrices with given marginals (row- and column-sums). The latter has been proven to be $\#P$-hard, [Dyer 1997], which casts strong doubt on the existence of an algorithm that would be polynomial with respect to not only $n$, but also the cardinalities $K_i$ of the involved multinomials.

**Computation using generating functions**

In the year following the appearance of Publication II, [Mononen 2008] developed a slightly faster algorithm for the same problem. However, its time complexity is of similar order and the algorithm is not polynomial in all input quantities. Interestingly, Mononen’s algorithm takes an entirely different approach, employing generating functions [Flajolet 2009].

The idea to apply generating functions to NML calculation problems is due to Petri Kontkanen, who developed a linear-time algorithm for computation of the (single variable) multinomial NML [Kontkanen 2007a]. This has been exploited in histogram density estimation in [Kontkanen 2007b]. For another application, [Mononen 2007]—slightly—improved the time complexity of NML computation for the naive Bayes network structure to $O(n^2)$, no longer depending on the cardinalities of the involved variables.

**Approximations**

While efficient algorithms for exact computation of the stochastic complexity have been developed—and successfully applied—for a number of relatively simple model classes, it remains a fact that NML computation is infeasible for many model classes that we would like to use. This gives rise to the question, whether in these cases we can at least calculate good approximations to this score.

As mentioned earlier, one can view the Bayesian information criterion (BIC, [Schwarz 1978]) as such an approximation. [Kontkanen 2003] compares it to two more refined approximations—Rissanen’s asymptotic expansion [Rissanen 1996] and the Szpankowski approximation—in the cases of a single multinomial variable (with varying cardinalities) and the naive Bayes network structure. The latter approximation Kontkanen derives based on Szpankowski’s theorem on the redundancy rate for memoryless sources [Szpankowski 2001], once more using the generating function trick. In all reported situations, the Szpankowski approximation turns out to be most...
accurate, while Rissanen’s asymptotic expansion is a better approximation than BIC.

For the case of general Bayesian networks—including the special case of Bayesian forests—the problem with NML computation is the fact that the sum of the regret does not factorize into local scores at each variable. [Myllymäki 2008] proposes to use the factorized Normalized Maximum Likelihood (fNML), an approximation to NML which encodes the data columnwise and normalizes each column separately, resulting in a code-length of

\[
l_{\text{fNML}}^c(D|n) = \sum_{i=1}^{m} l_{\text{NML}}^c(d_i|d_{\text{pa}_i}, n),
\]

with the definitions of Equation 3.26. This score can easily be computed, and is reported to perform favourably as compared to the marginal likelihood score. It remains unclear, how close to NML its factorized version fNML comes, since NML itself cannot be computed.

A similar approach is taken in [Silander 2009]. The sequential Normalized Maximum Likelihood (sNML), encodes the data rowwise and normalizes each sample separately:

\[
l_{\text{sNML}}^c(D|n) = \sum_{j=1}^{n} l_{\text{NML}}^c(d_j|d_1, \ldots, d_{j-1}, n).
\]

Sequential NML can be seen as a prequential score in the same way that Bayesian marginal likelihood can. But unlike the marginal likelihood, sNML does depend on the ordering of the encoded data samples, even when the chosen model class makes the i.i.d. assumption.

We may also combine fNML and sNML (‘fsNML’), and normalize after the encoding of each matrix entry. In all cases, each normalizing sum only goes over the parts of the data that appear to the left of the conditioning bar, which greatly simplifies computation.

The idea to normalize in smaller chunks, in order to make the resulting score efficiently computable, makes NML-derived scores applicable to a wide range of model classes. Sequential NML, for instance, has been successfully applied to regression problems [Rissanen 2010]. When deviating from the standard definition of NML in this way, the resulting code-length is no longer worst-case optimal nor does the regret (or, rather, the series of local regrets) remain independent of the data at hand. For this reason, we can no longer speak of the parametric complexity of a model class. However, methods derived in this fashion seem to work very well in practice.
3.7 Summary

We have seen, how Shannon’s source coding theorem and Kraft’s inequality tie probabilities to codelengths. Maximum probability becomes minimum codelength. The MDL principle is based on this observation, transforming the problem of data modeling into a problem of data compression.

Already in its simplest form—the so-called two-part codes—the MDL principle solves most of the problems we have encountered in Bayesian model class selection. Others are greatly simplified, made more explicit. Two-part codes offer a theoretically sound approach to data encoding, and enable us to avoid many pitfalls common in data modeling. We can compare any models (codes), regardless of their parametric structure, simply by comparing their compression capabilities. This can be done using only the data at hand, without any assumptions regarding the source they might have come from.

When we require an encoding to be universal in the sense that it can describe any given data (of appropriate form), then the shortest codelength we can achieve is, by definition, the Kolmogorov complexity. As its definition involves the use of a universal Turing machine, which may freely be chosen, Kolmogorov complexity is only defined up to an additive constant. Furthermore, it is incomputable. The theory behind it, however, has some important practical implications. It leads to the NML distribution, which often gives us the shortest codelength we can achieve in practice. This non-parametric distribution simulates the use of all models within a given model class—without being a member of it—using no prior on the model parameters. It can be also used as an objective tool to choose among competing model classes. Model class selection then boils down to the definition of a suitable model family to choose from, and computation of the stochastic complexity.

The NML distribution can be computed efficiently only for a few relatively simple model classes. But these classes do appear in real-world problems—often as subproblems—and a number of successful NML applications have been reported. In case we cannot compute the stochastic complexity exactly, this is no reason to throw in the towel. A variety of approximations have been developed and put to the test with encouraging results.

The following chapter introduces applications of MDL methods in the field of computational etymology, the study of the history of words. We are interested in phonetic sound change over time, across a family of kindred languages.
Chapter 4

Etymology

“Eurgh!”
—Arthur Dent, using a babel fish for the first time.

4.1 Motivation

Etymology is the study of the history of words. Our focus is on cross-language change of sounds, the way phonetics have developed over time within a family of languages descending from a common ancestor. Typically, this ancestor language is not known to us, as it dates far back into ancient past. The data serving as input to our methods takes on the form of cognate sets, groups of related words from kindred languages, as described in more detail in Section 4.2.

We introduce several models to investigate and evaluate these cognate sets. Our main point of departure is alignment of etymological data, i.e., identifying the symbols or sounds that correspond. In etymology, the alignment problem is different from alignment in Machine Translation [Och 2003], which seeks correspondence between words in the source and target languages. While there is some ambiguity also on the level of words, sounds appear in a much wider range of context. Moreover, in our problem setting we do not have a dictionary available to help in mapping sounds. Instead, we want to discover these correspondences, together with the contexts they appear in.

Given a raw set of etymological data, we aim to find the best alignment, in a sense yet to specified. Motivation for this starting point will be given in Section 4.3. As it turns out, our alignment models also include the rules of sound correspondence we are after, as it is these rules that enable us to
compress the data. The models, model classes and families we develop to this end are described in detail in Sections 4.4 and 4.5.

Sets of etymological data are found in digital etymological databases, such as ones we use for the Uralic language family. A database is typically organized into cognate sets. Each element of such cognate set is a word in one of the member languages of the family under consideration, all cognates forming such set are posited to be derived from a common origin and thus to be genetically related. This origin is some (unknown) word form in the assumed common ancestor language. The cognate sets are formed by linguists, and often there is debate about their correctness. While some word forms are clearly related, others are more distant, believed to belong to the set by some linguists, rejected by others. Each cognate set is—simplifyingly—assumed to have evolved from a protoform in a common ancestor language in a straight line, not to have been lost and reintroduced by borrowing. The languages in the family form a tree that describes their history of separation. This is a standard assumption and lean words, where detected, are not being included in the cognate sets.

Computational etymology poses several problems, including the discovery of regular sound correspondences across languages in a given language family and determination of genetic relations among groups of languages, both of which are subject of this work. Problems we only brush are the discovery of cognate sets and reconstruction of unobserved word forms. The latter splits further into diachronic reconstruction, i.e. reconstruction of protoforms for a hypothetical ancestor language, and synchronic reconstruction of word forms that are missing from a known language.

As we develop our alignment models at the sound or symbol level, we explicitly model correspondence of sounds. In the process of evaluation of these models, we also arrive at modeling relationships among entire languages within the family. Section 4.7 describes ways of inferring phylogenetic language trees, describing the assumed genetic interrelations between the languages in the family, from the models we have learned. Construction of phylogenies is studied extensively, e.g., by [Nakhleh 2005, Ringe 2002, Barbançon 2009]. Their work differs from ours in that it operates on manually precompiled sets of characters, which capture divergent features of languages within the family, whereas we operate at the level of sounds within words and cognate sets.

The problem of cognate discovery is addressed in [Bouchard-Côté 2009, Kondrak 2004, Kessler 2001]. Here we consider the etymological data—cognate sets—to be given. Different data sets may include different—even conflicting—cognate sets. We start out from a set of etymological data (or
4.1 Motivation

more than one such set) for a language family as given. Our focus is on the principle of recurrent sound correspondence, as in much of the literature, including [Kondrak 2002, Kondrak 2003]. This means that we assume that whenever a sound has changed in some language, then this has happened in some specific context throughout the entire language.

Reconstruction, such as the protoforms given in [Rédei 1991], seems to be the most demanding of the fore-mentioned problems. To this date, it has been done purely by hand, essentially by making educated guesses. Our imputation method, introduced in Section 4.6, provides a first approach to automatically reconstruct missing word forms of known languages. Reconstruction of protoforms is a subject of future research. We do believe, that the MDL principle, together with a suitably adjusted imputation procedure, can provide a means to this end.

Comparative and historical linguists have worked on problems of etymology for centuries. Their methods have been applied to various language families with immense labour. The obtained results have been subject to debate and disagreement, as every linguist has introduced her personal expertise and—inevitably—bias and subjectivity.

Computational linguistics provide means to analyze data far more efficiently. Moreover, only some major language families have extensively been studied from the etymological perspective, while many others have not. Language families such as Indo-European have received more attention than others and have been studied in greater detail, mainly because there is more relevant data that has been collected, and this data has been available to scholars for a longer time. For language families that have been paid lesser attention, automatic analysis will allow linguists to obtain results quickly, providing a foundation for further, more detailed investigation. We have been the first to study the Uralic language family by computational means, but no longer are we alone [Honkola 2013].

Minimizing description length also adds a level of objectivity to the analysis, as we can measure performance in bits. Of course the data itself, as well as the choice of the model family under consideration, remain choices to be made—an inescapable source of subjectiveness. Our methods may uncover previously unrecognized regularities, but even in case they only validate previously established theories, this is a useful result. Because computational approaches differ in nature from traditional linguistic methods, a matching result always serves as a non-trivial, independent confirmation of correctness of traditional methods, or adds its voice to an open debate.

Computational methods can provide valuable tools for the etymological
community. From a linguistic point of view, the methods can be judged by how well they model certain aspects of etymology, and by whether the automatic analysis produces results that match theories established by manual analysis. Imputation—reconstruction of previously deleted cognates—also provides an intuitive method of model quality evaluation, much like cross-validation in statistics.

From an information-theoretic point of view, we evaluate a model (class) by the code length it achieves. We will see that code length correlates well with the above linguistic criteria, proving our approach to be consistent.
4.2 The Data

Our methods are applicable to any reasonable collection of cognate sets from any language family. In this work we focus on the Uralic language family, using the StarLing Uralic database [Starostin 2005] originally based on [Rédei 1991]. So far, we have also run our algorithms on Suomen Sanojen Alkuperä (SSA, ‘The Origin of Finnish Words’, [Itkonen 2000]), a Finnish-centered etymological dictionary, and the StarLing Turkic database. For simplicity, we present our results—as well as all example alignments—only for StarLing Uralic, on which we will take a closer look in the following.

The StarLing Uralic database consists of 1898 cognate sets containing cognates from 15 languages. In this work, we restrict the data to the 10 languages from the finno-ugric branch, namely Estonian (EST), Finnish (FIN), Khanty (KHN), Komi (KOM), Mansi (MAN), Mari (MAR), Mordovian (MRD), Saami (SAA), Udmurt (UDM) and Hungarian (UGR). Each cognate set is derived from an entry in [Rédei 1991], an example of which is given in Figure 4.1. For each language, there may be multiple word forms from different dialects given. The data is mostly stemmed, containing relatively little extraneous morphological material. This is not the case with all data.

Figure 4.1: Original data entry corresponding to the Finnish word stem juokse- (‘to run’) in [Rédei 1991].

StarLing compiles these entries into a more digestible format, an example is given in Figure 4.2. Entries subject to disagreement among linguists are indicated by (one or more) question marks. In the example
of Figures 4.1–4.2, the Hungarian form 'ív' is questionable according to [Rédei 1991]. The StarLing database, however, marks all other entries as uncertain.

Figure 4.2: Preprocessed data entry as found in [Starostin 2005].

Finally, we simplify this further by arranging the data into a matrix, parts of which are shown in Figure 4.3. We ignore question marks (for now) and restrict each language to its most prominent dialect\(^1\). The resulting matrix has many missing entries, as cognate sets need not contain entries for all languages in the family. In fact, there are very few that do. Missing word forms are marked by dashes. Each language has its own alphabet, which can be either print or phonetic alphabets, depending on the database. We prefer the latter, since we are interested in sound change, not orthographic conventions.

Figure 4.3: Part of the cognate matrix serving as input to our methods.

\(^1\)We have also run our algorithms on data including the multiple dialects, but the aim here is to just give one example of a possible input.
4.2 The Data

Our basic assumption is, that phonetic change is a regular process. In other words, if along the way from an ancestor to the observed language some sound has undergone change, say, an 'a' changing into an 'o', then this happens throughout the language. Any such change may be context-dependent, but never random. But if the underlying process is in fact deterministic, then why use probabilistic models in the first place?

First of all, there are several sources of noise present in the data. It has been compiled by linguists with subjective views, and in many cases these views do conflict. There are also 'dubious' entries, indicated with a question mark (or multiple ones), which may be due to a weak semantic link, a violation of expected phonetic regularity, or a little of both. Even where cognate correspondence is undisputed, it may still only be partial. There may be morphological information contained in a word form, that simply does not correspond. The StarLing data has been stemmed (mostly), but that still leaves the problem of ossification. By this we mean that a morph that once carried meaning of its own has over time become an integral part of a word and therefore can no longer be stemmed away.

Secondly, we cannot expect to be able to recover all relevant, deterministic rules of sound change from the data. Over thousands of years many such changes will have occurred, back and forth, each single change conditioned on context which might be long lost, as it has undergone changes of its own. An additional source of complexity in language evolution is known as blocking. Some phonetic change occurring in a language may have exceptions where some word form would become indistinguishable from another existing word, in which case the unaltered form is preserved. Clearly, we cannot learn such mechanism from the data we are using. Moreover, the data is rather small. For some language pairs there are only a few hundred cognate sets with entries for both languages. Some symbols within a phonetic alphabet may only appear once or twice in the whole data.

These considerations give motivation to use probabilistic models for this data. Our models need to be capable of allowing exceptions from each rule implicit to them, since rules can be expected to hold only to some degree.
4.3 The Alignment Problem

While etymological datasets, containing cognate sets, are readily available, the alignment at the sound level between two or more word forms in a cognate set is almost never given explicitly, as is the case with StarLing.

Instead, in linguistic handbooks one finds general rules of derivation from a parent language to daughter languages. The handbook, in turn, typically provides a handful of examples of the application of each rule, at best also mentioning how pervasive the rule is, i.e., whether there are exceptions—examples that contradict the rule. The actual alignment for most related forms in the database are therefore implicit to these rules.

This creates several immediate problems. First, the user of the database is left to her own devices to determine precisely how any two or more words in a cognate set are related—which rules of derivation give rise to the observed forms? By this we mean a full sound-by-sound accounting of relationships in the observed word forms, leaving no sound unexplained.

Secondly, the presented word forms may contain morphological material that is etymologically unrelated. Some databases give 'dictionary' forms, which usually contain extraneous morphological affixes, and thereby obscure which subpart of a given word form stand in etymological relationship with other members in the cognate set, and which do not. While some affixes may be 'obvious', many are not. As mentioned in Section 4.2, the StarLing Uralic database is relatively clean in this respect, but it does contain a lot of ossified affixes, parts of the given cognates that need not etymologically relate.

Most seriously, the posited rules are usually insufficient to explain the totality of the observed data. This is due to the complex nature of the underlying derivation processes, which limits the ability of database creators to capture the regularities in full, and to make them explicit as rules. The given rules may also have exceptions that are not mentioned (or acknowledged) in the handbooks, yielding incomplete explanation.

In fact, rules given in handbooks are usually idealized abstractions, covering only the common examples of relationship. Typically, the rules described apply cleanly only in the simplest and most straightforward examples, while in much of the data the rules have unexplained exceptions, or fail to hold altogether. While some amount of exceptions or failure may be insufficient grounds to categorically dismiss a rule, we must also account for the exceptions.

A central idea behind our automatic analysis is to extract the rules of correspondence directly from the data, with no side information given. We use etymological handbooks such as [Lytkin 1973, Sinor 1997] only to verify
our results, not to guide learning. Therefore our results are independent of linguistic bias, as far as this bias is not already part of the data. In other words, all rules we learn are inherent in the given corpus. Unlike, e.g., [Kondrak 2004] we also utterly disregard the semantics of the involved cognates.

Given a set of cognates, we know (or assume) that its members are genetically related, in some way and at least in some parts. But this relation is not given on a sound-by-sound level, only on the level of complete words. Since we are interested in phonetic change, we need to first find out which sounds correspond to each other. This is the alignment problem. In its simplest form, we align single sounds for a pair of languages. We equate sounds and symbols, assuming phonetic notation. We speak of source and target languages, even though in many cases our models—being symmetric—do not require this distinction. Let $\Sigma$ be the source and $T$ the target alphabets, $|\Sigma|$ and $|T|$ their respective sizes.

At the symbol level, an alignment is then a pair $(\sigma : \tau) \in \Sigma \times T$ consisting of a source symbol $\sigma$ and a target symbol $\tau$. We call such pair an event in order to distinguish it from an alignment of single cognates or the whole corpus. Some symbols may align with themselves ($\sigma = \tau$), while others may have undergone changes during the time the two related languages have been evolving separately ($\sigma \neq \tau$). Clearly, with this type of one-to-one alignment alone we cannot align a source word $\sigma$ of length $|\sigma|$ with a target word $\tau$ of a different length $|\tau| \neq |\sigma|$. We also need insertions and deletions, for which reason we augment both alphabets with an empty symbol, denoted by a dot. We denote the augmented alphabets $\Sigma.$ and $T.$.

Typically, a sound will behave in a number of different ways depending on the context it appears in. For example, some symbol $\sigma$ from the source alphabet maps to the same sound $\tau_1 = \sigma$ in the target alphabet in most cases, but changes into a different sound $\tau_2 \neq \sigma$ inbetween vowels and gets deleted at the end of a word. The number of different choices for each sound can be assumed to be small.

We can now align word pairs such as $(\text{vuosi}, \text{al})$, meaning “year” in Finnish and Khanty, in any of the following

\[
\begin{align*}
\text{vuosi} & \quad | \quad | \quad | \quad | \\
\text{al} & \quad . \quad . \quad . \\
\text{vuosi} & \quad | \quad | \quad | \quad | \quad | \quad | \\
\text{a} & \quad \text{l} \quad . \quad . \\
\text{vuosi} & \quad | \quad | \quad | \quad | \quad | \quad | \\
\text{vuosi} & \quad | \quad | \quad | \quad | \quad | \quad | \\
\text{al} & \quad . \quad . \\
\text{vuosi} & \quad | \quad | \quad | \quad | \quad | \quad | \\
\text{v} & \quad \text{u} \quad \text{o} \quad \text{s} \quad \text{i} \quad . \\
\text{al} & \quad . \quad . \\
\text{vuosi} & \quad | \quad | \quad | \quad | \quad | \quad | \\
\text{vuosi} & \quad | \quad | \quad | \quad | \quad | \quad | \\
\text{al} & \quad . \quad . \\
\text{vuosi} & \quad | \quad | \quad | \quad | \quad | \quad | \\
\text{v} & \quad \text{u} \quad \text{o} \quad \text{s} \quad \text{i} \quad . \\
\text{al} & \quad . \quad . \\
\text{vuosi} & \quad | \quad | \quad | \quad | \quad | \quad | \\
\text{vuosi} & \quad | \quad | \quad | \quad | \quad | \quad | \\
\text{al} & \quad . \quad . \\
\end{align*}
\]

as well as another 58 ways we have counted.

The rightmost alignment, for example, consists of the symbol pairs $(\text{v:.})$, 85
(u:a), (o:.), (;:l), (s:.) and (i:.). Our objective is to find the best alignment for each pair (or set) of cognates in the data.

Applying the MDL principle, the goodness criterion will be the overall number of bits needed to encode the aligned data. We will verify in several ways that this in fact yields good models, as evaluated from a linguistic point of view.

The models presented in the following have been introduced in Publications III–VI. However, we take the freedom to deviate from these papers—both in content and in notation—whenever this seems to make the concepts more clear or more recent results suggest to do so.
4.4 Baseline Model and Extensions

Following the MDL paradigm, we need to encode the complete data $D$ in order to communicate it via a channel. As it is not meaningful to speak of an alignment without the underlying (unaligned) data $D$, we also speak of the alignment $D$. For a pair of languages, we want to minimize the code length of the ordered pairs of cognates together with the alignments we have chosen for each. Sending the pairs in order is an arbitrary choice with no effect on the alignments, the code length differs from that of unordered transmission by the additive constant $\log n!$, where $n$ is the number of cognates that source and target language share. For the receiver to be able to decode the message, the word boundaries must also be encoded. To this end we add the special character '$$' to both alphabets, meaning 'end of word', the augmented alphabets we denote by $\Sigma_\$$ and $T_\$$.

For the simplest of our model classes, which we call the 'baseline model', this character will only appear in the cognate-terminating event ($$ : $$). In some cases we will also need a symbol to denote the beginning of each word, we choose the character '##'. For the baseline model, we do not need to encode the cognate-initial event (## : ##), as we know where it will occur: at the beginning of the code, as well as after each event ($$ : $$) as long as we have not yet communicated all $n$ cognate pairs, but not elsewhere.

4.4.1 Baseline Model

Our simplest model class for the alignment problem considers the event space $\mathcal{E} = |\Sigma| \times |T| \cup (\$, \$)$ as a multinomial variable of cardinality $K = |\mathcal{E}| = |\Sigma| \cdot |T|$. The end-of-word symbol can only align with itself, while simultaneous deletion and insertion is not meaningful in this context, as it could only increase code length.

We encode the events using marginal likelihood with uniform prior. For an alignment $D = ((\sigma_1 : \tau_1), \ldots, (\sigma_n : \tau_n))$ of $D = ((\sigma_1, \tau_1), \ldots, (\sigma_n, \tau_n))$ with event counts $c(e)$ we therefore obtain a code length of

$$i_{MarLi}^K(D) = - \sum_{e \in \mathcal{E}} \log c(e)! + \log (|D| + K - 1)! - \log(K - 1)!,$$

where

$$|D| = \sum_{e \in \mathcal{E}} c(e)$$

is the total number of events in $D$, cf. Equation 2.16. Implicitly we have assumed that the number $n$ of common cognates is known to the receiver, such that the end of the message is unambiguous during decoding. Of
course, we could also start the message with a self-delimiting encoding of \( n \). This would add a constant to the codelength. Since this has no influence on the resulting model, we choose to ignore this term.

### 4.4.2 Learning Procedure

We learn an alignment of the corpus—all cognate pairs present in both languages under consideration—in the following way. Starting from a random alignment of all cognate pairs we iteratively realign one cognate pair \((\sigma_i, \tau_i)\) at a time. We first subtract all involved events from the counts. These decreased counts we denote \( c^{-i} \) and, for notational ease, the aligned data excluding the \( i \)th cognate pair \( D^{-i} \). We then align \((\sigma_i, \tau_i)\) optimally using dynamic programming [Bellman 1957] and reinsert the new events into the count vector. Note that this alignment is optimal only with respect to the 'frozen' model, based on \( D^{-i} \). For the complete \( D \)-model it is optimal in most cases, but in some cases only a good approximation to the optimum.

Dynamic programming requires that we find intermediate states, for which it does not matter (in terms of cost) how we got there. In our case, these states mark a point at which we have encoded the first \( u \) symbols of the source word and the first \( v \) symbols of the target word. We arrange these states in an alignment matrix \( A \) of size \( (|\sigma_i|+2) \cdot (|\tau_i|+2) \). Each cell stores the minimal achievable codelength of the corresponding partial alignment, recursively defined by

\[
\begin{align*}
a_{00} &= 0 \\
a_{u0} &= a_{u-1,0} + L((\sigma_u : :)|D^{-i}) \\
a_{0v} &= a_{0,v-1} + L((:: \tau_v)|D^{-i}) \\
a_{uv} &= \min\left\{ a_{u-1,v} + L((\sigma_u : :)|D^{-i}), a_{u,v-1} + L((:: \tau_v)|D^{-i}), a_{u-1,v-1} + L((\sigma_u : \tau_v)|D^{-i}) \right\} \\
& \quad \text{for } u, v \geq 1
\end{align*}
\]

(4.3)

and for \( u, v \geq 1 \) also which of the three choices (deletion, insertion, one-to-one alignment) achieves the minimum. For fixed \( i \), we denote the word lengths be \( N = |\sigma_i| \) and \( M = |\tau_i| \). The bottom rightmost cell \( a_{N+1,M+1} \) can only be reached from cell \( a_{NM} \). This transition corresponds to event \((\$ : \$)\). Cells \( a_{u,M+1} \) for \( u = 1..N \) and \( a_{N+1,v} \) for \( v = 1..M \) are disallowed, as any path through them would correspond to an alignment containing an event involving an end-of-word symbol together with some other symbol. Cell \( a_{N+1,M+1} \) then contains the cost of the best alignment, backtracking through \( A \) gives the alignment itself. Figure 4.4 visualizes the alignment matrix.
### 4.4 Baseline Model and Extensions

The cost of moving from one cell to another (to the right, down, or both) is given by the increase in codelength caused by adding the corresponding event $e$. These differences in codelength are

$$l(e|D^{-i}) = l(D^{-i} \cup \{e\}) - l(D^{-i})$$

$$= -\log(c^{-i}(e) + 1) + \log(|D^{-i}| + K),$$

corresponding to transition probabilities

$$P(e|D^{-i}) = 2^{-l(e|D^{-i})} = \frac{c^{-i}(e) + 1}{|D^{-i}| + K}.$$  

We realign all cognate pairs and update the model accordingly until the algorithm converges. Because this greedy method tends to get stuck in local minima, we use Simulated Annealing [Kirkpatrick 1983] with a suitable cooling schedule. Since the focus of this dissertation is on modeling, not optimization, we will not go into detail here. This learning scheme is common to all modeling approaches presented in this thesis. For some models, we have to make appropriate modifications which we will describe as needed.

---

<table>
<thead>
<tr>
<th></th>
<th>$#$</th>
<th>$\tau_1$</th>
<th>$\ldots$</th>
<th>$\tau_{v-1}$</th>
<th>$\tau_v$</th>
<th>$\ldots$</th>
<th>$\tau_M$</th>
<th>$$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$#$</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>$\sigma_1$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>$\ldots$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>$\sigma_{u-1}$</td>
<td>$a_{u-1,v-1}$</td>
<td>$\nrightarrow$</td>
<td>$\downarrow$</td>
<td>$a_{u-1,v}$</td>
<td></td>
<td></td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>$\sigma_u$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>$a_{u,v-1}$</td>
<td>$\rightarrow$</td>
<td>$a_{uv}$</td>
<td>x</td>
</tr>
<tr>
<td>$\ldots$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>$\sigma_N$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>$$</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
</tbody>
</table>

Figure 4.4: Alignment matrix $A$. Each cell $a_{uv}$ stores the codelength of the most probable alignment and a pointer to the cell we have come from to achieve it. Word beginnings are marked ‘#’, word endings ‘$’ and cells marked ‘x’ are disallowed.
4.4.3 Sanity Checking

We check the obtained codelengths against standard compressors (gzip, bzip2) and find that our models achieve shorter encoding, cf. Figure 4 of Publication IV. In this test, the input to the standard compressors has been the unaligned data. Our models also encode the alignments, which accounts for additional information. On the other hand, we can exploit the relatedness of the cognates, while standard compressors cannot. Since our models compress more, we conclude that this regularity outweighs the overhead from encoding the alignments. This is hardly surprising, but nonetheless soothing to observe.

The resulting alignments can also be checked by hand, using etymological handbooks and common sense. We find that in fact they already do make a lot of sense. There are no gold standards available for the alignment problem, since it is unclear what an optimal alignment should look like. Therefore, we cannot easily assign a score (apart from the codelength) to measure the quality of results. But the baseline model does find the vast majority of obvious correspondences.

Another way of looking at the results is through the $|\Sigma| \times |T|$ count matrices, examples of which are shown in Figures 2 and 3 of Publication III. We hope for these matrices to be sparse, as handbooks typically list only a small number of possible correspondences for each sound. Of course we also want to hit the right ones. Vowels should align with vowels, consonants with consonants, and in general sounds aligning with each other should be phonetically reasonably similar. For languages with similar alphabets we hope for a strong ‘diagonal’. Further, there should not be too many deletions and insertions.

Judged by these linguistic criteria, the baseline model is definitely on the right track. Starting from complete randomness, the codelength criterion guides the algorithm to meaningful alignments.

Finally, we wanted to know how resistant these models are to noise. To this end, we performed the following experiment. We aligned Finnish with itself, independently introducing noise into both the source and the target level. The contaminated data was produced by sweeping through the data on each level once, at each point adding a random symbol—uniformly drawn from $\Sigma = T$—with probability $\nu$ and reading off a symbol from the original data with probability $1 - \nu$. We hoped to be able to align each (original) source symbol $\sigma$ with the same symbol $\tau = \sigma$ on the target level, while the introduced noise symbols would be dealt with by deletions and insertions. Up to a noise level of $\nu = 0.85$, our models displayed the desired behaviour almost to perfection. For higher levels of noise the amount of
correctly found correspondences quickly decreased. This result proves the baseline model to be highly noise-resistant.

After these encouraging initial results with the baseline model, we now look at several extensions and improvements we have implemented.

### 4.4.4 NML

We want and expect the count matrix to be sparse, since typically a sound in the source language can correspond to only a small number of sounds on the target side, the context determining to which of these. In such case, the normalized maximum likelihood distribution yields slightly shorter codelength than the marginal likelihood, which is why we prefer to use it. With this simple model, it is easy to do the swap. The new codelength is

\[
I_{NML}^{K}(D) = -\sum_{e \in E} c(e) \log c(e) + \log R(K, |D|),
\]

where \( R(K, |D|) \) is the multinomial regret, a special case of Equation 3.27.

Note that, instead of the constant data size \( n \), here we have the total number of events, which does vary for different alignments due to insertions and deletions. Therefore, the parameters used during realignment are not given by the maximum likelihood as for the plug-in predictor (Eq. 3.29), and the change in codelength does not simplify. We have

\[
-\log P(e|D^{-i}) = L(D^{-i} \cup e) - L(D^{-i})
= (c^{-i}(e) + 1) \log (c^{-i}(e) + 1) - c^{-i}(e) \log c^{-i}(e)
+ \log R(K, |D^{-i}| + 1) - \log R(K, |D^{-i}|).
\]

Fortunately, we can precalculate the regrets for all possible total numbers of events and store them in a look-up table, instead of recalculating them every time we realign.

Switching from marginal likelihood to NML consistently decreases codelength. However, looking at the alignments and the count matrix, it is hard to tell the difference.

### 4.4.5 Codebook

As mentioned above, we expect the count matrix to be sparse. Therefore, it is beneficial to encode the positions of its non-zero entries separately, and subsequently the aligned data given this knowledge. Note that in the notation of this thesis—unlike that of Publications III and IV—this is not a
two-part code. Here we view the choice of the non-zero positions to define
a model class. Let \( \mathcal{E}^+ \subseteq \mathcal{E} \) be the set of positions of non-zero entries,
their number be \( K^+ = |\mathcal{E}^+| \). Then \( \mathcal{E}^+ \) is to be seen as the model class—or
codebook—we use to encode \( \mathcal{D} \). But before we can do so, we need to
communicate \( \mathcal{E}^+ \), which can be done in

\[
l(\mathcal{E}^+) = \log(K + 1) + \log \binom{K}{K^+} \tag{4.9}
\]

bits. We can then consider \( \mathcal{D} \) to come from a multinomial of cardinality
\( K^+ \).

The overall codelength—with the additional index 'CB' for codebook—then becomes

\[
l_{CB, \text{MarLi}}(\mathcal{D}) = l(\mathcal{E}^+) + l_{\text{MarLi}}^K(\mathcal{D})
= \log(K + 1) + \log \binom{K}{K^+} - \sum_{e \in \mathcal{E}^+} \log c(e)! + \log (|\mathcal{D}| + K^+ - 1)! - \log (K^+ - 1)!, \tag{4.10}
\]
in the marginal likelihood case, and

\[
l_{CB, \text{NML}}(\mathcal{D}) = l(\mathcal{E}^+) + l_{\text{NML}}^K(\mathcal{D})
= \log(K + 1) + \log \binom{K}{K^+} - \sum_{e \in \mathcal{E}^+} c(e) \log c(e) + \log R(K^+, |\mathcal{D}|) \tag{4.11}
\]
in the case of NML encoding.

The alignment procedure remains unchanged. Only the transition costs
(or probabilities) become slightly more complicated in cases where we have
to add a new entry to the codebook. Without writing it down explicitly,
be it mentioned that in both cases—\( e \in \mathcal{E}^+ \) and \( e \not\in \mathcal{E}^+ \)—the transition
probability is given by Equation 4.4 as the increase in codelength of adding
the corresponding event to \( \mathcal{D}^{-i} \).

Using such codebook significantly decreases codelength. At the same
time, the count matrices sparsify further. This is hardly surprising, since
adding a new event type to the codebook comes with additional cost. But
linguistic analysis (e.g., using handbooks) also reveals that the learned
alignments have greatly improved. We take this as evidence of the ade-
quacy of our approach.
4.4.6 Distinguishing Between Kinds of Events

Let us first some clarify some notation. We have spoken of events of being a single instance of some core alignment, say, (p:b). Event types in our notation are all such alignments throughout the aligned data $D$, that is, all instances the (p:b)-entry in the alignment matrix corresponds to. In the following, we also need event kinds, sets of alignment types that share a certain characteristic. We consider four different kinds, namely one-to-one alignments, deletions, insertions and word boundaries. The data space $E$ then splits into four parts, $E = E_{1-1} \cup E_{del} \cup E_{ins} \cup E_{\$}$.

In general, for $K$ kinds, we have $E = \bigcup_{\kappa=1}^{K} E_{\kappa}$. We denote the sizes of these subsets of the data space $K_{\kappa} = |E_{\kappa}|$, such that $K = \sum_{\kappa=1}^{K} K_{\kappa}$. Similarly, the event types actually occurring split into $E^+ = \bigcup_{\kappa=1}^{K} E^+_{\kappa}$ with sizes $K^+_{\kappa} = |E^+_{\kappa}|$. We do not expect different kinds of events to behave similarly. That is, the codebook is likely to contain quite different fractions of event types for the different kinds. Therefore it is beneficial to separately encode $K$ codebooks—one for each kind—of sizes $K^+_{\kappa}$. Encoding the data given these codebooks remains unchanged, yielding the new (NML) codelength

$$l_{CBs,NML}(D) = \sum_{\kappa=1}^{K} l(E^+_{\kappa}) + l_{NML}(D)$$

$$= \sum_{\kappa=1}^{K} \left( \log(K_{\kappa} + 1) + \log \left( \frac{K_{\kappa}}{K^+_{\kappa}} \right) \right) - \sum_{e \in E^+} c(e) \log c(e) + \log R(K^+, |D|)$$

(4.12)

with the index 'CBs' for (separate) codebooks, plural.

Once more, this reduces codelength, sparsifies the count matrix, and also improves the resulting alignments as seen from a linguist point of view.

4.4.7 Multiple Sound Alignment

So far, our models have not taken into account any of the context information on which phonetic rules generally condition. A first approach to do so is to align multiple sounds at a time. This is etymologically motivated, e.g. diphthongs in one language may correspond to single vowels in another, and has previously been done in [Kondrak 2003, Bouchard-Côté 2007].

While our MDL models find many ‘true’ multiple sound alignments (no gold standard exists, but some seem obvious), the actual strength of this approach lies in the context information it can capture. In order to exploit as much context information as possible in this way, we also model both
word boundaries explicitly, since they play an important role as conditioning context. As before, we denote the beginning of a word by ‘#’, its end by ‘$’. Encoding both initial and terminal word boundaries is a slight extension to what has been described in Publications III and IV. The MDL cost will be defined in such a way, that this does not lead to excess codelength.

We still regard the event space $E$ as a multinomial—when aligning $d$ symbols at a time it has cardinality $K = |E| \approx |\Sigma|^d |T|^d$. We choose to align at most two symbols at a time, such that this number is roughly a million. With the amount of data we have available, $d \geq 3$ seems unreasonable. An example of a two-to-two alignment—the word pair (tuomi, toom) meaning ‘bird cherry’ in Finnish and Estonian—is as follows.

```
#t  uo m   i$  
 |   | |   |
#t  oo m   $  
```

Here we have captured diphthong-to-long-vowel rule $(uo : oo)$, which can be seen as a correspondence truly involving two sounds on the source side. The other two multiple alignments (initial unvoiced plosive remains unvoiced (#t : #t) and terminal $i$ vanishes) involve context information.

We distinguish between 16 different kinds of possible event types, as listed below:

\[
\left\{ \begin{array}{lllll}
(# : #) & (#\sigma : #) & (# : #\tau) & (#\sigma : #\tau) \\
(\sigma : :) & (:. : \tau) & (\sigma_1\sigma_2 : :) & (:. : \tau_1\tau_2) \\
(\sigma : \tau) & (\sigma_1\sigma_2 : \tau) & (\sigma : \tau_1\tau_2) & (\sigma_1\sigma_2 : \tau_1\tau_2) \\
($ : $) & (\sigma$ : $) & (\sigma : \tau$) & (\sigma$ : \tau$) \\
\end{array} \right\}.
\]

The corresponding event subspaces are of largely differing sizes, event types involving two symbols on both source and target side outnumbering the simpler types. On the other hand, we expect few of these types ever to occur, since regular sound change requires relatively few rules. Therefore it becomes vital that we encode the codebooks separately, as described in Section 4.4.6.

The only real change to the cost function (apart from the definition of the event space) comes from the fact that we encode both word boundaries explicitly. But at each point of time during decoding, the receiver knows whether a new word is starting (after having read a ‘$’) or not. Therefore we can split data $D$ into two parts $D = D_{#} \cup D_{\rightarrow $}, where the first part consists of kinds of the top row in (4.13) and the latter of the rest. We can then encode these parts separately, and the receiver will be able to
merge them back together. Denote the event space and codebook sizes accordingly, \( K = K_\# + K_{\to S} \) and \( K^+ = K^+_\# + K^+_{\to S} \). The NML codelength then becomes

\[
l_{2-2}(D) = \sum_{\kappa=1}^{16} l(E^+_{\kappa}) + l^{K^+_{\#}}_{NML}(D_{\#}) + l^{K^+_{\to S}}_{NML}(D_{\to S}).
\] (4.14)

The alignment procedure, in essence, remains the same. However, the steps we are allowed to take in the alignment matrix of Figure 4.4 are now not only single steps right, down and diagonal, but also double steps into the same directions as well as right-and-down knight moves. Therefore, the minimum of Equation 4.3 is now taken over eight values. A word boundary may now be part of an aligned pair of symbols. The transition costs are once more given by Equation 4.4.

Two-to-two alignment further decreases codelength. It finds context-dependent rules of sound correspondence that could not be captured aligning only single sounds. Among them, e.g. for Finnish:Estonian, diphthongs \( uo, yö \) to long vowels \( oo, öö \), unvoiced plosives \( k, p, t \) remain unvoiced word-initially, elsewhere change into their voiced counterparts \( g, b, d \), and so on. The count matrix becomes too large to look at as such. We choose to only look at the one-to-one alignments in matrix form and at all others as a list. These lists are reasonably short, containing little ‘garbage’—entries that are not linguistically justified. Furthermore, all event types occurring more than once do make sense intuitively. The (one-to-one) count matrix becomes a lot more sparse, as the captured context rules reduce its entropy.

### 4.4.8 Separate Encoding of Affixes

To deal with the problem of poorly stemmed and/or ossified data described in Section 4.2, we need a means of encoding affixes without aligning them. Aligning parts of words that are not genetically related inevitably skews the distribution of events, and in doing so may keep a model from finding etymologically correct alignments in other parts of the corpus. These unrelated parts of cognates can be seen as noise in the data. From this point of view, we are simply facing a typical noisy data problem.

From the MDL perspective, noise is simply data that does not compress. When modeling data, noise often does have structure and is therefore compressible, but this structure is not of the type we are interested in. In our case, these ‘nuisance affixes’ share phonetic regularities of the language they belong to, but not the cross-language regularities we find by means of alignment. They can be encoded more compactly with a ‘naive’, single-language model.
The MDL way to separate noise from signal is to build two alternative models. One of them captures the regularities we wish to extract from the data, the other is a naive data model to account for parts of the data that do not display these regularities. The data is then encoded by a mixture of both, thereby separating them. This approach has been successfully applied to denoising problems, [Rissanen 2000, Roos 2005a]. We take a similar approach, briefly described in Section 6 of Publication IV. We define affix codebooks PCB (prefix codebook) and SCB (suffix codebook), encoding non-relating cognate parts in a naive, non-aligning way. We encode separate codebooks for source and target language, each in the same way. For example, the codelength of the source language prefix codebook is

$$l(PCB_{source}) = l(|PCB_{source}|) + l^{\Sigma+1}_{NML}(PCB_{source}),$$

(4.15)

where $l(|PCB_{source}|)$ is the length of a selfdelimiting encoding of the number of prefixes in the codebook, and $l^{\Sigma+1}_{NML}(PCB_{source})$ is the stochastic complexity of the codebook viewed as coming in an i.i.d. manner from a multinomial distribution of cardinality $|\Sigma| + 1$. The prefixes need to be encoded including their boundaries, hence the cardinality of the multinomial: the size of the source alphabet plus one for the boundary symbol.

We choose to encode the codebooks such that they reflect the order in which they appear in the aligned data, with possible repetitions. We therefore do not need to encode the the actual affixes when encoding the data, after the affix codebooks have been transmitted. We do, however, need to communicate whether an affix is present at any point, on source, target or both sides, which can be done by introducing special events. This makes the code of the affix-free data slightly longer, but in sum we gain.

Finding the optimal alignment of a pair of cognates can still be done with the dynamic programming procedure as described in Section 4.4.2. In addition to the previously mentioned steps through the alignment matrix, we are now allowed to do ’hyper-jumps’, that is, enter and exit the matrix at any given cell, at the associated cost coming from the implied change in codelength. An example of an affix-stemmed alignment is the following alignment of $(takki,takista)$ in Finnish and Estonian, assumed original meaning ’to be attached to’.

```
#t  a  kk  i  $  
|   |   |   |   |   |
#t  a  k  i  suffix 'sta$'
```
'Denoising’ the data in this way, makes the learned rules of correspondence cleaner. Although affixes are far from random and do contain information, this is not the information we are interested in, information about phonetic correspondence. Therefore—in this context—we may regard them as noise. The gain in codensity obviously depends on the corpus, for the StarLing Uralic database it is much smaller than for SSA. It is interesting as such, which parts of a cognate will be ’stemmed away’ in this automated fashion.

4.4.9 Multilingual Alignment

Ultimately, we want to be able to reconstruct ancestor languages. To this end, we need to be able to align more than two languages simultaneously, three at the very least, two descendant languages plus their common ancestor. In principle, our models generalize to $d$-fold alignment—simultaneous alignment of $d$ languages. Events become $d$-tuples of symbols, one from each involved language, and the alignment matrix becomes $d$-dimensional. But there is a fundamental problem to this approach. While the event space grows exponentially in $d$, the number of cognate sets with entries in all $d$ languages decreases rapidly. In order to learn anything of use, we must use all correspondence data available, which means that many cognates will be missing from one or more languages involved.

But how do we treat missing data? And how do we prevent the counts from becoming much smaller, when the data spreads over a larger event space, providing deteriorating support to an increasing number of parameters?

Our approach is to define the cost function as the sum over each involved pair of languages,

$$l(D^d) = \sum_{s<t} l(D^2_{st}),$$

where $D^2_{st}$ is the two-dimensional, aligned data restricted to source language $\Lambda_s$ and target language $\Lambda_t$. Whenever either of the cognates $\sigma_i$ and $\tau_i$ is missing, the corresponding item $i$ is not part of $D^2_{st}$. This definition is valid for any of the aforementioned model classes and, as all of these are symmetric by nature, it does not depend on the ordering of the languages in $D^d$.

The alignment matrix does become $d$-dimensional, therefore growing exponentially with $d$. The number of cells we can transition from to arrive at a cell $a_{uv}$ (cf. Figure 4.4) is now (at most) $2^d - 1$ when aligning single symbols, and $3^d - 1$ when allowing two symbols to be aligned together.
The requirements for dynamic programming to produce the optimal \(d\)-dimensional alignment are still given, but the minimum in the analogue of Equation 4.3 is now being taken over a larger number of values.

Section 4.7 introduces phylogenetic language trees. As we choose them to be binary—each observed language forms a leaf, while each protolanguage has exactly two children—we always deal with three languages at a time. For this reason, it suffices for our purposes to set \(d = 3\), which ensures efficient computation.

The cost function (4.16) is, strictly speaking, not a codelength. It is a sum of interdependently constrained codelengths, since we demand the two-fold alignments to be compatible. To illustrate this, let us look at the following example.

\[
\begin{array}{ccc}
  v & uo & s & i \\
  | & | & | & | \\
  v & oo & s . \\
  | & | & | & | \\
  . & a & l \\
\end{array}
\]

is a three-fold alignment of the triplet \((vuosi, voos, al)\) ('year') in Finnish, Estonian and Khanty. Three pairwise alignments of the same cognates might read as follows.

\[
\begin{array}{ccc}
  v & uo & s & i \\
  | & | & | & | \\
  v & oo & s . \\
  | & | & | & | \\
  . & a & l \\
\end{array} \quad \begin{array}{ccc}
  v & uo & s & i \\
  | & | & | & | \\
  v & oo & s . \\
  | & | & | & | \\
  . & a & l \\
\end{array} \quad \begin{array}{ccc}
  v & oo & s \\
  | & | & | & | \\
  . & a & l \\
  | & | & | & | \\
  . & a & l \\
\end{array}
\]

But, as these three are of different length, they do not correspond to any three-fold alignment and are therefore incompatible. In fact, we have to slightly modify the terms of the sum in (4.16), and realow events of type \((. . .)\), simultaneous insertion and deletion. Incompatibility may also appear in more subtle situations.

We observe, that the pairwise codelengths involved in (4.16) are slightly larger than what we can achieve without the compatibility constraint. In this respect, three-fold (or more-fold) alignment differs from all other extensions to the baseline model we have introduced, all of which aim at decreasing codelength. Here, we simply build a tool for reconstruction. This tool—a collection of compatible models—utilizes all available data. Cognates present in only some of the languages still contribute their share to the regularities captured by this collection, indirectly influencing all pairwise distributions.
4.5 Featurewise Context Modeling

In Section 4.4 we have presented our baseline model and a series of freely combinable extensions to it. These models perform surprisingly well when evaluated against any of the techniques described in Section 4.4.3 above or Sections 4.6–4.7 below. They perform well, even though there are two important features of the data they cannot capture.

4.5.1 Larger Context

First of all, one-to-one alignment cannot find any context dependent rules. Two-to-two alignment only takes a very limited context into account, namely at most one of the neighbouring symbols. However, many etymological rules of correspondence condition on the context of a sound less locally. For instance, vowel harmony in Finnish demands that front vowels \{\¨a, \¨o, y\} may not mix with back vowels \{a, o, u\} in the same (uncompounded) word. This is a long distance phonological assimilation process and the models of Section 4.4 are unable to capture this type of rule.

The two-to-two correspondence model was a first (and successful) approach to take context into consideration, but it cannot easily be extended to larger context. In Publications V and VI, we develop a family of context-aware models, which take on a different structure from what we have seen so far. To our knowledge, the models described in the following are the first to ever capture longer range context in etymological data.

4.5.2 Phonetic Features

A second shortcoming of the models presented so far is that they operate on the symbol level, while phonetic rules typically operate only on some aspect—or feature—of a sound. As a result, we had to learn a number of rules, when in fact there was only one underlying phonetic process involved. For example, the diphthong-to-long-vowel rule in Finnish:Estonian had to be learned twice: (uo : oo) and (yö : öö). This results in excess codelength as well as requires more data (support) in order to be captured by a model.

In the following, we operate on the feature space, where each symbol is represented as a vector of phonetic features. Each symbol participating in an alignment has a Type feature, a value in \{K, V, \_, \#\} denoting consonant, vowel, absence by deletion/insertion and word boundary. All consonants and vowels then have 4 features, each with a varying number of possible values as listed in Table 4.1.

The question may arise, why can the models presented in Section 4.4 not
operate on features instead of symbols? This is due to the fact that they cannot incorporate context. All the context they can see is the symbol itself (plus at most one neighbouring symbol for the two-to-two model), but they can see all of its features, although not separately. Take away this information and we cannot expect them to learn very much at all.

But for the context-aware models described below the situation is different. They are able to look at the relevant aspects of both intra-symbol and inter-symbol context. Each model class consists of a set of decision trees, one for each feature, encoding which aspects of the context are relevant to a specific feature of articulation. These models once more encode a pair of languages, but can also be extended to three or more languages in the way we have described in Section 4.4.9.

There is a number of possible variants for the context tree model classes, we give an example of a set of choices below.

4.5.3 Context Trees

We encode the separate features, in some fixed order, first source then target language. The relevant context is given by the model class, which holds a tree for each feature and each language level, source and target. The decision nodes of these trees correspond to a query of the value of a relevant feature at some position of either the source or the target cognate. We call the triplet \((P, L, F)\) a context, where \(P\) is one of the positions that
the model may query, \( L \) is the level (either source or target) and \( F \) is one of the possible features.

We choose to encode the symbols (i.e., their features) in the order they come. We can only query symbols (and features of the present symbol) already encoded. Also, we are blind to past insertions and deletions. This restriction needs to be made to ensure we can optimally align using dynamic programming\(^2\) as described in Section 4.4.2. Further, we encode the features in a specific order, starting with the **Type** feature.

The positions we allow are **Self** (of which we can only query features already encoded), **Previous Symbol** (which is not a dot), **Previous Consonant**, **Previous Vowel**, as well as **Self or Previous Consonant** and **Self or Previous Vowel** (previous iff **Self** is not a consonant/vowel). The definition of the set of positions a tree can query is crucial. Our choice is not the only one that is possible, nor is it the only one we have tried. It simply seems to be reasonable and working well.

For the consonant positions we can only query consonant features, vowel features for vowel positions. Both return ‘\( n/a \)’ if no appropriate symbol can be found, i.e., we run into the word boundary first. For the undecided positions **Self** and **Previous Symbol** we can query **Type** and any other feature, which may then also return ‘\( n/a \)’, if the symbol found at that position and level is of inadequate type to the query.

Each decision node/query splits the tree into a number of subtrees equal to the number of possible answers. In order to keep the trees from growing too wide, we also allow binary queries of the form \((P, L, F, V)\), where \( V \) is a given value and the return value is either **true** or **false**.

Each feature \( F \) at level \( L \) of a symbol \( \sigma \) or \( \tau \) then takes a path through the decision nodes, starting from the root and arriving at the leaf node corresponding to its relevant context, as the feature tree defines it. Each leaf node contains a distribution over the values of \( F \), as a multinomial of cardinality \(|F|\), the number of values for that feature. This distribution is used to encode \( F(\sigma) \), respectively \( F(\tau) \). Figure 4.5 shows a context tree, encoding the context relevant to feature \( V \) (voicedness) in Estonian, conditioned on Finnish.

Each path from the root to a leaf of such tree encodes a context rule. For instance, following the leftmost branch from the root lets us arrive at a leaf immediately. The corresponding rule states that, if a Finnish consonant is voiced (Finnish Itself Voiced = \( \oplus \)), then so is its Estonian counterpart,

\(^2\)A more recent implementation of the algorithm does not require this restriction, as long as we are allowed to query the **Type** feature only for the present and immediately preceding symbol.
with high probability (615 observed instances against 2).

4.5.4 Codelength

We encode the model class—the trees—in a naive way, by simply listing for each node whether it is a leaf or not using one bit and, if it is not, which triplet $(P, L, F)$ or quadruplet $(P, L, F, V)$ is being queried. Encoding this decision costs the logarithm of the number of choices we can make here. This number depends on the feature to be encoded, and with our choice of contexts that can be queried amounts to a little more than eight bits on average. For a model class $C$, a collection of 18 trees (two levels times Type plus four consonant and four vowel features), the corresponding codelength $l(C)$ is then the number of total nodes plus—roughly—eight times the total number of decision nodes.

This encodes the model class. Encoding the aligned data $D$ given the model class can be done at the leaves of each tree. At every point in time during encoding—as well as during decoding—we are aware of the context already encoded. This context together with the model class (the structures of the trees) determines the leaf into which $F(\sigma)$ (or $F(\tau)$) falls. We can therefore encode the data at the leaves, one leaf at a time, and the decoder will know in which order to merge these feature events back together in
order to recover the original data. The total codelength is then

\[ l_{context}(D, C) = l(C) + \sum_{L} \sum_{F} \sum_{\ell} l_{NML}^{[F]}(D_{|L,F,\ell}). \] (4.17)

For the data part of the code, we sum over both levels \( L \), all phonetic features \( F \), and all leaves \( \ell \) of the corresponding context tree in \( C \). For each leaf we apply the NML code to the data \( D_{|L,F,\ell} \)—restricted to the current level, feature and leaf—as a multinomial of cardinality \( |F| \).

### 4.5.5 Learning

To learn the model, we once more start from a random alignment of all cognates. We then build the decision trees as described below, to arrive at an initial model. This we use to realign all cognates as we have done in Section 4.3. Not being able to query dots (deletions/insertions) in the decision nodes ensures, that at any cell in the alignment matrix of Figure 4.4 the path we took to arrive at this cell has no influence on future transition costs, the requirement needed to ensure optimality of the alignment found by means of dynamic programming. After realigning a single pair of cognates, we only update the counts at the leaves of all feature trees, not rebuild those trees, which would slow the algorithm too much. We only rebuild the feature trees after realignment of the complete corpus. We alternate between tree rebuilding and corpus realignment until convergence. Again, to avoid premature convergence to a poor, but locally optimal solution, we employ simulated annealing with a suitable cooling schedule.

We learn the trees—given a complete alignment of the corpus—by iteratively splitting the feature data according to the cost-optimal decision in a greedy fashion. For given level \( L \) and feature \( F \), we start out by storing all corresponding feature events \( e \) at the root node of the tree, e.g. for the voicedness feature \( V \) in Estonian (aligned to Finnish, cf. Figure 4.5) we store data with counts

<table>
<thead>
<tr>
<th></th>
<th>801</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>821</td>
</tr>
</tbody>
</table>

In this example, there are 1.622 occurrences of Estonian consonants in the data, 801 of which are voiced. The best split the algorithm found was on ‘(Source, Self, V)’, resulting in three leaves (or roots of new subtrees). The data now splits according to this context into three subsets with counts

<table>
<thead>
<tr>
<th></th>
<th>+</th>
<th>-</th>
<th>n/a</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>615</td>
<td>135</td>
<td>51</td>
</tr>
<tr>
<td>-</td>
<td>2</td>
<td>764</td>
<td>55</td>
</tr>
</tbody>
</table>
For each of these new nodes we split further, until no further drop in total codelength can be achieved. A split costs about eight plus the number of decision branches in bits, the achieved gain is the drop in NML cost obtained from splitting the data.

4.5.6 Evaluation

As the model classes have become much more potent in that they can take larger context into account, the codelength decreases considerably. As we still align on the symbol level—even though we encode each symbol by its feature vector—we may again plot the count matrix. When we did so, we noticed some strange behaviour. While for some runs on some data the corresponding count matrix looks sensible (sparse), for others it does not, far from it. Looking at the actual alignments reveals the problem: in the latter cases the algorithm finds shifted alignments, such as

\[
\begin{array}{ccccc}
  v & u & o & s & i \\
  | & | & | & | & | \\
  \cdot & v & o & o & s
\end{array}
\]

In these cases it does so consistently, which does not hurt in terms of codelength. For any symbol, the most relevant context information is obviously in the symbol which it correctly corresponds to. Encoding source first, then target given the source, the information about the according feature of the corresponding symbol is available to the target, but not to the source. Shifting the alignment to the left reverses this situation. Shifting to the right (as in the example above) makes some of the source future available to the target. Of course, shifting the available context window does not make additional information available in sum. Overall, we do not gain from shifting (on average), but it does no harm, either. However, shifted alignments make the models less readable, as we have to undo the shift in our heads before we can read off rules from the context trees. Moreover, as stated above, the count matrices no longer display the desired structure. Obviously, this behaviour is unintended.

In order to still obtain a sound-by-sound alignment, we have recently developed a post-processing method that has produced encouraging initial results. The rationale behind it is that the largest portion of the information relevant to some sound should be in the corresponding sound on the opposite level. We measure the information gain at each node along the root-to-leaf path for every sound and use dynamic programming to find, for each word pair, the alignment that maximizes the sum of information gains over all pairs of sounds involved in it.
We can now look at the count matrices and we find they do make sense. But they only show some aspects of the models, as they do not capture the context information as the models do. The context rules can be read off the feature trees themselves. Every path from the root of a tree that ends in a leaf with a distribution of low entropy provides us a rule. For instance, the rule that unvoiced consonants in Finnish remain unvoiced in Estonian word-initially, can be represented by the path (in the Voicedness-tree for Estonian, see Figure 4.5)

\[(Source, Self, V) \rightarrow (Target, Previous, Type) \rightarrow^\# \left( \begin{array}{c|c} + & 1 \\ \hline - & 396 \end{array} \right) \] (4.18)

where the leaf distribution has very low entropy, 396 out of 397 instances of word-initial Estonian consonants corresponding to an unvoiced consonant in Finnish are unvoiced. Leaf nodes with high entropy represent unexplained variation.

### 4.5.7 Exploiting Monolingual Rules

Checking these rules, we also notice that some of them are not of the type we had hoped to find. While many are non-trivial rules of etymological correspondence, others are much more bland, such as the following rule for Finnish place of articulation of consonants.

\[(Source, Self, M) \rightarrow^{trill} \begin{pmatrix} \text{bilabial} & 0 \\ \text{labiodental} & 0 \\ \text{dental} & 52 \\ \text{retroflex} & 0 \\ \text{velar} & 0 \\ \text{uvular} & 0 \end{pmatrix} \] (4.19)

This simply means that all Finnish trills are dentally articulated. Trivial, as the only trill in the Finnish alphabet is ‘r’. In fact, rules of this type can be encoded more compactly by a sound map, an example of which is given in Table 4.2.

<table>
<thead>
<tr>
<th>Manner→</th>
<th>Voicedness→</th>
<th>plosive</th>
<th>nasal</th>
<th>lateral</th>
<th>trill</th>
<th>fricative</th>
<th>sibilant</th>
<th>glide</th>
<th>affricate</th>
</tr>
</thead>
<tbody>
<tr>
<td>bilabial</td>
<td>+</td>
<td>m</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>labiodental</td>
<td>+</td>
<td>n</td>
<td>l</td>
<td>r</td>
<td>v</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>dental</td>
<td>+</td>
<td>n</td>
<td>l</td>
<td>r</td>
<td>v</td>
<td>s</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>retroflex</td>
<td>+</td>
<td>n</td>
<td>l</td>
<td>r</td>
<td>v</td>
<td>s</td>
<td>j</td>
<td></td>
<td></td>
</tr>
<tr>
<td>velar</td>
<td>+</td>
<td>n</td>
<td>l</td>
<td>r</td>
<td>v</td>
<td>s</td>
<td>j</td>
<td></td>
<td></td>
</tr>
<tr>
<td>uvular</td>
<td>+</td>
<td>n</td>
<td>l</td>
<td>r</td>
<td>v</td>
<td>s</td>
<td>j</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 4.2: A map of Finnish consonants.
The sound maps for the involved languages can safely be assumed to be known to the receiver of the encoded data. A first—yet to be implemented—approach to exploit this information in encoding is the following. When building a feature tree, simply erase all feature events from the data stored at the root node, which are completely specified using the previously encoded features of the same symbol together with the sound map. There is no additional information needed to decode these instances and hence we can save some bits here. At the same time, the feature trees should become smaller, better readable. Also, when encoding the data at a leaf, we can ignore impossible values. For example, there are no retroflexes in Finnish. When the cardinality of the multinomial feature variable can be decreased in this way, the regret shrinks and the code gets shorter. Unfortunately, not all situations are as clear cut. Some events will be specified only partially. How to deal with this type of situation is a subject of future research.

Another interesting direction for the future is to combine separate models for each involved language together with a pure correspondence model. Context models as described above can be just as easily built for a single language, simply by building the trees without any alignment. The allowed contexts to be queried are then restricted to a single level. In this way, we can find phonetic rules within the modeled language, e.g. for Type in Finnish the rule

\[(\text{Source, Previous, Type}) \xrightarrow{\text{consonant}} \]
\[\rightarrow (\text{Source, Previous Vowel, Type}) \xrightarrow{n/a} \begin{pmatrix} \text{vowel} & 1332 \\ \text{consonant} & 0 \\ \# & 0 \end{pmatrix} \] (4.20)

stating that word-initially, there cannot be two consecutive consonants. Taking this information out of the correspondence model would yield smaller trees, which are better interpretable and provide larger data support at each leaf.
4.6 Imputation

We have introduced a series of MDL methods, producing codes of decreasing length. Information-theoretically, it is interesting to see how the linguistic assumption of regularity of phonetic change helps us to compress the given data. We have also evaluated these models by the rules of etymological correspondence of sounds they are able to find. Finally, we have checked the alignments they produce by hand. As there exists no gold standard for the alignment problem—and we do not know how any such standard could be defined—we are left to our own devices in judging their quality. While we find that both the discovered rules and the alignments found by the models closely follow the length of the codes associated to them, this way of evaluating 'by hand’ remains somewhat unsatisfactory.

For this reason, we have developed a more intuitively appealing, cross-validation type procedure of evaluation we call imputation. For a given model and a language pair—e.g., Finnish/Estonian—we hold out one word pair at a time and train the model on the remaining data. Then we show the hidden Finnish word to the model and let it guess, or impute, the corresponding Estonian form. We then compute the Levenshtein edit distance [Levenshtein 1966] between the imputed word and the withheld Estonian word, the minimum number of basic editing operations needed to produce one from the other. We impute all words of the target language in this way, sum the edit distances, and normalize by the number of symbols in the correct target data. This gives us the Normalized Edit Distance (NED).

Table 1 of Publication V lists the (symmetrized) NED scores achieved by the context model on all language pairs in the StarLing Uralic database. Figure 6 of Publication VI compares the NED scores achieved by the context model to those achieved by the baseline model with codebook.

For the baseline model (and its extensions), we can use a dynamic programming procedure similar to that used for alignment to obtain an imputed word form which, when the true, erased word is replaced with it, results in the shortest overall codelength. The alignment matrix now simplifies to a vector of length equal to the length of the given word in the source language, plus two for the word boundaries. Allowed transitions are steps of size one (one or two for a two-to-two model) and the associated cost is given by the minimum cost over all events $e$ which involve the corresponding symbol(s) on the source side. The target symbol(s) involved in the transitions of the cost-optimal path form the imputed word. In the same way, we can also impute conditioned on cognates in multiple languages, in which case the dimension of the imputation matrix equals the number of given word forms.
Following the same scheme with a context model does not necessarily produce the codelength-optimal imputation. Since the imputed target word contributes to the context—and therefore a partial imputation can influence future transition costs—the optimality requirement of dynamic programming is violated. Our solution to this problem is the following. At any point $i$ during imputation, corresponding to having read off $\sigma_1, \ldots, \sigma_i$ from the given source word, we have stored the $B$ best partial imputations. We now read off the next symbol $\sigma_{i+1}$ in all $|T|$ possible ways by encoding $\tau \in T$, using the model and all $B$ stored contexts. Out of these $B \cdot |T|$ new solutions, we again keep the $B$ ones with minimal cost. Theoretically, this does not guarantee optimal imputation, but in practice, for only one cognate given, we find that $B = 100$ suffices. We see this by checking that, say, $B = 1000$ yields the same solutions. In this way we can efficiently impute with the context model, even if optimality is no longer guaranteed.

Codelength and NED correlate very well. This is an encouraging indicator for codelength minimization to be a good approach. Our methods do not optimize NED directly, but operate on the codelength instead. Yet they produce models of low NED, a simple and intuitive measure of a model’s quality.

Imputation will become a necessary tool in reconstruction of unobserved ancestor languages, which is the ultimate goal of our research. Even if we do not yet know how to automatically reconstruct protoforms (and nor does anyone else, to our knowledge), we feel that imputation provides a promising approach to this task, which we further discuss in Chapter 5.

The normalized edit distance also defines a measure of closeness between related languages, which we will use to construct phylogenetic language trees, as we describe in the following Section.
4.7 Phylogenetic Language Trees

Another way to verify the soundness of our approach is to let the models induce phylogenetic trees of the involved languages, representing the structure of their genetic relatedness. These can then be compared to trees found in the literature. Examples of such trees, for the languages of our focus, are shown in Figure 5 of Publication IV (adapted from [Anttila 1989]) and Figure 1 of Publication V (adapted from the Encyclopedia Britannica). Although the two agree in large parts, there are some differences. A large collection of phylogenetic trees for this language family—and others—can be found at [Multitree 2009], displaying even more different views on the subject. The evolution of the Uralic language family is subject to debate among linguists. So once again, there is no gold standard to compare our results to, but we intend to add a voice to the discussion.

Phylogenetic trees are typically binary, rooted and inferred from distance matrices. The observed languages serve as the leaves of such tree, the inner nodes are unobserved, common ancestor languages to the leaves of the subtree rooted in them. Binary trees suite our purposes well, as the maximum number of neighbours we need to consider in reconstruction is minimal for them. Several algorithms to learn binary trees from given pairwise distances are readily available, such as the Unweighted Pair Group Method with Arithmetic Mean (UPGMA, [Murtagh 1984]), NeighborJoin [Saitou 1987] and the Quartet Tree method [Cilibrasi 2006]. The latter two actually produce unrooted trees, but there is a standard way to root them. Adding a ‘garbage node’—one with distance to all other nodes greater than any inter-language distance—provides a pointer to the position of the root. There are also several ways to define the language distances used to infer a tree.

Normalized edit distance is one obvious candidate. As this measure is asymmetric, we need to symmetrize the distance matrix, e.g. by taking the arithmetic mean over the distances in both directions. Figure 7 of Publication VI shows a phylogenetic language tree induced by NeighborJoin from the symmetrized normalized edit distances obtained by the context model.

To the information theorist, a more natural way to define pairwise languages distances is via the codelength itself. Not only does it involve the (minus log-) probability of the correct target word (under the optimal alignment) instead of the Levenshtein distance to the imputed version. As the latter is discrete, it can be regarded less informative. Codelength also involves the complexity of the learned rules of correspondence between the two languages. The means we use to turn codelengths into a distance mea-
sure is the **Normalized Compression Distance** (NCD) [Cilibrasi 2007]. For Languages $S$ and $T$, with corresponding common cognates $D_S$ and $D_T$, it is defined as

$$NCD(D_S, D_T) = \frac{l(D_S : D_T) - \min\{l(D_S), l(D_T)\}}{\max\{l(D_S), l(D_T)\}}$$

(4.21)

where $l(D_S : D_T)$ is the codelength achieved by the model we are using, and $l(D_S)$ and $l(D_T)$ are the codelengths achieved by a monolingual version of the same model. All models we have considered here can easily be restricted to encode a single language instead of two or more. Figure 6 of Publication IV shows a tree that was built by UPGMA with the NCD matrix for the baseline model. Figure 4.6 of this thesis, in turn, depicts a tree induced by NeighborJoin based on the normalized compression distances calculated using the context model, representing the phylogenetic structure of the Turkic language family.

![Phylogenetic Language Tree](image)

**Figure 4.6:** Phylogenetic language tree for the Turkic language family, induced by NeighborJoin from NCDs based on the context model of Section 4.5.

The phylogenetic languages trees induced by our methods are strikingly similar to the ones that can be found in the literature. The baseline model still makes a clear mistake by misplacing (the correctly found pair) *Mansi* and *Khanty*. The context model, however, makes no obvious mistakes, the
resulting trees lie within the variation among expert opinions, with equality for some.

We find this to be strong evidence for the validity of our approach. After all, we have induced these trees from raw data alone, using no prior assumptions, no semantic knowledge and no linguistic expertise in training the models.
Chapter 5

Summary and Current Work

“Ready are you? What know you of ready?”
—Yoda, The Empire Strikes Back.

The starting point of this thesis has been Bayesian reasoning, in particular its application to the task of model class selection. We have identified a number of problems related to this approach, such as the questionable assumption about the existence of a data generating distribution. Even if such distribution exists, it will not usually be a member of a model class under consideration, which can be especially harmful in supervised learning tasks such as classification. We further have to define prior distributions, both over the model classes we want to choose from and the models within each class. It is hard to do so in an objective way and, even in cases where we do have prior knowledge about the problem domain, the formulation of a prior can prove to be very difficult.

An elegant way around these problems is being offered by information theory, which enables us to look at data with no assumptions regarding their source. Following the minimum description length (MDL) principle, we regard the problem of maximizing probability as one of minimizing codelength, that is, data compression. Compression then is purely data-driven. As with any approach, we must first to define a suitable model class or model family to choose a class from. Then we define a method to describe the data as compactly as possible, an encoding scheme that gives good compression rate. This description method can be seen as corresponding to a prior of some sort, but typically this choice is more intuitive and straight-forward than the formulation of a prior.

The theory of Kolmogorov complexity provides proof for the fact that there can be no automated procedure to optimally choose a model class.
Regardless of the approach we are taking, we have to define a model family by hand, and choose an appropriate class from that. The MDL model class selection criterion is compression rate, we choose the class that achieves the shortest description length for the data at hand. But we can never be sure whether the class found in this way is optimal, model classes outside the family under consideration may achieve better compression rates. On the other hand, MDL methods make no prior assumptions about the nature of the data, its generating distribution or the suitability of the chosen model family or any of its members.

The problem of selecting a model within a chosen class or, equivalently, parameter estimation has turned into the problem of defining an encoding scheme. Where efficiently computable, we have a generic choice available, namely the normalized maximum likelihood (NML) distribution. Compared to the best model in the class under consideration, it achieves minimal excess codelength in the worst-case over all possible data. Under the assumption that data come from a universal distribution, Kolmogorov theory proves that the NML distribution also yields the shortest code on average, at least up to an additive constant depending on the universal Turing machine (UTM) with respect to which the universal distribution is defined. This assumption, while it may be called into question as such, is justified by the fact that any universal distribution, again up to a constant factor, dominates all other distributions.

This makes the NML distribution a valuable tool in data modeling. Unfortunately, computing this distribution is often infeasible in practice. But this does not hold for a number of relatively simple model classes. Publication II investigates the boundaries of NML computability for a specific family of Bayesian network models.

In case we cannot efficiently compute the NML distribution, we can often still use it for subproblems. Approximations of various type have also been reported to yield good model class selection criteria in an increasing number of applications. But with or without NML, we can always compare different encoding schemes objectively by looking at the resulting codelengths. Comparability is among the greatest strengths of the MDL approach, as it enables us to compare models of utterly different type and parametric structure by their compression capability.

Based on these findings, in Publications III–VI, we develop and evaluate a series of model families for the alignment problem arising in etymological data analysis. This is a novel approach, as for the first time information-theoretic methods are being applied to this problem. We utilize the MDL principle and compress the raw data together with the alignment we are
searching. In order to do so, we have to find and exploit the regularities of phonetic correspondence in the given data. This enables us to not only find good alignments, but also directly read off these rules from the models. The rules we find are probabilistic by nature. Although the basic, linguistically motivated assumption behind the problem is that inter-language change of sounds is a regular process, the data we are using contain noise from a variety of sources. Therefore, we cannot expect to find rules to completely explain the data in a deterministic manner.

Rules of phonetic variation are typically context-dependent, where the context relevant to a rule may restrict to the immediate neighbourhood of the sound in concerns, but often also involves longer-distance dependencies. The context model introduced in Section 4.5 is able to capture many such dependencies. It builds feature-wise decision trees, which explicitly encode the context it sees relevant to a rule. An obvious direction of further research is to make an even larger context available to these trees. For instance, we can model vowel harmony in Finnish only to some degree. Since in our present implementation, we can only query information about the vowel immediately preceding each position, we cannot ‘see through’ vowels \{i, e\} behaving neutrally with respect to this rule. Phonetic rules may also condition on aspects on the syllable level, such as stress. At present, we are unable to capture these rules.

A problem we face in this research lies in evaluation of the obtained results. Neither does a gold standard exist for the alignment problem, nor are there obvious, objective ways to compare the performance of our algorithms to other methods. We develop several ways to give evidence for the validity of our approach and the quality of our results. From an information-theoretic perspective, the achieved compression rate measures the quality of any model. Using codelength as a criterion, we can compare our models among each other. But we have no outside milestone to compare them against. This calls for further, linguistically motivated evidence to indicate the quality of our results.

Checking phonetic rules discovered by the models by hand, or even inspecting the obtained alignments on the cognate level, is troublesome and hardly leads to any objective criterion for comparison. On the cognate level, we introduce an imputation procedure that supplies an intuitive quality measure: the normalized edit distance (NED) between the true and the model-imputed word forms. On the corpus level, we let our models induce phylogenetic trees of the involved languages, which we then compare to expert-built trees found in the literature. These trees can be constructed by various algorithms that typically take a matrix of pair-wise language
distances as input. We can produce such matrices using NED, but also directly from the code lengths. Normalized compression distance (NCD) provides the desired means here.

In all tests we have performed our models do well. On top of that, the code length—the MDL criterion we optimize during learning—correlates strongly with any of the other criteria we have used in evaluation. Figure 5.1 shows a schematic diagram of the evaluation methods we have used, the bottom level containing the evaluation nodes.

![Figure 5.1: Flow chart of the used evaluation methods.](image)

Another direction, as mentioned in Section 4.5.7, is to separate phonetic rules within each single language—including information about the alphabet the language is using—from the correspondence model. This should improve learning, as the models become simpler and better supported by the data. Also, this would improve model readability, giving a clearer picture of the involved phonetic processes. However, this requires an encoding scheme that can combine the rules encoded in the multiple models to ends of code length minimization. At present it is unclear how to achieve this.

A more immediate application of our methods is that of cognate evaluation. Many of the entries in the corpora are marked uncertain. And while our models cannot evaluate the semantic aspects involved in cognate discovery, they do define a measure of phonetic similarity. We can modify the code length-based distance measure NCD to apply on the cognate level, conditioned on the learned models of a larger data. For cognates $\sigma \in S$
and \( \tau \in T \) we define

\[
NCD(\sigma, \tau) = \frac{l((\sigma : \tau)|\mathcal{M}(\mathcal{D}_S : \mathcal{D}_T)) - \min\{l(\sigma|\mathcal{M}(\mathcal{D}_S)), l(\tau|\mathcal{M}(\mathcal{D}_T))\}}{\max\{l(\sigma|\mathcal{M}(\mathcal{D}_S)), l(\tau|\mathcal{M}(\mathcal{D}_T))\}},
\]

(5.1)

where \( \mathcal{M}(\mathcal{D}_S) \), \( \mathcal{M}(\mathcal{D}_T) \) and \( \mathcal{M}(\mathcal{D}_S : \mathcal{D}_T) \) are the mono- and bilingual models learned from the given corpus. Ranking the cognate pairs according to this score, we may then decide to remove some improbable cases to arrive at a cleaner data set. Furthermore, this measure of degree of phonetic correspondence—as the model sees it—will be of linguistic interest in itself.

Finally, as mentioned earlier, we would ultimately like to be able to reconstruct ancestor languages which have been lost and can no longer be observed. We are able to generate phylogenetic language trees, good guesses of the history of language separation. The leaf nodes of these trees represent the languages of our corpus, while the inner nodes are unobserved. Alternatingly building models of correspondence between neighbouring nodes and re-estimating the word forms of the unobserved languages by means of imputation, we hope to be able to reconstruct the history all the way up to—in our case—‘Ur-Uralic’.
5 Summary and Current Work
References


References


<table>
<thead>
<tr>
<th>Year</th>
<th>Author(s)</th>
<th>Title</th>
<th>Publisher/Details</th>
</tr>
</thead>
</table>


References


Summary and Contributions

This doctoral dissertation is based on the following six publications, referred to as Publications I-VI. These are reprinted in the following. In all six papers, I am the first author, which reflects my significant contribution to each of them. It follows a listing with brief summary and contribution details.

Publication I

H.Wettig, P. Grünwald, T.Roos, P. Myllymäki, H.Tirri:
When Discriminative Learning of Bayesian Network Parameters Is Easy
Pp. 491-496 in Proceedings of the 18th International Joint Conference on
Artificial Intelligence, edited by G.Gottlob and T.Walsh. Morgan Kauf-

We investigate the relation between Bayesian network classifiers and logistic regression. We find that, in many cases that we explicitly identify, the two are equivalent in the sense that they encode the exact same set of conditional distributions. We further argue that the latter is to be preferred, as it instantiates its model parameters in a discriminative fashion. Empirical results backing up our claim did not fit into this (six page) publication, but are included in Chapter 2 of this dissertation.

Idea and implementation are of my signature. Theorems 3 and 4 are due to Teemu Roos and Peter Grünwald. Theorem 4 appearing in Section 2.6, a stronger result than that of Theorem 4 of the paper, is my work.
Publication II

H. Wettig, P. Kontkanen and P. Myllymäki:
Calculating the Normalized Maximum Likelihood Distribution for Bayesian Forests
Also published in:

This publication presents an algorithm to compute the normalized maximum likelihood (NML) distribution for tree-structured Bayesian Networks in polynomial time. The degree of this polynomial depends on the cardinality of the multinomials involved, which is unfortunate. There is, however, strong evidence for the inevitability of this, both in this publication and in [Mononen 2008] which—a year later—took an entirely different approach to the same problem.

The problem was suggested to me by Petri Myllymäki and Petri Kontkanen. Realization and implementation are of my doing.

Publication III-VI

H. Wettig and R. Yangarber:
Probabilistic Models for Aligning Etymological Data

H. Wettig, S. Hiltunen and R. Yangarber:
MDL-based Models for Aligning Etymological Data

H. Wettig, K. Reshetnikov and R. Yangarber:
Using context and phonetic features in models of etymological sound change
These four publications report a continuance of ongoing research. Naturally, there is considerable overlap to their content, as each subsequent publication builds up on the preceding ones. We introduce a series of models for the alignment of kindred words within a family of natural languages. We optimize MDL-based codelength criteria, which do not utilize any language-specific prior knowledge, resulting in utterly generic methods. The corresponding encoding schemes range from context-unaware modeling on the symbol level to featurewise modeling using context trees. We present our results and find that it is difficult to objectively evaluate them, as there is no golden standard alignment available. In fact, any attempt to define such standard would inevitably result in fierce debate among linguists. Therefore, we take on a number of approaches to verify the relevance of our findings. Of course—from the MDL perspective—short data encoding is a quality measure in a right of its own. We show that indeed, codelength correlates well with other, linguistically intuitive scores.

The alignment problem was put before me by Roman Yangarber, who also provided the linguistic expertise. All encoding schemes and algorithms are of my design. The research software has been implemented by Suvi Hiltunen and Javad Nouri, and some of the data pre-processing has been taken care of by Kirill Reshetnikov.