The UpWrite Predictor: A General Grammatical Inference Engine for Symbolic Time Series with Applications in Natural Language Acquisition and Data Compression

Jason Lloyd Hutchens, B.E. (Hons.)

This thesis is presented for the degree of
Doctor of Philosophy of
The University of Western Australia

Centre for Intelligent Information Processing Systems
Department of Electrical and Electronic Engineering
The University of Western Australia
Nedlands, WA
AUSTRALIA 6907

December 1999

Prologue

With the unreasonable petulance of mankind I rang the bell and gave a curt intimation that I was ready. Then I picked up a magazine from the table and attempted to while away the time with it, while my companion munched silently at his toast. One of the articles had a pencil mark at the heading, and I naturally began to run my eye through it. Its somewhat ambitious title was "The Book of Life," and it attempted to show how much an observant man might learn by an accurate and systematic examination of all that came in his way. It struck me as being a remarkable mixture of shrewdness and of absurdity. The reasoning was close and intense, but the deductions appeared to me to be far fetched and exaggerated.

The writer claimed by a momentary expression, a twitch of a muscle or a glance of an eye, to fathom a man's inmost thoughts. Deceit, according to him, was an impossibility in the case of one trained to observation and analysis. His conclusions were as infallible as so many propositions of Euclid. So startling would his results appear to the uninitiated that until they learned the processes by which he had arrived at them they might well consider him as a necromancer.

"From a drop of water," said the writer, "a logician could infer the possibility of an Atlantic or a Niagara without having seen or heard of one or the other. So all life is a great chain, the nature of which is known whenever we are shown a single link of it. Like all other arts, the Science of Deduction and Analysis is one which can only be acquired by long and patient study, nor is life long enough to allow any mortal to attain the highest possible perfection in it. Before turning to those moral and mental aspects of the matter which present the greatest difficulties, let the inquirer begin by mastering more elementary problems. Let him, on meeting a fellow-mortal, learn at a glance to distinguish the history of the man, and the trade or profession to which he belongs. Puerile as such an exercise may seem, it sharpens the faculties of observation, and teaches one where to look and what to look for. By a man's finger-nails, by his coat-sleeve, by his boots, by his trouser-knees, by the callosities of his forefinger and thumb, by his expression, by his shirtcuffs—by each of these things a man's calling is plainly revealed. That all united should fail to enlighten the competent inquirer in any case is almost inconceivable."

"What ineffable twaddle!" I cried, slapping the magazine down on the table; "I never read such rubbish in my life."

"What is it?" asked Sherlock Holmes.

"Why, this article," I said, pointing at it with my eggspoon as I sat down to my breakfast. "I see that you have read it since you have marked it. I don't deny that it is smartly written. It irritates me, though. It is evidently the theory of some armchair lounger who evolves all these neat little paradoxes in the seclusion of his own study. It is not practical. I should like to see him clapped down in a third-class carriage on the Underground, and asked to give the trades of all his fellow-travellers. I would lay a thousand to one against him."

"You would lose your money," Holmes remarked calmly. "As for the article, I wrote it myself."

"You!"

"Yes; I have a turn both for observation and for deduction. The theories which I have expressed there, and which appear to you to be so chimerical, are really extremely practical - so practical that I depend upon them for my bread and cheese."

"And how?" I asked involuntarily.

"Well, I have a trade of my own. I suppose I am the only one in the world. I'm a consulting detective, if you can understand what that is. Here in London we have lots of government detectives and lots of private ones. When these fellows are at fault, they come to me, and I manage to put them on the right scent. They lay all the evidence before me, and I am generally able, by the help of my knowledge of the history of crime, to set them straight. There is a strong family resemblance about misdeeds, and if you have all the details of a thousand at your finger ends, it is odd if you can't unravel the thousand and first."

A Study in Scarlet
SIR ARTHUR CONAN DOYLE

Abstract

"Come, Watson, come!" he cried. "The game is afoot."

The Return of Sherlock Holmes
SIR ARTHUR CONAN DOYLE

This dissertation is concerned with the development of a general Grammatical Inference Engine for symbolic time series—a device which is capable of automatically constructing a predictive model from arbitrary symbolic data. We are particularly interested in the situation where the data is natural language text, and would like to draw conclusions about language acquisition in human beings from this work, but we deliberately avoid making any language-specific assumptions in the design of the Grammatical Inference Engine, lest they adversely affect its generality.

This work stems from pattern recognition, something at which the human brain is particularly adept. We use the UpWrite, a modelling framework initially developed for the task of image recognition via syntactic techniques, to bootstrap a simple predictive model with higher level structure. Fairly general information processing techniques are used to discover this higher level structure from the sequence of predictions made about the data by the predictive model. We refer to the bootstrapped predictive model as the UpWrite Predictor.

Our implementation of the UpWrite Predictor uses the novel techniques of agglutination and agglomeration to iteratively construct symbol sequences and symbol classes which constitute higher level structure in the data, with the result that words and syntactic categories are successfully discovered in natural language text. The fact that simple information theoretic measures may be used to uncover structure of this sort from the sequence of predictions made by a simple predictive model, with minimal assumptions made about the underlying data, lends strength to the argument that at least some aspects of human language acquisition may be explained without resorting to the notion of universal grammar.

The UpWrite Predictor may potentially be of use in many applications, and we explore one of these, data compression, in this dissertation. We show that the performance of the standard PPMC data compressor may be improved by using the UpWrite Predictor to abstract the data prior to compression taking place. During our exploration of the data compression problem, we introduce several modifications and additions to traditional

PPM techniques, including a wildcard language model, which makes use of additional equivalence classes which traditional Markov models discard, and a goal-oriented language model, which is able to constrain the predictions it makes based upon knowledge of future events, and is therefore able to capture long-distance dependencies in natural language text.

Contents

List of Figures

List of Tables

Acknowledgements

> "Thank you!" said Holmes. "Thank you!" and as he turned away, it seemed to me that he was more nearly moved by the softer human emotions than I had ever seen him.
>
> *The Return of Sherlock Holmes*
> SIR ARTHUR CONAN DOYLE

Completing postgraduate studies is a frightening prospect. I have constantly found myself rebounding between two opposing beliefs: that I have not done sufficient work to warrant writing a dissertation, and that I have covered so much ground in so many diverse areas that I would have a lot of trouble deciding what to put in and what to leave out. Both of these concerns have proved true to varying degrees, and both are a consequence of my propensity to procrastinate, remarkable even by postgraduate standards. The only solution was to sit down and write, and see what came of it. I sincerely hope that the results speak for themselves.

I would like to thank all of the members of CIIPS, the Centre for Intelligent Information Processing Systems at The University of Western Australia. Housed within the Department of Electrical and Electronic Engineering, CIIPS has been my home since 1992, when, along with fellow second-year undergraduate students Danny Goodman and Bruce Cooper, I talked Professor Yianni Attikiouzel, director of CIIPS, into letting us perform "virtual reality research". This resulted in a poor-man's VR system, consisting of a bicycle crash helmet, a pair of monochromatic viewfinders salvaged from broken video cameras, two Amiga computers (one for each eye, synchronised via a serial link) and a bunch of potentiometers for measuring the position and orientation of the victim's head. Seven years later, I realise that the opportunity given to us by Yianni on that first fateful day has enabled me to touch upon a lot of exciting areas, and to develop ideas which I otherwise would not have had the chance to. It has also meant that this dissertation has been a long time in the making, and the fact that I've finally finished writing is due in no small part to a pep-talk given to me by Yianni at just the right time. Thanks for everything, Yianni.

Dr. Michael Alder, of the Department of Mathematics at UWA, has been my supervisor since 1994. I am incredibly indebted to him for his advice, and for affording me the freedom to walk my own path. No doubt I have often proved a disappointing apprentice, and this tome won't necessarily alter his opinion on that matter, but I do not think I'd have made it this far if it hadn't been for Mike. I won't beat around the bush—Mike, your presence has been an inspiration, and is one thing that has made this little endeavor worthwhile.

My thanks go to other members of CIIPS, especially our numerous administrative staff over the years, not the least of all Violetta Cetrullo and Brenda Churchill, who have provided lots of help to fledgling thesis writers such as myself. Chris deSilva has often given me food for thought, has expressed enthusiasm for my work, and has kindly proof-read many draft chapters. I would also like to thank Roberto Togneri, Gareth Lee, Poh Lian Choong, Sok Gek Lim, Tony Zaknich, Ramachandran Chandrasekhar and Keith Godfrey for the interesting comments they've made about my work over the years, and for all the help they've given me.

My fellow postgraduate students have provided help, support, and, most welcome of all, frequent distraction. I would like to thank Paul (Wil) Williams, Bruce (Brewski) Cooper, Sonny Tham, Daniel Harvey, Robert (Ram) McLaughlin, Gareth Cook, Phil Dunstan and Raymond Low for being great friends who have always been around to chat to, and to bounce ideas (and other objects) off. Patrick Hew was a great help in his role as an unsolicited proof-reader, something for which I'm profoundly grateful, and I would also like to thank the numerous postgraduate and undergraduate students who have passed through CIIPS on their way to bigger and better things. I fondly remember you all.

I would like to thank my parents, who have been kind enough to provide me with a home, and who have given me their love and understanding, even though they have not really understood what it is I've been doing locked away in my bedroom for twelve hours each day for the past four months. Last, but not least, I wish to express my extreme gratitude to my beloved D., who has provided me with much-needed support during the difficult times, and who has given me the greatest reason of all for completing my studies.

JASON HUTCHENS
Perth, Western Australia,
December 1999

Chapter 1

Introduction

> The past and the present are within the field of my inquiry, but what a man may do in the future is a hard question to answer.
>
> *The Hound of the Baskervilles*
> Sir Arthur Conan Doyle

1.1 Introduction

The human brain is particularly adept at pattern recognition. It is capable of recognising the faces of friends and family, of distinguishing a quiet spoken word against louder ambient noise, of detecting and acting upon cause-effect relationships in the environment, and of perceiving and understanding language, both spoken and written. On occasion the human brain overextends itself; detecting structure where none exists, a situation familiar to those who have found themselves bedridden with a bad dose of influenza, and who have stared, in a delirious state of mind, at the grain of a wooden wardrobe. The brain is constantly looking for patterns, and the poor patient may have fleeting glimpses of something animal moving beneath the whorls of the timber.[1]

This propensity for detecting structure in chaos no doubt exists because doing so is useful for the survival of the organism, and the process of evolution tends to favour such mechanisms. Neanderthal statisticians who refused to jump up the nearest tree upon seeing a lion for the first time, having only seen tigers in the past, unfortunately did not live long enough to ensure the survival of their DNA. The rest of us have been left with the ability to chunk: to summarise our knowledge of the world at any particular instant, to generalise based upon past experience, and to make inferences about what is likely to happen in the future.

The evolution of language is a recent phenomenon, geologically speaking, and it seems likely that its very existence is a product of the general information processing abilities of the human brain. Our thesis rests on this belief—that simple computational pattern recognition, which makes no assumptions about the data being processed, may be used to unlock some of the rich structure which underlies natural language.

1.2 Motivation

Models of natural language are used in a variety of applications, and interest in such models is growing day by day, as Information Technology becomes more and more pervasive. Applications of models of natural language abound, and range from language understanding, text generation and machine translation to speech recognition, data compression and author identification. Language models are typically crafted with a particular application in mind, and it is easy to fall into the trap of attempting to achieve slight advances in performance by tweaking model parameters, or developing new algorithms to do the tweaking on one's behalf.

We believe that real advances in the study of natural language modelling will not come until we focus on the development of a general language acquisition device which is applicable to all of the standard problems, but which does not make any assumptions about the data it is likely to see. Such an approach would enable us to explore the limits of our language model prior to the introduction of application-specific constraints, and, more importantly, would have the potential to shed some light on the human language acquisition process.

1.3 The Problem

We are attempting to develop a general Grammatical Inference Engine—an algorithm which is capable of automatically constructing a model for some data. To make the problem somewhat easier, we restrict ourselves to the case where the data being modelled is a symbolic time series. Ideally, the Grammatical Inference Engine will find structure in the data automatically, and will use this structure to create a hierarchical model which is able to describe the data on a global scale. Although we are interested in natural language applications, we acknowledge that this work must begin with very simple data sources in the hope of generalising to more complicated ones once a suitable framework has been developed.

We will show that very simple predictive models, which make predictions about the next symbol in the data based upon a local context of symbols, are all that is required to build such a system. Information theoretic measures may be used to find structure in symbolic time series data from nothing more than the sequence of predictions made by such models, and this structure can then be used used to bootstrap the model, in a process known as the UpWrite.

The UpWrite enables us to design a system which begins with a very simple local model of the data and controllably grows this model in order to describe the data on a global scale, allowing the model to make generalisations about data seen in the past in order to better predict the future. The result is a hierarchy of models which describe the data on all levels, allowing applications to choose the granularity of description that they require.

We aim to show that the UpWrite Predictor, the predictive model which results from the application of the UpWrite to a low order Markov model, is able to make better predictions than more traditional predictive models due to its ability to incorporate higher level structure extracted from the data. One application which we focus on is that of adaptive statistical data compression, a stimulating and challenging field which puts tight constraints upon the predictive model, and which nicely illustrates the communication problem. Our intention, however, is that the UpWrite Predictor should be capable of being used in a myriad of applications.

It would be unreasonable to deny that the statistical regularities of natural language are not used, in part, to aid the language acquisition process in human beings. If structure is there to be found, it is almost certain that the process of evolution would have favoured brains which are capable of extracting this structure and using it to aid the language acquisition process. It is our belief that the human brain is a particularly good predictive model, and that fairly generic information processing techniques are used in the human brain to fairly specific ends.

1.4 The Sherlock Corpus

The *Sherlock corpus* is used throughout this dissertation when presenting the results of experiments which require a reasonably sized collection of natural language text. It was formed by downloading all of the Sherlock Holmes stories, by Sir Arthur Conan Doyle, from the Oxford Text Archive, which is located on the World Wide Web at `http://www.ota.ox.ac.uk/`, stripping them of all SGML processing directives and newline characters, and concatenating the resulting files together. The Sherlock corpus is a 3536088 byte file which contains 666696 words from a vocabulary of 21005 words. It contains a small percentage of errors due to the optical character recognition process which was used to construct the electronic versions of the Sherlock Holmes stories stored in the Oxford Text Archive, and no attempt was made by us to eliminate these errors. The version of the corpus used in this dissertation may be downloaded from the World Wide Web by following the appropriate link from the Web site of this dissertation [1].

Various sections of the Sherlock corpus are used when performing experiments, and these were formed via Unix commands, as summarised below.

`Test`: This is a 1147 byte file which is used primarily for the evaluation of various modelling techniques, and is shown in its entirety in figure 1.1. It is formed by the command `head -c 1147 Sherlock > Test`.

`Train`: This is most of the Sherlock Corpus, with the `Test` section removed. It is formed by the command `tail -c 3534940 Sherlock > Train`.

`Small`: This is the first 100000 bytes of the `Train` section of the Sherlock corpus, and is used primarily when we need to train predictive models quickly. It is formed by the command `head -c 100000 Train > Small`.

`Sentence`: This is the first sentence of the Sherlock corpus, "To Sherlock Holmes she is always the woman." Many plots and diagrams are given over this sentence. It is formed by the command `head -c 44 Sherlock > Sentence`.

> To Sherlock Holmes she is always the woman. I have seldom heard him mention her under any other name. In his eyes she eclipses and predominates the whole of her sex. It was not that he felt any emotion akin to love for Irene Adler. All emotions, and that one particularly, were abhorrent to his cold, precise but admirably balanced mind. He was, I take it, the most perfect reasoning and observing machine that the world has seen, but as a lover he would have placed himself in a false position. He never spoke of the softer passions, save with a gibe and a sneer. They were admirable things for the observer - excellent for drawing the veil from men's motives and actions. But for the trained reasoner to admit such intrusions into his own delicate and finely adjusted temperament was to introduce a distracting factor which might throw a doubt upon all his mental results. Grit in a sensitive instrument, or a crack in one of his own high-power lenses, would not be more disturbing than a strong emotion in a nature such as his. And yet there was but one woman to him, and that woman was the late Irene Adler, of dubious and questionable memory.

FIGURE 1.1: The `Test` section of the Sherlock Corpus in its entirety.

We are interested in the problem of general Grammatical Inference, and for this reason we occasionally perform experiments using a version of the Sherlock Corpus which has all whitespace and punctuation removed, and in which all characters are converted to their uppercase equivalents. This makes segmentation of the data into words a non-trivial task, and this is something that we hope the UpWrite Predictor can accomplish automatically. The various sections of the *SHERLOCK corpus* may be formed from the corresponding sections of the Sherlock corpus by using the sequence of piped Unix commands `cat Section | tr -d -c '[:alnum:]' | tr '[:lower:]' '[:upper:]' > SECTION`.

We have been using the Sherlock corpus for many years, and originally selected it because it was the largest freely-available collection of English text we could find. We think it is appropriate that Sherlock Holmes himself was a master of inference and deduction, and was greatly interested in ciphers and secret codes. We pay homage to Sir Arthur Conan Doyle's creation by quoting extracts from his stories in the epigraphs at the beginning of each chapter in this dissertation, and more extensively in the prologue and epilogue.

1.5 Original Contributions

Some of the original contributions made during the course of our research are

- application of the UpWrite technique to the modelling of symbolic time series, and the subsequent development of the UpWrite Predictor, a novel predictive model;
- extending and developing various algorithms for discovering symbol sequences and symbol classes in arbitrary data;
- an important insight into the problem of syntactic category acquisition which may potentially result in a successful algorithm for classifying natural language words into classes;
- application of the UpWrite Predictor to the problem of adaptive statistical data compression;
- many novel modifications of and additions to the various techniques used in traditional adaptive statistical data compressors, including new escape mechanisms, alternative equivalence classifications and a more theoretically sound exclusion process; and
- the development of a goal-oriented language model which is capable incorporating information about future events, and therefore may be used to generate data which exhibits long-distance dependencies by "filling in the blanks" in a template.

1.6 Structure of the Dissertation

The next three chapters of this dissertation serve to provide an introduction to Information Theory, the inference of predictive models, and the UpWrite. We begin in chapter 2 by giving an historical overview of Information Theory, and introducing the various information theoretic measures which we shall be using when we develop the UpWrite Predictor. This is followed in chapter 3 by a description of the grammatical inference problem for predictive models, a discussion of the inference of Markov models, and the presentation of the techniques of smoothing and back-off which are used in the n-gram language models of speech recognition systems to address some of the problems associated with the basic Markov model. The UpWrite is then introduced in chapter 4, using a simple image recognition problem to illustrate its power.

These three introductory chapters provide us with the tools necessary for the development of the UpWrite Predictor in chapter 5. In this chapter we show how information theoretic measures may be applied to the sequence of predictions made by a simple predictive model to find two types of structure in data, sub-objects and quotient-objects, which in natural language text may correspond to sequences of characters which form words and classes of words which form syntactic categories. We give an historical account of work performed in the automatic extraction of this kind of structure from data, and we introduce a few novel techniques of our own. We then show how the UpWrite may be used to incorporate the structure found by these techniques into the predictive model in a process of abstraction which results in a hierarchical description of the data, and which serves to improve the performance of the predictive model on novel data.

The performance of the UpWrite Predictor is evaluated in chapter 6, where the results of a series of experiments performed on a variety of artificial corpora generated by simple phrase-structure grammars are used to assess the ability of the UpWrite Predictor to find the structure inherent in the data. We also look at the performance of the UpWrite Predictor at modelling natural language text, and we give evidence of its ability to find words and syntactic categories by using it generatively and inspecting the resulting output.

Data compression is introduced in chapter 7, and we give a thorough overview of the modelling technique of Prediction by Partial Matching, or PPM, which is pervasive in this domain. We introduce some novel modifications of and additions to the standard methods used in PPM, including a goal-oriented language model which is capable of using knowledge about future events in order to increase its scope, enabling it to capture long-distance dependencies in data, before showing how the UpWrite Predictor may be successfully applied to the data compression problem by abstracting the data prior to compression taking place.

In the final chapter of this dissertation, chapter 8, we summarise the material presented, discuss our results, draw conclusions from them, and speculate on possible future work.

Notes

[1] The author may confirm via anecdotal evidence that this is indeed the case, and recommends the reader verify this with his or her own experiments.

References

[1] Jason Hutchens' PhD web site. Available at:
`http://ciips.ee.uwa.edu.au/~hutch/phd/`

Chapter 2

Information Theory

> A confederate who foresees your conclusions and course of action is always dangerous, but one to whom each development comes as a perpetual surprise, and to whom the future is always a closed book, is indeed an ideal helpmate.
>
> *The Casebook of Sherlock Holmes*
> SIR ARTHUR CONAN DOYLE

2.1 Introduction

Information Theory is a relatively new science which emerged from Claude Shannon's pioneering work on communication systems in the 1940's. The primary goal of Information Theory is the quantification of information, something which enables the design and analysis of efficient communication systems. The fundamental unit of information is the *bit*, which represents the amount of information required to distinguish between two equally likely events, and which should be familiar to anyone experienced with computers.

A quantitative information measure can be derived by likening information processing to an act of prediction, whereby a model makes a prediction about which symbol is likely to occur next in the data being processed. We refer to such a model as a *predictor*, or a *predictive model*. The *information* supplied to the predictive model by the symbol which actually does occur next is a function of the probability which the predictive model assigned to that symbol in the course of making its prediction. Information is a property which is supplied to the predictive model by the data, and the amount of information received by the predictive model is dependent on the model itself. For this reason we find it intuitive to refer to the information supplied to the predictive model by the next symbol in the data as the *surprise* experienced by the predictive model upon finding out what the symbol actually is.

A second information theoretic measure introduced by Shannon is that of *entropy*, which is a property of the predictive model itself. The entropy of a predictive model is the expected value, with respect to the model, of the information which will be supplied to the model by the next symbol in the data. The entropy of a predictive model is maximum when the model assigns an equal probability of occurrence to all symbols, and minimum when the predictive model assigns a probability of 1 to a particular symbol, and a probability of 0 to the remaining symbols. For this reason we find it intuitive to refer to the entropy of a predictive model as its *uncertainty*.

In the present work we are primarily concerned with modelling an information processing device based on our beliefs of how the human brain functions. Information Theory provides the mathematical framework necessary for pursuing such a line of research. It gives us a language for talking about such models, for comparing them to one another, and for extracting structure from the data they process.

2.1.1 What is Information?

Intuitively we understand information to be that intangible something which arises when knowledge is applied to data. A meteorologist, for example, may use her knowledge of meteorological phenomena to make a forecast of tomorrow's minimum and maximum temperatures based upon measurements taken from anemometers, rain gauges, thermometers and suchlike. The application of the meteorologist's knowledge to the observed data has resulted in information, and this information is relative to the meteorologist herself—another person may have regarded the measurements from the weather station as nothing more than a meaningless sequence of numbers.

Knowledge is something which can be both innate and attained by experience. When we construct a model, innate knowledge is specified in the form of the assumptions that we make. As the model is applied to an information processing problem, it may adapt to the data which it observes, modifying its knowledge in the process, and thus 'learning' from its experience.

It is often confusing to use terms like information, knowledge and experience when speaking about a model; it smacks of anthropomorphisation, and it may cause one to draw incorrect conclusions, and to suggest fallacious theories, merely on the strength of the loaded meanings inherent in the terminology. We cannot avoid using the terminology of Information Theory—doing so would require us to devise an obscure terminology of our own. Instead, we stress that Information Theory itself is not concerned with semantics or meaning in any way.[2]

2.1.2 Why is Information Important?

Information, in the technical sense of the word, is important primarily because it provides a measure of how 'interesting' a piece of data is. Data which has high information with respect to the predictive model is data which should be investigated—it may be fed back into the model to allow it to alter its knowledge, and the importance which it places on future events.

The information measure is directly related to the minimum number of bits required to unambiguously represent the data with respect to the predictions made by the predictive model, and this enables the data to be compressed for efficient storage and transmission; one real-world application of Information Theory which we shall meet in chapter 7.

2.1.3 Overview

We begin this chapter with an historical account of Information Theory, and then proceed to discuss the role of a predictor in a communication system. We then rigorously define the notation and information theoretic measures which we shall be using throughout this thesis, and we conclude by discussing some of the more esoteric philosophical issues raised by a theory of information.

2.2 Shannon's Mathematical Theory of Communication

In 1941 Claude Shannon, now regarded as the father of Information Theory, was studying various communications problems at the Bell Telephone Laboratories in Princeton, New Jersey. Motivated in part by the war effort, his research culminated in 1948 with the publication of the seminal work "A Mathematical Theory of Communication"; an event which marked the birth of a new science [12].

Shannon's work was preceded by that of R.V.L. Hartley, who, twenty years earlier, had recognized some essential aspects of the information measure Shannon would later develop rigorously [6].[3] Hartley realized that observation of a symbol's value generates information only if that value is one of several that the symbol could have taken on—that is, if the symbol is the value of a random variable. He wrote that [6]

> ... in estimating the capacity of the physical system to transmit information we should ignore the question of interpretation, make each selection perfectly arbitrary, and base our result on the possibility of the receiver's distinguishing the result of selecting any one symbol from that of selecting any other. By this means the psychological factors and their variations are eliminated and it becomes possible to set up a definite quantitative measure of information based on physical considerations alone.

Hartley was searching for a method which would allow the capacity of various systems to carry information to be compared. He proposed a quantitative measure of information by considering a message composed of z discrete random variables, each of which has D possible values. Intuitively, the information conveyed by these symbols should be z times the information conveyed by a single symbol, yet the message itself has D^z possible values. This suggests the logarithmic function as an appropriate information measure, in which case the information content of the message may be expressed as $z \log D$.

Definition 2.1. *The* Hartley Information *of a discrete random variable X is therefore* $\log D$*, where D is the number of possible values of X.*

Today it is common to calculate logarithms to base 2 when measuring information, but it is often forgotten that the selection of base is entirely arbitrary, and does nothing more than specify the units that the information content will be expressed in. When calculated to base 2, the information is expressed in *bits*.[4] Throughout this dissertation all logarithms will be calculated to base 2 unless otherwise stated.

The Hartley Information provides an answer to many simple technical problems. For example, a telephone switching system that is to service 8 customers requires $\log_2 8{=}3$ bits of information to uniquely identify each customer. The fundamental problem with Hartley's measure, however, is that it neglects to take the relative frequency of events into account, meaning that rare events are considered to supply the same amount of information as frequent events.

Shannon was functioning as a communications engineer, studying the problem of noise in communication systems, when he established a theory which allowed the construction of codes which were tolerant to noisy transmission systems. He considered a communications system to consist of six components, illustrated in the block diagram of figure 2.1. These six components are

- an *information source*, which produces the message which is to be communicated;
- a *transmitter*, which transforms the message into a signal fit for transmission;
- a *channel*, which is the medium over which the transformed messages are transmitted;
- a *noise source*, which perturbs the signal during transmission;
- a *receiver*, which performs the inverse operation of the transmitter; and
- a *destination*, which is the intended recipient of the message.

This formulation of a communication system is very general; it applies equally well to a conversation between two human beings, the transmission of television signals and the storage of data on magnetic media for retrieval at some later date.

Like Hartley before him, Shannon was searching for a measure of information which would enable him to compare the performance of two communication systems. Shannon wanted a measure which would give, in some sense, an indication of how much information is produced by the information source; a measure which would indicate the degree of choice the information source has when selecting an event to generate, and the degree of uncertainty an observer has about the outcome. He began by assuming that the information source is stochastic, so that each message it is capable of producing has an

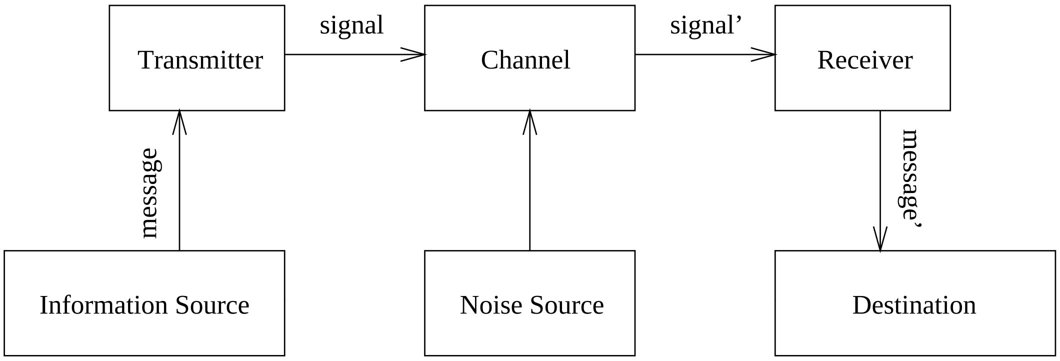

FIGURE 2.1: Shannon's model of a communication system.

associated probability of occurrence. The important insight made by Shannon was that an information measure should be a function of the probability of the message.

Hartley had previously stated that the logarithmic function is the most natural choice for calculating the information content of the message, and Shannon derived this rigorously, assuming that the desirable properties of an information measure are

- that it is continuous with respect to the probability distribution over all possible messages;
- that, for a uniform distribution, the information measure should monotonically increase with the cardinality of the set of events; and
- that if an event is broken down into a series of events, the information measure of the original event should be the weighted sum of the information measure of each event in the series.

Shannon applied the thermodynamical concept of entropy to the communication system as a measure of uncertainty about the message that the information source is going to emit.[5] We shall rigorously define Shannon's entropy measure, along with a variety of other useful information theoretic measures, in section 2.4.

2.3 Shannon's Guessing Game

In 1951 Shannon published a paper which developed some of his thoughts concerning prediction, natural language, and the language competence of human beings [11]. Shannon wished to approximate the entropy of the English language with respect to the model which is implicit in the human brain,[6] and he achieved this by asking human subjects to progressively predict the next letter in an English sentence, an experiment which is now commonly referred to as the *Shannon Game* [10].

The Shannon Game is used to approximate the entropy of the language model implicit in the human brain by measuring the number of guesses required by human subjects to correctly guess each letter in the test sentences. We shall use the Shannon Game to give an example of prediction, and to discuss the measures of surprise and uncertainty.

Consider the incomplete sentence shown below, and imagine that an anonymous experimenter has asked you to utter the word which you expect to see next.

The cat sat on the ...

It is likely that the word which first entered your head upon reading this sentence was 'mat'. There are several reasons for this. You may have seen the sentence "the cat sat on the mat" many times before, and this prior experience increases your expectations of seeing it again. Your experience of the world dictates that cats may sit on various things, so even if you have never seen a sentence exactly like this one before, you can narrow down the possibilities. Your uncertainty of the identity of the next word in the sentence is therefore quite low.

You now inform the anonymous experimenter that you expect to see the word 'mat' next. The anonymous experimenter lets you know that your guess is incorrect.

You might now proceed to guess such words as 'floor', 'chair' or 'carpet', because these too are things that cats might sit on, or you might guess words such as 'hat', 'gnat' or 'rat', because they satisfy the rhyming constraint and happen to be mildly amusing. The anonymous experimenter, much to your chagrin, continues to shake his head as you make your guesses.

After a while you tire of this process, at which stage the anonymous experimenter informs you that the next word in the sequence was actually 'ceiling'.

The word 'ceiling' forms a perfectly grammatical sentence—the sentence "the cat sat on the ceiling" is satisfactory as far as a formal grammar of English is concerned. But to the subject of such an experiment the appearance of the word is surprising, and it is likely that you consider it to form a nonsense sentence. The appearance of the word 'ceiling' it is likely to have surprised you more than the appearance of the words 'mat', 'rat' or even 'floor' would have.

Such a demonstration shows that the model of the English language inside your head, whatever it may be, has additional constraints to those imposed by the formal grammars of English developed by the linguistics community.[7] It also indicates that the surprise you experience upon discovering the identity of the next word in the sentence is a crude measure of your prediction competence.

Shannon gave bounds for the entropy of the model of natural language implicit in the human brain by collecting statistics about the number of attempts human subjects required to correctly guess the next letter in a collection of English sentences. His estimate gave an upper bound of 1.3 bits per letter and a lower bound of 0.6 bits per letter, which agrees with modern experimental results [2].[8]

2.4 Information Theoretic Measures

We shall now formally define the notation and the information theoretic measures which we shall be using throughout this dissertation. The reader should beware that some of our terminology is non-standard; particularly our use of the terms information and entropy. Shannon considered information to be a change in entropy, and stated that "information is that which reduces uncertainty". Our definitions of information and entropy differ, in that we define information as an *a posteriori* property of the data with respect to a predictive model, and we define the entropy as an *a priori* property of the model itself, expressed as the expected value of the information.

We recommend the text by Thomas Cover and Joy Thomas as an excellent reference on Information Theory [5].

2.4.1 Notation

Definition 2.2. *An* alphabet *is a finite set containing at least two distinct elements known as the* symbols. *Let* $\mathcal{A} = \{x_1, x_2, \ldots, x_i, \ldots\}$ *represent the alphabet, and let* $|\mathcal{A}|$ *denote the cardinality of the alphabet.*

Example 2.1. *The set of ASCII characters constitutes an alphabet.*

Remark 2.1. *Although we will usually be dealing with an alphabet of characters, we will occasionally be working on the word level when processing natural language data, in which case the set of words will be referred to as the alphabet rather than the dictionary.*

Definition 2.3. *Let* $s_z = x_1, x_2, \ldots, x_i, \ldots, x_z$*, with* $x_i \in \mathcal{A} \ \forall \ x_i$*, be a sequence of* z *symbols which we shall refer to as a* symbolic time series *or as the* data.

Example 2.2. *For the alphabet of ASCII characters, any file in the ASCII format is a valid symbolic time series. The Sherlock corpus is therefore a symbolic time series in the alphabet of ASCII characters.*

Remark 2.2. *We will often informally refer to the data as a* corpus, *a* text *or a* string, *particularly when talking about natural language data.*

Definition 2.4. *A* predictive model $\mathcal{M}$ *is an algorithm which is capable of making a prediction about the next symbol* x_i *in the symbolic time series* s_z *in the form of a probability distribution over the alphabet. We denote the probability which the predictive model assigns to the next symbol in the data as* $P(x_i|\mathcal{M}, s_{i-1})$*, where* s_{i-1} *represents the portion of the data which precedes the next symbol, referred to as the* history.

Remark 2.3. *An* n^{th}*–order Markov model satisfies this definition, and we shall introduce such models in the next chapter.*

Remark 2.4. *We shall also refer to predictive models as* predictors.

2.4.2 Probability Theory

Predictive models are normally constructed by measuring the relative frequencies of symbols in various contexts, as observed in a training corpus, and we shall discuss the construction of predictive models in this manner in the next chapter. We must tread carefully when referring to relative symbol frequencies and symbol probabilities; a common trap is to regard them as being the same thing. This is not the case.

Probability theory is used to quantify our expectations about the future, and these expectations are represented as a *probability distribution* over all possible future events. This distribution may be arrived at in one of two ways; as a measure of the frequencies of outcomes in random experiments, or as our degrees of belief in propositions which do not involve random variables. The first kind of probability is easy to come by; we can simply make observations of an event in order to approximate its probability, as is the case when inferring the maximum likelihood model from a training corpus. The second kind of probability is subjective rather than objective; it forces us to reveal our *a priori* assumptions, to be explicit about our confidence in our beliefs.

Definition 2.5. *Baye's rule is defined in equation 2.1, where $\mathcal{M}$ is a predictive model and $\mathcal{D}$ is the observed data. The* a priori *probability of the model is given by $P(\mathcal{M})$, and $P(\mathcal{M}|\mathcal{D})$ gives the* a posteriori *probability of the model with respect to the observed data. Application of Baye's rule allows us to update our degree of belief in model $\mathcal{M}$ in the light of observed data, and provides a practical method of selecting one predictive model over another.*

$$P(\mathcal{M}|\mathcal{D}) = \frac{P(\mathcal{M})P(\mathcal{D}|\mathcal{M})}{P(\mathcal{D})} \tag{2.1}$$

Remark 2.5. *Although the predictive model $\mathcal{M}$ may have been inferred from observations of actual data, $P(\mathcal{M})$ must be arrived at by other, possibly quite subjective, means. The power of the Bayesian approach is that it forces us to be explicit about our prejudices. In situations where we have no reason for preferring one predictive model over another, the uniform distribution may be used as the Bayesian prior.*

Remark 2.6. *Note that $P(\mathcal{D})$ is included merely to normalise the equation, to ensure that $P(\mathcal{M}|\mathcal{D})$ is a valid probability distribution. This may be achieved by setting $P(\mathcal{D}) = \sum_{\mathcal{M}} P(\mathcal{D}|\mathcal{M})$.*

Example 2.3. *Consider a biased coin, with $P(heads) > P(tails)$. Without any* a priori *knowledge of this particular coin, it seems reasonable to assume that heads and tails are equally likely. Now let M_m represent the model which predicts $P(heads) = m$ with $m \in [0, 1]$. Baye's rule requires us to provide an* a priori *distribution over the space of models. If we were to choose the uniform distribution, it would mean that we have no prior assumptions, and are willing to approximate $P(heads)$ from the data alone. If we* *are completely confident, and set $P(\mathcal{M}_{\frac{1}{2}}) = 1$ with $P(\mathcal{M}_m) = 0$ for all other m, no number of observations will alter our belief that $\mathcal{M}_{\frac{1}{2}}$ is the correct model. In reality the prior we choose will fall somewhere between these two extremes.*

We shall be using the Bayesian approach in chapter 7 when we explore some alternative methods for combining the predictions made by a family of predictive models into a single prediction, for use in data compression systems.

2.4.3 Information Theoretic Measures

Definition 2.6. *The* information supplied to the predictive model by the next symbol in the data *is denoted $I(x_i|\mathcal{M}, s_{i-1})$, where x_i is the symbol which follows the history s_{i-1} and $\mathcal{M}$ is the predictive model, and is given by the negative logarithm, taken to base 2, of the probability of the symbol x_i following the history s_{i-1} according to the predictive model $\mathcal{M}$, that is,*

$$I(x_i|\mathcal{M}, s_{i-1}) = -\log_2 P(x_i|\mathcal{M}, s_{i-1}) \tag{2.2}$$

Example 2.4. *Consider a model $\mathcal{M}_a$ which predicts the value of a coin toss with the uniform prediction $P(heads|\mathcal{M}_a, s_{i-1}) = P(tails|\mathcal{M}_a, s_{i-1})$, regardless of the history of previous coin tosses s_{i-1}. Each coin toss will result in exactly one bit of information being supplied to the model, since $-\log_2 \frac{1}{2} = 1$.*

Remark 2.7. *The information supplied to the predictive model by the next symbol in the data specifies the minimum number of bits required to describe the symbol with respect to the model in an unambiguous way, and may be understood informally to represent the* surprise *the predictive model receives upon discovering what the next symbol in the data actually is.*

Definition 2.7. *The* instantaneous entropy of the predictive model *is given by the expected value of $I(x_i|\mathcal{M}, s_{i-1})$, and is denoted $H(\mathcal{M}, s_{i-1})$,[9] where s_{i-1} is the history and $\mathcal{M}$ is the predictive model, that is,*

$$H(\mathcal{M}, s_{i-1}) = \sum_{x_i \in \mathcal{A}} P(x_i|\mathcal{M}, s_{i-1}) I(x_i|\mathcal{M}, s_{i-1}) \tag{2.3}$$

Example 2.5. *The entropy of the $\mathcal{M}_a$ of example 2.4 is 1 bit, since the prediction made by the model is the uniform distribution. Consider a second model $\mathcal{M}_b$ which makes the predictions $P(heads|\mathcal{M}_b, s_{i-1}) = \frac{3}{4}$ and $P(tails|\mathcal{M}_b, s_{i-1}) = \frac{1}{4}$. The entropy of this model will be $-\frac{3}{4}\log_2\frac{3}{4} - \frac{1}{4}\log_2\frac{1}{4} = 0.81$ bits.*

Remark 2.8. *We intuitively think of $H(\mathcal{M}, s_{i-1})$ as the* uncertainty *the predictive model $\mathcal{M}$ has about the next symbol in the data, given that s_{i-1} is the history.*

Remark 2.9. *A symbol assigned zero probability by the predictive model supplies an infinite amount of information to the model should it occur, but this does not affect the entropy calculation, as*

$$\lim_{P(x_i)\to 0^+} P(x_i)\log_2 P(x_i) = 0 \tag{2.4}$$

Remark 2.10. *The entropy of the predictive model is a maximum when it assigns an equal probability to all symbols in the alphabet, and equal to*

$$H_{max} = \log_2 |\mathcal{A}| \tag{2.5}$$

Definition 2.8. *The* average information supplied to the predictive model by the data *is denoted by $\overline{I}(s_z|\mathcal{M})$, where z is the number of symbols in the data s_z and $\mathcal{M}$ is the predictive model, and is given by dividing the information supplied to the predictive model by the data by the number of symbols in the data, that is,*

$$\overline{I}(s_z|\mathcal{M}) = -\frac{1}{z}\log_2 P(s_z|\mathcal{M}) \tag{2.6}$$

Remark 2.11. *$P(s_z|\mathcal{M})$ can be decomposed into a product of the predictions made by the predictive model $\mathcal{M}$ about each symbol in the data s_z, and we shall define this decomposition, and show how it may be used to derive the standard Markov model, in the next chapter.*

Example 2.6. *Consider using $\mathcal{M}_b$ of example 2.5 to make predictions about a fair coin, so that heads and tails occur with equal frequency in the data s_z. The average information supplied to the model M_b by the sequence of coin tosses s_z will be $-\frac{1}{2}\log_2\frac{3}{4}-\frac{1}{2}\log_2\frac{1}{4}=1.21$ bits per symbol, while the average information supplied to the model M_a of example 2.4 by s_z will be 1 bit per symbol.*

Remark 2.12. *The average information of the data with respect to the predictive model may be used as a performance measure, and it represents the minimum number of bits required to express s_z in terms of the predictions made by the model. We shall use this measure in chapter 7 when we use predictive models in data compression systems.*

Example 2.7. *If we were to use $\overline{I}(s_z|\mathcal{M})$ as a performance measure in the previous example, M_a would be deemed the better model, as $\mathcal{M}_a$ is "less surprised about the data" than $\mathcal{M}_b$.*[10]

Definition 2.9. *The* perplexity of the predictive model with respect to the data *is denoted $PP(\mathcal{M}|s_z)$, where s_z is the data and $\mathcal{M}$ is the predictive model, and is given by the probability of the data with respect to the predictive model raised to the average information supplied to the predictive model by the data, that is,*

$$PP(\mathcal{M}|s_z) = P(s_z|\mathcal{M})^{\overline{I}(s_z|\mathcal{M})} \tag{2.7}$$

Remark 2.13. *When the IBM Speech Recognition Group developed a language model for their Tangora speech recognition system, the perplexity of the model with respect to the text was introduced as a performance measure [1].*

Remark 2.14. *The perplexity is a monotone function of the average information, and gives the average number of equiprobable symbols which the predictive model must choose between when making a prediction [4]. We shall not be using the perplexity as a performance measure in this dissertation, we define it here only to mention its relationship to the average information.*

2.5 A Philosophical Discussion

Information Theory has found many applications in numerous diverse fields, and there can be no doubt that a general theory of information is of great use in today's information-driven society, and that many more applications of Information Theory are likely to emerge in the future. Indeed, we shall show that information theoretic measures may be used to discover structure in data, and that this structure may be used to bootstrap a predictive model in a process known as the UpWrite. Information Theory, together with the UpWrite, may potentially contribute to the development of a general learning algorithm, something which would mark the genesis of computational intelligence.

Information Theory has formed a basis for theories of learning, including complexity theory, and, more generally, Norbert Wiener's Cybernetics [13]. One author has gone so far as to form, in a non-rigorous sense, a quantum theory in which information is a fifth dimension, and entropy is a field of force [7]. The use and abuse of Information Theory is widespread, and not restricted to the sciences by any means. The introduction of a quantitative measure of information saw its misapplication throughout the humanities, and it is therefore not surprising that the linguistics community of today treats Information Theory with caution, and applies it carefully.

Generally speaking, the entropy of a system rises to a maximum whenever it is in a state of complete and utter disorder, and such systems are inherently uninteresting. In thermodynamics the maximum entropy state is achieved when the system is in equilibrium, which certainly does not seem interesting at a macroscopic scale. Higher entropy states appear roughly the same on a macroscopic level, with many possible arrangements of elements at a microscopic scale. Frozen water has less entropy than water vapour, while a sensible model of the English language has a lower entropy than a model which deems all characters equally likely.

Constraints must be imposed on a system if it is to exhibit behaviour which we would consider interesting. A fish tank containing layers of different coloured fluids is more interesting than one containing a featureless murky liquid, and is evidence of some higher order process at work. Similarly, the constraints which bind the possible forms of sentences in natural languages result in symbol strings which are inherently more interesting than those generated by a uniformly random source, and the higher order processes at work may be uncovered, if only partially, by investigating the strings themselves.

Constraints impose order on a system which otherwise would be completely chaotic, and this imposition of order has the side-effect of introducing redundancy—the fact that the elements of the system are ordered with respect to one makes redundancy inevitable, and its existence in a system is, according to John von Neumann, evidence of complexity [3].

Redundancy, according to Lila Gatlin, comes in two flavours; context-free and context-sensitive [3]. An example of the former is the statistical rule that some symbols occur more frequently than others. Gatlin suggests that this type of redundancy exists to safeguard against error. Context-sensitive redundancy, however, is the extent to which the symbols have departed from their state of independence, and permits variety.

It is a curious thing that intelligent beings create complex systems, and it is even more thought-provoking to suggest that such systems can arise spontaneously, in a self-generating way. According to Campbell [3], John von Neumann believed that such behaviour is a foregone conclusion once the system breaks the so-called "complexity barrier".

In developing the UpWrite Predictor, we shall apply the principles of Information Theory to simple predictive models in an attempt to uncover higher level structure in data, and we shall use this structure to augment the model, increasing its predictive power, in a method borrowed from syntactic pattern recognition—the UpWrite. This is an attempt to model the complex systems which generated the data, with a long-term goal being the development of a computational model of natural language acquisition in human beings.

2.6 Summary and Conclusion

In this chapter we have given a brief historical overview of Information Theory, beginning with Shannon's model of a communication system. We then introduced the notation which we shall be using throughout this thesis, and defined the various information theoretic measures which we have found to be of value in the problem of automatically discovering structure in arbitrary data. We concluded with a rather subjective philosophical overview of why Information Theory is useful in the lofty goal of developing artificially intelligent systems.

Notes

[2] Even though Information Theory itself is not concerned with semantics or meaning, we shall see that it is possible to design a simple predictive model which learns semantic categories in natural language text, to a degree, by applying simple information theoretic measures to the sequence of predictions it makes.

[3] Shannon acknowledges the pioneering work of Hartley in his classic paper.

[4] The terminology 'bit' was coined by J.W. Tukey by contracting the term "binary digit", and was first mentioned in Shannon's landmark paper.

[5] It is interesting to note that Boltzmann referred to thermodynamical entropy as relating to "missing information".

[6] Although this may seem like a controversial statement, it is obvious that the human brain must contain a model of natural language of some sort, because human beings are capable of both generating and understanding natural language utterances.

[7] When considering which words may follow the segment "the cat sat on the ...", it is unlikely that you considered all possible noun phrases. Rather, your prior knowledge imposes additional constraints above and beyond those of a grammar which a linguist might come up with. A formal grammar may only be able to predict that a noun-phrase is expected, which could be anything from 'mat' to "colourless green idea".

[8] Entropy, of course, is a property of a model, and averaging results from a group of different models therefore seems slightly suspicious. It should be noted however that some researchers happily refer to the *entropy of a language*. This is confused, but not intolerably so, if we regard the language as generated by a model.

[9] Although the adoption of the H symbol to denote entropy is borrowed from thermodynamics, we agree with James Massey's suggestion that it be thought of as a belated honour to Hartley [9].

[10] Note that this Minimum Information criterion is equivalent to the standard Maximum Likelihood criterion.

References

[1] A. Averbuch, L. Bahl, R. Bakis, P. Brown, G. Daggett, S. Das, K. Davies, S. Degennaro, P. de Souza, E. Epstein, D. Fraleigh, F. Jelinek, B. Lewis, R. Mercer, J. Moorhead, A. Nadas, D. Nahamoo, M. Picheny, G. Schichman, P. Spinelli, D. Van Compernolle, and H. Wilkens. Experiments with the Tangora 20,000 word speech recognizer. Technical report, IBM T.J. Watson Research Center, 1986.

[2] Peter F. Brown, Stephen A. Della Pietra, Vincent J. Della Pietra, Jennifer C. Lai, and Robert L. Mercer. An estimate of an upper bound for the entropy of English. *Computational Linguistics*, 18(1):31–40, 1992.

[3] Jeremy C Campbell. *Grammatical Man: Information, Entropy, Language and Life.* Pelican Books, 1984.

[4] Eugene Charniak. *Statistical Language Learning.* MIT Press, 1993.

[5] Thomas M. Cover and Joy A. Thomas. *Elements of Information Theory.* John Wiley & Sons, 1991.

[6] R.V.L. Hartley. Transmission of information. *The Bell System Technical Journal*, VII(3):535–563, July 1928.

[7] Arie S. Issar. *From Primeval Chaos to Infinite Intelligence: On Information as a Dimension and on Entropy as a Field of Force.* Avebury, 1995.

[8] Frederick Jelinek. *Probabilistic Information Theory: Discrete and Memoryless Models.* McGraw-Hill, 1968.

[9] James L. Massey. Applied digital Information Theory. Lecture notes, 1992.

[10] Martin Redington and Nick Chater. The guessing game: A paradigm for artificial grammar learning. In A. Ram and K. Eiselt, editors, *Proceedings of the Sixteenth Annual Conference of the Cognitive Science Society*, pages 745–749, 1994.

[11] C.E. Shannon. Prediction and entropy of printed English. *The Bell System Technical Journal*, XXX(1):50–64, January 1951.

[12] Claude E. Shannon and Warren Weaver. *The Mathematical theory of Communication.* University of Illinois Press, 1949.

[13] Norbert Wiener. *Cybernetics, or Control and Communication in the Animal and the Machine.* John Wiley & Sons, 1961.

Chapter 3

Inference of Predictive Models

> "You see, my dear Watson"—he propped his test-tube in the rack, and began to lecture with the air of a professor addressing his class—"it is not really difficult to construct a series of inferences, each dependent upon its predecessor and each simple in itself."
>
> *The Return of Sherlock Holmes*
> SIR ARTHUR CONAN DOYLE

3.1 Introduction

The information theoretic measures introduced in the previous chapter may be used to find structure in data from the sequence of predictions made by a predictive model, and the usual method of constructing such a model is to infer it from a training corpus. This process is a particular case of what is called *grammatical inference* by the syntactic pattern recognition community [4], which may also be referred to as *computational language acquisition.*

Observation of natural language suggests that letters and words occur with varying frequencies [5, 11], and predictive models inferred from a large text corpus are able to capture this property. Such models have many applications, and are used today in speech recognition systems and adaptive statistical data compressors. Both the speech recognition and data compression communities have independently developed similar grammatical inference techniques to construct the predictive models they require.

3.1.1 Speech Recognition

A speech recognition system transcribes an utterance by selecting the most probable word string W in light of the observed acoustic evidence A, as in equation 3.1.

$$\hat{W} = arg \max_{W} P(W|A) \tag{3.1}$$

Application of Bayes' rule allows us to express $P(W|A)$ as in equation 3.2. A speech recognition system therefore consists of two components—an acoustic model which is capable of estimating $P(A|W)$, and which is inferred from a large number of utterances, and a language model which is capable of estimating $P(W)$, and which is inferred from a large corpus of natural language text.[11]

$$P(W|A) = \frac{P(W)P(A|W)}{P(A)} \tag{3.2}$$

An *n-gram language model* is typically used to estimate $P(W)$. The frequencies of word n-tuples are collected from a training corpus, and are used to estimate a probability distribution over all words in the alphabet for contexts of $n-1$ words. This probability distribution is *smoothed* to ensure that every word in the alphabet is assigned a non-zero probability by the n-gram language model. Smoothing is usually performed by linearly interpolating the probability estimates of j-gram language models for $0 \leq j \leq n$. Complicated optimization procedures are typically applied in order to determine locally optimal interpolation weights; the fact that the grammatical inference process is computationally expensive as a result is of little consequence, as the predictive model used in speech recognition systems remains static during use.

3.1.2 Data Compression

Adaptive statistical data compressors work by using *arithmetic coding* to efficiently encode each symbol in the data with respect to a predictive model. In order to avoid the overhead of transmitting this model along with the encoded representation of the data, adaptive statistical data compressors infer the model incrementally from the data as it is being compressed.

A *PPM language model*, where PPM is an abbreviation of *Prediction by Partial Matching*, is typically used in such data compressors—the predictions made by a 3^{rd}–order Markov model are smoothed either by *blending* them with the predictions of lower order Markov models, in a process identical to that of linear interpolation, or by using a technique known as *escape* to gradually fall-back from the 3^{rd}–order Markov model to the lower order Markov model which assigns a non-zero probability to the next symbol in the data.

The predictive model used in adaptive statistical data compressors is inferred adaptively from the data being compressed, and this means that the techniques of blending and escape are required to be computationally efficient, so that compression may complete in reasonable time.

3.1.3 Overview

We begin this chapter with a brief description of the process of grammatical inference before introducing the general technique used to infer predictive models from data. The standard Markov model is then introduced, and we discuss some of the problems it faces. The techniques of *smoothing* and *back-off*, which are used to combine the predictions made by several predictive models of various orders, are then presented. We defer a discussion of the PPM language model until chapter 7, where we shall present the data compression process in detail.

3.2 Grammatical Inference

A *Grammatical Inference Engine* is a device which automatically constructs a grammar from a training corpus. Whether or not such a device is used in the human brain during natural language acquisition is a topic of much controversy. It is our position that the human brain does infer natural language from scratch, rather than making use of a universal grammar which is present from birth, and we base this belief on our experience that fairly general information processing techniques may be used to find fairly specific kinds of language-like structure in data. We introduce a computational model which is capable of uncovering some of this structure in chapter 5.

Apart from contributing to the debate on natural language acquisition in human beings, a general Grammatical Inference Engine would be of great use in applications where it is not possible to design a grammar by hand. Eugene Charniak states that general grammatical inference is useful because [2]

- the analysis is grounded in real data, and therefore makes good use of the available evidence, while avoided unfounded assumptions;
- it may be used in applications where a lack of perfection is of no consequence;
- it can produce useful results in a reasonable amount of time;
- an expert may not be available, and hence standard linguistic techniques for designing a grammar cannot be applied;
- the model can adapt to new data as it is observed; and
- many applications require a stochastic grammar, and it may be unreasonable to expect a human expert to provide us with probability estimates.

Our approach is one of selecting a type of grammar which is easy to infer from data, and which is capable of capturing both the context-free and context-sensitive redundancy observed in natural language. A simple local-context model such as a Markov model is sufficient to achieve this, and we shall show that Markov models may be trivially inferred from data. A technique known as the UpWrite, which we shall introduce in the next chapter, may then be used to abstract the data in a way which improves the ability of the predictive model inferred from it to generalise about previously unseen data, resulting in the formation of a more powerful model. This approach differs from that of traditional syntactic pattern recognition, where a grammar considered appropriate to the data is designed by hand, with the result that grammatical inference may not be such a trivial problem.

3.3 The Stochastic Grammatical Inference Process

A *stochastic grammar* is a model which is capable of assigning a probability to an arbitrary symbolic time series $s_z = x_1, x_2, \ldots, x_z$. It is common to decompose the probability $P(s_z)$ using Bayes' rule, as in equation 3.3, where $s_{i-1} = x_1, x_2, \ldots, x_{i-1}$ is the history. It should be noted that this decomposition is not unique—there are many possible decompositions, all of them equally valid. We select the one shown in equation 3.3 merely because it describes a predictive model, in that the probability of a symbol x_i is conditional on the history of preceding symbols.

$$P(s_z) = \prod_{i=1}^{z} P(x_i|s_{i-1}) \tag{3.3}$$

The inference problem for predictive models is therefore one of estimating the conditional probabilities of the right-hand side of equation 3.3. This is difficult as i increases, as it becomes more and more likely that the history s_{i-1} will not have been observed in the training corpus at all. The common solution to this problem is to group histories into a finite set of *equivalence classes*.

Definition 3.1. *An* equivalence class of strings *is a set of strings which are deemed to be similar according to some measure. We use $\Phi(s_{i-1})$ to denote the equivalence class to which the string s_{i-1} belongs.*

The decomposition of equation 3.3 may now be expressed as in equation 3.4. Our problem is now one of selecting an appropriate equivalence classification.

$$P(s_z) \approx \prod_{i=1}^{z} P(x_i|\Phi(s_{i-1})) \tag{3.4}$$

Frederick Jelinek states [8] that the equivalence classification chosen must necessarily draw a compromise between

- being sufficiently refined to provide adequate information about the history; and

- providing classes which occur frequently enough in the training corpus so that the probability $P(x_i|\Phi(s_{i-1}))$ can be reliably estimated.

The equivalence classification traditionally used in both the speech recognition and data compression fields is to classify histories according to their most recent context of n symbols. This is the *Markovian assumption*, and predictive models which use this equivalence classification are known as *n^{th}–order Markov models.*

3.4 Markov Models

An n^{th}–order Markov model makes the equivalence classification shown in equation 3.5.

$$\Phi(s_{i-1}) = \langle x_{i-n}, \dots, x_{i-1} \rangle \tag{3.5}$$

Remark 3.1. *We refer to the substring $\langle x_{i-n}, \dots, x_{i-1} \rangle$, which the n^{th}–order Markovian equivalence classification maps all histories $s_{i-1} = x_1, x_2, \dots, x_{i-1}$ onto, as a* context. *We delimit contexts with the angled brackets $\langle$ and $\rangle$, and we often delimit substrings and other short symbolic time series using the same notation.*

Definition 3.2. *A* Markov model *consists of a state space $\mathcal{S} = \{y_1, \dots, y_c\}$, a unique starting state $y_s \in \mathcal{S}$, and a probability distribution of transitions between states $P(y_b|y_a)$ $\forall\ y_a, y_b \in \mathcal{S}$.*

Definition 3.3. *An* n^{th}–order Markov model *contains one state for every context, with a transition between two states $\langle x_{i-n}, \dots, x_{i-1} \rangle$ and $\langle x_{i-n+1}, \dots, x_i \rangle$ occurring with probability $P(x_i|x_{i-n}, \dots, x_{i-1})$, and emitting the symbol x_i.*

The n^{th}–order Markov model $\mathcal{M}$ estimates the probability of the data s_z as in equation 3.6.

$$P(s_z|\mathcal{M}) \approx \prod_{i=1}^{z} P(x_i|\langle x_{i-n}, \dots, x_{i-1} \rangle) \tag{3.6}$$

Remark 3.2. *A Markov model is capable of assigning a probability to an arbitrary symbolic time series by performing the decomposition of equation 3.6. Markov models may also be used* generatively *to create data according to the probabilities embodied by the model. This process proceeds by beginning in an arbitrary state, following a transition out of this state at random, in accordance with the probabilities of the transitions which lead out of the state, emitting the appropriate symbol, and iterating. We shall be using predictive models generatively in chapters 6 and 7 in order to inspect the data they produce. This enables us to get an intuitive feel of how well the predictive model captures the essence of the data it was inferred from.*

Example 3.1. *Figure 3.1 shows a simple 2^{nd}–order Markov model which is capable of generating pseudo-English sentences such as "The man ate the apple, and the woman ate a peach."*

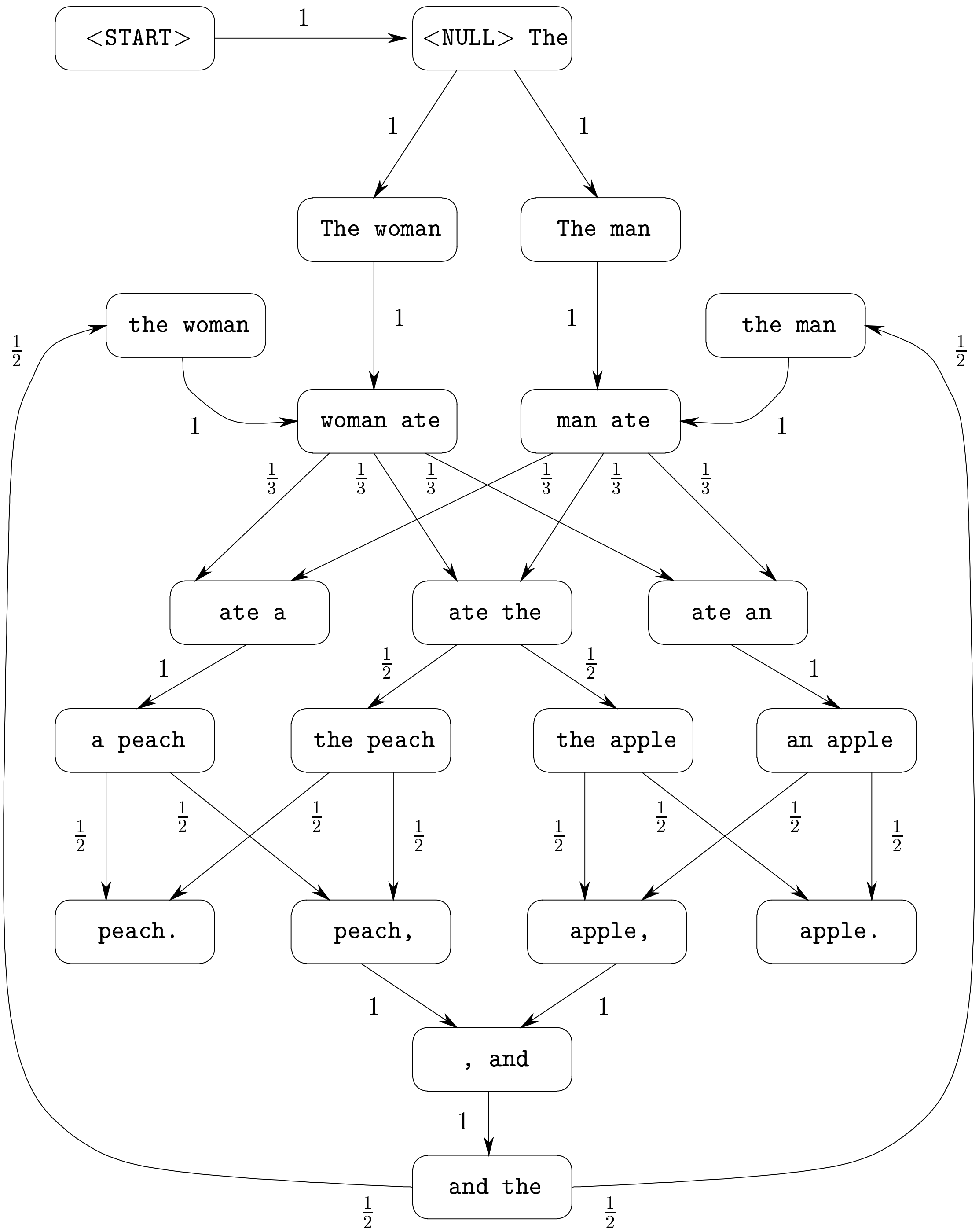

FIGURE 3.1: A simple 2^{nd}–order Markov model which is capable of generating pseudo-English sentences.

It is now possible to derive the maximum-likelihood estimate of the probabilities of the right-hand side of equation 3.6 from the normalised frequency with which the substring $x_{i-n}, \dots, x_i$ occurs in the training corpus. This is achieved as in equation 3.7, where $C(s_i)$ denotes a count of the number of occurrences of the substring s_i in the training corpus.

$$P(x_i|\langle x_{i-n}, \dots, x_{i-1}\rangle) \approx \frac{C(x_{i-n}, \dots, x_i)}{C(x_{i-n}, \dots, x_{i-1})} \tag{3.7}$$

Remark 3.3. *We refer to the substring $\langle x_{i-n+1}, \dots, x_i\rangle$, which is of length n, as an n-*gram*. The n-gram language model is so named because the inference process consists of counting the occurrences of n-grams in the training corpus. An n^{th}–order Markov model is equivalent to an $(n+1)$-gram language model.*

Remark 3.4. *We refer to a 1-gram language model as a* unigram language model*, a 2-gram language model as a* bigram language model *and a 3-gram model as a* trigram language model.

Speech recognition systems normally incorporate 2^{nd}–order Markov models, while 3^{rd}–order Markov models are usually used in the PPM language model of adaptive statistical data compressors. We shall introduce the PPM language model in chapter 7; until then we shall be concerned only with the modelling techniques used in speech recognition systems, as they will be sufficient to illustrate the problems faced by the inference of predictive models in general.

3.5 Problems With Markov Models

The average information supplied to a predictive model by the data, as defined in equation 2.6, may be used to evaluate the performance of the predictive model. Consider the plot shown in figure 3.2, which was generated by inferring character level n-gram language models, for various n, from the entire Sherlock corpus, and evaluating the performance of each of the resulting predictive models by measuring the average information provided to the model by the `Small` section of the Sherlock corpus.

It can be seen that the performance of an n-gram language model improves as n is increased when the model is used to make predictions about portions of the corpus from which it was trained. This is not surprising; larger contexts constrain the possible values of x_i, and, for sufficiently large n, $P(x_i|\mathcal{M}, s_{i-1}) = 1 \ \forall \ x_i \in s_z$.

It need not be said that in practical applications we do not have the opportunity to infer predictive models from the data on which they will be applied—the entire point of grammatical inference is to construct a predictive model which is capable of making generalisations about hitherto unseen data. Consider figure 3.3, which plots the performance of character level n-gram language models which were inferred from the `Train` section of

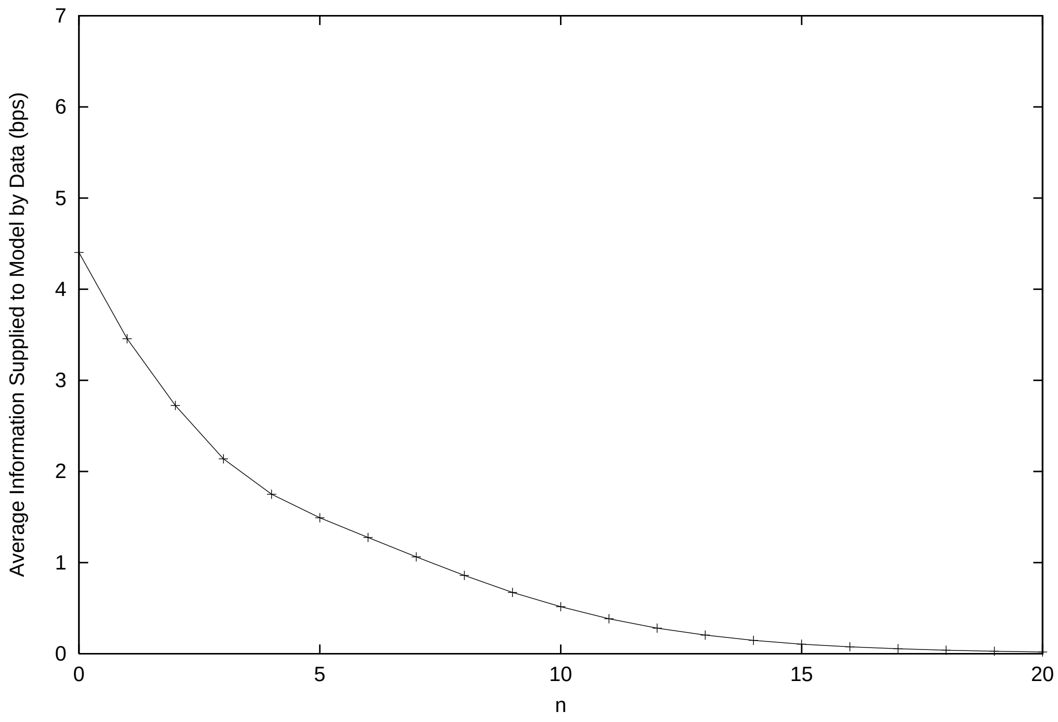

FIGURE 3.2: Plot of the performance of n-gram language models, for various n, over a portion of the data from which they were inferred.

the Sherlock corpus only, and which were then evaluated over the `Small` section of the Sherlock corpus. In order to avoid problems arising from cases where $P(x_i|\mathcal{M}, s_{i-1}) = 0$, we set $P(x_i|\mathcal{M}, s_{i-1}) = \frac{1}{|\mathcal{A}|}$ in situations where this would have occurred.[12]

It turns out that the performance of the n-gram language model is almost identical to that of the previous example for $n \leq 3$, but after this it begins to level off, and degrades for $n > 5$. This is due to the fact that as n is increased, more and more contexts which did not occur in the training corpus are observed in the testing corpus.

Three of the main problems suffered by Markov models, when used as predictive models, are

the zero-frequency problem which occurs when the model is unable to assign a non-zero probability to the next symbol in the data, with the result that the symbol provides an infinite amount of information to the model should it occur;

the sparse data problem which occurs when an insufficient amount of data was used for inference, resulting in unreliable probability estimates; and

the local context problem in which the model fails to take long-range dependencies which exist in the data into account.

In the following sections we shall describe each of these problems in detail, after which we will briefly discuss the two methods of *smoothing* and *back-off* which are used in

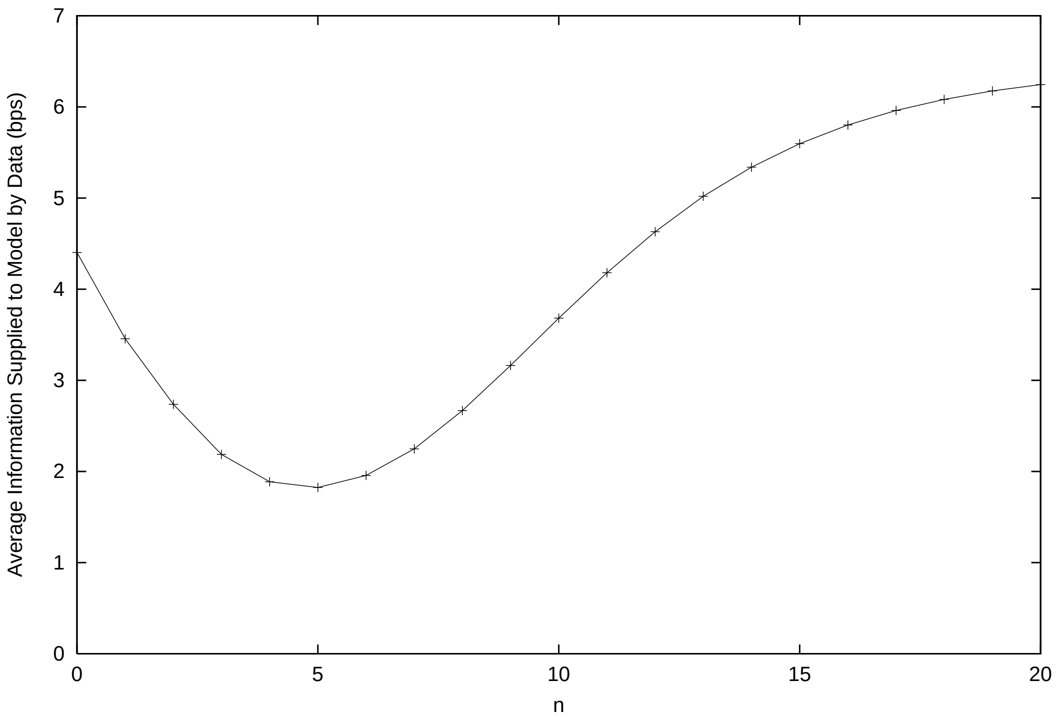

FIGURE 3.3: Plot of the performance of n-gram language models, for various n, over novel data.

the predictive models of speech recognition systems to combine the predictions made by Markov models of various orders into a single prediction, eliminating the zero-frequency problem altogether, and reducing the problem due to sparse data. We shall defer a potential solution to the third problem to chapter 5, where we introduce *UpWrite Predictor*.

3.5.1 The Zero-Frequency Problem

The zero-frequency problem occurs when a predictive model encounters a context which was not observed during inference and, as a consequence, assigns a zero probability to the symbol which actually occurs next in the data [3, 10]. This results in the probability $P(x_1, x_2, \ldots, x_z)$ being calculated as zero, causing a catastrophic failure whereby an infinite amount of information is supplied to the predictive model by the data.

The zero-frequency problem may be due to the appearance of a novel symbol in a previously observed context, or by the appearance of a novel context which consists of a hitherto unseen arrangement of known symbols. We shall analyse the nature of these two situations by studying a simple artificial data source.

The Occurrence of a Novel Symbol

Consider a symbolic time series which consists of a repeated concatenation of the substring `aab`. Figure 3.4 illustrates a portion of this data, where the occurrence of the novel symbol `c` may, for instance, be due to the presence of noise (*i.e.* it may be a corrupted `a`).

```
aabaabaabaabacbaabaab
```

FIGURE 3.4: The hitherto unseen symbol `c` occurs in data.

Let us assume that a trigram language model $\mathcal{M}$ has been inferred from a version of this data in which the symbol `c` did not occur. This model will be unable to assign a non-zero value to $P(\mathtt{c}|\mathcal{M},\mathtt{ba})$, $P(\mathtt{b}|\mathcal{M},\mathtt{ac})$ and $P(\mathtt{a}|\mathcal{M},\mathtt{cb})$. In general, an n-gram language model will be rendered useless over a sequence of n symbols whenever a single novel symbol occurs, beginning at the novel symbol itself and continuing for the context length, during which the novel symbol appears in the context and, therefore, the context is also novel.

The Occurrence of a Novel Context

Consider a different portion of the same data, shown in figure 3.5, where the occurrence of the novel context $\langle\mathtt{bb}\rangle$ may also be due to corruption of the data by noise.

```
aabaabaabaababbaabaab
```

FIGURE 3.5: The hitherto unseen context $\langle\mathtt{bb}\rangle$ occurs in data.

In this situation, even though all of the symbols in the data were observed during inference, the fact that the context $\langle\mathtt{bb}\rangle$ is novel means that $P(\mathtt{a}|\mathtt{bb})$ cannot be assigned a non-zero value by the trigram model. Furthermore, in this particular example, the model will be unable to assign non-zero values to both $P(\mathtt{b}|\mathtt{ba})$ and $P(\mathtt{b}|\mathtt{ab})$, since an `a` symbol has only previously occurred in those contexts.

3.5.2 The Sparse Data Problem

The sparse data problem is related to the zero-frequency problem in that it is a consequence of the fact that only a finite amount of data is available for inference. An n^{th}–order Markov model requires more training data as n is increased in order to observe each context sufficiently often so that a reliable probability estimate can be made.

In reality the size of training corpus required by an n^{th}–order Markov model for even moderate values of n is prohibitive, and simply unattainable for many applications. The result of this is that the predictions made by n^{th}–order Markov models tend to be rather sparse probability distributions for large n.[13]

3.5.3 The Local Context Problem

The third problem suffered by Markov models is caused by the equivalence classification they use—the models only take a local context into account, and are therefore blind to any long-range features in the data.

Consider, for example, the following sentence, which is taken from "The Adventures of Sherlock Holmes" by Sir Arthur Conan Doyle.

> He took an orange from the cupboard, and tearing it to pieces he squeezed out the pips upon the table.

In this sentence, the seventeenth word, 'pips', is far more dependent on the fourth word, 'orange', as it is on the two words immediately preceding it. A Markov model of sufficiently high order to take this long-distance dependency into account would suffer from the other two problems we have mentioned.

3.6 Smoothing

The language model used in speech recognition systems addresses the zero-frequency problem and the sparse data problem by *smoothing* the predictions made by various orders of Markov model together in order to guarantee that $P(x_i|\mathcal{M}, s_{i-1}) > 0 \ \forall \ x_i \in \mathcal{A}$. This process is akin to that of *blending* used in the PPM language model of adaptive statistical data compressors. We shall discuss blending in chapter 7.

Smoothing is achieved as in equation 3.8. The predictions made by j-gram language models[14] for various j, denoted by $\mathcal{M}_j$, are interpolated by taking a weighted sum, where the interpolation weights are functions of the context $\langle x_{i-n+1}, \ldots, x_{i-1}\rangle$, and satisfy $\lambda_j(\langle x_{i-n+1}, \ldots, x_{i-1}\rangle) > 0 \ \forall \ j \in [1, n]$, and $\sum_{j=1}^{n} \lambda_j(\langle x_{i-n+1}, \ldots, x_{i-1}\rangle) = 1$. We may consider the interpolation weights to define a probability distribution over the set of j-gram language models $\{\mathcal{M}_1, \ldots, \mathcal{M}_n\}$.

$$P(x_i|\mathcal{M}, s_{i-1}) \approx \sum_{j=1}^{n} \lambda_j(\langle x_{i-n+1}, \ldots, x_{i-1}\rangle) P(x_i|\mathcal{M}_j, s_{i-1}) \tag{3.8}$$

Smoothing via linear interpolation incorporates the predictions of low order Markov models, which tend to make the most of the available data, and the sparse-data problem is reduced as a consequence. Although smoothing eliminates the zero-frequency problem due to novel contexts, the appearance of a novel symbol will still result in an infinite amount of information being supplied to the model. Data compressors address this by introducing another model, $\mathcal{M}_{-1}$, which makes the uniform prediction, while speech recognition systems ensure that no symbol occurs in the testing corpus which has not already been observed in the training corpus.

We now require methods for estimating the interpolation weights. Many candidate techniques exist, but we shall consider only one—*Baum-Welch optimization*, which is a particular case of the *Expectation-Maximisation algorithm.*

3.6.1 Baum-Welch Optimization

Baum-Welch optimization, also known as the *forward-backward algorithm*, is commonly used to determine locally-optimal interpolation weights in the language model of speech recognition systems [6–8]. Baum-Welch optimization begins by regarding the smoothed language model of equation 3.8 as a single *Hidden Markov Model*, or *HMM*. We shall not describe the Baum-Welch algorithm in detail; for that we recommend Jelinek's book [8].

Definition 3.4. *A* Hidden Markov Model *consists of a state space* $\mathcal{S} = \{y_1, \ldots, y_c\}$*, a unique starting state* $y_s \in \mathcal{S}$*, a probability distribution of transitions between states* $P(y_b|y_a)\ \forall\ y_a, y_b \in \mathcal{S}$*, and a probability distribution over the output alphabet* $\mathcal{A}$ *associated with each transition* $P(x_i|y_a, y_b)$*.*

Remark 3.5. *A* Hidden Markov Model *is a Markov model in which the state sequence used to generate an output string is unknown—that is, more than one state sequence is possible for any given output string.*

The probability of the data $s_z = x_1, \ldots, x_z$ may be expressed as the sum of the individual probabilities of all possible generations of that data, as given in equation 3.9.[15]

$$P(s_z) = \sum_{y_1,\ldots,y_z} \prod_{i=1}^{z} P(y_i|y_{i-1})P(x_i|y_{i-1}, y_i) \tag{3.9}$$

Figure 3.6 shows a portion of the HMM which corresponds to a trigram language model which has been smoothed via linear interpolation. The transition from the state $\langle x_{i-2}, x_{i-1}\rangle$ to the state $\langle x_{i-1}, x_i\rangle$ may occur in three different ways, corresponding to the three transitions in the unigram, bigram and trigram language models. The three states corresponding to the interpolation weights λ_j are referred to as *null states*, and the three transitions to these null states from the state $\langle x_{i-2}, x_{i-1}\rangle$, represented by dashed lines in the diagram, are referred to as *null transitions.* These transitions emit no symbols when they are traversed.

By applying equation 3.9 we may calculate the probability of the Hidden Markov Model ending up in state $\langle x_{i-1}, x_i\rangle$ given that it started in state $\langle x_{i-2}, x_{i-1}\rangle$, and we find that this corresponds to the probability $P(x_i|\mathcal{M}, s_{i-1})$ of equation 3.8, for $n = 3$.

Initially the probabilities of the null transitions in the HMM are unknown, while those of the remaining transitions are approximated via standard maximum-likelihood techniques. A large portion of the training corpus is used for this inference, with the remainder being *held out* so that it may be used to estimate the values of the interpolation weights.

Ideally, we would like to choose the interpolation weights as in equation 3.10, where $\mathcal{M}_\lambda$ represents an HMM with a particular set of interpolation weights λ, and s_z is the

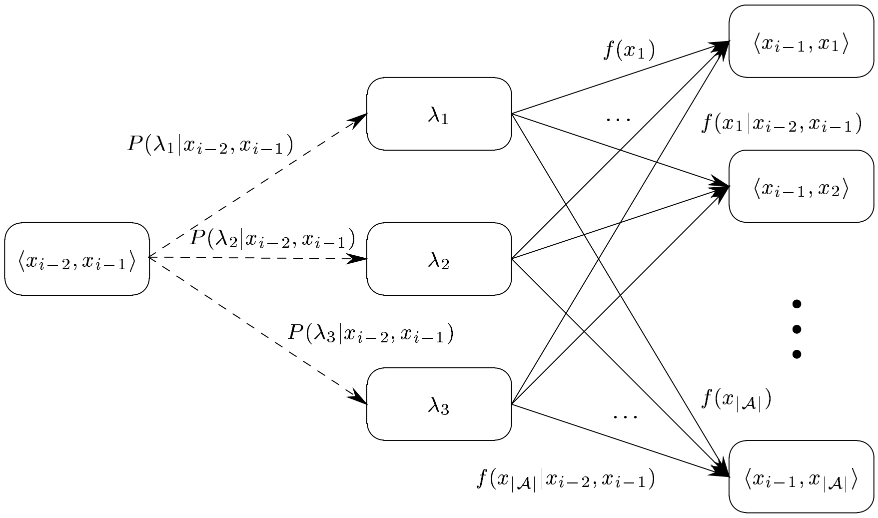

FIGURE 3.6: Part of the HMM formed from the linear interpolation of three Markov models.

observed data. The process of estimating the interpolation weights now becomes one of estimating the probabilities $P(\lambda_j|x_{i-2}, x_{i-1})$—the probability associated with each null transition in the HMM—such that the probability of the observed data is maximised.

$$\hat{M}_\lambda = arg \max_\lambda P(s_z|\mathcal{M}_\lambda) \tag{3.10}$$

The Baum-Welch algorithm may be used to find locally-optimal values of the interpolation weights. It is based upon the assumption that if the state sequence $y_1, \ldots, y_k$ which generated the observed data s_z is known, where $k > z$ due to the presence of null transitions, then the transition probabilities could be estimated via the standard maximum-likelihood technique by counting the number of times each transition was used in the generation. In an HMM, however, many possible state sequences are capable of generating the same data. In this case, the counting function $C(y_{i-1}, y_i)$, which is used to estimate $P(y_i|y_{i-1})$, is expressed as a function of $P(y_i|y_{i-1})$ itself!

The Baum-Welch algorithm proceeds by making an initial guess at the values of the interpolation weights, and then gradually refining this initial estimate by finding all possible state sequences which could have generated the held-out portion of the training corpus, and using the modified count functions described above to re-estimate the null transition probabilities. This process is repeated until the values of the interpolation weights converge.

3.7 Back-off

A second technique used in the language model of speech recognition systems is that of *back-off*. Instead of taking the weighted sum of predictions made by several Markov models to determine the value of $P(x_i|\mathcal{M}, s_{i-1})$, the probability assigned to x_i by exactly one Markov model in the set $\{\mathcal{M}_1, \dots, \mathcal{M}_n\}$ is used, with the model selected to estimate the probability of a particular x_i being determined by gradually falling back from higher order Markov models to lower order ones. This process is similar to that of *escape* in the PPM language model of adaptive statistical data compressors. We shall discuss escape in chapter 7.

Various back-off techniques exist. In this section we shall consider one—Katz's back-off procedure, which has been found to often out-perform smoothed language models which use the Baum-Welch algorithm to optimize the interpolation weights. Katz's back-off procedure is equivalent to a process of *non-linear interpolation*, whereby the frequency estimates of rare events are *discounted*, with the freed probability mass being redistributed amongst unobserved events.

3.7.1 Katz's Back-Off Procedure

Katz's back-off procedure is a heuristic nonlinear interpolation technique which has been found to out-perform smoothed language models which use the Baum-Welch algorithm to optimize their interpolation weights [8, 9]. It works by quantifying the belief that high frequency counts are more accurate than low frequency counts by *discounting* the relative frequency of a symbol in a context via a discounting function $d(C(x_{i-n+1}, \dots, x_i))$, which is a function of the number of times the n-gram has been observed, as in equation 3.11, and re-distributing the remaining probability mass among events which have not yet been observed in the context.

$$P(x_i|\langle x_{i-n+1}, \dots, x_{i-1}\rangle) \approx d(C(x_{i-n+1}, \dots, x_i))\frac{C(x_{i-n+1}, \dots, x_i)}{C(x_{i-n+1}, \dots, x_{i-1})} \tag{3.11}$$

Katz chose to set $d(r) = 1$ for $r > K$, where K is some predetermined count threshold and r is an n-gram count, in the belief that for sufficiently large counts the relative frequency is an adequate estimate of the probability. Katz suggests that K be set to 5 or so [9]. The discounting function used for the remaining r which satisfy $1 \leq r < K$ is chosen to ensure that the probability assigned to unseen n-grams is that specified by the *Good-Turing estimate* [8].

This results in the general back-off procedure shown in equation 3.12, where the probability of a symbol x_i according to the model $\mathcal{M}_j$ and the history s_{i-1} is expressed as a discounted version of the relative frequency $f(x_i|x_{i-n+1}, \dots, x_{i-1})$, as encapsulated in the function Q_T, which is a Good-Turing type function.

$$P(x_i|\mathcal{M}_j, s_{i-1}) \approx \begin{cases} f(x_i|x_{i-j+1}, \dots, x_{i-1}) & \text{if } C(x_{i-j+1}, \dots, x_i) \geq K, \\ \alpha Q_T(x_i|x_{i-j+1}, \dots, x_{i-1}) & \text{if } 1 \leq C(x_{i-j+1}, \dots, x_i) < K, \\ \beta P(x_i|\mathcal{M}_{j-1}, s_{i-1}) & \text{otherwise.} \end{cases} \tag{3.12}$$

Fallback to the model $\mathcal{M}_{j-1}$ occurs when this relative frequency is zero, and the weights $\alpha \in [0,1]$ and $\beta \in [0,1]$ are chosen to ensure that the prediction made by the predictive model is a valid probability distribution. In the situation where the fallback process reaches $\mathcal{M}_1$, it is assumed that $P(x_i|M_1, s_{i-1}) = f(x_i)$. Katz calculates the weights α and β directly, as functions of the observed n-gram frequencies, in preference to optimizing their values via other techniques. Details of Katz's technique may be found in Jelinek's book [8].

3.8 Summary and Conclusion

In this chapter we have introduced the notion of a grammatical inference engine, and we have discussed the problems faced by Markov models, which may be inferred from data trivially. We then briefly discussed the two solutions proposed by the speech recognition community to these problems; smoothing and back-off. The details of these techniques are not important to our thesis, although we shall be presenting the equivalent techniques of blending and escape, developed by the data compression community, when we construct our own adaptive statistical data compressor in chapter 7.

Notes

[11] $P(A)$ is merely a normalising constant which ensures that $P(W|A)$ is a valid probability.

[12] This means that the prediction made by the predictive model is no longer a valid probability distribution, but this is of no consequence in the example; we merely seek to guarantee that $\overline{I}(s_z|\mathcal{M})$ remains finite.

[13] Sparse in that most of the elements of the probability distribution will be zero.

[14] Recall that a j-gram language model is equivalent to a Markov model of order $j-1$.

[15] $P(s_z)$ may be calculated from an HMM in a recursive manner by constructing a *trellis* which represents the generation of data s_z over time. This technique is presented in Jelinek's book [8].

References

[1] L. Bahl, F. Jelinek, and R. Mercer. A maximum likelihood approach to continuous speech recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-5(2), March 1983.

[2] Eugene Charniak. *Statistical Language Learning*. MIT Press, 1993.

[3] John G. Cleary and W.J. Teahan. Experiments on the zero frequency problem. In James A. Storer and Martin Cohn, editors, *Data Compression Conference (DCC '95)*, pages 52–61, 1995.

[4] King Sun Fu. *Syntactic methods in pattern recognition.* Academic Press, 1974.

[5] Zellig Harris. *A Theory of Language and Information: A Mathematical Approach.* Oxford University Press, 1991.

[6] Frederick Jelinek. Principles of lexical language modeling for speech recognition. Technical report, IBM T.J. Watson Research Center.

[7] Frederick Jelinek. Self-organized language modeling for speech recognition. Technical report, IBM T.J. Watson Research Center.

[8] Frederick Jelinek. *Statistical Methods for Speech Recognition.* MIT Press, 1997.

[9] Slava M. Katz. Estimation of probabilities from sparse data for the language model component of a speech recognizer. *IEEE Transactions on Acoustics, Speech and Signal Processing*, ASSP-35(3), March 1987.

[10] Ian H. Witten and Timothy C. Bell. The zero-frequency problem: Estimating the probabilities of novel events in adaptive text compression. *IEEE Transactions on Information Theory*, 37(4):1085–1094, July 1991.

[11] George Kingsley Zipf. *Human Behavior and the Principle of Least Effort: An Introduction to Human Ecology.* Addison Wesley, 1949.

Chapter 4

An Introduction to the UpWrite

> Everything which had been disconnected before began at once to assume its true place, and I had a shadowy presentiment of the whole sequence of events.
>
> *The Memoirs of Sherlock Holmes*
> Sir Arthur Conan Doyle

4.1 Introduction

The *UpWrite* is a process for constructing a hierarchical representation of data. It works by iteratively extracting higher level structure from the representation of the data at one level of the hierarchy in order to determine how the data should be represented at the next level of the hierarchy. Similarities between local models of the data at one level of representation are used to extract this higher level structure.

The basic goal of the UpWrite is to progressively abstract the data in such a way as to preserve its necessary features, with the intention of using the resulting representation in applications such as image recognition.

Consider, for example, the image shown in figure 4.1. This image is a low-resolution bitmap of a triangle, and the lowest level representation of such an image, as far as a computer program is concerned, is as a two-dimensional matrix of binary values, corresponding to the black and white pixels in the bitmap. This is the level of representation at which any algorithm which looks for objects such as triangles in images must begin.

To a human being, however, other levels of representation are immediately apparent. We might say that the next level of abstraction is that of line segments. We can recognise three line segments in the image, and we may additionally recognise three vertices; points which exist solely based upon the relationship of these line segments with one another. This first level of representation, from a collection of pixels to a collection of line segments and vertices, has abstracted the image by grouping together pixels which are, in some sense, parts of the same higher level object.

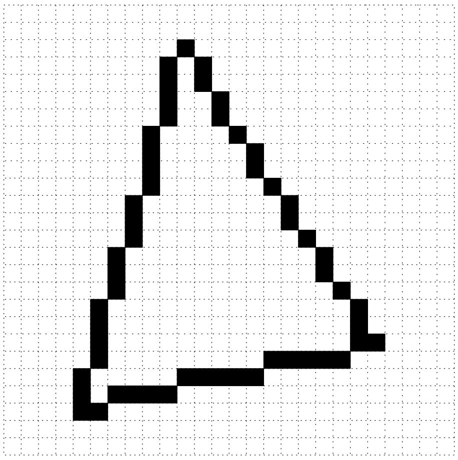

FIGURE 4.1: An binary image of a triangle has several levels of representation, the lowest level of which is an array of bits.

At the next level of abstraction, a human being may say that the three line segments and the three vertices describe a single object—the triangle. A computer program which is able to construct a hierarchical model of a triangle in the same way may be capable of determining whether or not a novel image belongs to the set of images of triangles.

The UpWrite can be used to develop a computer program which is able to achieve this classification with a high level of success, without knowing what a triangle is *a priori*. The resulting program is generic enough to be extended to non-trivial image classification problems.

4.1.1 Relationship to Predictive Models

The UpWrite was inspired in part by the observation that many patterns exhibit a quasi-linguistic structure which is reminiscent of the structure of natural language. Our thesis rests on the belief that the UpWrite, which is a fairly general framework for finding structure in data, and progressively constructing a hierarchical model of that data, may be applied to simple predictive models.

4.1.2 Overview

We begin this chapter with a brief historical overview which serves to introduce the UpWrite and its relationship to the field of syntactic pattern recognition. We then define the UpWrite rigorously; a non-trivial task given that the technique itself has not yet been fully developed, and that no definite reference exists at the time of writing. We then illustrate the application of the UpWrite to a simple pattern recognition problem, that of classifying polygons, before concluding with a brief discussion of real-world image recognition applications which successfully use the UpWrite, and a discussion of how the UpWrite may be of use in the current work.

4.2 Historical Background

4.2.1 Syntactic Pattern Recognition

The UpWrite is related to the field of *syntactic pattern recognition*, which was initiated by King Sun Fu when he observed that images tend to be hierarchically structured in ways which are analogous to the structure of natural languages [9–11]. The syntactic approach to pattern recognition differs from the more prevalent decision-theoretic approach in that instead of representing a pattern as a feature vector, it attempts to construct a global description of the pattern in terms of its constituent parts, and represent this description as a string of symbols. The symbols themselves are discrete, and the alphabet of symbols is selected by a human being based upon a study of the type of data being modelled.

Example 4.1. *Consider the bitmap of a triangle shown in figure 4.1. A candidate alphabet of symbols for describing images such as this is* $\mathcal{A} = \{\mathtt{N}, \mathtt{NE}, \mathtt{E}, \mathtt{SE}, \mathtt{S}, \mathtt{SW}, \mathtt{N}, \mathtt{NW}\}$*, where each of these symbols represents a line segment of some predetermined length oriented in accordance with the associated compass direction.*

Various techniques exist whereby a description string may be formed from a pattern, but these tend to be unsatisfactory, with the result that similar patterns do not necessarily produce similar description strings.[16] This is probably due to the fact that images tend not to be very string-like.

One commonly used technique for the construction of a description string from a pattern is that of *chain coding*, whereby the border of the pattern is traced, and the closest matching symbol to the current local portion of the pattern is emitted at each stage of this border tracing process. The resulting description string is obviously dependent on where in the image the border tracing procedure began, and preprocessing techniques offer solutions to this problem.

Example 4.2. *Consider the bitmap of a triangle shown in figure 4.1, and the alphabet of the previous example. A possible chain-coding of the triangle which began at the black pixel closest to the lower left border of the image, and proceeded in an upwards direction, is* `N,N,NE,N,N,SE,SE,S,SE,SE,W,W,W,SW`.

The recognition process is one of determining to which language the string of symbols representing the pattern belongs. Each class of patterns which can be recognised by the system will have an associated grammar, and this grammar may be used to accept or reject the string which represents a novel pattern (or, if the grammar is stochastic, to assign this string a probability). In some cases these grammars may be designed by hand, but it is often preferable to automatically construct them from a set of training examples via a process of *grammatical inference*. This recognition technique is in contradistinction to the decision-theoretic approach to pattern recognition, in which classification is performed by partitioning a feature space.

4.2.2 The UpWrite

In the 1970's Michael Alder became interested in modelling cognitive processes in ways which are consistent with our knowledge of the central nervous system and the findings of Cognitive Psychology. His primary interest was to develop a system which was capable of automatically selecting salient features which would enable a hierarchical representation of the structure in images to be created, thus avoiding the problem in traditional syntactic pattern recognition in which a human being is required to select the alphabet of primitive symbols.

Alder was motivated by the quasi-linguistic features of information processing in the human brain, and a discontent with the classical neural network model. The work of Hubel and Wiesel and Blakemore, described in David Marr's excellent "Vision" [15], suggested that primitive feature extraction occurs at a low level of the nervous system, and this led Alder to develop a framework referred to as the UpWrite.

In the early 1990's Alder, together with Christopher deSilva and Yianni Attikiouzel, authored a series of technical reports describing an early version of the UpWrite [1, 2, 7], and in [1] they write that

> ... what we seek is not simply a system which can recognise, say, hand-printed or cursive characters, nor one which uses high level descriptions of the characters, but one which generates high level descriptions automatically. Ideally, it might generate a decomposition of the letter A which would amount to recognising three strokes with appropriate relations between them; even more idealistically it might go on to find words and hence to be able to 'read' badly distorted characters simply by using context.

The overall aim of the UpWrite is the formation of a satisfactory theory of learning which may be applied to various problems. Alder, deSilva and Attikiouzel summarised the hierarchical nature of the UpWrite in [2], where they write that

> ... human being use many levels of description of events and entities. For example, one may specify, in principle, the sequence of hand movements over a piece of paper, or one may say "he signed a cheque". These may be alternative descriptions, of use to different people ... parallel to the levels of description, there are levels of acquisition ...

One of the main features of the UpWrite is that instead of selecting the grammar which is most suited to the current pattern recognition problem, with the result that the selected grammar may be difficult to automatically infer from data, a simple local model which is conducive to inference is selected instead, and techniques are then used to find higher level structure in the data with respect to the simple local model itself, and bootstrap the model to a higher level of abstraction. The implication of this approach is that identical pattern recognition techniques may be applied to problems which initially appear to be quite different in nature.

The UpWrite has been successfully applied to a wide range of image processing tasks, some of which we shall describe near the end of this chapter, and its consistently good performance suggests that it is a good framework for syntactic pattern recognition, in a more general setting than that prescribed by Fu.

4.3 The UpWrite Process

The UpWrite is an attempt to address the problems of syntactic pattern recognition while avoiding the pitfalls of Fu's programme. Specifically, the UpWrite generalises beyond string languages, as they are thought to be too restrictive for the modelling of images, while avoiding the introduction of the human brain into the loop. The UpWrite represents an object hierarchically, at different levels of abstraction, and defines an automatic inference procedure which is used to generate higher levels of description in a process which is essentially bottom-up [5].

Two major types of structure are modelled by the UpWrite. Sub-objects are ordered sets of objects, while quotient-objects are equivalence classes of objects. For example, words formed from character sequences are sub-objects, while syntactic categories of words are quotient-objects.

4.3.1 The Sub-Object UpWrite

The *Sub-Object UpWrite* is based on the general notion that the observed data may be partitioned into a collection of sub-objects such that these sub-objects correspond to the primitives of some higher level representation of the data. For example, a set of pixels in an image may correspond to a line segment, and a set of characters in natural language text may correspond to a word.

Incorporating sub-objects into a predictive model has the advantage of increasing the context available to the model, because a single sub-object represents a sequence of lower level symbols. For example, consider figure 4.2, which shows a sequence of characters UpWritten to a sequence of character pairs. If we momentarily disregard the sparse data problem, it is obvious that an n-gram language model inferred from the UpWritten version of the data will better predict what is coming next, as this n-gram language model is equivalent to a $2n$-gram language model inferred from the lower level representation of the data.

We are concerned with the modelling of symbolic time series, and we shall therefore assume that a sub-object is a sequence of symbols in the following definitions. We shall often informally refer to sub-objects as *symbol sequences*.

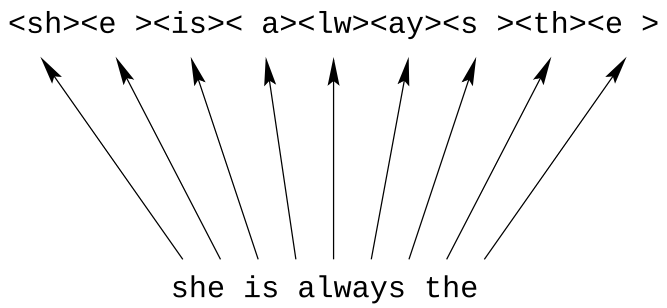

FIGURE 4.2: The Sub-Object UpWrite extends the context available to a predictive model.

Definition 4.1. *The* Sub-Object UpWrite *is a mapping* $\langle x_1, \ldots, x_j \rangle \mapsto y_i$ *between a symbol sequence* $\langle x_1, \ldots, x_j \rangle$ *of symbols taken from some alphabet* $\mathcal{A}_1$ *to a symbol* y_i *in some new alphabet* $\mathcal{A}_2$ *such that the observed data* $s_z = x_1, \ldots, x_z$, $x_i \in \mathcal{A}_1$ *can be represented as a higher level symbolic time series* $t_k = y_1, \ldots, y_k$, $y_i \in \mathcal{A}_2$, $k \leq z$.

Definition 4.2. *The* Sub-Object DownWrite *is the corresponding inverse relation.*

Remark 4.1. *We shall describe the process of mapping between strings of symbols taken from* $\mathcal{A}_1$ *to the symbols in* $\mathcal{A}_2$ *as* UpWriting the alphabet, *and the process of representing the observed symbol sequence* s_z *in terms of this higher level alphabet as* UpWriting the data.

Remark 4.2. *UpWriting the data presents a parsing problem, as more than one UpWritten version of the data may be possible. We shall see that the parsing problem may be eliminated by UpWriting the data on the fly, as sub-objects are being extracted, when we present the* UpWrite Predictor *in the next chapter.*

Remark 4.3. *The Sub-Object DownWrite is a trivial process, as for each symbol* $y_i \in \mathcal{A}_2$, *a corresponding symbol sequence* $\langle x_1, \ldots, x_j \rangle$, $x_i \in \mathcal{A}_1$ *exists such that* $\langle x_1, \ldots, x_j \rangle \mapsto y_i$. *We may DownWrite the data* $t_k = y_1, \ldots, y_k$, $y_i \in \mathcal{A}_2$ *by replacing each symbol* y_i *in the data with the symbol sequence* $\langle x_1, \ldots, x_j \rangle$.

Example 4.3. *Let* $\mathcal{A}_1$ *be the alphabet of ASCII characters, and* s_z *be the Sherlock corpus. Let* $\mathcal{A}_2$ *be the alphabet which contains a symbol for every distinct alphanumeric substring in* s_z, *and every distinct non-alphanumeric substring in* s_z. *This defines a Sub-Object UpWrite from an alphabet of characters to an alphabet of words (*i.e. *strings of alphanumeric characters) and non-words (*i.e. *strings of whitespace, punctuation and so on). The symbol sequence* t_k, *representing the UpWritten version of the Sherlock corpus, corresponds to the Sherlock corpus at the word level.*

Sub-Objects may exist at higher levels of representation than the word level in natural language text, due to the fact that sequences of words form phrases, clichés, sentences, paragraphs, chapters *et cetera*. Furthermore, it is not clear that the word level should be the first level of UpWrite performed—it is likely that morphemes will be discovered before we find the words themselves.

4.3.2 The Quotient-Object UpWrite

The *Quotient-Object UpWrite* is based on the notion that symbols may be assigned to equivalence classes such that these equivalence classes correspond to the primitives of some higher level representation of the data. For example, a quotient-object of 'things' in an image may correspond to the class of straight line segments, while a quotient object of words in natural language text may correspond to the class of verbs.

Incorporating quotient-objects into a model has the advantage of making the most out of the data available, as a single context of quotient-objects corresponds to many contexts of lower level objects, and therefore will be observed more frequently in the data. For example, consider figure 4.3, which shows a string of words UpWritten to a string of word classes, where each class contains two words. It is obvious that an n-gram model inferred from the UpWritten version of the data will observe each context of sub-objects at least as frequently as all possible contexts of words which are UpWritten to that context.

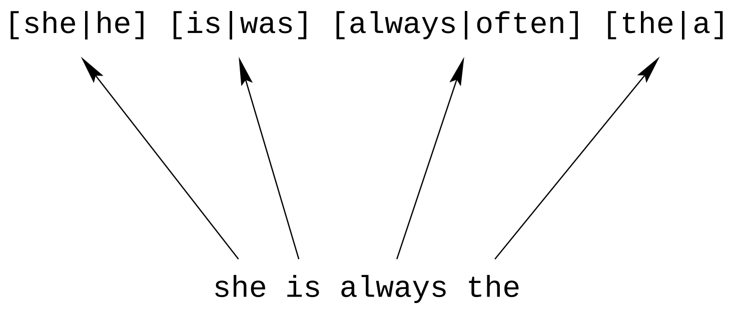

FIGURE 4.3: The Quotient-Object UpWrite results in contexts which are observed more frequently.

We shall often informally refer to quotient-objects as *symbol classes*.

Definition 4.3. *The* Quotient-Object UpWrite *is a mapping* $\{x_1, \dots, x_j\} \mapsto y_i$ *between an equivalence class of symbols* $\{x_1, \dots, x_j\}$ *taken from some alphabet* $\mathcal{A}_1$ *to a symbol* y_i *of some new alphabet* $\mathcal{A}_2$ *such that the observed data* $s_z = x_1, \dots, x_z$, $x_i \in \mathcal{A}_1$ *can be represented as a higher level symbolic time series* $t_z = y_1, \dots, y_z$, $y_i \in \mathcal{A}_2$.

Remark 4.4. *The* Quotient-Object DownWrite *is not unique, since it is not possible to determine which lower level symbol* x_i *should serve as the DownWritten version of the* *symbol y_i. It is possible to generate possible DownWritten versions of t_z by selecting a symbol $x_i \in \{x_1, \dots, x_j\}$ at random. The corresponding DownWritten version of t_z, which is not unique, has the property that its UpWrite is t_z, and the DownWrite therefore provides an indication of how well the UpWrite has captured salient features of the data being modelled.*

Remark 4.5. *We shall find it useful to consider the equivalence class $\{x_1, \dots, x_j\} \mapsto y_i$ to be stochastic—that is, to associate a non-zero probability $P(x_i \in \{x_1, \dots, x_j\})$ with each symbol in the equivalence class. In this case, the* Quotient-Object DownWrite *proceeds by selecting a symbol $x_i \in \{x_1, \dots, x_j\}$ at random in accordance with the probability distribution over the equivalence class.*

Example 4.4. *Let $\mathcal{A}_1$ be the alphabet of words and non-words which appear in the Sherlock corpus, as found by the Sub-Object UpWrite of the previous example. Let $\mathcal{A}_2$ be the alphabet which contains a symbol for the class of symbols in $\mathcal{A}_1$ which are alphanumeric at the lowest level, and a symbol for the class of symbols in $\mathcal{A}_1$ which are non-alphanumeric at the lowest level. This defines a Quotient-Object UpWrite from an alphabet of words to an alphabet of word categories, and the UpWritten symbol sequence t_z corresponds to an alternating sequence of quotient-objects which represent words and quotient-objects which represent non-words.*

In practice, we would be more interested if the quotient-objects found in natural language text corresponded to syntactic categories such as noun, verb, article, adjective and so on. In fact, we shall soon show that the quotient-objects found by some techniques tend to be quasi-semantic in nature, and it is not uncommon to find classes containing proper names, inanimate objects, verbs describing human motion and so on.

4.3.3 Discovering Sub-Objects and Quotient-Objects

We shall devote a considerable portion of the next chapter to the problem of finding sub-objects and quotient-objects in symbolic time series. In the meantime, we shall briefly discuss how these problems may be solved in a general sense.

A sub-object may be found by observing that a particular set of objects are strongly correlated with one another. In a symbolic time series, for example, a sequence of symbols which occurs frequently would be a candidate sub-object. Obviously some sort of model is needed to measure the correlation between objects, and the UpWrite requires that a local model of the data be used. This requirement is made partially due to the fact that it enables parallel processing of the data, as many processing units can model different portions of the data independently, partially due to the fact that such models are typically easy to infer from data, and partially due to the fact that the UpWrite itself will extend the reach of each of these local models by constructing a hierarchical representation of the data.

A quotient-object may be found by observing that one object may be replaced with another in a particular context without adversely affecting the power of the local model to describe the data. The two objects may then be considered to be equivalent in that the performance of the local model upon encountering either of the objects is similar.

The UpWrite has a few disadvantages. The first of these is the necessity to train the model in order for it to be of any use, although this requirement is not necessarily a bad thing. The second disadvantage is due to the fact that the UpWrite constructs a hierarchical model of the data. Optimizing the hierarchical model over all levels of the hierarchy is computationally intractable, and therefore the UpWrite from one level of the hierarchy to the next, higher level, tends to be greedy, in that the performance of the model between the levels is optimized, to the possible detriment of the model overall.

4.4 An Example: Classifying Polygons

We shall now give an example of the UpWrite in action. This example is based on the work of Robert McLaughlin and Michael Alder [5,17,19,20,22], and is purely speculative in that it is not the result of an actual computational implementation.

Consider a pattern recognition system which is required to determine whether a particular image of a polygon is a triangle or a square. To make its job easy, we assume that the images are binary, that exactly one polygon appears in each image, and that there are no noise effects apart from those due to the discretization process.

Clearly this pattern recognition problem borders on the trivial. However, we shall demonstrate that the UpWrite solves it elegantly, and avoids the use of heuristics entirely, meaning that the solution is general enough to be applied to quite different pattern recognition problems which are considerably more complicated.

In order to solve this problem using the UpWrite, we require

- a model which describes the data locally at various levels of abstraction;
- a process for finding line segments and vertices from local descriptors of the pixels in an image;
- a process for finding squares and triangles from the local descriptors of line segments and vertices; and
- a classification scheme which is capable of deciding whether a hitherto unseen polygon is a square, a triangle or something else entirely.

The reader may wish to pause and consider how a problem such as this would be solved traditionally. It is very likely that a traditional solution would be specifically tailored to look for lines, triangles and squares in images, and would therefore be fairly specific to this particular problem. The usual method employed by scientists and engineers when faced with a problem such as this is to inspect the images by eye, and decide, via introspection, which features of the image allow for their classification, to encode these features in some manner, perhaps by rules and tests, and finally to apply a standard technique for classification. This precludes automation, and restricts the generality of the approach.

It will be shown that the UpWrite is able to find lines, triangles and squares in a very general sense, and does so without human intervention. The UpWrite therefore suffers fewer restrictions than more traditional approaches to pattern recognition.

4.4.1 Selecting a Local Model

The UpWrite is a process which represents data at different levels of abstraction, and which does so automatically. This is achieved by modelling the data locally, at each level of representation, and by combining these local models in various ways to find higher level structure, a bootstrapping process which uncovers global structure at its highest level.

Consider the image of figure 4.4. We shall assume that this image is expressed as a bitmap of black and white pixels, as it was in figure 4.1. The lowest level description of the triangle is therefore a two-dimensional array of symbols taken from the alphabet $\{0, 1\}$, where 0 represents a white pixel and 1 represents a black pixel. For classification of various images to be possible, we need to abstract this data in a way which decreases its degrees of freedom while preserving its structure.

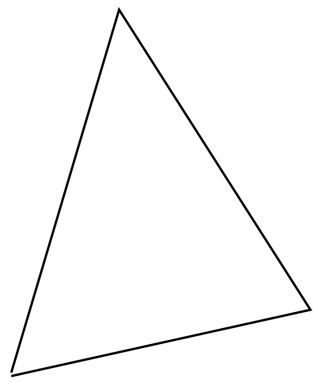

FIGURE 4.4: A line drawing of the triangle of figure 4.1.

In order to extract higher level structure from this data, we require a model which is capable of describing a local region of pixels in a way which allows us to measure the correlation between two adjacent regions in order to decide whether they are part of the same higher level object. We would like to measure this correlation in an information theoretic way; by predicting what we expect to see nearby a particular neighbourhood of pixels. Local *Gaussian Models* possess these necessary features. Although the choice of Local Gaussian Models is not ideal, we do not know of a better alternative at the moment.

4.4.2 Local Gaussian Modelling

A local *Gaussian Model* abstracts the data by calculating the normalized 1^{st}– and 2^{nd}–order moments for local neighbourhoods of pixels. This is equivalent to considering the black pixels in the image to be vectors in $\mathbb{R}^2$, and calculating the centroid and covariance matrix of a local neighbourhood of these vectors. The resulting local model, which may be represented as a vector in the UpWrite space $\mathbb{R}^5$, may be shown graphically as an ellipse positioned at the centroid, with an orientation determined by the major eigenvector of the covariance matrix, and a width and length determined by the two eigenvalues.

Figure 4.5 shows the result of using local Gaussian Models to describe the data. The triangle has been UpWritten from a lower level representation of 55 vectors in $\mathbb{R}^2$, corresponding to the black pixels of figure 4.1, to 16 vectors in $\mathbb{R}^5$.

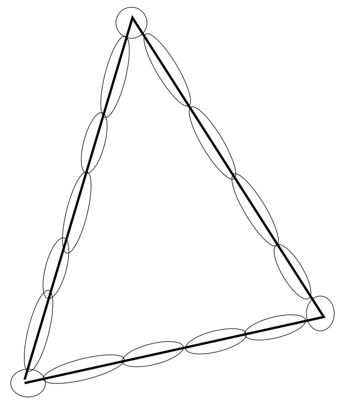

FIGURE 4.5: A line drawing of a triangle with local Gaussian Models superimposed.

This first level of UpWrite has compressed the representation of the triangle by using 80 numbers to describe it instead of 110 numbers. It has also uncovered some structure—we observe that the ellipses positioned along line segments tend to be long and thin, while those positioned at the vertices tend to be quite circular. This first level of UpWrite is an example of the Sub-Object UpWrite, as we have mapped a set of pixels into a single higher level Local Gaussian Model.

Gaussian Models may be used as predictors. We can, for example, calculate the probability of a particular symbol being black or white using a Gaussian Model, and this enables us to DownWrite the image by colouring each pixel a shade of grey according to the probability assigned to it by the model. This process, if applied to the current model, would produce a triangular fuzz. DownWriting the higher level representation of the data allows us to confirm that the structure of the data which we consider to be important has been preserved.

4.4.3 Finding Lines and Vertices

The triangle is now represented by 16 vectors in $\mathbb{R}^5$. A second level of UpWrite may be performed in order to carry this abstract representation of the triangle further.

As we previously observed, ellipses which lie along line segments tend to be long and thin. If they lie on the same line segment, they have similar orientations, and they are spatially nearby. We can therefore find line segments in the data in at least two different ways.

Cluster the vectors: A clustering algorithm may be applied to find clusters of vectors which are similar according to some metric. Vectors which are assigned to the same cluster may turn out to belong to the same line segment.

Entropic chunking: A sequence of local Gaussian Models may be used to predict what the next model in the sequence should look like. If this prediction proves to be erroneous, we have reached the end of some region of higher level structure. Until then, we join the sequence of models together. This approach may find line segments, and even curved lines depending on the tolerance threshold used to decide whether a prediction is erroneous or not.

Figure 4.6 shows the results of this second level of UpWrite, which has successfully identified the three line segments in the image. This second stage of UpWrite is another example of the Sub-Object UpWrite; we have mapped a sequence of local models into a single Gaussian Model at a higher level of abstraction.

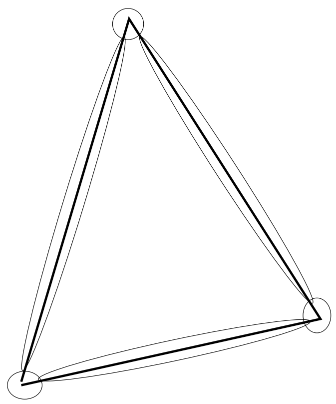

FIGURE 4.6: A line drawing of a triangle with local Gaussian Models describing the lines and vertices.

The vertices of the triangle, represented by circular ellipses in the figure, were not UpWritten at all at this stage of the process. The vertices are areas of high entropy in the image, and may be considered to be separators between line segments.

4.4.4 Finding Triangles and Squares

In order to determine whether a set of vectors in high-dimensional space represents a triangle, a square, or something else entirely, we need to perform one further level of UpWrite; the Quotient-Object UpWrite.

We might intuitively describe a triangle as "three line segments and three vertices" and a square as "four line segments and four vertices". We would therefore like our algorithm to automatically decide that all line segments are the same kind of thing, and all vertices are the same kind of thing. If we are successful in doing this, we will have come full-circle, once again describing our data using a binary alphabet!

Contextual information may be used to classify the vectors which represent line segments and vertices into one of these two classes. We know, for example, that vectors representing vertices tend to occur at either end of a line segment, and this information may be enough to provide a classification. We shall not bother with exactly how the classification process is achieved; we shall simply assume that it has been achieved, and that we are left with a single vector in some high-dimensional space which describes the triangle in a very abstract way.

4.4.5 Classification

If we apply this UpWrite process to lots of different images of triangles, and lots of different images of squares, and if the UpWrite process preserves enough structure about these shapes so that they may be distinguished at the highest level of abstraction, we will find clusters of vectors in this high-dimensional space. Classification is then a trivial process of deciding, via some measure of similarity, which of the two clusters a novel vector belongs to, if any. Our problem of distinguishing triangles from squares is solved.

In the process of solving this trivial pattern recognition problem, we have represented each image at four levels of abstraction. We therefore have quite powerful models of our images which may be applied to many different problems, such as deciding whether a particular shape is the same as one we have already seen but at a different scale, for example. We also have a pattern recognition technique which can classify a new polygon merely by training the system on a collection of these shapes too, and ensuring that the highest level of abstraction is specific enough to allow it to be distinguished satisfactorily from triangles and squares. If not, we would need to change the representation by adding additional constraints.

4.5 Real-World Examples

The example we have presented was trivial, and served only to illustrate in a fairly general way how the principles behind the UpWrite process may be used in a classification problem. Implementation of such an algorithm is not as straightforward as it may have sounded, and many different factors need to be taken into account. Even so, the UpWrite has proved itself to be a powerful, robust pattern recognition technique, and we shall illustrate this by briefly describing two examples of its application to real-world problems.

4.5.1 Identifying Aircraft

Robert McLaughlin, Michael Alder and Christopher deSilva have successfully applied the UpWrite to various image recognition tasks which may initially appear to constitute quite different problems. In 1995 they took a system which had been developed to distinguish between silhouettes of left and right hands [18], and applied it to the problem of identifying various aircraft in infra-red images [18, 21].

The system which classified images of hands into two classes, left hand and right hand, worked by inferring a hierarchical model of a hand by modelling the fingers and palm with low order central moments, and modelling the six vectors which result from this process (corresponding to the five fingers and the palm) as a single vector in some high-dimensional space. This system achieved a 97.4% accuracy when classifying novel images. It was then shown that the same algorithm could successfully distinguish between silhouettes of various aircraft with essentially no modifications—classification of novel images was completely successful in this case.

A similar algorithm was used to classify images of four different aircraft. The images were formed by thresholding an infra-red image of an aircraft model which had been rotated arbitrarily in three dimensions, resulting in a success rate of 71% for delta-wing aircraft and 61% for non-delta-wing aircraft.

4.5.2 Distinguishing Calvin From Hobbes

Sok Gek Lim [14], Michael Alder and Philip Dunstan [8] studied the problem of detecting intruders by classifying moving shapes as recorded by a video camera. Motion was detected by taking the difference between two consecutive video frames, and a border-trace of the resulting difference image was modelled using the UpWrite technique. The system correctly classified images of human heads up to 85% of the time, and correctly classified images which did not contain a human head up to 95% of the time.

The system was applied, with essentially no modifications, to the problem of distinguishing between the comic strip characters Calvin and Hobbes, created by Bill Watterson [13, 14]. Comic panels containing one or other of the characters were subjected to border tracing, and were then modelled via the UpWrite process. Novel images of Calvin were correctly classified 80% of the time, while novel images of Hobbes were correctly classified 99% of the time, using a trivial classification technique of separating the two highest level clusters with a single hyperplane.

Lim and Alder speculate on the features that the system seemed to be using for classification, and they write that [13]

> ... we think the system is picking up Calvin's hair spikes, and possibly Hobbes' whiskers. Such correlations are plainly detectable in principle. Additional evidence on the matter of which features have been extracted from the data is supplied by an examination of the cases where the classification failed. The system has trouble when Calvin wears a hat or has his hair brushed.

The fact that an image classification technique using the UpWrite may be successfully applied to a completely different problem, with no essential changes apart from the low level image processing required to capture the images in the first place, is indicative of its power and generality.

4.5.3 Other Work

The UpWrite is a relatively unknown syntactic pattern recognition technique, and most of the research in this area has been performed within the Centre for Intelligent Information Processing Systems at The University of Western Australia. With the intention of disseminating information about this promising approach to the pattern recognition problem, we would like to draw the reader's attention to the work of Paul Williams [23], Robert McLaughlin [16], Sok Gek Lim [14] and Patrick Hew [12], with the papers of Alder *et. al.* [3–5] providing a rigorous overview of the UpWrite.

4.6 Summary and Conclusion

In this chapter we have introduced the UpWrite, a powerful framework for the hierarchical extraction of syntax from data. We have given a brief history of the UpWrite, illustrated it via a trivial image classification problem, and briefly discussed some real-world examples of its use in image recognition.

We are interested in the problem of constructing a predictive model which automatically adapts to the data it is processing in order to describe it in a way which is efficient, and which preserves the high level features necessary for classification while abstracting low level features which are overly specific in that they adversely effect its performance. In the next chapter we shall show that the UpWrite may be used to increase the power of a predictive model, by detecting structure in the data via information theoretic techniques, and then by incorporating this structure into a higher level version of the model by UpWriting the data. This is fundamentally a bootstrapping process; the model increases the context available to it, making better use of the limited data available, by incorporating structure which it discovers itself.

Notes

[16] In the case of chain coding, for example, a single noisy pixel in an image can drastically effect the resulting description string.

References

[1] Michael Alder, Christopher deSilva, and Yianni Attikiouzel. Automatic knowledge acquisition. Technical Report TR90-14, The Centre for Intelligent Information Processing Systems, Department of Electrical & Electronic Engineering, The University of Western Australia, Australia 6907, 1990.

[2] Michael Alder, Christopher deSilva, and Yianni Attikiouzel. On the automatic generation of higher levels of description. Technical Report TR90-15, The Centre for Intelligent Information Processing Systems, Department of Electrical & Electronic Engineering, The University of Western Australia, Australia 6907, 1990.

[3] Michael D. Alder. Inference of syntax for point sets. In E. S. Gelsema and L. N. Kanal, editors, *Pattern Recognition in Practice IV: Multiple Paradigms, Comparative Studies and Hybrid Systems*, number 16 in Machine Intelligence and Pattern Recognition, pages 45–58. Elsevier Science B.V., June 1994.

[4] Michael D. Alder and Christopher deSilva. Topological stochastic grammars. In *IEEE International Symposium on Information Theory*, July 1994.

[5] Michael D. Alder, Gek Lim, and Christopher J.S. deSilva. The syntax of images. Technical Report TR95-01, The Centre for Intelligent Information Processing Systems, Department of Electrical & Electronic Engineering, The University of Western Australia, Australia 6907, 1995.

[6] Mike Alder. Stochastic grammatical inference. Master's thesis, Department of Mathematics, The University of Western Australia, Australia 6907, 1988.

[7] C.J.S. deSilva, M.D. Alder, and Y. Attikiouzel. Automating knowledge engineering. Technical Report TR90-03, The Centre for Intelligent Information Processing Systems, Department of Electrical & Electronic Engineering, The University of Western Australia, Australia 6907, 1990.

[8] Phillip Dunstan, Gek Lim, and Michael D. Alder. Real-time head detection. In *Proceedings of the IEEE International Conference on Neural Networks*, volume 5, pages 2217–2221, November 1995.

[9] King Sun Fu. *Syntactic methods in pattern recognition.* Academic Press, 1974.

[10] King-Sun Fu and Taylor L. Booth. Grammatical inference: Introduction and survey - part I. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-8(3), May 1986.

[11] King-Sun Fu and Taylor L. Booth. Grammatical inference: Introduction and survey - part II. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-8(3), May 1986.

[12] Patrick Chisan Hew. *Pixels to Strokes to Digits.* PhD thesis, Department of Mathematics, The University of Western Australia, Australia 6907, 1999.

[13] Gek Lim and Michael D. Alder. Calvin and Hobbes. Technical Report TR95-02, The Centre for Intelligent Information Processing Systems, Department of Electrical & Electronic Engineering, The University of Western Australia, Australia 6907, 1995.

[14] Sok Gek Lim. *Visual Object Shape Recognition Using Hierarchical Syntax Extraction.* PhD thesis, Department of Electrical & Electronic Engineering, The University of Western Australia, Australia 6907, 1997.

[15] David Marr. *Vision.* W.H. Freeman and Company, 1982.

[16] Robert A. McLaughlin. *Intelligent algorithms for finding curves and surfaces in real world data.* PhD thesis, Department of Electrical & Electronic Engineering, The University of Western Australia, Australia 6907, 1999.

[17] Robert A. McLaughlin and Michael D. Alder. Syntactic pattern recognition of simple shapes. In *Proceedings of the Australian and New Zealand Conference on Intelligent Information Systems*, pages 5–9, 1993.

[18] Robert A. McLaughlin and Michael D. Alder. Recognising aircraft: Automatic extraction of structure by layers of quadratic neural nets. In *Proceedings of the IEEE International Conference on Neural Networks*, volume 7, pages 4288–4293, June 1994.

[19] Robert A. McLaughlin and Michael D. Alder. Recognising cubes in images. In E. S. Gelsema and L. N. Kanal, editors, *Pattern Recognition in Practice IV: Multiple Paradigms, Comparative Studies and Hybrid Systems*, number 16, pages 45–58. Elsevier Science B.V., June 1994.

[20] Robert A. McLaughlin and Michael D. Alder. The Hough Transform and the UpWrite: a comparison. In *Proceedings of the IEEE International Conference on Neural Networks*, volume 1, pages 146–151, 1995.

[21] Robert A. McLaughlin and Michael D. Alder. Recognition of infra red images of aircraft rotated in three dimensions. In *Proceedings of the Australian and New Zealand Conference on Intelligent Information Systems*, pages 82–87, November 1995.

[22] Robert A. McLaughlin and Michael D. Alder. The Hough Transform versus the Up-Write. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(4):396–400, April 1998.

[23] Paul S. Williams. *The automatic hierarchical decomposition of images into sub-images for use in image recognition and classification.* PhD thesis, Department of Electrical & Electronic Engineering, The University of Western Australia, Australia 6907, 1999.

Chapter 5

Design and Implementation of the UpWrite Predictor

"I daresay my face looked as bewildered as yours did just now when first I read this message. Then I reread it very carefully. It was evidently as I had thought, and some secret meaning must lie buried in this strange combination of words."

The Memoirs of Sherlock Holmes
SIR ARTHUR CONAN DOYLE

5.1 Introduction

We shall now apply the UpWrite concept to symbolic time series, by modelling the data using a simple predictive model, and using measures from Information Theory to discover symbol sequences and symbol classes which constitute higher level structure in the data. This structure may then be UpWritten to the root symbols of a higher level alphabet; a process which creates a new, UpWritten form of the symbolic time series.

Another predictive model may then be inferred from this alternative representation of the data, and the UpWrite process repeated to an even higher level. This produces a family of predictors, each of which operate at different levels of abstraction of the original data. This family of predictors, collectively referred to as the *UpWrite Predictor*, may then be applied to various problems.

5.1.1 Overview

In this chapter we specify our design of the UpWrite Predictor, and explain how two types of structure, symbol sequences and symbol classes, may be found in symbolic time series via the application of information theoretic measures to the sequence of predictions made by a predictive model. This is followed by a description of our implementation of the

UpWrite Predictor, and a discussion of the various decisions which had to be made during its implementation.

5.2 The UpWrite Predictor

Our aim in developing the UpWrite Predictor is to construct a modelling technique which is generic enough to find structure in all sorts of symbolic time series, and which is able to exploit this structure in order to improve its own performance. We hope to achieve this by applying information theoretic measures to the predictions made by a simple predictive model in order to find structure in the data, and then by UpWriting the data with respect to the structure discovered. We would like the UpWrite Predictor to be capable of being used in a wide range of applications, including data compression, speech recognition, and various natural language processing tasks. In order to achieve this, we require that

- it should make as few assumptions as possible about the data;
- the only information available to the predictive model should be the data itself, and the alphabet that it is represented in;
- it should not rely too heavily on a particular form of predictive model;
- the predictive model used in the UpWrite Predictor should not make assumptions about a particular application;
- it should be possible to use the UpWrite Predictor adaptively;[17] and
- *ad hoc* methods should be avoided wherever possible.

Figure 5.1 shows the proposed structure of the UpWrite Predictor. It consists of a chain of Alphabet-Model-UpWriter modules which are connected together by the data emitted and the predictions made by the previous module in the chain.[18] Each module represents a higher level of abstraction of the data and all of the modules, when taken together, constitute a single hierarchical predictive model of the data.

Each model in the UpWrite Predictor is a simple predictive model, inferred from the history available to it, and capable of making predictions about the next symbol in the data in the form of a probability distribution over the alphabet. The UpWriter applies information theoretic measures to the sequence of predictions made by the previous module in the chain in order to extract two sorts of structure, symbol sequences (sub-objects) and symbol classes (quotient-objects), from the data emitted by the previous module. This structure is added to the alphabet of the current module, and the predictive model in that module is inferred from the UpWritten data that results. The predictions made by the predictive model are fed back into the UpWriter, so that it may correct any mistakes it may have made during the UpWrite process. It should be noted that parsing and UpWriting

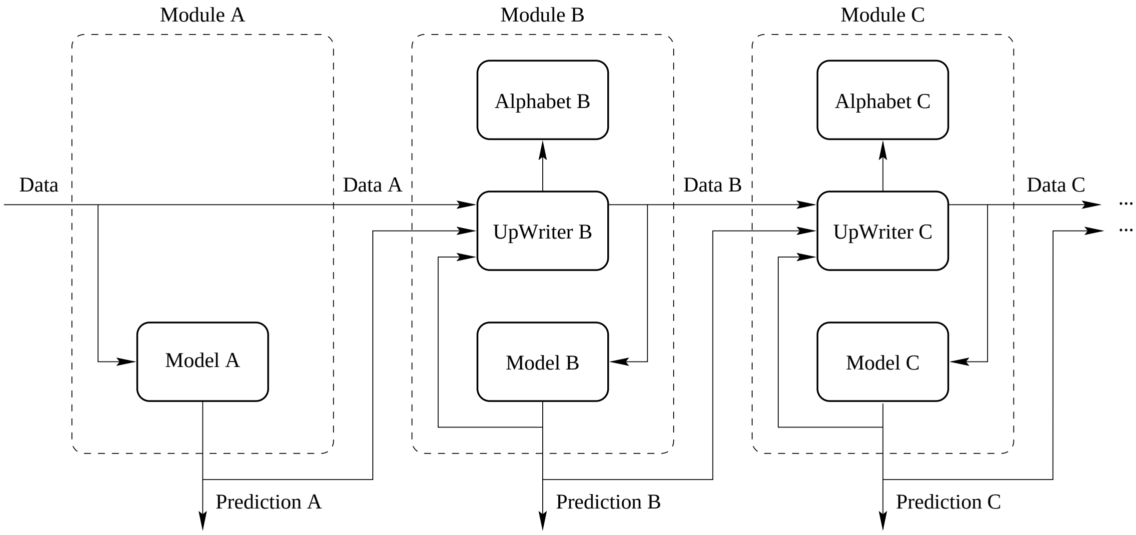

FIGURE 5.1: The proposed structure of the UpWrite Predictor.

are the same process, as the data available to each predictive model is the instantaneous output of the UpWriter in that module.

Each module is also capable of DownWriting, since its alphabet contains a mapping between the symbols in it and the symbol sequences and symbol classes which they correspond to.[19] More importantly, it is capable of DownWriting its predictions, which is imperative if the UpWrite Predictor is to be used for various real-world applications. For example, a data compressor may require that the predictions of the various modules are DownWritten to predictions over the lowest level alphabet, and a spell-checker certainly would.[20]

Having given the design of the UpWrite Predictor, we must now state that better techniques exist for discovering symbol sequences and symbol classes in data than simply looking for patterns in the predictions made by predictive models. Information about which symbols can occur between certain pairs of symbols may make the task of finding symbol classes considerably easier, for instance. We shall discuss other techniques for finding structure in data, but we shall not incorporate these into our implementation, although this certainly would be possible, because we are more interested in developing a modelling system which can augment itself with structure found from its own predictions than we are with optimizing the performance of each of the individual components of such a system. Our reason for handicapping ourselves in this way is to show that generic information processing mechanisms can discover quite specific structures in data, without having to be designed to do so.

5.3 Discovering Symbol Sequences

One feature of many types of data is that certain sub-objects of symbols, which we informally refer to as *symbol sequences*, occur with high frequency. For instance, in English text, at the character level, the character sequence $\langle \mathtt{t}, \mathtt{h}, \mathtt{e} \rangle$ occurs much more frequently than the character sequence $\langle \mathtt{q}, \mathtt{x}, \mathtt{y} \rangle$. Forming a new symbol out of frequently occurring symbol sequences has the advantages of increasing the context available to the predictive model, and allowing further structure to be uncovered at a higher level by providing a form of scaffolding which enables a bootstrapping process.

We are therefore interested in the problem of automatically discovering such symbol sequences in data, and, due to our interest in the UpWrite Predictor, we wish to develop techniques for using the predictions of simple predictive models to uncover this structure.

The process of finding symbol sequences in symbolic time series is often referred to as *chunking*, in which case the symbol sequences themselves are known as *chunks*.[21] When the data is natural language text, or natural language utterances, the task is commonly referred to as *segmentation*.

The instantaneous entropy of a predictive model is generally recognised as a good indicator of where the boundary between two words lies in natural language text. The points where the uncertainty of the model as defined by its instantaneous entropy is high generally correspond to boundaries between two chunks. We may therefore discover symbol sequences in data by thresholding the instantaneous entropy of the predictive model. We refer to this technique as *thresholded entropic chunking*.

The information provided to the predictive model by the next symbol in the data is a good indicator of whether or not the symbol is part of the current chunk or the beginning of a new chunk. Symbols which are deemed to be part of the current chunk may be paired with the most recent symbol in the context to form a symbol pair in a process we dub *agglutination*.

We shall present a brief overview of this area of research before describing how the two techniques of thresholded entropic chunking and agglutination work. We shall also introduce a third third technique which discovers symbol sequences in data by finding separator symbols, such as the space character in English text.

5.3.1 Performance Measures

In order to evaluate the various techniques for discovering symbol sequences which we shall present in this section, we introduce three measures: *recall*, *accuracy* and *coverage*, as defined in equations 5.1, 5.2, and 5.3. In these equations $\mathcal{C}$ represents the alphabet of symbol sequences found by the algorithm being evaluated, $\mathcal{W}$ represents the set of 'correct' symbol sequences, as determined by a human being, and $\mathcal{S}$ represents the *multiset* of symbol sequences which form the data used for testing. A multiset $\mathcal{S}$ associates a count with its elements, and $|\mathcal{S}|$ is defined as the sum of these counts, therefore giving the number of 'correct' symbol sequences in the testing data.

The measures of accuracy and recall have previously been used to evaluate the success of algorithms for discovering sequential structure in language [15].

$$recall = \frac{|\mathcal{S} \cap \mathcal{C}|}{|\mathcal{S}|} \tag{5.1}$$

$$accuracy = \frac{|\mathcal{W} \cap \mathcal{C}|}{|\mathcal{C}|} \tag{5.2}$$

$$coverage = \frac{|\mathcal{W} \cap \mathcal{C}|}{|\mathcal{W}|} \tag{5.3}$$

The recall measures the proportion of symbol sequences in the testing data which were correctly identified by the algorithm, the accuracy measures the proportion of the symbol sequences found by the algorithm which are correct, and the coverage is the proportion of correct symbol sequences which were found by the algorithm.

5.3.2 Previous Work

There has been a small amount of work performed on the automatic segmentation of utterances, particularly in the study of language acquisition in infants. Zellig Harris[22] is probably the most notable person to have pursued this line of research [5], and Gerry Wolff has studied the segmentation of natural and artificial languages [25,26]. Harris applied an *ad-hoc* technique which approximates thresholded entropic chunking, while Wolff applied an *ad-hoc* technique which approximates agglutination.

There has also been some work that gives biological evidence of these processes. Hayes and Clark showed that human subjects could identify word boundaries in artificial speech after a short period of listening to it [7], while Kutas and Hillyard published results that seem to indicate that the human brain experiences 'surprise' when listening to sentences which violate expectations [11], suggesting something like an information theoretic model implemented in the central nervous system.

Zellig Harris

The famous structural linguist Zellig Harris showed as early as 1955 that it is possible to locate word boundaries in utterances via a stochastic process [5]. Harris achieved this by measuring the number of different phonemes which could occur after the first n phonemes of an utterance, and segmenting the utterance when this *successor count* reached a peak. He stated that the fact that such a technique worked

> ... would suggest to us that sentences can be segmented into morphemic elements, even if we did now know beforehand that such elements exist in language.

Like Shannon a few years earlier [22], Harris realised that the corpus required to gather these statistics would be prohibitively large, and that the best source of data was therefore a human informant. He did not consider restricting the context to a local one—all of the phonemes, from the beginning of the utterance onwards, were used for segmentation.

We are not aware of any modern computational implementation of Harris' technique, so we have written one ourselves. In order to do this, we needed to introduce the concept of a local context in order to guarantee that our estimate of the successor count is based on a sufficient number of observations. We also needed to rigorously define what constitutes a 'peak' in the successor count—it appears that Harris performed the segmentation by eye. We define a peak as a maximum value which occurs between two minimum values such that the smallest difference between the maximum value and the two minimum values exceeds three. In cases where two or more maximum values exist[23], the rightmost one is chosen. This definition serves to eliminate errors due to small fluctuations in the successor count.

We performed two experiments; one on the Sherlock corpus, and one on the SHERLOCK corpus. In both cases the entire corpus was used to infer the successor counts of each local context, and results were gathered over the `Test` section of each corpus. In order to present Harris' technique in the best light, we chose to perform the experiment for a range of context lengths, and we present results for the context length which performed the best.[24] Curiously, this optimal length was 1 for the Sherlock corpus, and 5 for the SHERLOCK corpus. We believe this is due to the fact that whether or not the previous symbol in the data was whitespace is sufficient for segmentation, but, when no whitespace exists, more contextual information needs to be taken into account.[25]

The experiments were conducted by breaking the corpora into their constituent sentences, and pre-pending sufficient 'null' symbols to each sentence in order to provide sufficient context for the first non-null symbol. A Markov model of the required context length was inferred from this data, and used to segment the `Test` section of each corpus in accordance with Harris' algorithm. Results of these experiments are presented in table 5.1. Figures 5.2 and 5.3 plot the successor count over the first sentence of the corpora, illustrating that peaks in the count do indeed correspond to word boundaries.

Corpus	**Recall**	**Accuracy**	**Coverage**
Sherlock	75.00%	51.67%	65.96%
SHERLOCK	45.89%	23.26%	30.53%

TABLE 5.1: Results of Harris' segmentation algorithm.

We decline analysis of these results, as our intention is not to criticize or improve upon Harris' algorithm, but merely to illustrate that it works. With the benefit of hindsight, it is plainly obvious that Harris was approximating the instantaneous entropy of a predictive model, since the entropy is related to the number of different symbols which can occur next.[26] We shall show that genuine thresholded entropic chunking yields superior results.

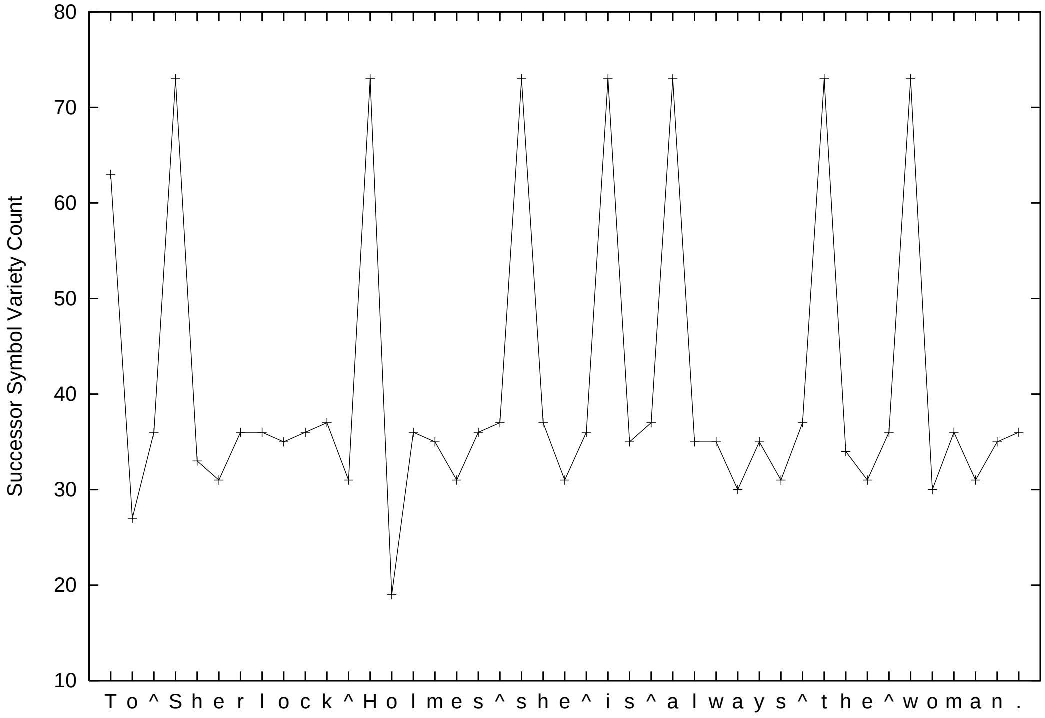

FIGURE 5.2: Successor count over the `Sentence` section of the Sherlock corpus, for a context of 1 symbol.

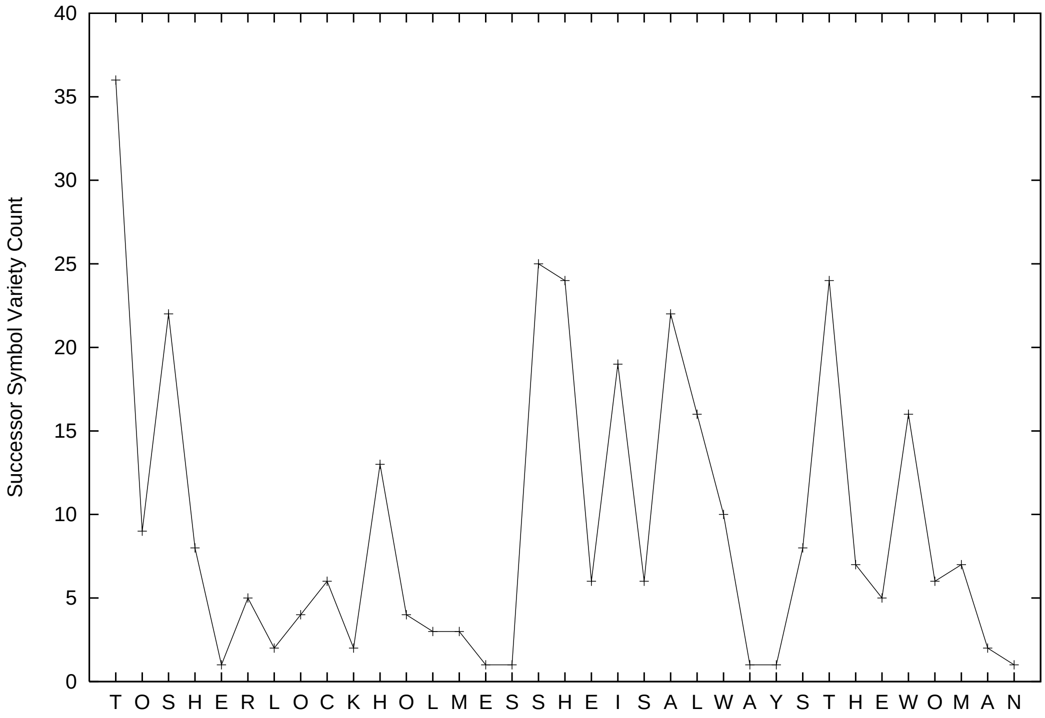

FIGURE 5.3: Successor count over the `SENTENCE` section of the SHERLOCK corpus, for a context of 5 symbols.

Kutas and Hillyard

It is widely accepted that language comprehension involves the continual formulation and update of hypotheses concerning which words are likely to appear in the immediate future—this is the act of prediction. Marta Kutas and Steven Hillyard tested this theory of language comprehension by measuring event-related brain potentials in human subjects [11]. They state that

> ... extensive human research [has shown] that certain components recorded from the scalp are sensitive to a person's expectations. In particular, unexpected or surprising stimuli are typically followed after some 300 to 600 msec by a positive ERP component known as P300.

Their experiment involved asking subjects to read 160 different seven-word sentences a word at a time. A random 25% of these sentences ended in a semantically inappropriate word, although all of the sentences were syntactically correct. It was found that the semantically deviant sentences typically elicited a large negative component after the final word in the sentence had been read.

It is indeed encouraging that the notion of 'surprise' is detectable in human brain function, and it is tempting to take this as evidence that the idea of segmentation via thresholded entropic chunking is fundamentally correct [8].

Hayes and Clark

John Hayes and Herbert Clark studied the ability of the human brain to segment auditory data via a chunking mechanism (which they refer to, more properly, as clustering [7]). Their interest in this process began with observation of visual phenomenon, and they write that

> ... even to a careful observer, an animal who remains motionless may merge perfectly into its background, but it will be quite visible as soon as it moves. When it moves, the correlations among the elements within its visual boundaries are much stronger than correlations between it and its surroundings. Given a clustering mechanism, the same difference in correlation would establish the boundaries of auditory objects, that is, words. We very much prefer to appeal to such general cognitive processes in explaining linguistic phenomena than to more special processes which may apply uniquely to language.

We very much agree with the final sentence of the above quotation, and we find it interesting that the study of visual cognition inspired Hayes and Clark, as it did the UpWrite.

Hayes and Clark devised a set of phonemes by generating complex sounds in three or four voices on a computer, and modulating these sounds in various ways. These phonemes were divided into vowels and consonants, depending upon the degree of variation between the beginning and end of the phoneme (vowels had little variation), and these were used to create a small set of words, of between 6 and 8 phonemes each, which began with a consonant, and thereafter alternated between vowel and consonant. Subjects listened to a continuous stream of sound, formed by concatenating these words at random, for approximately 45 minutes. Subsequent testing revealed that the subjects were able to recognise a word sequence as familiar more readily if pauses were inserted into the stream at the appropriate places.

This result suggests that the human brain is capable of segmenting unfamiliar data, even in the absence of further information about what constitutes a valid symbol sequence. In order to obtain a subjective view of the process, Hayes and Clark participated in their experiment themselves, and reported that

> ... at first, the sound stream seems quite amorphorous and featureless. After perhaps a minute of listening, an event—perhaps a phoneme or part of a phoneme—stands out from the stream. When the event has recurred several times, the listener may notice that it is typically preceded or followed by another event. The combination of events can in turn be related to events that happen in its neighborhood. Recognition of a word, then, seems to proceed from perceptually distinctive foci outwards in both directions towards the word boundaries. Presumably, the process would tend to stop at the word boundaries because the correlations across the boundaries are weak.

Human beings are capable of detecting patterns in other sources of auditory data, such as the recurrent themes in a fugue, and this ability, along with the result of Hayes and Clark, suggests that a general segmentation mechanism must be present in the human brain, even in situations where a referent is lacking.[27] The subject's eye view of the experiment given by Hayes and Clark is suggestive of a process of agglutination.

Gerry Wolff

Gerry Wolff developed a computational technique for the segmentation of artificial text in an attempt to explain the results of Hayes and Clark. [25] He later applied his technique to the segmentation of natural language [26]. Although Wolff was aware of the work of Zellig Harris, he approached the task of segmentation as one of agglutination as opposed to thresholding the value of some measure.

Wolff's algorithm, MK10, proceeds by scanning over symbolic time series, keeping a count of the number of times each pair of adjacent symbols appears. Whenever the count of a particular pair exceeds some predetermined threshold, the pair is added to the alphabet as a new symbol, all counts are cleared, and the process is repeated, in a process akin to the UpWrite [25].

Each symbol pair is stored along with the number of symbols which needed to be scanned in order to form it. This provides a basic measure of the strength of association between the symbols in the pair, and allows each symbol pair to be decomposed hierarchically[28] and viewed as a dendrogram.

Figures 5.4 and 5.5 show hierarchical decompositions of the first sentence of both the Sherlock and SHERLOCK corpora, as formed by our implementation of Wolff's algorithm, plotted with a logarithmic vertical axis.[29] Note that, in accordance with the MK10 algorithm, new symbols were formed out of symbol pairs which had been seen 10 times. Increasing this tolerance may produce superior results, but optimization of Wolff's algorithm is beyond the scope of this dissertation. In both figures it is apparent that words have been found at various levels of the hierarchy, but it is not obvious where the resulting dendrogram should be cut in order to segment the data.

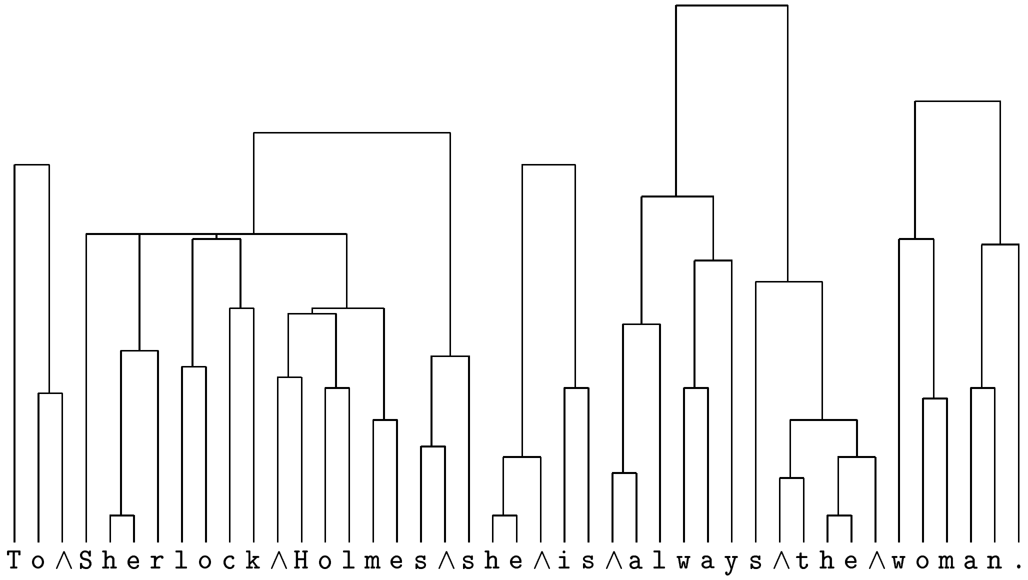

FIGURE 5.4: Dendrogram formed by Wolff's algorithm over the `Sentence` section of the Sherlock corpus.

In a later paper, Wolff described the MK10H algorithm, which counts the occurrence of each adjacent symbol pair over an entire corpus, and forms a new symbol out of the most frequent pair, choosing arbitrarily if there are several such pairs [26]. This algorithm avoids the problem of having to select a tolerance threshold, but, on the other hand, it does not seem capable of providing a correlation measure, apart from the frequency with which the pair occurred.

In table 5.2 we present the results of segmenting the `Test` section of both the Sherlock and SHERLOCK corpora using Wolff's MK10 algorithm. Segmentation was performed by cutting the dendrogram at the point which maximised the recall measure.

FIGURE 5.5: Dendrogram formed by Wolff's algorithm over the SENTENCE section of the SHERLOCK corpus.

Corpus	Recall	Accuracy	Coverage
Sherlock	32.69%	16.75%	23.40%
SHERLOCK	20.29%	16.28%	21.88%

TABLE 5.2: Results of Wolff's agglutination algorithm.

Results indicate that although Wolff's algorithm does not segment the data as well as Harris', it does have the advantage of providing a hierarchical view of the correlation between various parts of the data. It is surprising that the algorithm performs poorly on the Sherlock corpus—this seems to be due to the fact that the whitespace character has a strong correlation with the characters that begin and end words, and is therefore glued fairly arbitrarily to either the beginning or the end of a word fairly on in the process. This much is apparent from the dendrogram of figure 5.4.

Wolff's algorithm is similar to our proposed technique of forming symbol sequences via agglutination, as the information provided to the predictive model by the next symbol in the data, contextual on the current symbol, is inversely proportional to the number of times the corresponding symbol pair has been observed. We shall show that the process of agglutination yields superior results.

Other Work

Several other researchers have pursued the problem of segmentation within the overall framework of attempting to understand how language is acquired, and many of these have attempted to build computational models of the process. We acknowledge the work of Hideki Kozima [10], Jenny Saffran, Elissa Newport and Richard Aslin [20], Walter Stolz [24], Doug Beeferman, Adam Berger and John Lafferty [1], Jeffrey Reynar and Adwait Ratnaparkhi [19], Ian Witten and Craig Nevill-Manning [15,16] and Steven Finch [4].

5.3.3 Algorithms for Discovering Symbol Sequences

We shall present three techniques which may be used to discover symbol sequences in data using nothing more than the sequence of predictions made by a simple predictive model. The first of these is an obvious solution to the problem which may be applicable in certain circumstances, while the remaining two are rigorous versions of the the algorithms of Zellig Harris and Gerry Wolff respectively,

The identification of separator symbols: In written English, whitespace is used to separate words from one another, and various forms of punctuation are used to separate structure at higher levels. Accurate identification of such separator symbols renders the problem of finding symbol sequences trivial. A predictive model may identify separator symbols by observing which symbols cause it the greatest amount of uncertainty about what is coming next.

Segmentation: When separator symbols do not exist in the data, more sophisticated techniques are required to uncover symbol sequences. The first of these, segmentation, searches for boundaries between two symbol sequences—regions of the data where a separator symbol may reasonably be inserted. A predictive model may be used to determine such regions by segmenting its instantaneous entropy whenever this exceeds a predetermined value.

Agglutination: Rather than searching for boundaries between symbol sequences, we may equally well search for symbol pairs which form parts of the same symbol sequence. Symbol sequences may then be formed by gluing such symbol pairs together progressively. The information provided to the predictive model by the next symbol in the data may be regarded as a measure of independence between the most recent symbol in the context and the symbol which follows it. The symbol which provides the least amount of information to the predictive model may be glued onto the most recent symbol in the context in order to form a new symbol sequence. This process is tantamount to beginning with a symbolic time series which is completely segmented, with a boundary occurring between every symbol pair in the data, and gradually removing boundaries between symbol pairs in order to progressively uncover symbol sequences.

5.3.4 Identifying Separator Symbols

The segmentation of data is a trivial process when it is already segmented, as is the case with printed English, where whitespace separates words. The automatic discovery of such separator symbols would provide a more robust method of extracting symbol sequences from data than presupposing which symbols function as separators.[30]

In natural language text we find that the entropy of a predictive model peaks immediately upon encountering a separator symbol. This suggests a rather *naïvé* method of determining which symbols these are—we can merely measure the average uncertainty of the predictive model as it scans the data, and mark symbols which result in an abnormally high uncertainty as being candidate separator symbols.

Figure 5.6 plots the ten characters in the Sherlock corpus which cause the highest entropy in a 2^{nd}–order Markov model. It can be seen that the whitespace character $\wedge$ causes a significantly higher entropy than the other characters: $H(\wedge) = 4.59$ bits, while $H(\mathtt{E}) = 3.93$ bits—a difference of 0.66 bits. This is strong evidence that the whitespace character is a separator symbol.

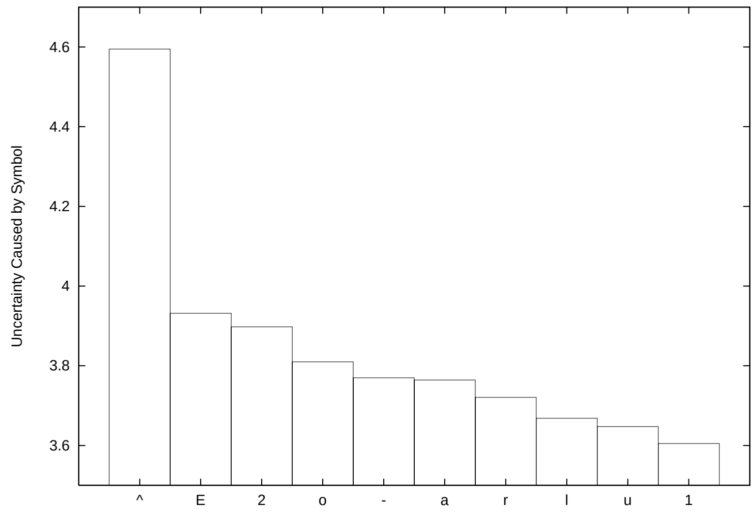

FIGURE 5.6: The ten characters which cause the highest uncertainty in a 2^{nd}–order Markov model over the Sherlock corpus.

The level of uncertainty of a 2^{nd}–order Markov model caused by characters in the SHERLOCK corpus, as shown in figure 5.7, indicates that no separator symbol exists for that text, simply because the difference between the two highest entropies is a mere 0.06 bits. It is interesting to note that in this case the symbols `E`, `D`, `S` and `O`, which are four of the ten symbols which cause the highest uncertainty in the predictive model, are found at the end of 46.71% of the words in the corpus. This is an encouraging vindication of our choice of the instantaneous entropy of a predictive model as the primary indicator of the location of boundaries between symbol sequences in the data.

As we are interested in implementing an UpWrite Predictor which is capable of finding structure in all sorts of data, we shall not use this technique to find symbol sequences. In applications which are concerned only with processing natural language texts, however, a method for discovering separator symbols automatically may be of some use.

5.3.5 Segmentation

The instantaneous entropy of a predictive model fluctuates as it scans across a symbolic time series, and we observe in natural language text that this entropy reaches maximum

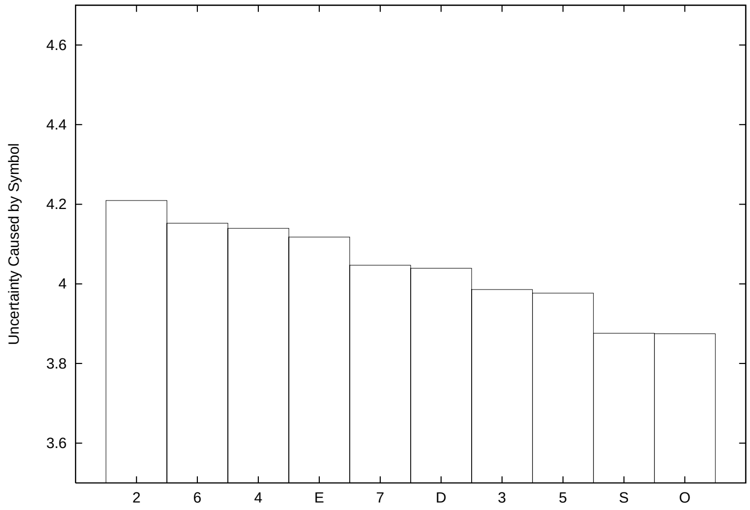

FIGURE 5.7: The ten symbols which cause the highest uncertainty in a 2^{nd}–order Markov model over the SHERLOCK corpus.

values at the beginning of words. This is illustrated by figure 5.8, which plots the entropy of a 1^{st}–order Markov model, trained on the Sherlock corpus, as it scans across the `Sentence` section of that corpus. It is clear that a well-chosen threshold will result in perfect segmentation of the text.

Word boundaries are not so clearly marked in natural language text which is devoid of whitespace and punctuation, as is apparent from figure 5.9. It is obvious that techniques other than thresholded entropic chunking are required to successfully segment data such as this.

The only problem with discovering symbol sequences via thresholded entropic chunking is that a threshold value needs to be selected *a priori*. If the threshold value is set too low, no symbol sequences will be found; if it is set too high, the entire data will be deemed to constitute a single symbol sequence. We shall not introduce any strategies for estimating good thresholds here, apart from mentioning that, without any additional information about the data, a threshold of $log_2|A|$ has been found to be as good as any other, and would function well for the plot shown in figure 5.8.

Thresholded entropic chunking works remarkably well on natural language text when whitespace is present between words. On data where no such separator symbol is present, though, it is generally impossible to find a single ideal threshold value. It is common for very long anomalous[31] symbol sequences to be extracted, along with symbol sequences which we would consider to be useful, and procedures are then required for identifying and rejecting anomalous symbol sequences. The feedback mechanism present in the UpWrite Predictor, illustrated in figure 5.1, enables such a process.

Table 5.3 summarises the results of various experiments performed on the two Sherlock corpora. As usual, we have reported the best results without offering any solution to the

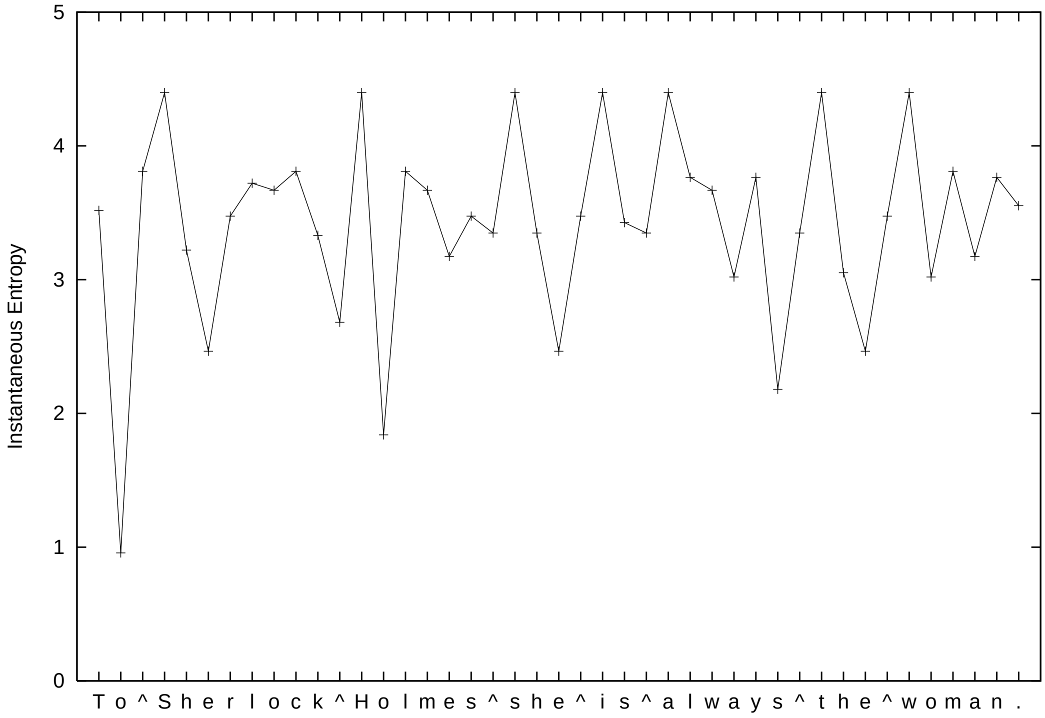

FIGURE 5.8: Instantaneous entropy of a 1^{st}–order Markov model over the `Sentence` section of the Sherlock corpus.

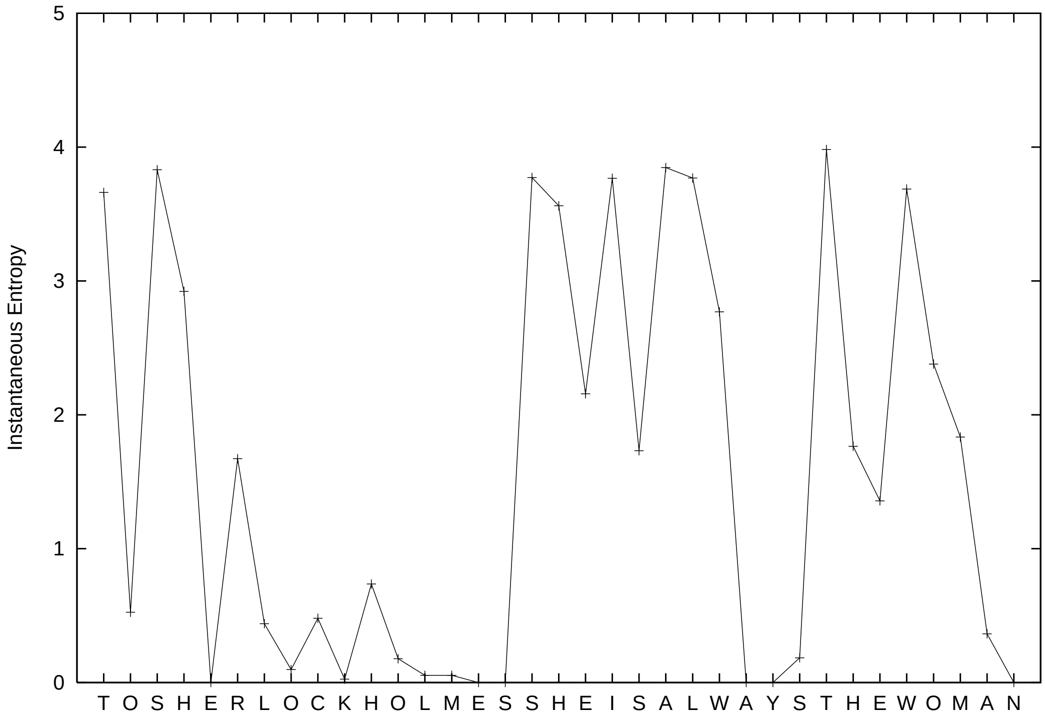

FIGURE 5.9: Instantaneous entropy of a 1^{st}–order Markov model over the `SENTENCE` section of the SHERLOCK corpus.

problem of automatically finding the model parameters to reproduce them. We do this in order to give an upper-bound in performance which can be reasonably expected from the algorithm.

Corpus	Recall	Accuracy	Coverage
Sherlock	100.0%	100.0%	100.0%
SHERLOCK	52.66%	36.18%	41.98%

TABLE 5.3: Results of thresholded entropic chunking.

When segmenting the `Test` section of the Sherlock corpus, a Markov model of order 1 with an entropy threshold of 4 bits was found to produce perfect results. When segmenting the `TEST` section of the SHERLOCK corpus, a Markov model of order 5 with an entropy threshold of 2.5 bits produced the best results. We note that these models take into account the same amount of context as the ones we used to illustrate Harris' algorithm. The results of thresholded entropic chunking, however, are superior.

⟨SHERLOCKHOLMES⟩
⟨ITWAS⟩
⟨MENT⟩⟨ION⟩
⟨QUESTION⟩⟨ABLE⟩
⟨THE⟩⟨YWERE⟩
⟨IT⟩⟨AKE⟩

TABLE 5.4: Some examples of erroneous symbol sequences discovered by thresholded entropic chunking on the `SHERLOCK` corpus.

It is worth considering the nature of the mistakes the algorithm made when segmenting the SHERLOCK corpus. Words, parts of words, and concatenations of words and word-parts were all discovered, and this result is disheartening for the researcher trying to extract words from the data automatically, as it means that words can never be completely recovered via a thresholding process alone. Table 5.4 gives some examples of erroneous symbol sequences formed by the algorithm. The first two of these are common word pairings which have been formed into a single symbol sequence, which is quite reasonable and perhaps even useful. The next two examples are single words which have been split in twain, but, again, the location of the break in the word seems quite reasonable, and the word parts could be considered to be morphemes. The final two examples are truly erroneous—a concatenation of two words has been split at an inappropriate location.

This final error may be attributed to the fact that the algorithm is blind to the context of symbols which follows each hypothesised boundary in the data, and therefore must immediately decide whether to insert a boundary or not, without the advantage of look-ahead. The algorithm greedily forms the symbol sequences ⟨`THE`⟩ and ⟨`IT`⟩, which correspond to common English words, at the expense of forming poor symbol sequences immediately afterwards.

It has been suggested to us that segmentation on the basis of the information provided to the predictive model upon observation of the next symbol in the data may produce superior results. This has been investigated, and it was found that performance was considerably poorer than thresholded entropic chunking. The information measure does have the advantage of taking into account symbols on both sides of the hypothesised boundary, and we shall use it to form symbol sequences in a process of agglutination.

5.3.6 Agglutination

Segmentation via thresholded entropic chunking has the advantage of taking into account as much context as is deemed necessary, but the fact that this context must precede the hypothesized boundary results in various types of unavoidable errors. An agglutination algorithm which takes into account the correlation between adjacent pairs of symbols in the data, and which effectively inserts a boundary between them if their correlation is weak, may avoid these problems. An iterative agglutination algorithm in the style of Wolff's has the additional advantage of gradually extending the range of the context available to the model as more symbol sequences are added to the alphabet.

The information provided to the predictive model by the next symbol in the data may be considered to be a measure of the degree of independence between the most recent symbol in the context and the next symbol, with low values of information signifying a high degree of correlation between these symbols. Figure 5.10 plots the instantaneous information provided to a 1^{st}–order Markov model trained on the entire Sherlock corpus as it makes predictions over the `Sentence` portion of that corpus, with figure 5.11 showing a similar plot for the SHERLOCK corpus. In both cases, it is immediately apparent that symbol sequences cannot be discovered by thresholding the instantaneous information provided to the model by the data.

In figure 5.10, the symbol pair $\langle \mathtt{H}, \mathtt{o} \rangle$ is the most correlated, as the information provided to the predictive model by the symbol `o` when the most recent symbol in the context is `H` is minimum. Similarly, the symbol pair $\langle \mathtt{H}, \mathtt{E} \rangle$ is a good candidate for agglutination in the plot of figure 5.11.

The agglutination process works by finding the most correlated symbol pair in the data, forming a new symbol sequence from that pair, UpWriting the data with respect to this new symbol sequence, and iterating until some stopping criterion is reached. This procedure is similar to that introduced by Wolff, with the main difference being the method used to calculate the correlation between adjacent symbols.

In figure 5.12 we show the dendrogram formed when agglutination is performed over the `Sentence` section of the Sherlock corpus. The process was halted when the `Sentence` section of the Sherlock corpus was UpWritten to a single higher level symbol. Figure 5.13 shows a similar dendrogram formed from the SHERLOCK corpus.

It is interesting that both dendrograms shown have a very similar structure. In fact, the agglutination process makes identical errors in each case—the first character of the

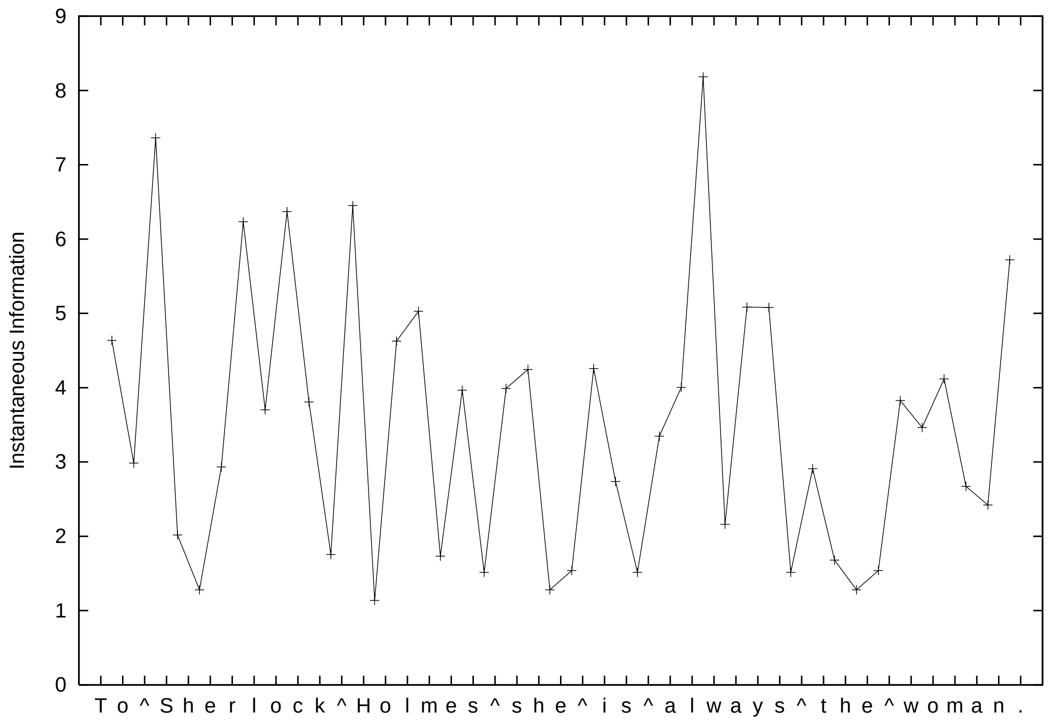

FIGURE 5.10: Instantaneous information provided to a 1^{st}–order Markov model by the `Sentence` section of the Sherlock corpus.

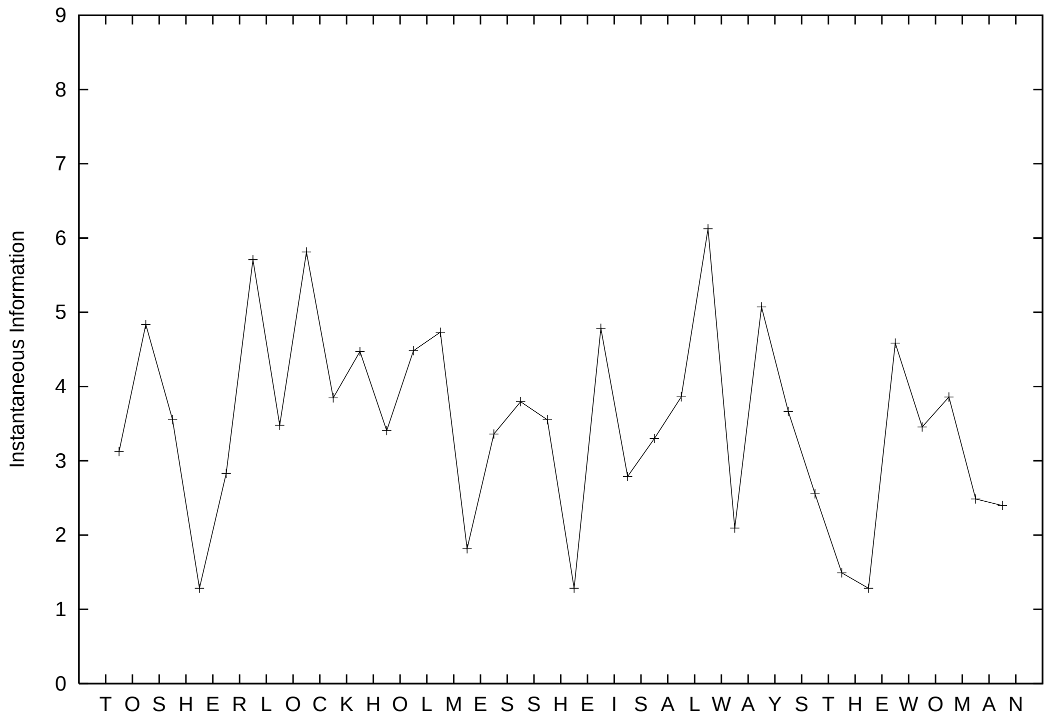

FIGURE 5.11: Instantaneous information provided to a 1^{st}–order Markov model by the the `SENTENCE` section of the SHERLOCK corpus.

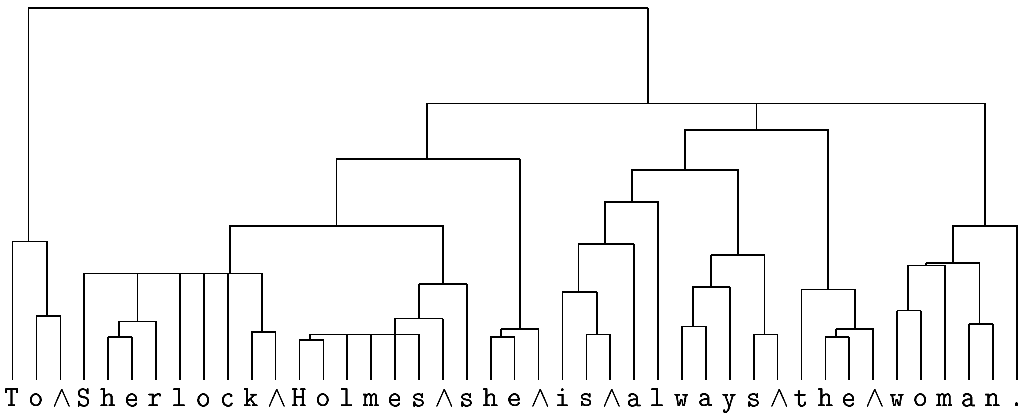

FIGURE 5.12: Dendrogram formed by agglutination using the information to measure correlation over the `Sentence` section of the Sherlock corpus.

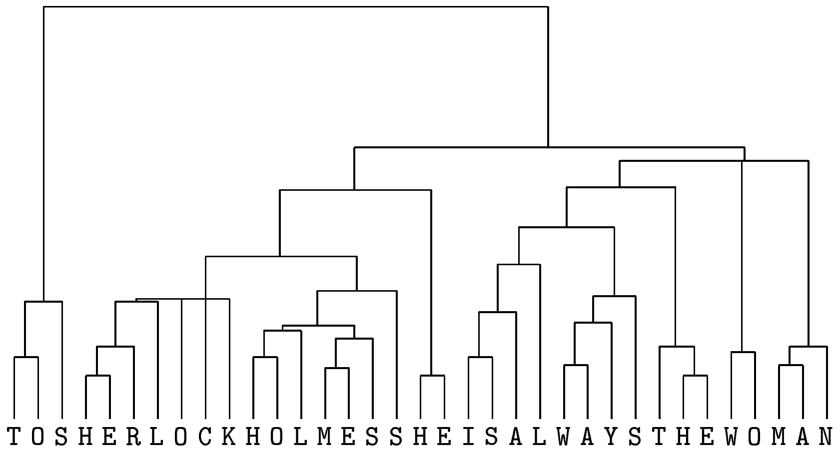

FIGURE 5.13: Dendrogram formed by agglutination using the information to measure correlation over the `SENTENCE` section of the SHERLOCK corpus.

word 'she' is incorrectly glued to the word 'Holmes', and the word 'always' is incorrectly split in two. Furthermore, on the SHERLOCK corpus, the first character of the word 'Sherlock' is incorrectly glued to the word 'To'. Apart from these errors, the hierarchical structure which results from the agglutination process seems quite reasonable.

Results for using the agglutination process to discover symbol sequences in the `Test` section of the Sherlock and SHERLOCK corpora are given in table 5.5. In both cases, the information was measured with respect to a 1^{st}–order Markov model, and the dendrogram was cut at the point which maximised the recall measure.

Corpus	**Recall**	**Accuracy**	**Coverage**
Sherlock	38.46%	18.78%	26.24%
SHERLOCK	46.86%	22.73%	23.44%

TABLE 5.5: Results of agglutination.

These results are superior to those of Wolff's algorithm, but they are particularly poor when compared to those of thresholded entropic chunking. This is due to the fact that although the agglutination process is quite successful at forming words, these words are formed at different levels of the dendrogram, and segmenting the data by merely cutting the dendrogram at some point therefore does not work particularly well, resulting in a mixture of words, word parts and word concatenations. The fact that results on the SHERLOCK corpus are superior to results on the Sherlock corpus is attributable to the fact that a 1^{st}–order Markov model was used in each case, and the existence of whitespace in the Sherlock corpus limits the context available to the model.

Agglutination using the information provided to the model by the next symbol in the sequence as a measure of the correlation between adjacent symbol pairs has the advantages of using the predictions of a predictive model, and of incorporating information from both sides of the posited chunk boundary. However, it is a slow process, due to the fact that a single symbol sequence is added to the alphabet on each iteration.

5.4 Discovering Symbol Classes

A second feature of many types of data is that certain quotient-objects of symbols, which we informally refer to as *symbol classes*, occur in similar contexts. For example, in English text, at the word level, we find that various syntactic categories of words exist, including verbs, nouns, adjectives, adverbs *et cetera*. Incorporating structure of this sort into a predictive model enables a process of generalisation, whereby several lower level contexts are UpWritten to the same higher level context. This generalisation process lessens the problems due to sparse data, because observations made of one symbol may be used to draw conclusions about every symbol in the class to which it belongs.

The problem of automatically discovering symbols classes in data is considerably more difficult than that of discovering symbol sequences, and we have made less progress in this area than we would have liked. Our interest in using the predictions made by a simple predictive model in order to determine what the symbol classes are has proven to be a handicap, but it is felt that the problem would remain difficult even if this restriction was lifted.

The process of finding symbol classes often involves finding clusters of vectors in a high-dimensional space, and for this reason it is informally known as *clustering.* When the data under consideration is natural language text or natural language utterances, the process is referred to as *syntactic category acquisition.*

Information theoretic measures, such as the Kullback-Leibler divergence, may be used to determine the similarity between the probability distributions given as predictions by a predictive model.[32] We may find the two most similar predictions by examining the predictions made by the predictive model for every possible context. A new symbol class may be formed from the most recent symbol of each of the two contexts which resulted in the two most similar predictions, and this process may be iterated, in a similar fashion to the agglutination process for discovering symbol sequences, to uncover larger symbol classes. We refer to this process as *agglomeration.*

The predictions themselves may be considered to represent noisy approximations to stochastic symbol classes, and in this case a large number of similar predictions provides evidence that the corresponding symbol class exists. A symbol class may then be formed from the centroid vector of an appropriately large cluster of similar predictions. We refer to this process as *clustering.*

We begin this section by presenting a brief overview of this area of research before considering the reasons why the problem is so difficult, and offering some insights which may prove helpful in the future. We then present the two methods of agglomeration and clustering for discovering symbol classes.

5.4.1 Previous Work

Several groups have pursued the problem of devising an algorithm which is capable of discovering symbol classes in a corpus of data, and all of the approaches investigated are similar in that they tend to consist of the following three stages.

- Construct a *distribution vector* for each of the symbols which is being classified;
- Cluster the space of distribution vectors using methods of numerical taxonomy [23] in order to determine the symbol classes;
- Assign each symbol in the alphabet to some number of these classes.

This distributional approach was common in linguistics prior to the Chomskian revolution, with Zellig Harris pioneering the field [6]. Harris writes that

> ... the distribution of an element [is] understood as the sum of all its environments.

Algorithms for classifying symbols generally define a symbol's environment as a local context which typically consists of the two symbols which precede it and the two symbols which follow it. Distribution vectors are formed by collecting statistics about the relative frequencies of contexts observed for each symbol in the alphabet. Classes are then formed from symbols which occur in similar contexts, according to the results of clustering these distribution vectors.

We shall present a brief overview of previous work on this problem, focusing on the pioneering work of George Kiss [9], who effectively clustered the predictions of a 1^{st}–order Markov model to determine symbol classes, followed by the work of Hinrich Schütze [21] and Redington, Finch and Chater citeRedington6, who both used similar methods of discovering symbol classes from distribution vectors in high-dimensional space.

George Kiss

In 1973 George Kiss proposed a model, conducive to computational implementation, in order to account for the learning of syntactic categories from positive data, based upon current theories of child language acquisition [9].

The basic assumption made by Kiss was that words can be classified based upon statistical regularities observed in a training corpus. Kiss' model is based upon a "word store" in which nodes corresponding to words are connected with unidirectional links, with the strength of each link in this *association network* corresponding to the normalised frequency with which the target word of the link follows the source word. Kiss notes that

> the network built up by this learning process is a particular physical representation of the Markov chain inherent in the input corpus.

A separate set of bidirectional links are established between the nodes in the network following the inference of this implicit 1^{st}–order Markov model. This second set of links reflects the similarity between words, determined by measuring the similarity between the *distribution vectors* corresponding to the two words which the link connects. The distribution vector of a word is determined by activating the node of the network corresponding to the word, and measuring the activations of all its neighbouring nodes. This amounts to determining the prediction made by the implicit 1^{st}–order Markov model when the word appears in the context.

Kiss used $1 - C(x_a||x_b)$ as a similarity measure between two words x_a and x_b, where $C(x_a||x_b)$ is the *Canberra metric*, as defined in equation 5.4, where $d(x_i|x_a)$ is the normalised frequency with which the word x_a is followed by the word x_i in the training corpus, as determined from the distribution vector of x_a. The distribution vectors are truncated by excluding all elements which fall below some predetermined threshold, and this has the effect of excluding unreliable probability estimates based on infrequent events. Kiss states that this process reflects the property of 'forgetting' in biological memory.

$$C(x_a||x_b) = \sum_{x_i \in \mathcal{A}} \frac{|d(x_i|x_a) - d(x_i|x_a)|}{d(x_i|x_a) + d(x_i|x_b)} \tag{5.4}$$

The final stage of the process is to form classes of words, and Kiss achieves this by using the principles of numerical taxonomy to hierarchically cluster words deemed similar [23]. Uni-directional connections between words and classes are then formed, with the strength of these connections being determined by calculating the average similarity between the word and all words in the class. Kiss defended his choice of model when he wrote that

> I am arguing here for a kind of "bootstrapping" process, which gets under way using some simple machinery, and then makes use of the results to improve itself.

We find the similarities between this statement and the UpWrite process to be quite striking. Furthermore, Kiss' algorithm is unique among those presented in this section in that it is capable of assigning a single word to multiple classes, and it quantifies the degree of membership which a word has to a particular class via the strength of the link between the word and the class.

Experiments were performed using a corpus of transcribed child directed speech, and 31 words were selected for further analysis. Results indicated that word classes exhibited some of the expected structure, but association between words and clusters tended to be weak, and some of the expected clusters did not separate from each other. Kiss stated that word classes could in fact only be detected by applying "sensitive methods". Further experiments performed with all words in the corpus resulted in a clear-cut organisation of nouns, and a less cohesive clustering of verbs.

Hinrich Schütze

Hinrich Schütze presented a method for inducing the parts of speech of words from a corpus of natural language text [21]. His method works by forming distribution vectors for each word by collecting frequency counts for a context of two predecessor words and two successor words. Initially, this process is limited to the five thousand most common words in the corpus. This results in $5,000$ distribution vectors in $\mathbb{R}^{20000}$, and these vectors are clustered using the cosine of the angle between vector pairs as a measure of their similarity. In order that this clustering process should be time efficient, a singular value decomposition is performed in order to map the vectors into $\mathbb{R}^{15}$ such that their similarity with one another is preserved.

Schütze used a different method of forming distribution vectors in a second phase in order to classify the remaining words. The 278 most frequent words in the corpus were used as features, as were the 222 classes found by the algorithm in the first phase when operating on the remaining 4722 most frequent words in the corpus. This process resulted in a distribution vector in $\mathbb{R}^{2000}$ for each word yet to be classified, and these vectors were mapped into $\mathbb{R}^{10}$ prior to clustering being performed. Schütze effectively applied a bootstrapping process by using existing classes as scaffolding, something which facilitates the classification of infrequent words.

The classes discovered by this algorithm range from the reasonable to the noisy, and Schütze observes that

> ... ambiguity is a problem for the vector representation used here, because the two components of an ambiguous vector can add up in a way that makes it by chance similar to an unambiguous word of a different syntactic category.

This problem is caused by the fact that more than one word class often occurs within any given context, and we shall analyse the ramifications of this in section 5.4.2.

Redington, Chater and Finch

Martin Redington, Nick Chater and Steven Finch have also investigated the acquisition of syntactic categories from a corpus of natural language text via distributional methods [18]. They write that

> ... the problems involved in computationally acquiring many aspects of language from realistic linguistic input are indeed formidable, and have led many to argue that the majority of linguistic knowledge must be innate.

Redington, Chater and Finch aimed to show that distributional statistics provide sufficient evidence about the syntactic category of some words, even in the absence of other, potentially highly informative, sources of information. Their method collects statistics about the distribution of contexts for each word being classified, where the contexts used consist, as with Schütze's algorithm, of the two preceding words and the two succeeding words.

The *Spearman Rank Correlation Coefficient* is used to measure the similarity between the resulting distribution vectors, rescaled so that its value falls in the range $[0, 1]$, and standard methods of numerical taxonomy are used to determine a hierarchical clustering of the vectors [23]. The Spearman Rank Correlation Coefficient is defined in equation 5.5, where $r_a(i)$ is the rank assigned to the i^{th} element of the distribution vector of x_a, and n denotes the number of elements in each distribution vector. The rank $r_a(i)$ is determined by assigning a rank of 1 to the element of the vector which has the lowest value, assigning higher ranks to elements of higher values, and assigning an average rank to several elements which share the same value.

$$\rho_{ab} = 1 - 6 \sum_{i=1}^{n} \frac{[r_a(i) - r_b(i)]^2}{n(n^2 - 1)} \tag{5.5}$$

Experiments were performed using the CHILDES corpus of child-directed speech. The most frequent 1000 words in the corpus were classified using the most frequent 150 words to provide context, resulting in 1000 distribution vectors in $\mathbb{R}^{600}$. Results indicate that 37 clusters were formed, with the most populous of these corresponding to standard syntactic categories such as verbs, nouns, prepositions, adjectives and so on, with a degree of misclassification resulting from syntactic ambiguity, which the algorithm does not account for.

Redington, Chater and Finch analysed the contribution of different portions of the context to the quality of the resulting word classes, and found that generally the closer a context word is to the word being classified, the more information it provides about its syntactic category. The also found that although the best results were obtained by using a context which contained both preceding and succeeding words, words in the preceding context were generally much more useful than those in the succeeding context.

Other Work

Other researchers have worked on the problem of the automatic assignation of symbols to classes, and we acknowledge the work of Hang Li and Naoki Abe [13], Timothy Cartwright and Michael Brent [3], and Fernando Pereira, Naftali Tishby and Lillian Lee [17].

5.4.2 The Problem of Noise due to Ambiguity

Before describing our methods of agglomeration and clustering, we would like to consider why the problem of discovering symbol classes in data is so difficult. Consider the phrase-structure grammar shown in figure 5.14, which generates a symbol sequence over a lowest level alphabet of four symbols. We would like to recover the three symbol classes $(1) \mapsto [\mathtt{A}|\mathtt{B}]$, $(2) \mapsto [\mathtt{B}|\mathtt{C}]$ and $(3) \mapsto [\mathtt{C}|\mathtt{D}]$.

$$
\begin{array}{lll}
\# & \mapsto & (1)(4)(2)(5)(3)(6) \\
1 & \mapsto & \mathtt{A}|\mathtt{B} \\
2 & \mapsto & \mathtt{B}|\mathtt{C} \\
3 & \mapsto & \mathtt{C}|\mathtt{D} \\
4 & \mapsto & (1)|(3) \\
5 & \mapsto & (1)|(2) \\
6 & \mapsto & (3)|(2)
\end{array}
$$

Sample: `ACBCDBAABBCDABBACCAACBDBBBBBDCABCCDCBCCCCBBC` ...

FIGURE 5.14: A phrase-structure grammar which generates a sequence of symbols which exhibit several kinds of ambiguity.

The first thing to note is that these symbol classes are ambiguous in that the symbols `B` and `C` belong to two classes; this is presumably a feature of many types of data, and ambiguity of this kind obviously occurs in natural languages. A second form of ambiguity is due to the fact that any given symbol class may be followed by exactly two others, with equal probability, due to the fact that meta-classes, such as (5), exist in the grammar. For example, $P((1)|(2)) = \frac{1}{2}$ and $P((2)|(2)) = \frac{1}{2}$. This second type of ambiguity also exists in natural languages—consider, for example, the fact that an adjective may be followed by either an adjective or a noun in a sequence of English words.

This second form of ambiguity at the level of symbol classes means that the observations which we actually make at the lowest level representation of the data (which is the level at which we must necessarily begin) will consist of a convex combination of some unknown number of symbol classes. For example, $P(\mathtt{B}|(2)) = \frac{1}{2}P(\mathtt{B} \in (1)) + \frac{1}{2}P(\mathtt{B} \in (2))$.

Let us continue this exposition by considering the nature of the space of probability distributions. This is the space in which clustering is to be performed, as we are interested in finding clusters of predictions. Each element of a probability distribution is non-negative, and the sum of all elements is unity. Therefore, a probability distribution represented as a vector in $\mathbb{R}^{|\mathcal{A}|}$ will lie on the $(|\mathcal{A}| - 1)$–*simplex*—the region of space defined by all possible points which satisfy the constraints placed on vectors which represent probability distributions.

Consider the predictions made by a predictive model about a string of symbols taken from a binary alphabet. If we consider the predictions to be vectors in $\mathbb{R}^2$, we will find that all of the vectors lie on the the simplex corresponding to the line segment which has $(0, 1)$ and $(1, 0)$ as its end-points, as shown in figure 5.15. Similarly, for a ternary alphabet, the predictions made by a predictive model will lie in the region of space corresponding to the equilateral triangle which has $(0, 0, 1)$, $(0, 1, 0)$ and $(0, 0, 1)$ as its vertices, as in figure 5.16. In general, over the alphabet $\mathcal{A}$, all predictions lie in the region of space corresponding to the $(|\mathcal{A}| - 1)$–simplex, in $\mathbb{R}^{|\mathcal{A}|}$.

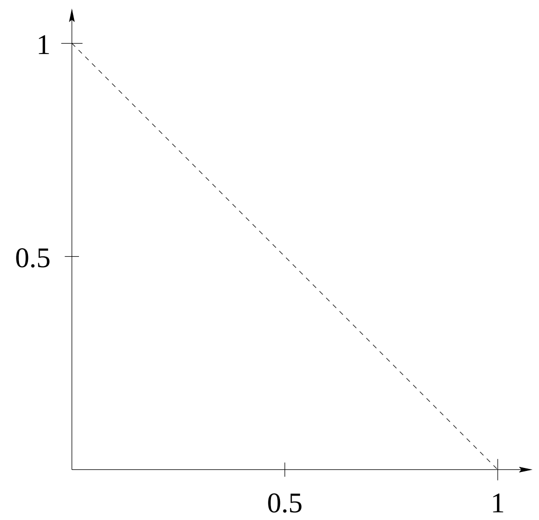

FIGURE 5.15: A simplex in $\mathbb{R}^2$ is the line segment connecting $(0, 1)$ and $(1, 0)$.

We are now in a position to consider how the vectors may be distributed within the space. We do this by recalling that all of the probability distributions we observe will be

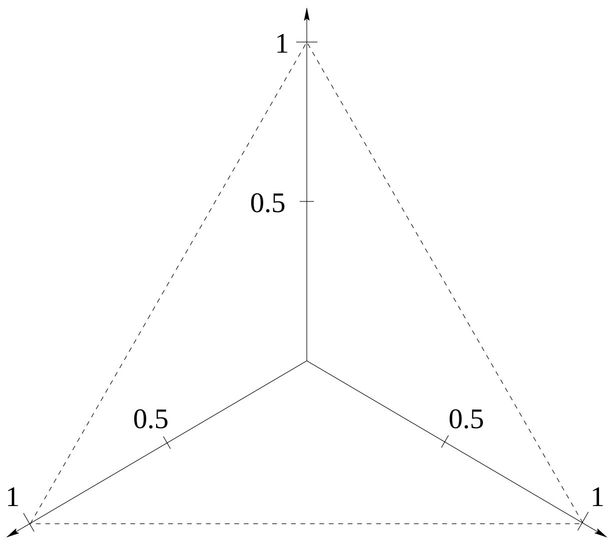

FIGURE 5.16: A simplex in $\mathbb{R}^3$ is the region of space corresponding to the equilateral triangle which has $(0,0,1)$, $(0,1,0)$ and $(1,0,0)$ as its vertices.

convex combinations of some unknown number of probability distributions at a higher level of abstraction. This means that each observed probability distribution will lie somewhere within the simplex which has as its vertices the k higher level probability distributions which the lower level probability distribution is a convex combination of. The exact location of the observed probability distribution within this simplex is determined by the coefficients of the convex combination, and these are equivalent to a probability distribution over the higher level alphabet.

Another way of thinking about this is that the observed probability distributions are the DownWritten versions of higher level probability distributions. For example, the higher level probability $P(\mathtt{B}|(2))$ is DownWritten to the lower level probability $\frac{1}{2}P(\mathtt{B} \in (1)) + \frac{1}{2}P(\mathtt{B} \in (2)) = \frac{1}{2}$, as we have previously shown. We assume, of course, that each symbol class may also be represented by a probability distribution over the lower level alphabet, and this is true in the case where the symbol classes are stochastic.

It is therefore apparent that the observed probability distributions are not noisy versions of a single underlying symbol class—they are noisy convex combinations of some unknown number of symbol classes. The evidence we observe is therefore rather indirect; instead of looking for clusters of points, we should be looking for the set of points which define the vertices of some unknown number of k–simplices, for various k. Obviously there are many candidate point sets, with the unit vectors of the $(|\mathcal{A}| - 1)$–simplex being a possible (and uninteresting) solution. Therefore, we should search for a solution according to the principle of Occam's razor; we want to find the vertices which describe the observed distribution of vectors well, while at the same time minimizing the combined description length of these vertices and the data relative to the k-simplices which they define.[33]

As an example of the sort of data we have been talking about, we show in figure 5.17 a plot of the vectors which correspond to the predictions made by a 2^{nd}–order Markov model over the data defined by the grammar of figure 5.14.[34] In this diagram, the four vertices of the $(|\mathcal{A}| - 1)$–simplex are denoted by crosses, and connected with solid lines. These vertices define a tetrahedron in $\mathbb{R}^4$, and all possible probability distributions fall within this simplex. The probability distributions corresponding to the three symbol classes of the grammar are denoted by circles, and it is obvious that the sixteen vectors[35] which represent the predictions made by the 2^{nd}–order Markov model all fall on the simplex defined by these three vectors. An algorithm which searched for these vertices in the manner prescribed above would successfully recover the three symbol classes, while any standard clustering algorithm would not.[36]

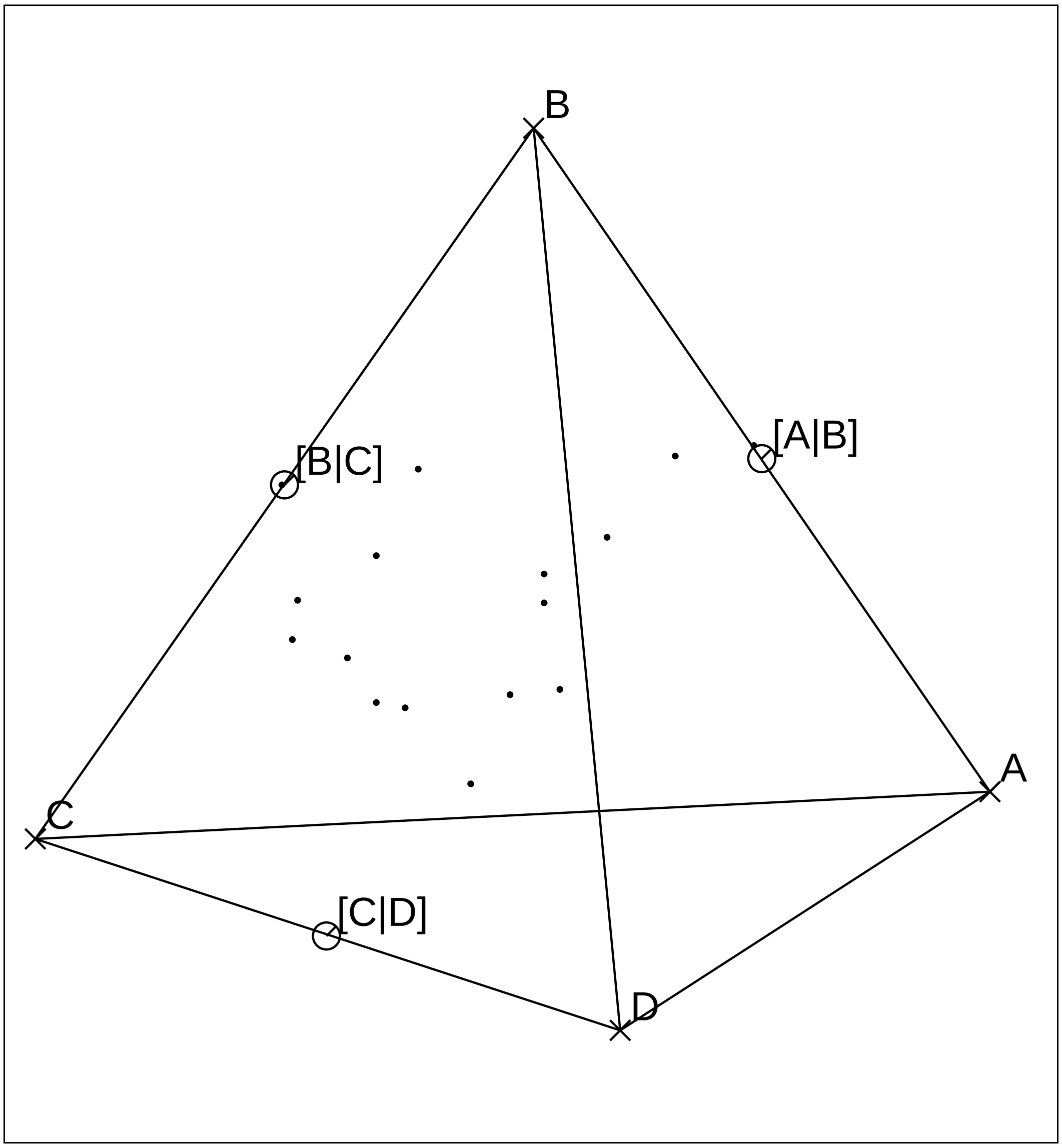

FIGURE 5.17: The sixteen vectors representing predictions made by a 2^{nd}–order Markov model over data generated by the grammar of figure 5.14 lie on the 2–simplex which is defined by the three vectors which represent the symbol classes, and this, in turn, lies within the 3–simplex in $\mathbb{R}^4$.

Note that increasing the context available to the predictive model will serve to disambiguate the predictions it makes to an extent, and that symbol classes will therefore be more readily obtained by clustering the predictions made by higher order Markov models. However, this has the disadvantage of limiting the number of observations made per context, and the sparse data problem raises its ugly head. A more intelligent technique of finding symbol classes is required, and we hope that the insight presented above marks the genesis of such a technique.

A clustering mechanism which takes account of the fact that the vectors being clustered are likely to be distributed in this way may be more successful in finding symbol classes in data than the traditional approaches we have previously presented. Alas, we have not yet been able to explore this promising line of work, and we only mention it here to draw the reader's attention to the nature of the problem, and to suggest a possible technique for solving it.

5.4.3 Algorithms for Discovering Symbol Classes

We are now in a position to consider methods for discovering symbol classes in data; we are interested in methods which achieve this using the predictions made by a simple predictive model, and nothing more. Both of the methods we shall consider involve treating the predictions made by the predictive model as vectors in $\mathbb{R}^{|\mathcal{A}|}$,[37] and clustering these vectors in order to determine what the symbol classes should be.[38] The two methods differ only in the way the symbols are assigned to classes once the clusters have been formed.

Agglomeration: The most recent symbols in the contexts which correspond to the predictions in a particular cluster may be considered to belong to the same symbol class. This is motivated by the fact that similar predictions are likely to be made upon encountering any of the symbols in a particular class.

Clustering: The *centroids* of the various clusters found may be considered to be stochastic symbol classes. This is motivated by the fact that each prediction made by the predictive model may be considered to be a noisy version of some underlying symbol class.

In both cases stochastic symbol classes are formed, in the case of agglomeration this is achieved by gathering frequency statistics about the symbols in each class as the UpWrite is being performed. Stochastic symbol classes are desirable because the probability of a symbol being a member of a class may be used to DownWrite the predictions made by the predictive model, something which is useful in many applications. Note that neither of these techniques attempt to address the problems of noise due to ambiguity—we have reluctantly assigned this problem to future work.

We had originally planned to use an information theoretic measure to calculate the similarity between two probability distributions—the Kullback-Leibler divergence may have been suitable, for instance. Our experiments have indicated that the measure used is fairly inconsequential, and we have therefore chosen to use the standard Euclidean distance due to the fact that it is relatively straightforward to compute.

It should be noted that our technique of clustering the predictions made by a predictive model is equivalent to forming a distribution vector of contexts, where the context *follows* the symbol being classified. That is, the prediction made by a model is equivalent to a distribution over all such contexts. Redington, Chater and Finch studied the performance of their classification scheme for various contexts, and concluded that the context which we have implicitly chosen produces poor results [18]. On the other hand, Kiss' algorithm made use of an identical distribution vector [9]. Furthermore, the fact that the agglomeration algorithm is iterative does tend to make better use of the available data than traditional algorithms, which form symbol classes in a single pass.

5.4.4 Agglomeration

Both of the methods we propose for discovering symbol classes in data are based on the notion that clues as to what these symbol classes are are contained in the predictions made by simple predictive models. The first method, agglomeration, is similar to the agglutination technique for finding symbol sequences in data. The general idea of agglomeration is that symbols may belong to the same class if the predictive model makes similar predictions about which symbols are likely to follow them. The technique consists of finding the two most similar symbols in the data, forming a new class out of them, UpWriting the data with respect to this new class, and iterating. During the UpWrite process, frequency counts for the symbols in the class are collected, enabling us to estimate the probability of a particular symbol appearing given that we know which class occurred, resulting in a hierarchy of stochastic symbol classes.

The motivation for this technique is that we wish to use a simple predictive model to find structure in the data, but this predictive model may not be sufficiently powerful to find the required structure in a single pass of the data. We therefore favour a 'skeptical' approach; the single most likely class is added each iteration, with the data being UpWritten at the end of each iteration. This process bootstraps the discovery of less likely classes, with the classes which have been found already functioning as scaffolding. Agglomeration may be considered to be a form of hierarchical clustering, with the *similarity matrix* [23] being recalculated whenever a pair of symbols is joined, not by the standard techniques, but by UpWriting the data, inferring a new language model, and calculating the similarity matrix afresh. This results in dendrograms which look different to those formed from a standard hierarchical clustering process, as the iterative nature of our algorithm often results in classes becoming more similar than the average similarities between their elements.

For large alphabets, the agglomeration algorithm can be quite slow. We therefore restrict the algorithm so that it only considers the 1000 most frequently occurring symbols on each iteration—this means that our job is one of finding the two most similar vectors out of up to 1000 vectors in $\mathbb{R}^{|\mathcal{A}|}$. This restriction does speed up the algorithm substantially, and, contrary to first appearance, it does not mean that only the most frequent 1000 symbols will be classified. Each iteration of the algorithm removes two symbols from the alphabet and replaces them with a single new symbol, meaning that the algorithm will eventually assign every symbol in the alphabet to a class.

It is considerably more difficult to evaluate the performance of an algorithm whose job it is to discover symbol classes, as we simply do not know what the desired classes look like. We therefore decline to give any sort of performance criteria of the algorithm, preferring to evaluate its performance by eye.

The agglomeration algorithm was used to find symbol classes in the Sherlock corpus at both the character level and the word level. When operating at the word level, the algorithm converted all words to their uppercase equivalents prior to inferring the model in order to make the most of the limited data available. In both cases a simple 1^{st}–order Markov model was used, and no attempt was made to smooth the predictions it made.

Figures 5.18 and 5.19 show two of the hierarchical classes formed at the character level by the agglomeration algorithm, the first corresponding to the class of digits, and the second corresponding to a class of various forms of punctuation. Apart from the two classes illustrated, two other classes of punctuation were found—one of these contained the two closing brackets `)` and `]`—and a single large class containing all members of the modern Roman alphabet, apart from `I`, was also formed.

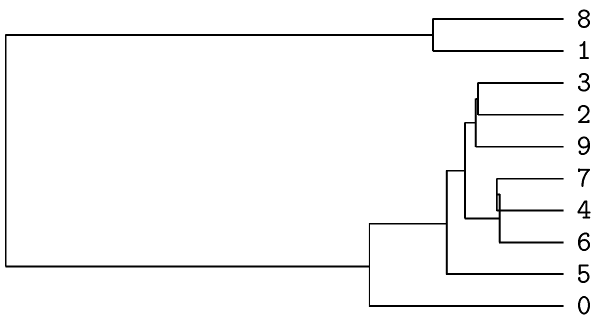

FIGURE 5.18: The class of digits discovered from the Sherlock corpus at the character level by the agglomeration algorithm.

It is interesting to note that the digits `1` and `8` are deemed to be more similar to each other than they are to the remaining eight digits. This is almost certainly due to the fact that many of the numerical sequences in the Sherlock corpus denote dates in the nineteenth century. The class of punctuation symbols shown in figure 5.19 consists mostly of symbols which signify the beginning of a new phrase, including the two opening brackets `(` and `[`. The inclusion of the symbol `I` in this class is attributable to the fact that the the optical character recognition process used to form the Sherlock corpus occasionally made an error whereby the symbol `[` appeared rather than the symbol `I`.

FIGURE 5.19: One of several classes of punctuation characters discovered from the Sherlock corpus at the character level by the agglomeration algorithm.

Figures 5.20 to 5.27 show eight of the classes formed by the agglomeration algorithm at the word level. These range from the good, as is the case with the class of figure 5.20, which contains two separate groups of words which are very similar to one another, to the rather esoteric, as is the case with the class of figure 5.27, which appears to have been formed from many ambiguous contexts.

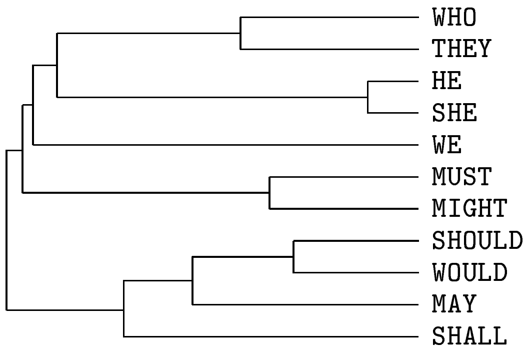

FIGURE 5.20: A class containing pronouns which are used to describe persons and groups of persons, and auxiliary verbs which are used to express possibility, as discovered from the Sherlock corpus at the word level by the agglomeration algorithm.

It is interesting that all of these classes are very strongly semantic in nature, and this validates our earlier claim that algorithms which are designed to find word classes in natural language text invariably find classes of a quasi-semantic sort. The class of figure 5.26 is a good example of this; the class contains words which describe parts of buildings and parts of towns, such as `WINDOW`, `ROOM`, `HOTEL` and `STREET`. Classes of these sorts are potentially much more useful than classes which correspond to coarse syntactic categories such as noun and verb, as they allow a predictive model to make quasi-semantic generalisations.

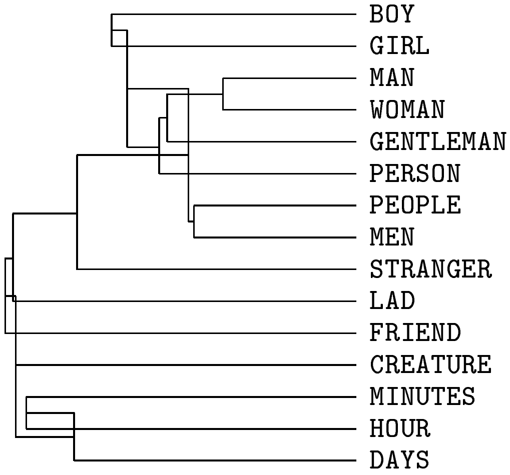

FIGURE 5.21: A class containing nouns which are used to describe human beings, and nouns which are used to indicate periods of time, as discovered from the Sherlock corpus at the word level by the agglomeration algorithm.

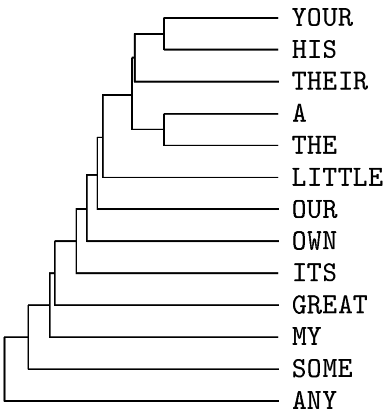

FIGURE 5.22: A class containing, among other things, the definite and indefinite articles, and attributive possessive pronouns, as discovered from the Sherlock corpus at the word level by the agglomeration algorithm.

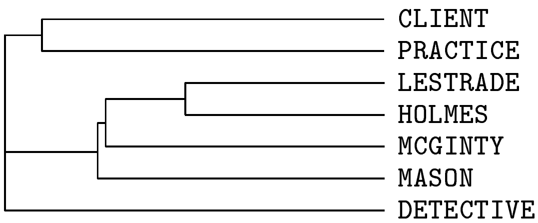

FIGURE 5.23: A class containing proper nouns, among other things, as discovered from the Sherlock corpus at the word level by the agglomeration algorithm.

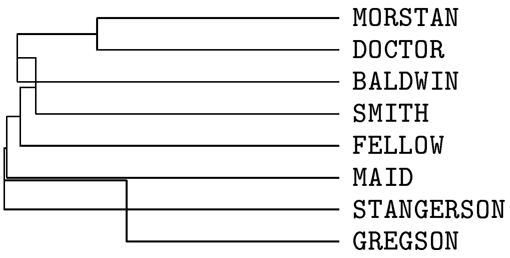

FIGURE 5.24: A class containing proper nouns, among other things, as discovered from the Sherlock corpus at the word level by the agglomeration algorithm.

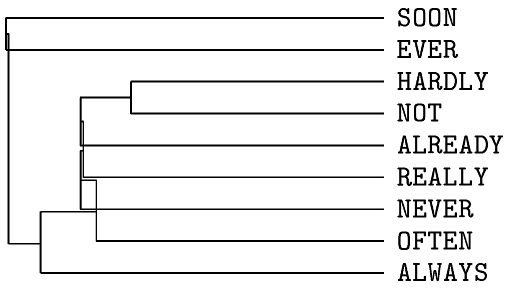

FIGURE 5.25: A class containing for the most part adverbs which express time-relative possibilities, as discovered from the Sherlock corpus at the word level by the agglomeration algorithm.

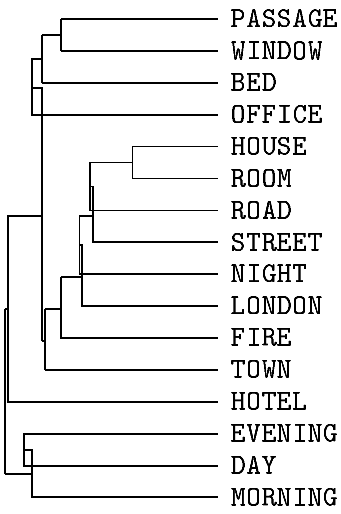

FIGURE 5.26: A class containing for the most part nouns which describe parts of buildings, parts of towns, and times of day, as discovered from the Sherlock corpus at the word level by the agglomeration algorithm.

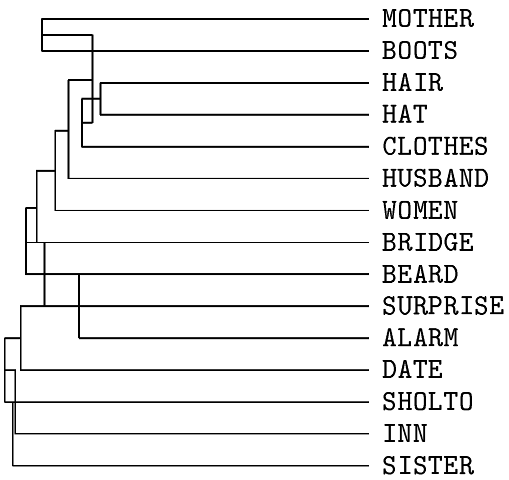

FIGURE 5.27: A rather esoteric class containing, amongst other things, nouns which describe parts of the body, clothing and familial relationships, as discovered from the Sherlock corpus at the word level by the agglomeration algorithm.

The advantages of the agglomeration algorithm are that it is computationally efficient (as it restricts the process to the most frequently occurring symbols), it mirrors the agglutination technique in that it is skeptical, greedy and iterative, and it finds rather good, albeit noisy, symbol classes. Its major disadvantage is that, although the classes are stochastic, a particular symbol can belong to, at most, a single class.

5.4.5 Clustering

A second method of using the predictions made by a simple predictive model to find symbol classes is simply to assume that each prediction, when expressed as a probability distribution, is a noisy version of exactly one stochastic symbol class. A cluster of probability distributions can then be regarded as strong evidence that the centroid vector of that cluster is itself the noise-free version of the symbol class to which the members of the cluster are approximations.

The motivation for this technique is that the symbol classes found are immediately stochastic, and that they are contextually constrained, with the consequence that a symbol may belong to different symbol classes in different contexts.

Our technique is to infer a simple 1^{st}–order Markov model from a corpus, and to apply standard hierarchical clustering methods to the predictions made by this predictive model. Clusters may be recovered from the resulting hierarchy by specifying a minimum similarity threshold, with the centroids of the resulting clusters representing new symbol classes which may then be used to UpWrite the data. These classes are context-sensitive, as they are based on predictions which are themselves context sensitive; the UpWrite will take this into account, meaning that a particular symbol may be UpWritten to one of many classes depending upon the value of the symbol which immediately precedes it.

To speed up the hierarchical clustering process, we decided to restrict the symbols in the context to the 1000 most frequently occurring symbols, as was the case for agglomeration. The classes are formed from the centroids of the clusters which result, and may therefore contain any of the symbols in the alphabet, not just the most frequent 1000.

The clustering algorithm was used to find symbol classes in the Sherlock corpus at the word level only, with all words converted to their uppercase equivalents in order to make better use of the limited data available. As before, a simple 1^{st}–order Markov model was inferred from the corpus, and no attempt was made to smooth its predictions.

Figures 5.28 to 5.31 show four of the symbol classes found by this process. The figures indicate the context words for which predictions were made, and list the most frequent ten words according to the centroid vector of the resulting cluster, which was regarded as being a probability distribution in its own right. For instance, in figure 5.28, the predictions made by the model upon encountering each of the five words `TELL`, `TOLD`, `ASK`, `LET` and `EXCUSE` were clustered, and the ten most probable words in the resulting symbol class, listed by decreasing probability, are `ME`, `YOU`, `US`, `HIM`, `THE`, `HER`, `IT`, `MY`, `FOR` and `THEM`.

$$[\texttt{TELL}|\texttt{TOLD}|\texttt{ASK}|\texttt{LET}|\texttt{EXCUSE}]\ \mathit{predicts} \left\{ \begin{array}{l} \texttt{ME} \\ \texttt{YOU} \\ \texttt{US} \\ \texttt{HIM} \\ \texttt{THE} \\ \texttt{HER} \\ \texttt{IT} \\ \texttt{MY} \\ \texttt{FOR} \\ \texttt{THEM} \end{array} \right.$$

FIGURE 5.28: A symbol class containing pronouns used to refer to persons as its most frequent elements, as discovered in the Sherlock corpus at the word level by clustering the predictions made by a 1^{st}–order Markov model.

It is immediately obvious that the set of context words themselves represent valid word classes, and it is these that were found by the agglomeration technique. It is our opinion that agglomeration produces far superior classes, and this may be due to the fact that it *is not* context sensitive, allowing more observations to be taken into consideration when forming a symbol class.

$$[\texttt{TEN}|\texttt{FIVE}|\texttt{FEW}|\texttt{NINE}]\ \textit{predicts} \left\{ \begin{array}{l} \texttt{MINUTES} \\ \texttt{YEARS} \\ \texttt{DAYS} \\ \texttt{O'CLOCK} \\ \texttt{HOURS} \\ \texttt{WORDS} \\ \texttt{HUNDRED} \\ \texttt{MILES} \\ \texttt{.}\wedge \\ \texttt{-} \end{array} \right.$$

FIGURE 5.29: A symbol class containing units of time measurement as its most frequent elements, as discovered in the Sherlock corpus at the word level by clustering the predictions made by a 1^{st}–order Markov model.

$$[\texttt{YOU}|\texttt{I}|\texttt{WE}|\texttt{THEY}|\texttt{HE}|\texttt{SHE}|\texttt{WHO}]\ \textit{predicts} \left\{ \begin{array}{l} \texttt{HAD} \\ \texttt{HAVE} \\ \texttt{WAS} \\ \texttt{COULD} \\ \texttt{WILL} \\ \texttt{ARE} \\ \texttt{AM} \\ \texttt{CAN} \\ \texttt{WOULD} \\ \texttt{WERE} \end{array} \right.$$

FIGURE 5.30: A symbol class containing the words `HAVE` and `HAD`, and auxiliary verbs which express possibilities as its most frequent elements, as discovered in the Sherlock corpus at the word level by clustering the predictions made by a 1^{st}–order Markov model.

$$[\texttt{LOOK}|\texttt{LOOKED}|\texttt{GLANCE}|\texttt{ARRIVED}|\texttt{STARED}|\texttt{GLANCED}]\ \textit{predicts}\ \begin{cases}\texttt{AT}\\ \texttt{OUT}\\ \texttt{OF}\\ \texttt{INTO}\\ \texttt{ROUND}\\ \texttt{UPON}\\ \texttt{UP}\\ \texttt{IN}\\ \texttt{AFTER}\\ \texttt{BACK}\end{cases}$$

FIGURE 5.31: A symbol class containing prepositions which describe motion or position as its frequent elements, as discovered in the Sherlock corpus at the word level by clustering the predictions made by a 1^{st}–order Markov model.

Apart from the symbol classes shown, which are rather good, the clustering method also found a very large number of similar classes, all of which contained the indefinite article, the definite article, and common pronouns and punctuation characters in various ratios. The preponderance of such classes would almost certainly adversely affect the performance of any algorithm which used them, as the total number of symbols in the alphabet would not be decreased by their inclusion, and the predictor would not be able to generalise about the data significantly as a result. Classes of nouns tended not to be formed by the algorithm.

The advantages of forming symbol classes by finding clusters of prediction are that the resulting symbol classes are context sensitive and immediately stochastic, and that the process may be performed in a single pass of the data. However, the results of the algorithm are inferior to those of agglomeration, and this may be a consequence of the fact that a rather unconstraining context of one previous symbol was used.

5.5 The Final Structure of the UpWrite Predictor

We are now in a position to present the structure of the UpWrite Predictor as we have implemented it. The reader should note that different implementations are possible; we selected ours based upon the fact that we hope to test the performance of the UpWrite Predictor as part of an adaptive statistical data compressor, and this requires a predictive model which may be used adaptively. There are several problems which need to be addressed when implementing the UpWrite Predictor.

- A simple predictive model needs to be selected, and a default alphabet needs to be determined;

- Methods need to be chosen for discovering symbol sequences and symbol classes in the data at all levels of abstraction;
- The UpWrite and DownWrite processes need to be defined with respect to these methods;
- A technique for removing anomalous symbol sequences and symbol classes from the higher level representation of the data needs to be devised; and
- A stopping criterion needs to be determined.

We shall now discuss the various decisions that were made during the implementation of the UpWrite Predictor. We postpone evaluation of our implementation until the next chapter, in which we shall be performing a series of experiments to test its performance.

5.5.1 Selecting the Predictive Model

The UpWrite Predictor makes use of a family of predictive models, each of which process the data at a different level of abstraction. These models make predictions which are used both internally and externally, to find structure in the data being processed, and to provide information about the data to whatever application is making use of the UpWrite Predictor.[39]

The UpWrite Predictor is intended to be used both statically (*i.e.* trained on a corpus prior to being applied to a problem) and adaptively. For this reason, the predictive model employed should be time-efficient, and should be able to make predictions prior to observing any data.

Using an extremely simple model has the advantage that the performance of the final UpWrite Predictor will be more easily evaluated; we need not worry about how the performance of the predictor changes as the parameters of the model are varied if the model has no parameters. We are therefore interested in choosing the simplest model which is able to assign a non-zero probability to every symbol, even when no data has been observed.

This model will still need to be context-dependent, and it will still need to express the relative frequencies of the symbols in the alphabet. These two constraints are evident in all complex systems, according to Lila Gatlin [2], and must be included if symbol sequences and symbol classes are to be extracted by any of the methods presented in sections 5.3 and 5.4.

For these reasons we choose to use a simple 1^{st}–order Markov model in the UpWrite Predictor. In order to ensure that its predictions are smoothed, we apply *Laplace's Law of Succession*, whereby all frequency counts, for every context in the predictive model, including those yet to be observed, are initialised to 1. Using such a short context has the additional advantage of making the best best use of the limited amount of data available for inference.

5.5.2 Discovering Symbol Sequences and Symbol Classes

The predictive model we have selected uses a context of a single symbol, and this makes it perfectly suited to the two methods of agglutination and agglomeration. Both of these methods are highly iterative in nature, in that they construct a hierarchical description of the data by UpWriting it gradually. We consider this property to be a distinct advantage. Our implementation of the UpWrite Predictor is therefore highly iterative itself—on each iteration the most promising symbol sequence and the most promising symbol class is discovered from the data, and either one of these two types of structure may then be used to UpWrite the data.

The iterative nature of the methods of agglutination and agglomeration also makes them ideal for use in an adaptive system, as the most promising symbol sequence and the most promising symbol class can be constantly monitored and updated during inference—the lone context of the predictive model which is updated with the next symbol in the sequence is the only context which needs to be re-evaluated to determine whether or not better candidate symbol sequences and symbol classes exist.

Finally, our decision to form symbol sequences via agglutination alleviates the parsing problem, as we shall soon see.

5.5.3 UpWriting and DownWriting

UpWriting symbols to symbol classes is trivial, as each symbol belongs to exactly one class. UpWriting is therefore a matter of replacing each occurrence of the symbol in the data with the class to which it belongs. The DownWrite is not unique for symbol classes, but it is possible to generate data which has the property that its UpWritten form is identical to that of the data under consideration. We may 'DownWrite' a symbol class by selecting a symbol from the class at random, according to the probabilities of the symbols in that class. We shall find this useful when evaluating the performance of the UpWrite Predictor, as it allows us to use the UpWrite Predictor generatively.

UpWriting symbol pairs to symbol sequences is slightly more problematic, as an ambiguous sequence of symbols may be encountered. Because our implementation of the UpWrite Predictor is iterative, with only a single symbol sequence being UpWritten on each iteration, and because these symbol sequences always consist of a pair of symbols,[40] ambiguity only occurs when the symbol sequence being UpWritten consists of a pair of identical symbols.

Consider, for example, the data `AAA` and the symbol sequence $\langle \texttt{AA} \rangle$. There are two possible ways in which this data can be UpWritten, either as $\langle \texttt{AA} \rangle\texttt{A}$ or as $\texttt{A}\langle \texttt{AA} \rangle$. We choose to UpWrite data in a greedy manner, by UpWriting symbol sequences as soon as they are encountered while scanning the data. This decision was motivated in part by the fact that an adaptive system must necessarily scan the data in this way, and that therefore a greedy parser is the only possible solution to this parsing problem in certain applications.[41] In general, though, the UpWrite Predictor circumvents the parsing problem by UpWriting the data on the fly, while structure is being discovered.

DownWriting symbol sequences is trivial; we merely emit the sequence of lower level symbols which the higher level symbol represents.

5.5.4 Correcting Mistakes via Feedback

The UpWrite Predictor features a feedback mechanism within each module, as shown in figure 5.1, whose purpose is to correct any errors which the UpWriter may have made while discovering symbol sequences and symbol classes. The inclusion of this mechanism was motivated by the fact that the UpWrite is hierarchical in nature, and yet the process of finding structure in data tends to occur at a single level of this hierarchy—new symbols are added to the alphabet without consideration of the overall implications of their existence.

The feedback mechanism allows us to *tentatively* add new symbols to the alphabet, evaluate the performance of the higher level predictive model which is inferred from the UpWritten data, and decompose any higher level symbols which prove detrimental to its performance.

Consider, for example, the symbol sequence shown in figure 5.32. A low order Markov model which discovers symbol sequences in this data via agglutination may produce the UpWritten form of this data shown in figure 5.33. A model inferred from the UpWritten data will, assuming that sufficient data is available for probability estimation, exhibit a high level of surprise upon encountering the symbol `EAR` in the context `THEY`. This may be sufficient evidence to deem the symbol pair $\langle \texttt{THEY}, \texttt{EAR} \rangle$ erroneous. The feedback mechanism may then be used to ask the UpWriter to correct this symbol pair, and this may be achieved by decomposing the recently added symbol sequence $\langle \texttt{THE}, \texttt{Y} \rangle$ into its constituent symbols. The correct symbol sequence $\langle \texttt{THE}, \texttt{YEAR} \rangle$ may then be formed via agglutination at a later stage.

`THE|Y|CAME|THE|Y|EAR|LATER.`

FIGURE 5.32: Example data which is to be UpWritten with the symbol sequence $\langle \texttt{THE}, \texttt{Y} \rangle$.

`THEY|CAME|THEY|EAR|LATER`

FIGURE 5.33: A greedy UpWriting process results in an erroneous chunk.

While it is obvious that a feedback mechanism of this sort would prove beneficial, we have not implemented anything so ambitious in the version of the UpWrite Predictor which we have developed. Instead, we decided to simply form a candidate symbol sequence and a candidate symbol class on each iteration of the algorithm, and evaluate the performance of two hypothetical UpWrite Predictors, one of which incorporates the candidate symbol sequence, and the other of which incorporates the candidate symbol class. The UpWrite Predictor which was found to perform the best effectively selects the type of structure which will be incorporated, and the process is then iterated. This process uses the feedback mechanism of the UpWrite Predictor to tentatively UpWrite the data with two candidate symbols, and then discards the symbol which results in the greatest performance degradation.

In the static case, inference is performed on a training corpus, and evaluation is performed by measuring the average information provided to the predictive model by a separate testing corpus. In the adaptive case, evaluation can be performed on the most recent portion of the data with respect to a predictive model inferred from the remaining data in the history.

5.5.5 Stopping Criterion

It seems intuitive that a suitable stopping criterion for the UpWrite Predictor would be to constantly monitor its performance and cease the UpWrite process as soon as this performance degrades. However, we have found that the performance of our particular implementation occasionally degrades immediately. This is due to the fact that the predictive model used is of a very low order, and the structure needed to improve the performance of such a model can sometimes only be found after several iterations of the algorithm, during which the processes of agglutination and agglomeration gradually form useful structure from the detrimental structure extracted at the beginning of the process, which serves as scaffolding.

A suitable stopping criterion is therefore not as easy to find as may have been imagined. In our experience we find that the UpWrite process may be stopped when a gradual degradation in performance is observed; this is usually indicative that all useful structure has been found, and that *hapax legomena* are now being uncovered. Their addition to the model serves to water down the statistics, resulting in a slight performance degradation on each iteration of the algorithm.

For some applications it may not be necessary to specify a stopping criterion at all. If one can afford the time, the UpWrite Process will halt naturally as soon as the entire data has been UpWritten to a single symbol. When this occurs the hierarchical description of the data is complete, and applications may select a level of abstraction *a posteriori*. Needless to say, it is difficult to imagine a circumstance when this approach is viable, unless the data being modelled consists of a relatively small number of symbols.

5.6 Summary and Conclusion

In this chapter we have shown how the UpWrite concept may be applied to a simple predictive model. The result is the UpWrite Predictor, a language model which uses none of the standard methods to improve its performance; rather, it bootstraps itself with structure discovered in the data from its own predictions. We have shown that two types of structure, symbol sequences and symbol classes, may be successfully uncovered in data via the gradual iterative processes of agglutination and agglomeration. In the next chapter we shall perform a variety of experiments using the implementation of the UpWrite Predictor which we have presented.

Notes

[17] Although for many tasks it is reasonable to train the predictive model before applying it to a problem, in some applications, particularly that of data compression, it is desirable to train the predictive model on the fly, while it is being applied to the problem.

[18] Module A implicitly knows the lowest level alphabet, which, for the purposes of this dissertation, is usually the set of bytes. Furthermore, Module A has no UpWriter, because it uses the lowest level alphabet instead of an UpWritten version of a lower level alphabet.

[19] Recall that DownWriting a symbol class is not defined, but lower level data can be generated by choosing a member of the symbol class at random, in accordance with the probabilities of the various symbols in the class.

[20] Each module will make its predictions at different times, and the problem of combining DownWritten versions of these predictions into a single prediction is therefore non-trivial.

[21] The term *chunk* was introduced by Miller in 1956 as a unit of immediate memory, and it is a vital concept of human information processing research [12, 14].

[22] Zellig Harris was Noam Chomsky's mentor, and was a well-known structural linguist.

[23] Such cases occur only when the successor count plateaus.

[24] The best performing context length was chosen to be the one which yielded the highest value of the *recall*.

[25] Note that the average word length in the Sherlock corpus is 4.18 characters, and 4.43 characters in the `Test` section. It seems probable that the optimal context length is related to the average word length, since the context will be observed more frequently if all of it falls within the same word.

[26] This is true only if all the symbols are equiprobable. In reality, the entropy is also related to the probability distribution over these symbols, something which Harris did not take into account, possibly due to the impossibility of getting frequency information from a human informant.

[27] A *referent* is a relationship with a linguistic form, such as a word or a sentence, and something in the real word, such as the object or action that is being referred to.

[28] This hierarchical decomposition is possible because the symbols which constitute the symbol pair may be lower level symbol pairs themselves.

[29] For clarity, the plot was truncated, with the most similar symbol pairs appearing to be much more closely correlated than they actually are.

[30] Consider, for example, the text formed by performing a search-replace function on whitespace, as with the command "`cat Sherlock | tr '[:space:]'  '*'`". Automatic identification of separator symbols would make it possible to easily segment this text, by recognizing that the asterisk character functions as a separator symbol in this instance.

[31] Anomalous in that they are infrequent, often only occurring once in the data.

[32] We shall show, however, that the Euclidean distance is a sufficient similarity measure.

[33] This should be possible, because knowledge of the k-simplices reduces the degrees of freedom of the observed data.

[34] This plot is a projection from $\mathbb{R}^4$ onto $\mathbb{R}^2$, as produced by the `xgobi` program.

[35] There are sixteen distinct vectors corresponding to the predictions made after the sixteen possible contexts observed by the 2^{nd}–order Markov model. Note that two of these vectors lie very near the symbol classes [`B`|`C`] and [`A`|`B`], and are therefore difficult to distinguish in the diagram.

[36] The problem is compounded by the fact that the bottom-most symbol class in the diagram is separated from the vectors which correspond to predictions by a significant amount.

[37] Actually, such a vector lies on the $(|\mathcal{A}|-1)$–*simplex* in $\mathbb{R}^{|\mathcal{A}|}$ as we have previously shown.

[38] We have made the implicit assumption throughout that clustering vectors of some sort is the best way of discovering symbol classes, but other, non-stochastic approaches exist. For instance, see Wolff's SNPR algorithm [27].

[39] Even so, an application which makes use of the hierarchical representation of the data formed by the UpWrite Predictor may supply its own predictive model which it can use externally for whatever purpose is required.

[40] This is due to the method of agglutination, which forms symbol sequences from highly correlated adjacent pairs of symbols.

[41] This is true of adaptive statistical data compressors, for example. The receiver must be able to parse the data in an identical fashion to the transmitter, which means that the transmitter cannot look-ahead into the sequence in order to make a parsing decision. We meet such systems in chapter 7.

References

[1] Doug Beeferman, Adam Berger, and John Lafferty. Text segmentation using exponential models. In *Proceedings of the Second Conference on Emperical Methods in Natural Language Processing*, 1997.

[2] Jeremy C Campbell. *Grammatical Man: Information, Entropy, Language and Life.* Pelican Books, 1984.

[3] Timothy A. Cartwright and Michael R. Brent. Syntactic categorization in early language acquisition: Formalizing the role of distributional analysis. *Cognition*, 62:121–170, 1997.

[4] Steven Paul Finch. *Finding Structure In Language.* PhD thesis, University of Edinburgh, 1993.

[5] Zellig S. Harris. From phoneme to morpheme. *Language*, 31:190–222, 1955.

[6] Zellig S. Harris. Distributional structure. In Henry Hiz, editor, *Papers on Syntax*, pages 3–22. Kluwer Boston, 1981.

[7] John R. Hayes and Herbert H. Clark. Experiments on the segmentation of an artificial speech analogue. In John R. Hayes, editor, *Cognition and the Development of Language*, pages 221–234. John Wiley & Sons, Inc., 1970.

[8] Jason L. Hutchens. Natural language grammatical inference. Honours thesis, Department of Electrical & Electronic Engineering, The University of Western Australia, Australia 6907, 1994.

[9] G.R. Kiss. Grammatical word classes: A learning process and its simulation. *Psychology of Learning and Motivation*, 7:1–41, 1973.

[10] Hideki Kozima. Text segmentation based on similarity between words. In *Proceedings of the* 31^{st} *Annual Meeting of the Association of Computational Linguistics*, pages 286–288, 1993.

[11] Marta Kutas and Steven A. Hillyard. Reading senseless sentences: Brain potential reflects semantic incongruity. *Science*, 207(11):203–205, January 1980.

[12] W.J.M. Levelt. Hierarchical chunking in sentence processing. *Perception & Psychophysics*, 8:99–103, 1970.

[13] Hang Li and Naoki Abe. Word clustering and disambiguation based on co-occurrence data. In *Proceedings of the* 36^{th} *Annual Meeting of the Association for Computational Linguistics and the* 17^{th} *International Conference on Computational Linguistics*, volume 2, pages 749–755, 1998.

[14] George A. Miller. The magical number seven, plus or minus two: Some limits on our capacity for processing information. *The Psychological Review*, 63(2):81–97, March 1956.

[15] Craig G. Nevill-Manning. *Inferring Sequential Structure.* PhD thesis, University of Waikato, 1996.

[16] Craig G. Nevill-Manning and Ian H. Witten. Identifying hierarchical structure in sequences: A linear-time algorithm. *Journal of Artificial Intelligence Research*, 7:67–82, 1997.

[17] Fernando Pereira, Naftali Tishby, and Lillian Lee. Distributional clustering of English words. In *International Colloquium on Grammatical Inference*, pages 5/1–3, April 1993.

[18] Martin Redington, Nick Chater, and Steven Finch. Distributional information: A powerful cue for acquiring syntactic categories. *Cognitive Science*, 22(4):425–469, 1998.

[19] Jeffrey C. Reynar and Adwait Ratnaparkhi. A maximum entropy approach to identifying sentence boundaries. In *Proceedings of the Fifth Conference on Applied Natural Language Processing*, March 1997.

[20] Jenny R. Saffran, Elissa L. Newport, and Richard N. Aslin. Word segmentation: The role of distributional cues. *Journal of Memory and Language*, 35:606–621, 1996.

[21] Hinrich Schütze. Part-of-speech induction from scratch. In *Proceedings of the* 31^{st} *Annual Meeting of the Association for Computational Linguistics*, pages 251–258, 1993.

[22] C.E. Shannon. Prediction and entropy of printed English. *The Bell System Technical Journal*, XXX(1):50–64, January 1951.

[23] Peter H.A. Sneath and Robert R. Sokal. *Numerical Taxonomy.* W.H. Freeman and Company, 1973.

[24] Walter Stolz. A probabilistic procedure for grouping words into phrases. *Language and Speech*, 8:219–235, 1965.

[25] J. G. Wolff. An algorithm for the segmentation of an artificial language analogue. *British Journal of Psychology*, 66(1):79–90, 1975.

[26] J. G. Wolff. The discovery of segments in natural language. *British Journal of Psychology*, 68:97–106, 1977.

[27] J. Gerard Wolff. Language acquisition, data compression and generalization. *Language & Communication*, 2(1):57–89, 1982.

Chapter 6

Experiments with the UpWrite Predictor

> "I have no data yet. It is a capital mistake to theorize before one has data. Insensibly one begins to twist facts to suit theories, instead of theories to suit facts."
>
> *The Adventures of Sherlock Holmes*
> Sir Arthur Conan Doyle

6.1 Introduction

The UpWrite Predictor has been designed, and our particular implementation of it described, so we are now in a position to evaluate its performance. Ultimately we would like to show that the UpWrite Predictor is capable of finding structure in all sorts of data, and that it may be used in a wide range of real-world applications. However, given that this dissertation has seen the development of the UpWrite Predictor as a new paradigm for the modelling of symbolic time series, we shall content ourselves with evaluating its performance on a range of artificially generated data, and a small corpus of natural language text, leaving exploration of applications of the technique for future work.

We choose to perform experiments with data generated by simple phrase-structure grammars, and this choice is motivated in part by the fact that such grammars have been used in the past to evaluate the performance of other language acquisition algorithms [4]. Phrase-structure grammars are capable of embodying the two types of structure which the UpWrite Predictor has been designed to detect. In all cases where a phrase-structure grammar is used to provide data for experimental purposes, we create a testing corpus from 10000 generations of the grammar and a training corpus from 1000 generations of the grammar. The final length of these corpora is dependent on the average length of a generation in the phrase-structure grammar being used.

Note that throughout this chapter experiments shall be performed using a 1^{st}–order

UpWrite Predictor only; although it is possible to use higher order Markov models in the UpWrite Predictor we are will not be exploring the performance of such models, as our interest lies in improving the performance of simple predictive models by augmenting the models with structure found in the data by other means.

6.1.1 Overview

In this chapter we use simple phrase-structure grammars to test the performance of the UpWrite Predictor at acquiring symbol sequences, symbol classes, and a combination of the two, both in cases where the symbol classes are ambiguous, and in cases where they are not. We then proceed to evaluate the performance of the UpWrite Predictor on the seven texts developed by Gerry Wolff in order to evaluate the performance of his SNPR algorithm [4]. In all cases we specify the phrase-structure grammar used in a way consistent with Wolff. Following this, we evaluate the performance of the UpWrite Predictor on natural language text, and we show that an intuitive feeling for the performance of the UpWrite Predictor can be obtained by *eye-balling* the DownWritten form of the abstracted version of the data it creates, and the data which it generates. These experiments go some way to illustrating the potential advantages of the UpWrite technique in the domain of automatic natural language acquisition.

6.2 Discovering Symbol Sequences

The phrase-structure grammar shown in figure 6.1 was designed with the intention of creating a basic data source capable of generating data containing unambiguous higher level symbol sequences. It was used to generate a training corpus 30000 bytes in length, and a testing corpus 3000 bytes in length.

$$
\begin{array}{lll}
\# & \mapsto & (1)|(2)|(3) \\
1 & \mapsto & \texttt{AAA} \\
2 & \mapsto & \texttt{BBB} \\
3 & \mapsto & \texttt{CCC}
\end{array}
$$

Sample: `CCCBBBCCCAAAAAACCCBBBBBBCCCBBBCCCAAABBBAAA` ...

FIGURE 6.1: The phrase-structure grammar used to generate data for testing the acquisition of symbol sequences.

The UpWrite Predictor was inferred from the training corpus, and its performance on the testing corpus was evaluated on each iteration. The structure found by the UpWrite Predictor which resulted in the lowest performance degradation was selected, both corpora were UpWritten, and the process was iterated.

The plot shown in figure 6.2 shows how the performance of the UpWrite Predictor changes as the data is UpWritten to higher levels of abstraction. Note that the first three iterations of the algorithm result in the greatest gains in performance, and that performance continually improves until the thirty-eighth iteration, at which stage it begins to gradually degrade. The best performance the UpWrite Predictor attains is 40.8% better than the performance of a 1^{st}–order Markov model, even though the UpWrite Predictor is nothing more than a 1^{st}–order Markov model which processes a higher level version of the data constructed from an analysis of its own sequence of predictions.

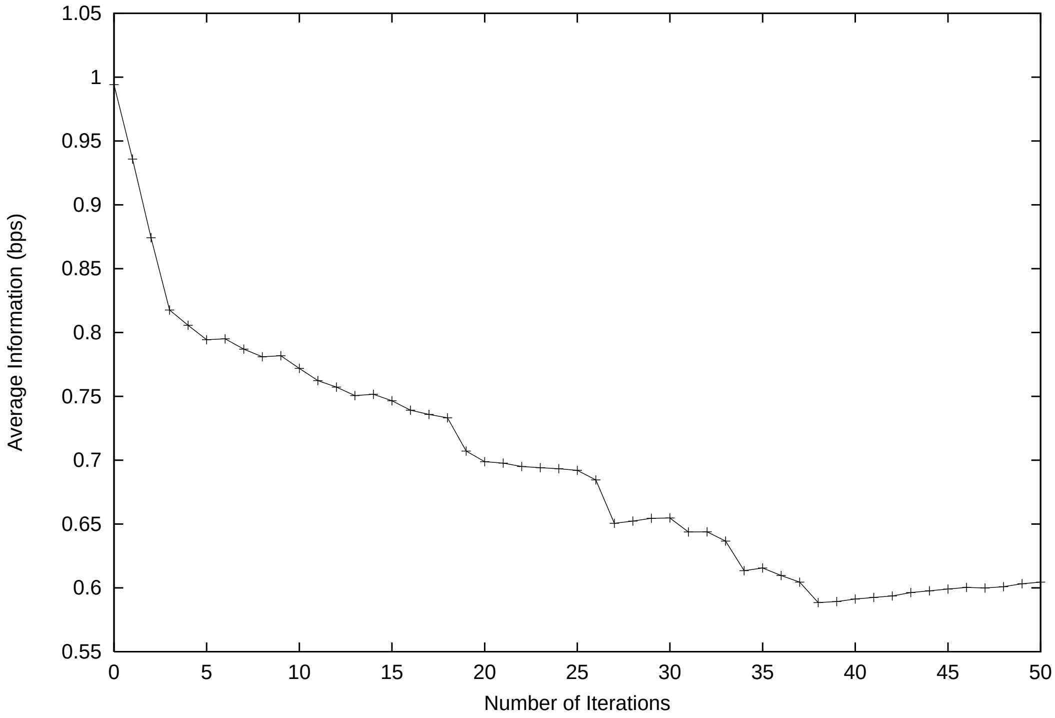

FIGURE 6.2: The performance of the UpWrite Predictor at the end of each iteration as evaluated on data generated by the phrase-structure grammar of figure 6.1.

Table 6.1 shows the first thirteen higher level symbols which were discovered by the UpWrite Predictor and used to UpWrite the data, in the order of discovery. Note that the first three UpWrites performed were with respect to the symbol sequences `CC`, `BB` and `AA`, and it is not surprising that the greatest performance gain is observed at this stage, as these symbol sequences provide enough context to enable the UpWrite Predictor to make a low entropy prediction about the next symbol in the data.

Unfortunately, the three symbol sequences which exist in the grammar of figure 6.1 were not found by the UpWrite Predictor. This is due to the fact that the corpora contain long repetitions of the symbols `A`, `B` and `C`, with the consequence that in the UpWritten representation of the corpora the symbol `AA`, for example, may be followed by the symbols `A`, `AA`, `BB` and `CC`, while the symbol `A`, for example, may be followed only by `BB` or `CC`. This is a side-effect of the form of greedy parsing used to UpWrite the data.

It should be noted that the longer repetitions of symbols such as `AAAAAA`, `BBBBBB` and `CCCCCC` are eventually discovered and UpWritten by the algorithm, and the points at which this occurs correspond to the noticeable performance improvements at the nineteenth, twenty-seventh and thirty-fourth iterations.

⟨CC⟩
⟨BB⟩
⟨AA⟩
⟨BAA⟩
⟨BCC⟩
⟨BAAA⟩
⟨BAAAA⟩
⟨BAAAAAA⟩
⟨BCCC⟩
⟨BCCCC⟩
⟨BCCCCCC⟩
⟨CAA⟩
⟨CBB⟩

TABLE 6.1: A list of the structure found by the UpWrite Predictor on data generated by the phrase-structure grammar of figure 6.1, shown in the order of acquisition.

6.3 Discovering Symbol Classes

In order to test the performance of the UpWrite Predictor in discovering symbol classes in data, the simple phrase-structure grammar of figure 6.3 was used to generate a 30000 byte training corpus and a 3000 byte testing corpus, and the UpWrite Predictor was inferred from and evaluated on these corpora in the standard manner.

$$
\begin{array}{lll}
\# & \mapsto & (1)(2)(3) \\
1 & \mapsto & \mathtt{A|B} \\
2 & \mapsto & \mathtt{P|Q} \\
3 & \mapsto & \mathtt{X|Y}
\end{array}
$$

Sample: `APYAQYAPXBPYBPYBPXBQYAQXAQYAPXBQXAPYBPXBPX` ...

FIGURE 6.3: The phrase-structure grammar used to generate data for testing the acquisition of unambiguous classes.

Note that the data generated by the grammar contains three unambiguous symbol classes. It also contains a single symbol sequence at a higher level than these symbol classes. Table 6.2 lists the five higher level symbols which were discovered by the UpWrite Predictor, in the order of acquisition, and it can be seen that the first three of these correspond to the three symbol classes of the grammar, while the last of these corresponds to the higher level symbol sequence (1)(2)(3). Once the data has been UpWritten using this structure, no more symbol classes exist to be found, and so the UpWrite Predictor halts.

It should be noted that UpWriting the data with the structure discovered by the UpWrite Predictor doesn't significantly alter its performance, as evaluation is performed over the lowest level alphabet, meaning that predictions made at a higher level must be DownWritten. Symbol classes were discovered merely because the addition of symbol sequences such as `AQ` would have adversely affected performance. If, however, evaluation

`[P|Q]`
`[A|B]`
`[Y|X]`
`⟨[P|Q][Y|X]⟩`
`⟨[A|B][P|Q][Y|X]⟩`

TABLE 6.2: A list of the structure found by the UpWrite Predictor on data generated by the phrase-structure grammar of figure 6.3, shown in the order of acquisition.

was performed by measuring the performance of the UpWrite Predictor on the higher level data, *without* DownWriting the predictions, we would find that the performance of the UpWrite Predictor would improve dramatically with the addition of the three symbol classes. This evaluation procedure is equivalent to a process of lossy compression, as the exact form of the lower level data is not recoverable from the higher level predictions, and we shall use this evaluation procedure in section 6.9 when we generate data using an UpWrite Predictor inferred from natural language text.

A second phrase-structure grammar, shown in figure 6.4, was used to test the ability of the UpWrite Predictor to acquire symbol classes which are ambiguous insofar that some of the symbols in the alphabet belong to more than one symbol class. A training corpus of 30000 bytes and a testing corpus of 3000 bytes were generated from the grammar, and the performance of the UpWrite Predictor was then evaluated.

#	$\mapsto$	(1)(2)(3)
1	$\mapsto$	`A|B|P`
2	$\mapsto$	`P|Q`
3	$\mapsto$	`X|Y|Q`

Sample: `BPYBPXBQXAPYAQQAPYPPXPPYBPXBQQPQQPQQAPXBQY` ...

FIGURE 6.4: The phrase-structure grammar used to generate data for testing the acquisition of ambiguous classes.

In figure 6.5 we show a plot of the performance of the UpWrite Predictor on this data, and it is immediately obvious that the performance of the UpWrite Predictor degrades initially before improving—it is this phenomenon which makes determining a suitable stopping criterion difficult.

Table 6.3 lists the first nine symbols found by the UpWrite Predictor from this data. The first three symbols found correspond to unambiguous symbol classes, identical to those of the previous example. An immediate problem is apparent—because the UpWrite Predictor assigns a symbol to one, and only one, symbol class, every occurence of the symbols `P` and `Q` in the data are UpWritten to the symbol class `[P|Q]`, making it impossible to discover the ambiguous symbol classes at a later iteration of the algorithm.

Even so, the discovery of the symbol sequences shown does serve to disambiguate the predictions made by the UpWrite Predictor, and its performance improves as a consequence of their inclusion. Overall, though, the performance of the algorithm in discovering ambiguous symbol classes is poor, and this is to be expected in the light of the discussion

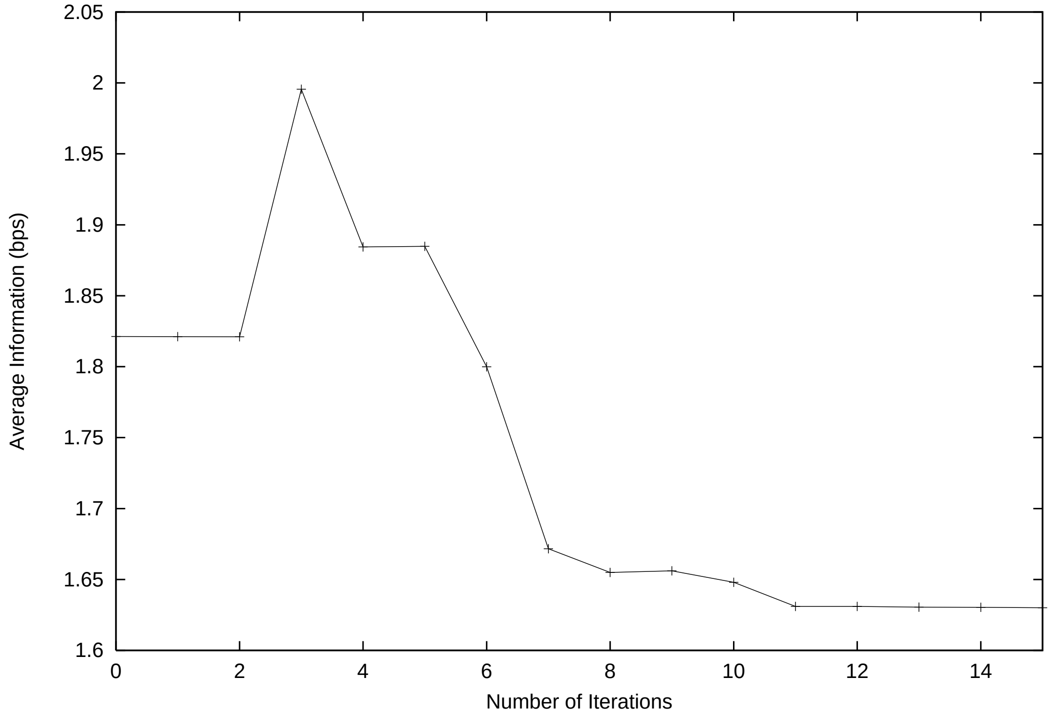

FIGURE 6.5: The performance of the UpWrite Predictor at the end of each iteration as evaluated on data generated by the phrase-structure grammar of figure 6.4.

[Y|X]
[A|B]
[P|Q]
⟨[A|B][P|Q]⟩
⟨[A|B][P|Q][Y|X]⟩
⟨[A|B][P|Q][P|Q]⟩
⟨[P|Q][P|Q]⟩
⟨[P|Q][P|Q][Y|X]⟩
⟨[P|Q][P|Q][P|Q]⟩

TABLE 6.3: A list of the structure found by the UpWrite Predictor on data generated by the phrase-structure grammar of figure 6.4, shown in the order of acquisition.

as to the nature of this problem in the previous chapter. The UpWrite Predictor is further hampered due to the fact that a context of only one symbol is not sufficient to disambiguate its predictions—better performance would be attained with a larger context, or the ability to contextually constrain the symbol classes discovered.

6.4 Finding Both Types of Structure

We are now interested in evaluating the ability of the UpWrite Predictor to find both symbol classes and symbol sequences in data. The phrase-structure grammar of figure 6.6 was used to generate a training corpus of 90000 bytes and a testing corpus of 9000 bytes. The generated data exhibits symbol sequences at two levels, and unambiguous symbol classes.

#	$\mapsto$	(1)(2)(3)
1	$\mapsto$	`AAA|BBB`
2	$\mapsto$	`PPP|QQQ`
3	$\mapsto$	`XXX|YYY`

Sample: `AAAQQQXXXBBBQQQYYYBBBQQQYYYAAAQQQXXXAAAPPP` ...

FIGURE 6.6: The phrase-structure grammar used to generate data for testing the acquisition of symbol sequences and unambiguous symbol classes.

In figure 6.7 we plot the performance of the UpWrite Predictor as it is inferred from this data. This plot has an interesting 'staircase' structure, descending from an initial performance equivalent to that of an ordinary 1^{st}–order Markov model, and levelling off after eleven iterations of the algorithm.

All of the symbols discovered by the UpWrite Predictor, and used to UpWrite the data, are listed in table 6.4—the algorithm halted after these symbols were discovered due to the fact that no symbol classes remained to be found. All of the lower level symbol sequences are discovered by the algorithm first; the 'staircase' nature of the plot being explained by the fact that incorporation of symbol sequences such as `XX` provides a significant performance improvement, due to the fact that it allows the model to make a low entropy prediction about the next symbol in the data,[42] while incorporation of longer symbol sequences such as `XXX` do not alter the performance of the model.[43]

Once the symbol sequences had been discovered by the UpWrite Predictor, and used to UpWrite the data, the three symbol classes were found. The incorporation of these classes into the alphabet do not significantly alter the performance of the UpWrite Predictor, but the incorporation of larger symbol sequences would adversely affect the performance by a significant amount, which explains why the symbol classes are added. Once the data has been UpWritten with respect to the three symbol classes, the higher level symbol sequence (1)(2)(3) is uncovered in two iterations of the algorithm. The performance of the UpWrite Predictor in this simple example is perfect.

The phrase-structure grammar shown in figure 6.8 was used to generate a similar data

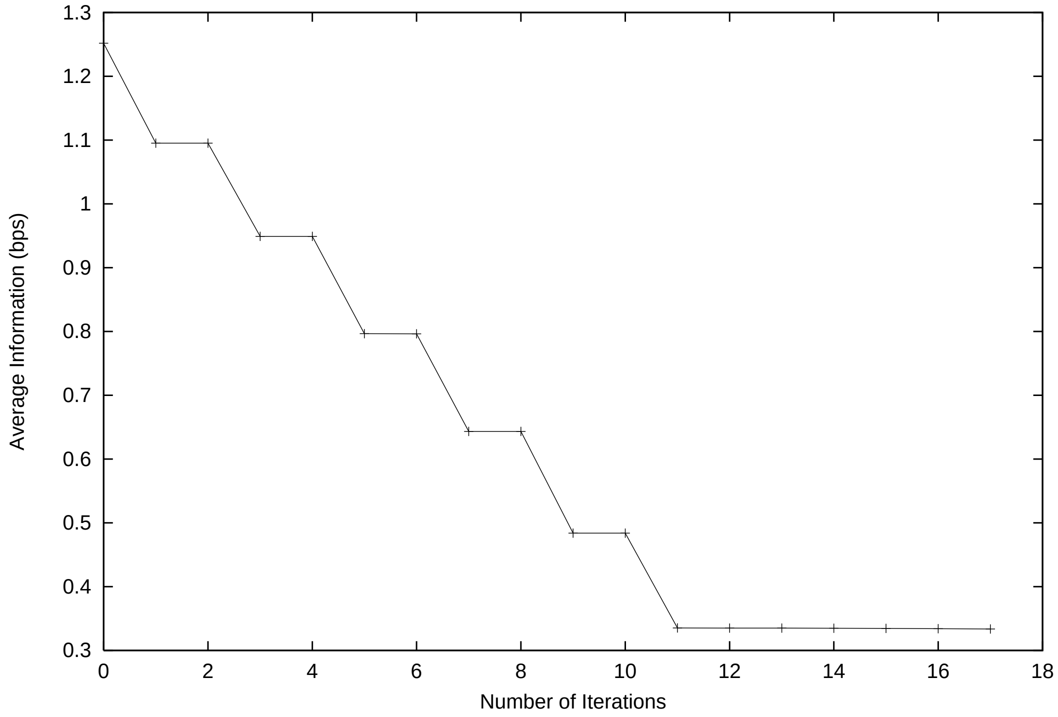

FIGURE 6.7: The performance of the UpWrite Predictor at the end of each iteration as evaluated on data generated by the phrase-structure grammar of figure 6.6.

⟨XX⟩
⟨XXX⟩
⟨AA⟩
⟨AAA⟩
⟨QQ⟩
⟨QQQ⟩
⟨PP⟩
⟨PPP⟩
⟨BB⟩
⟨BBB⟩
⟨YY⟩
⟨YYY⟩
[AAA|BBB]
[XXX|YYY]
[QQQ|PPP]
⟨[AAA|BBB][QQQ|PPP]⟩
⟨[AAA|BBB][QQQ|PPP][XXX|YYY]⟩

TABLE 6.4: A list of the structure found by the UpWrite Predictor on data generated by the phrase-structure grammar of figure 6.6, shown in the order of acquisition.

sequence to that shown of the previous example, but in this case the symbol classes were made ambiguous in two ways: some of the symbol sequences belong to more than one symbol class, and two higher level symbol sequences exist, meaning that a particular class may be followed by more than one other class. As we discussed in the previous chapter, it is very difficult indeed to devise an algorithm which can recover symbol classes of this sort from data.

#	$\mapsto$	(1)(2)(3)|(1)(3)(2)
1	$\mapsto$	`AAA`|`BBB`|`PPP`
2	$\mapsto$	`PPP`|`QQQ`
3	$\mapsto$	`XXX`|`YYY`|`QQQ`

Sample: `AAAQQQYYYBBBYYYQQQBBBXXXPPPPPPQQQYYYAAAYYY` ...

FIGURE 6.8: The phrase-structure grammar used to generate data for testing the acquisition of symbol sequences, ambiguous symbol classes and 'phrases'.

The plot shown in figure 6.9 illustrates the performance of the UpWrite Predictor on this data, and we notice that the plot initially exhibits the same staircase structure of the previous example, with a slight degradation in performance occurring once the performance has levelled off.

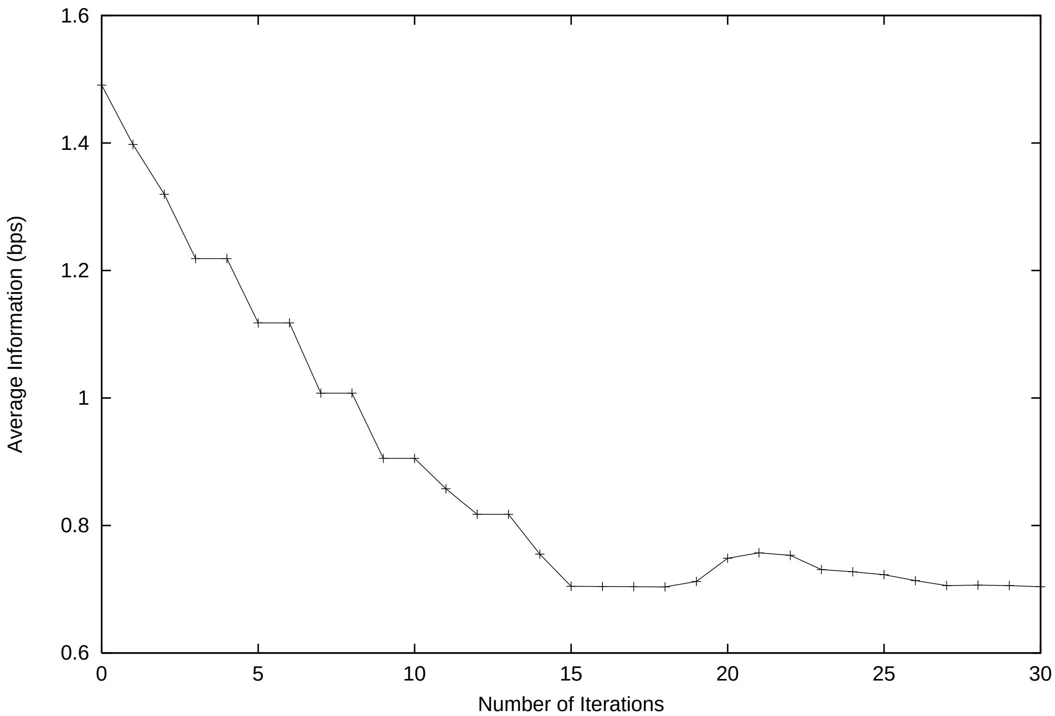

FIGURE 6.9: The performance of the UpWrite Predictor at the end of each iteration as evaluated on data generated by the phrase-structure grammar of figure 6.8.

The first twenty-two symbols found by the UpWrite Predictor from this data are listed in table 6.5 in the order in which they were discovered. We note that the algorithm begins by discovering portions of the two symbol sequences which are ambiguous in the sense that they belong to more than one symbol class. The fact that they were discovered first is no doubt because they occur more frequently in the data. The remaining four unambiguous symbol sequences are then discovered, and this corresponds to the 'staircase' region of the plot. The UpWrite Predictor then proceeds to discover some rather curious symbol sequences. Not only are the correct ambiguous symbol sequences PPP and QQQ discovered, but their concatenations, PPPPPP and QQQQQQ, are also discovered. This is due to the fact that the ambiguous nature of the correct symbol sequences causes them to frequently occur in pairs in the data.

⟨PP⟩
⟨QQ⟩
⟨BB⟩
⟨BBB⟩
⟨YY⟩
⟨YYY⟩
⟨AA⟩
⟨AAA⟩
⟨XX⟩
⟨XXX⟩
⟨PPP⟩
⟨PPPP⟩
⟨PPPPPP⟩
⟨QQQ⟩
⟨QQQQ⟩
⟨QQQQQQ⟩
[YYY|XXX]
[BBB|AAA]
[PPPPPP|BBB|AAA]
[PPP|QQQ]
[QQQQQQ|YYY|XXX]
[PPPPPP|BBB|AAA|PPP|QQQ]

TABLE 6.5: A list of the structure found by the UpWrite Predictor on data generated by the phrase-structure grammar of figure 6.8, shown in the order of acquisition.

Once these symbol sequences have been discovered, classes are formed from the four unambiguous symbol sequences AAA, BBB, XXX and YYY. The symbol class [PPP|QQQ] is also formed, as are two symbol classes which are almost identical to the ambiguous classes of the grammar used to generate the data. It is at this stage that the performance of the algorithm begins to degrade. It is indeed encouraging that the ambiguous symbol classes were recovered to an extent, and this seems to be due to the fact that two higher level symbol sequences (1)(2)(3) and (1)(3)(2) exist in the data. These higher level symbol sequences, unfortunately, were not discovered by the algorithm.

6.5 Performance on a Random Source

We shall end our examination of the performance of the UpWrite Predictor with a look at its performance on data generated by a random memoryless source. The simple phrase-structure grammar of figure 6.10 was used to generate a training corpus 10000 bytes in length, and a testing corpus 1000 bytes in length, with the generated data consisting of a random sequence of symbols taken from an alphabet of six symbols.

$\# \mapsto$ A|B|P|Q|X|Y
Sample: `XQPAPYAPABBQBPBYXQAXQAQXBPQPYBPBYQXXQQQBBB` ...

FIGURE 6.10: The phrase-structure grammar used to generate a random sequence of symbols.

The performance of the UpWrite Predictor on this data is plotted in figure 6.11, and it is plain that structure found in the training corpus by the UpWrite Predictor serves only to degrade its performance on the testing corpus. This is the exact behaviour we expect to see in the UpWrite Predictor when it is inferred from such data—any structure discovered in the training corpus is due only to statistical irregularities caused by the fact that only a finite amount of data is available for inference, and the statistical irregularities which exist in the training corpus do not generalise across to the testing corpus.

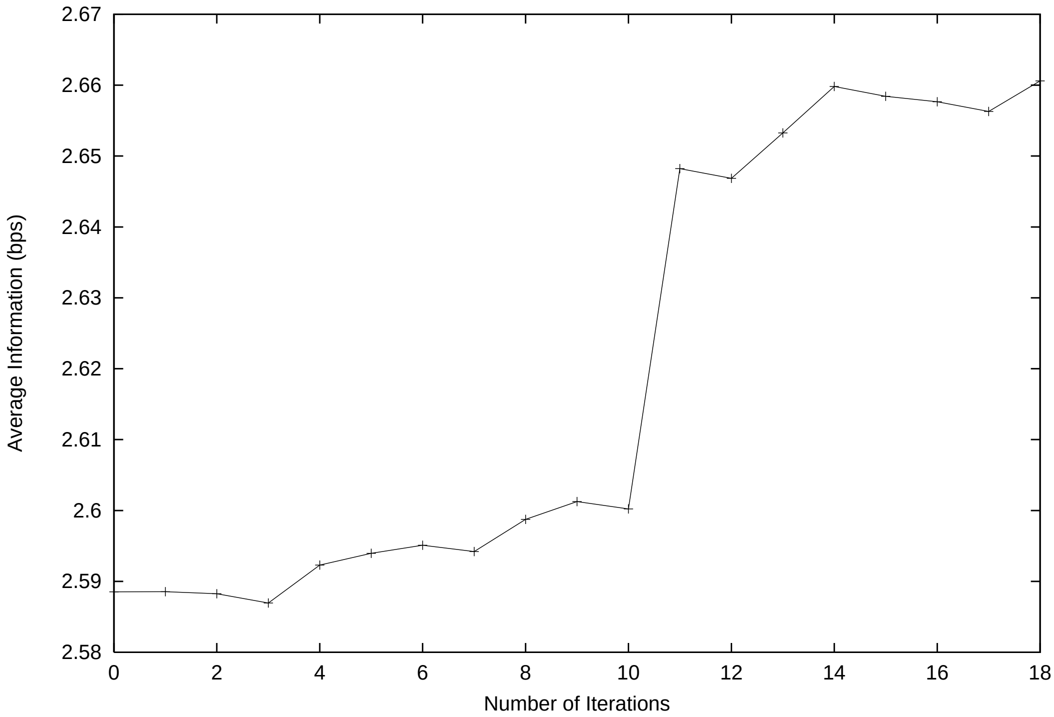

FIGURE 6.11: The performance of the UpWrite Predictor at the end of each iteration as evaluated on data generated by the phrase-structure grammar of figure 6.10.

We decline to give a list of the symbols discovered by the UpWrite Predictor in the random data, as doing so would not be enlightening.

We have evaluated the performance of the UpWrite Predictor using data generated by various simple phrase-structure grammars, with the results that the UpWrite Predictor performs about as well as we could hope it to, given that it is based on a 1^{st}–order Markov model, and is unable to find contextually constrained symbol classes. We would now like to proceed to performing experiments on slightly more complicated data.

6.6 Evaluation on Quasi-English Data

Gerry Wolff devised seven artificial texts of a quasi-English sort in order to evaluate the performance of his algorithm, SNPR, which is capable of discovering symbol sequences and symbol classes in data [4]. We shall discuss Wolff's SNPR algorithm briefly in the next chapter, where we present Wolff's thesis that language acquisition, and learning in general, may be viewed as a process of data compression. Wolff devised SNPR in an attempt to explain the language acquisition process in human beings, and considers the seven artificial texts which we shall be performing experiments with in this section to be "analogues of the linguistic data available to children".

Due to the number of iterations performed in the experiments which follow, it is not practical for us to list all of the structure uncovered by the UpWrite Predictor. Instead, we choose to show only the structure which exists in the UpWritten version of the training corpus when the algorithm is stopped, together with the occasionally symbol sequence or symbol class in order to illustrate the order in which certain other symbols were discovered. All remaining structure found by the UpWrite Predictor served as *scaffolding* during inference, but, having done its job of enabling the discovery of higher level structure, no longer appears in the UpWritten form of the data.

6.6.1 Text 1

In figure 6.12 we reproduce the phrase-structure grammar used by Wolff [4] to generate Text 1. This grammar was used to generate a training corpus 109954 bytes in length and a testing corpus 11002 bytes in length. Wolff devised this corpus in order to generate data which contained a discontinuous dependency between the symbol sequences `PUT` and `ON`.

#	$\mapsto$	(1)`PUT`(2)`ON`|(1)`MADE`(2)
1	$\mapsto$	`JOAN`|`LIZ`|`YOU`|`HE`
2	$\mapsto$	`SOME`|`THEM`|`FOUR`|`IT`

Sample: `YOUPUTITONHEPUTTHEMONHEMADEITLIZMADESOME`...

FIGURE 6.12: The phrase-structure grammar used to generate Text 1.

The plot shown in figure 6.13 shows that the performance of the UpWrite Predictor improves steadily during the first twenty-five iterations of the algorithm, after which it begins to degrade slightly.

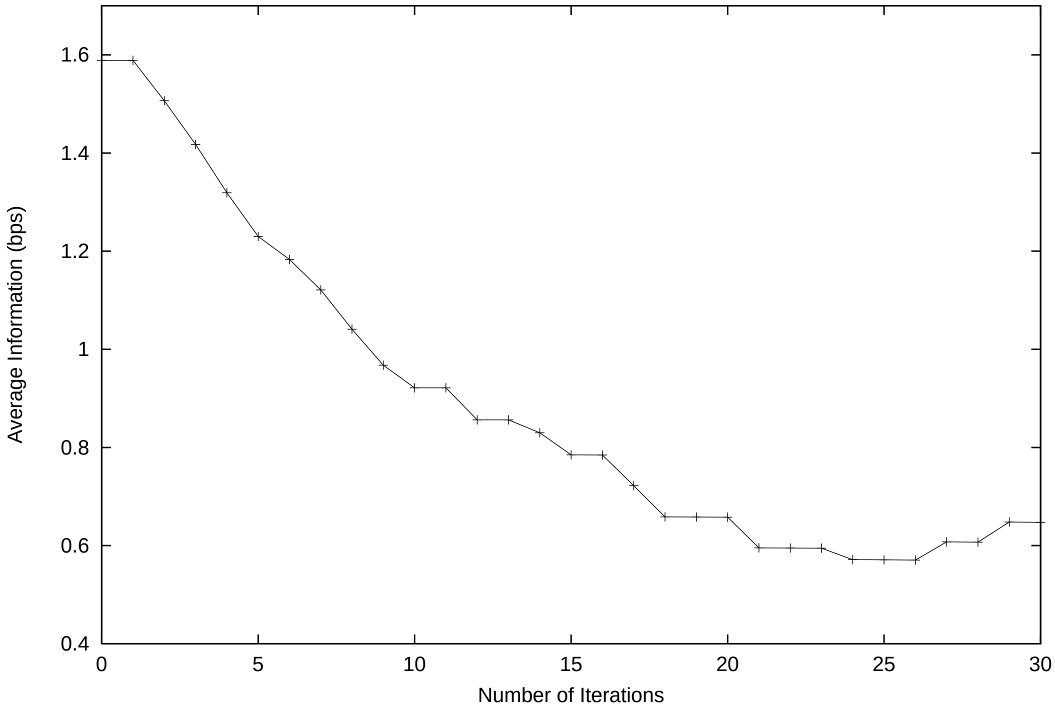

FIGURE 6.13: The performance of the UpWrite Predictor at the end of each iteration as evaluated on data generated by the phrase-structure grammar of figure 6.12.

In table 6.6 we list some of the structure found in the data by the UpWrite Predictor—for clarity we include the iteration at which the structure was discovered by the algorithm and used to UpWrite the data. The structure not shown tended to consist mostly of parts of the symbol sequences shown in the list, and functioned as scaffolding so that the longer symbol sequences which correspond to words in Wolff's grammar could be found.

The UpWrite Predictor finds all eleven words which exist according to the grammar which was used to generate the data. It then forms the symbol sequence `THEMON`, and this behaviour may be explained by the fact that the symbol sequence `THEM` is ambiguous in the generated data, as it also occurs in generations such as `YOUMADEITHEMADESOME`. The inclusion of the sequence `THEMON` therefore serves to improve the performance of the UpWrite Predictor, as it disambiguates the context. The UpWrite Predictor forms the two symbol classes implicit in the data, but neglects to add the ambiguous[44] symbol sequences `HE`, `THEM` and `IT` to these classes. The symbol class [`THEM`|`ON`|`THEMON`] serves to correct the error made by the UpWrite Predictor when it formed the symbol sequence `THEMON` prior to assigning `THEM` to a symbol class. The symbol sequences `PUT` and `MADE` are assigned to a class because they each occur between the classes (1) and (2), and this appears to be a sensible classification. The consequence of this classification is that the UpWrite Predictor fails to capture the discontinuous dependency implicit in the data.

The UpWrite Predictor then proceeds to form symbol sequences which correspond to the high level generations in the grammar, with the exception that these symbol sequences

4 ⟨PUT⟩
7 ⟨JOAN⟩
10 ⟨SOME⟩
11 ⟨HE⟩
13 ⟨MADE⟩
16 ⟨FOUR⟩
18 ⟨IT⟩
19 ⟨THEM⟩
20 ⟨LIZ⟩
22 ⟨ON⟩
23 ⟨YOU⟩
24 ⟨THEMON⟩
26 [JOAN|LIZ|YOU]
28 [SOME|FOUR]
29 [THEM|ON|THEMON]
30 [PUT|MADE]
35 ⟨[JOAN|LIZ|YOU][PUT|MADE][SOME|FOUR⟩
36 ⟨[JOAN|LIZ|YOU][PUT|MADE]IT⟩
37 ⟨[JOAN|LIZ|YOU][PUT|MADE][THEM|ON|THEMON][JOAN|LIZ|YOU][PUT|MADE][SOME|FOUR]⟩

TABLE 6.6: A list of the structure found by the UpWrite Predictor on data generated by the phrase-structure grammar of figure 6.12, shown in the order of acquisition.

are affected by the errors made early on in the inference process. The symbol sequences constructed at iterations 35, 36 and 37 cover many of the generations of the original grammar, the notable exception being generations which begin with the symbol sequence `HE`, and they are capable of generalising to new generations, such as `YOUPUTSOME` and `JOANMADETHEMON`.

6.6.2 Text 2

In the phrase-structure grammar shown in figure 6.14, reproduced from [4], the symbol sequence `IT` belongs to two symbol classes. This grammar was used to generate a training corpus of 121338 bytes and a testing corpus of 12205 bytes.

\# ↦ (1)(2)ES(3)|(4)(2)(3)
1 ↦ IT|TOM|JANE|JOAN
2 ↦ MISS|WATCH
3 ↦ BIRDS|LIZ|THEM|IT
4 ↦ YOU|MEN|WE|THEY
Sample: TOMMISSESTHEMYOUWATCHBIRDSTHEYMISSLIZ ...

FIGURE 6.14: The phrase-structure grammar used to generate Text 2.

The plot of figure 6.15 indicates that although the performance of the UpWrite Predictor slightly degrades after the first iteration of the algorithm, performance improves rapidly after that, and we start to see it begin to level off at the twentieth iteration, and degrade at the forty-first.

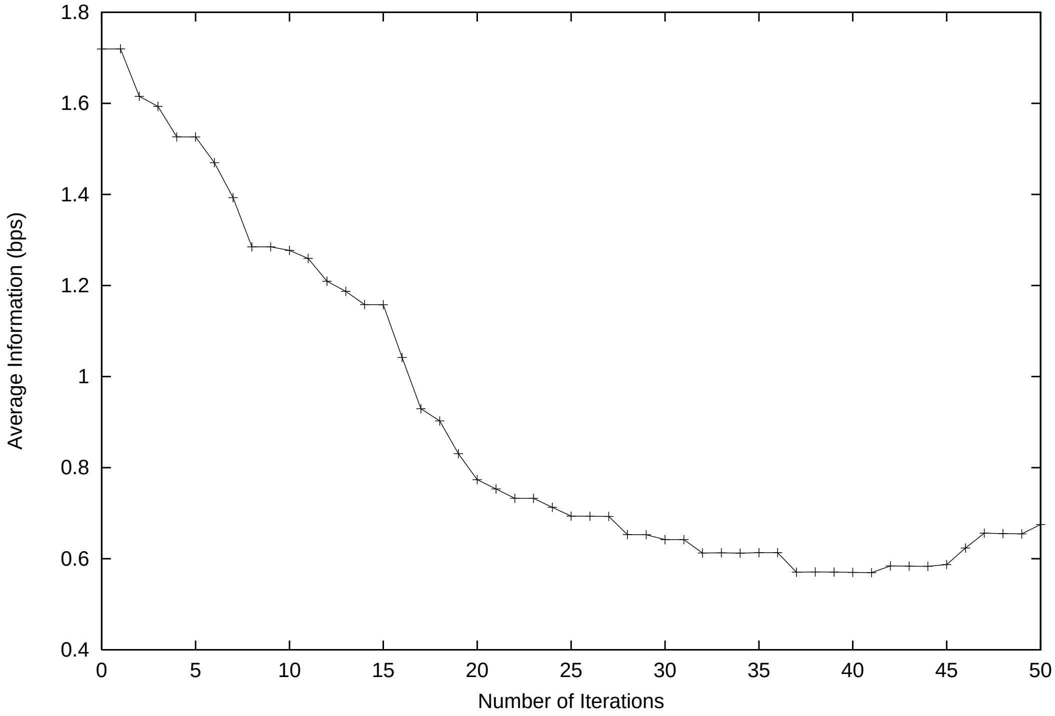

FIGURE 6.15: The performance of the UpWrite Predictor at the end of each iteration as evaluated on data generated by the phrase-structure grammar of figure 6.14.

Some of the structure discovered in this data by the UpWrite Predictor is listed in table 6.7, along with the iteration in which it was found. We see that the UpWrite Predictor forms most of the symbol sequences in the grammar to begin with, including the symbol sequence `ES` which it joins to the the symbol sequence `MISS` on the twenty-eighth iteration, and to the symbol sequence `WATCH` on the thirty-second.

The performance of the UpWrite Predictor at finding symbol classes in this data is relatively poor. This is partially due to the fact that the symbol class [`WE`|`MEN`] is used to form phrases in iterations 30 and 31, which means that the symbol sequences `YOU` and `THEY` are omitted from that symbol class permanently. In fact, a second symbol class is created from these two symbol sequences in iteration 40.

The presence of the ambiguous symbol sequence `IT` seems to hinder the acquisition of symbol classes, and this symbol sequence is not included in a symbol class until iteration 52, which is the only time a correct symbol class was formed by the algorithm. It should also be noted that the formation of the incorrect symbol class [`YOU`|`THEY`|`JANE`|`TOM`] is reasonable, as it represents a combination of some of the members of symbol class (1) and symbol class (4) in the grammar, both of which begin generations, and both of which are followed by symbol class (2).

Finally, note that the algorithm formed the incorrect symbol class [`L`|`B`] at the first iteration, and this resulted in the incorrect symbol sequences ⟨[`L`|`B`]`IZ`⟩ and ⟨[`L`|`B`]`IRDS`⟩ being formed at a later stage. This behaviour may be explained by the fact that the

1 [L|B]
9 ⟨WATCH⟩
11 ⟨WE⟩
15 ⟨YOU⟩
18 ⟨MISS⟩
19 ⟨IT⟩
20 ⟨ES⟩
22 ⟨JANE⟩
26 ⟨MEN⟩
27 ⟨JOAN⟩
28 ⟨MISSES⟩
29 [WE|MEN]
30 ⟨[WE|MEN]WATCH⟩
31 ⟨[WE|MEN]MISS⟩
32 ⟨WATCHES⟩
33 ⟨[L|B]IYZ⟩
34 ⟨[L|B]IRDS⟩
35 ⟨JOANMISSES⟩
36 ⟨JOANWATCHES⟩
37 ⟨THEM⟩
38 ⟨TOM⟩
39 ⟨THEY⟩
40 [YOU|THEY]
48 [JANE|TOM]
51 [YOU|THEY|JANE|TOM]
52 [IT|[L|B]IZ|⟨[L|B]IRDS⟩|THEM]

TABLE 6.7: A list of the structure found by the UpWrite Predictor on data generated by the phrase-structure grammar of figure 6.14, shown in the order of acquisition.

two symbols `L` and `B` are only ever followed by the frequently occurring symbol `I` in the data, and therefore the two symbols are greedily classified by the agglomeration process. The presence of this incorrect symbol class causes the UpWrite Predictor to generalise to productions such as `THEYMENMISSBIZ` and `THEYWATCHESLIRDS`.

6.6.3 Text 3

The phrase-structure grammar of figure 6.16, reproduced from [4], exhibits a hierarchical organisation in that the symbol class (1) may be decomposed either to the sequence of symbol classes (2)(3) or the symbol sequence `JOHN`. This grammar was used to generate a training corpus 121035 bytes in length, and a testing corpus 12179 bytes in length.

#	$\mapsto$	(1)(4)(5)|(6)(7)(8)
1	$\mapsto$	(2)(3)|`JOHN`
2	$\mapsto$	`A`|`THE`
3	$\mapsto$	`BOY`|`GIRL`
4	$\mapsto$	`LIKES`|`ATE`
5	$\mapsto$	`FISH`|`MEAT`
6	$\mapsto$	`WE`|`THEY`
7	$\mapsto$	`WALK`|`RUN`
8	$\mapsto$	`FAST`|`SLOWLY`

Sample: `AGIRLATEFISHTHEYWALKFASTJOHNATEMEAT` ...

FIGURE 6.16: The phrase-structure grammar used to generate Text 3.

The plot of the performance of the UpWrite Predictor on this data, shown in figure 6.17, shows that a steady performance improvement occurs over the first sixty iterations of the algorithm, with a gradual performance degradation occurring after that.

Table 6.8 lists some of the structure found in the data by the UpWrite Predictor, together with the iteration during which the structure was found. We see that the UpWrite Predictor forms phrases beginning with the symbol sequence `JOHN` fairly early on in the peace, and this is due to the fact that the hierarchical symbol class (1) means that one-half of the productions which begin with the symbol class (1) will begin with the symbol sequence `JOHN`, while the other one-half begin with one of the four possible DownWritten versions of the symbol sequence (2)(3). The frequency with which the symbol sequence `JOHN` appears in the data is enough to warrant the formation of these phrases.

The UpWrite Predictor only finds a single correct symbol class, although the symbol class [`BOY`|`GIRL`|`THEGIRL`|`THEBOY`] seems reasonable, and is a result of the fact that the symbol sequences ⟨`THEGIRL`⟩ and ⟨`THEGIRL`⟩ were formed prior to any classification taking place. The existence of this erroneous symbol class results in the UpWrite Predictor generalising to productions such as `GIRLATEMEAT`.

4	⟨JOHN⟩
7	⟨MEAT⟩
10	⟨RUN⟩
11	⟨GIRL⟩
16	⟨BOY⟩
25	⟨JOHNLIKES⟩
28	⟨JOHNATE⟩
35	⟨JOHNATEFISH⟩
36	⟨JOHNATEMEAT⟩
39	⟨FAST⟩
41	⟨SLOWLY⟩
42	⟨FISH⟩
44	⟨LIKES⟩
45	⟨RUNFAST⟩
46	⟨RUNSLOWLY⟩
47	⟨WE⟩
49	⟨WALK⟩
52	⟨THEY⟩
53	⟨WALKFAST⟩
54	⟨WALKSLOWLY⟩
55	⟨LIKESFISH⟩
56	⟨LIKESMEAT⟩
58	⟨THEGIRL⟩
59	⟨THEBOY⟩
60	[THEGIRL|THEBOY]
67	[WE|THEY]
73	[BOY|GIRL|THEGIRL|THEBOY]

TABLE 6.8: A list of the structure found by the UpWrite Predictor on data generated by the phrase-structure grammar of figure 6.16, shown in the order of acquisition.

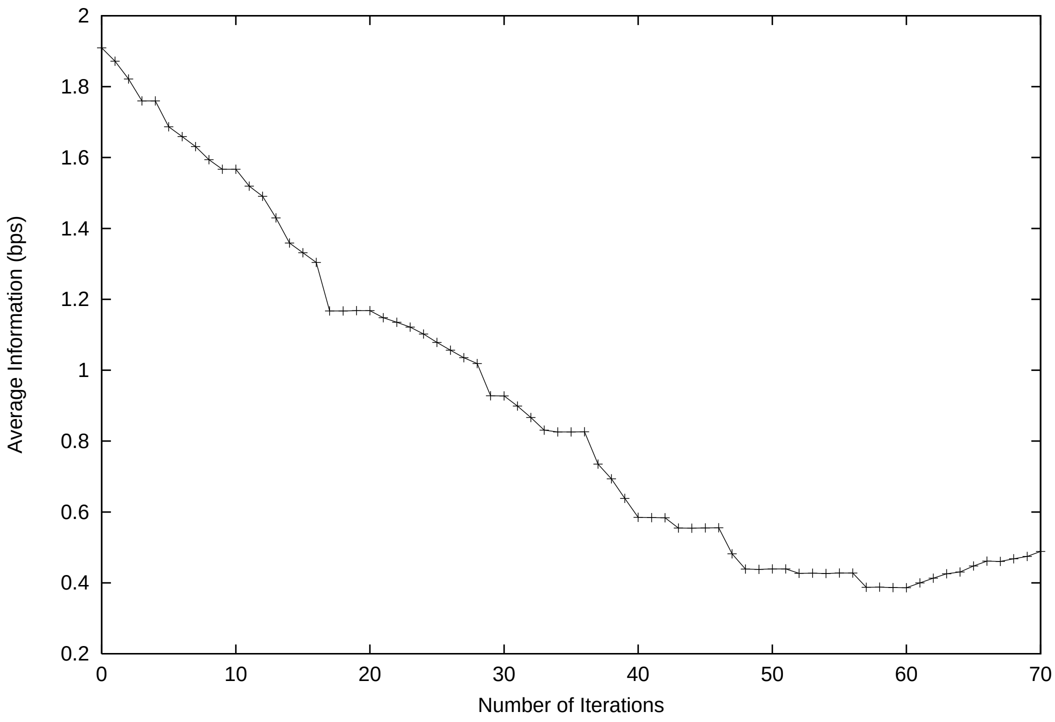

FIGURE 6.17: The performance of the UpWrite Predictor at the end of each iteration as evaluated on data generated by the phrase-structure grammar of figure 6.16.

6.6.4 Text 4

In the phrase-structure grammar shown in figure 6.18, reproduced from [4], the null symbol ϕ reflects the optional nature of the symbol sequence `SOME` in generations which incorporate the symbol class (3). This grammar was used to generate a training corpus 125141 bytes in length and a testing corpus 12452 bytes in length.

#	$\mapsto$	(1)(2)(3)(4)|(5)(6)(7)
1	$\mapsto$	`BOB`|`MARY`
2	$\mapsto$	`LIKES`|`ATE`
3	$\mapsto$	ϕ|`SOME`
4	$\mapsto$	`FISH`|`MEAT`
5	$\mapsto$	`WE`|`THEY`
6	$\mapsto$	`WALK`|`RUN`
7	$\mapsto$	`FAST`|`SLOWLY`

Sample: `WEWALKSLOWLYMARYATEFISHMARYLIKESSOMEFISH` ...

FIGURE 6.18: The phrase-structure grammar used to generate Text 4.

The performance of the UpWrite Predictor inferred from this data is plotted in figure 6.19, and it can be seen that performance steadily improves until the forty-first iteration, at which point the performance of the UpWrite Predictor begins to degrade.

Some of the structure discovered by the UpWrite Predictor in this data is listed in table 6.9, along with the iteration in which it was found. It can be seen that all thirteen

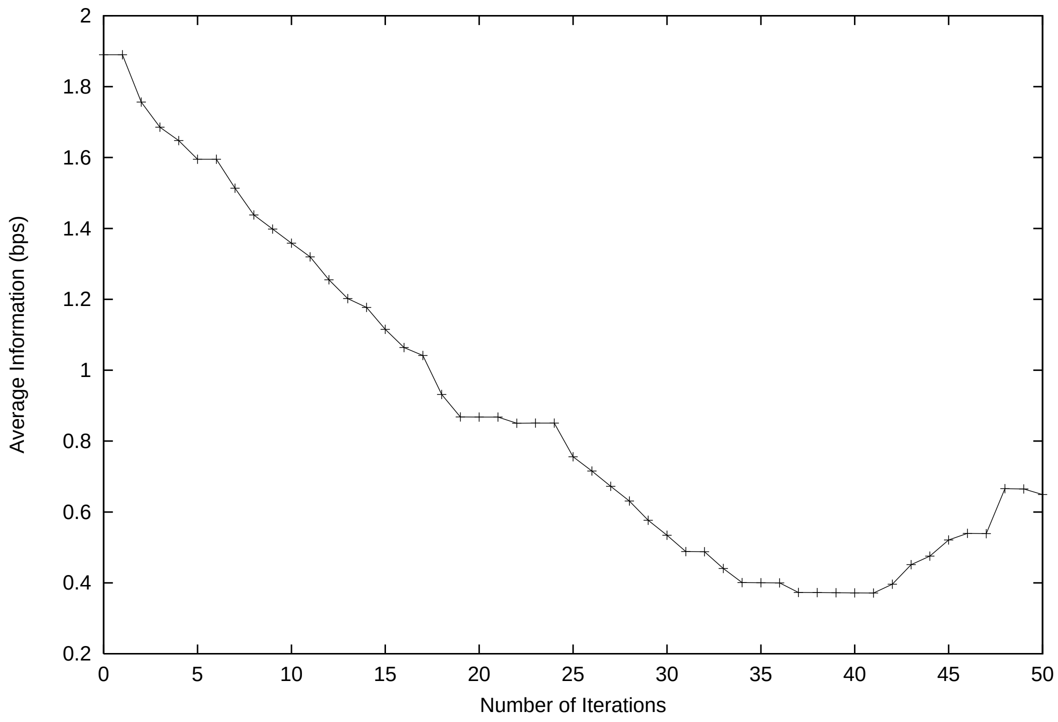

FIGURE 6.19: The performance of the UpWrite Predictor at the end of each iteration as evaluated on data generated by the phrase-structure grammar of figure 6.18.

symbol sequences implicit in the data are found by iteration 38, and the two higher level symbol sequences ⟨THEYWALK⟩ and ⟨THEYRUN⟩ are also formed by this point, only to be grouped together in a symbol class.

The algorithm finds the three correct symbol classes [FISH|MEAT], [MARY|BOB] and [ATE|LIKES], and also forms a combined symbol class consisting of the symbol classes (4) and (7) in the grammar, each of which end productions. A similar phenomenon occurs in the formation of the symbol class [WE|MARY|BOB], which excludes the symbol sequence THEY due to the fact that it was used to form the phrases ⟨THEYWALK⟩ and ⟨THEYRUN⟩ prematurely. The absence of the symbol class [WALK|RUN] is also explained by the premature formation of these phrases. The symbol sequence SOME is never classified, and this is the behaviour which we would expect, given its optional nature.

The combination of the various symbol classes in the grammar results in the inferred UpWrite Predictor generalising to productions such as WEATEFAST and MARYWALKSLOWLY.

6.6.5 Text 5

The phrase-structure grammar shown in figure 6.20 was used by Wolff to create Text 5 and Text 6, with the exception that some of the productions were banned from Text 6, in order to test the ability of the SNPR algorithm to generalise to productions unseen during training [4]. We generated a training corpus 125342 bytes in length and a testing corpus 12458 bytes in length from this grammar.

5	⟨MARY⟩
6	⟨RUN⟩
9	⟨FISH⟩
17	⟨FAST⟩
21	⟨THEY⟩
23	⟨THEYWALK⟩
24	⟨THEYRUN⟩
26	⟨SOME⟩
27	⟨MEAT⟩
28	⟨ATE⟩
32	⟨BOB⟩
35	⟨LIKES⟩
36	⟨SLOWLY⟩
37	⟨WE⟩
38	⟨WALK⟩
39	[THEYWALK|THEYRUN]
40	[FISH|MEAT]
41	[MARY|BOB]
45	[FAST|SLOWLY|FISH|MEAT]
48	[ATE|LIKES]
50	[WE|MARY|BOB]

TABLE 6.9: A list of the structure found by the UpWrite Predictor on data generated by the phrase-structure grammar of figure 6.18, shown in the order of acquisition.

#	$\mapsto$	(1)(2)(3)|(4)(5)(6)
1	$\mapsto$	DAVID|JOHN
2	$\mapsto$	LOVES|HATED
3	$\mapsto$	MARY|SUSAN
4	$\mapsto$	WE|YOU
5	$\mapsto$	WALK|RUN
6	$\mapsto$	FAST|SLOWLY

Sample: JOHNLOVESMARYDAVIDHATEDMARYYOURUNSLOWLY ...

FIGURE 6.20: The phrase-structure grammar used to generate Text 5 and Text 6.

The performance of the UpWrite Predictor on this data is shown in the plot of figure 6.21, and it can be seen that a fairly steady performance improvement occurs up until the thirty-fourth iteration of the algorithm, at which point it levels off.

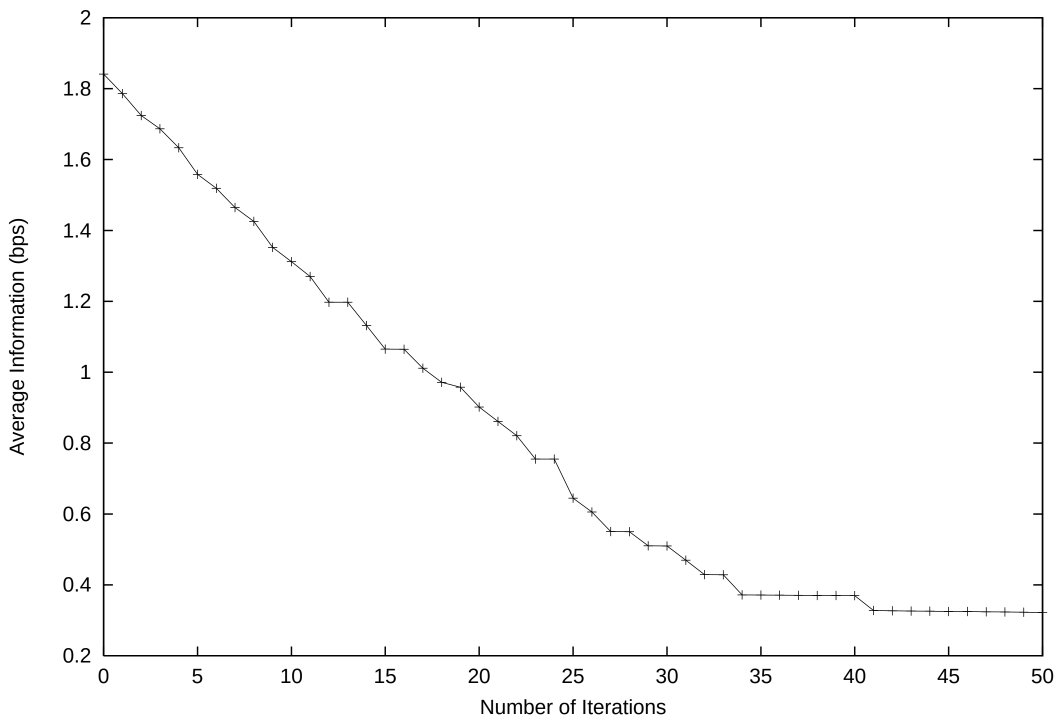

FIGURE 6.21: The performance of the UpWrite Predictor at the end of each iteration as evaluated on data generated by the phrase-structure grammar of figure 6.20.

Some of the structure found by the UpWrite Predictor is listed in table 6.10, together with the iteration in which it was found. It can be seen that the UpWrite Predictor found ten of the twelve symbol sequences implicit in the data by iteration 36, and found four correct symbol classes: [`HATED`|`LOVES`], [`JOHN`|`DAVID`], [`RUN`|`WALK`] and [`YOU`|`WE`]. The remaining two symbol classes weren't discovered due to the fact that phrases were formed from the symbol sequences which are members of these classes in iterations 40, 41, 47 and 49. Overall, the performance of the UpWrite Predictor on this example is very good, with the result that the inferred UpWrite Predictor is capable of generating identical data to the original grammar.

6.6.6 Text 6

Wolff formed Text 6 by removing all occurrences of the symbol sequences `JOHNLOVESMARY` and `WEWALKFAST` from Text 5 [3]. We performed this modification on the training corpus of the previous example, resulting in a new training corpus 111296 bytes in length. The testing corpus remained untouched, enabling us to evaluate the ability of the UpWrite Predictor to generalise to data unseen during training.

4	⟨JOHN⟩
8	⟨HATED⟩
13	⟨DAVID⟩
16	⟨FAST⟩
19	⟨MARY⟩
22	⟨RUN⟩
31	⟨YOU⟩
32	⟨SUSAN⟩
34	⟨LOVES⟩
36	⟨SLOWLY⟩
37	[HATED|LOVES]
38	[JOHN|DAVID]
40	⟨[JOHN|DAVID][HATED|LOVES]SUSAN⟩
41	⟨[JOHN|DAVID][HATED|LOVES]MARY⟩
42	⟨WALK⟩
43	⟨WE⟩
44	[RUN|WALK]
45	[YOU|WE]
47	⟨[YOU|WE][RUN|WALK]FAST⟩
49	⟨[YOU|WE][RUN|WALK]SLOWLY⟩

TABLE 6.10: A list of the structure found by the UpWrite Predictor on data generated by the phrase-structure grammar of figure 6.20, shown in the order of acquisition.

Figure 6.21 plots the performance of the UpWrite Predictor on this data. It can be seen that performance improves less rapidly than in the previous example, and levels off at a higher level than in the previous example. This is almost certainly due to the fact that the UpWrite Predictor is supplied with more information by the testing corpus, which contains symbol sequences unseen during training.

Some of the structure found by the UpWrite Predictor in this experiment is listed in table 6.11. It is interesting to note that the symbol class [DAVID|JOHN] is formed fairly soon after the symbol sequences ⟨DAVID⟩ and ⟨JOHN⟩ have been discovered by the algorithm. The formation of this class enables the UpWrite Predictor to generalise to the symbol sequence JOHNLOVESMARY, which was unseen during training, and the symbol sequence found in iteration 54 embodies this ability to generalise.

Unfortunately the UpWrite Predictor forms no other symbol classes. All possible phrases apart from WEWALKFAST and WEWALKSLOWLY are formed, meaning that the UpWrite Predictor failed to generalise to the second symbol sequence which was unseen during training. In fact, the absence of these two symbol sequences from the training corpus hindered the acquisition of symbol classes significantly.

6.6.7 Text 7

The final text specified by Wolff was generated by the phrase-structure grammar shown in figure 6.23, reproduced from [4], and has the property that the symbol sequence VERY may

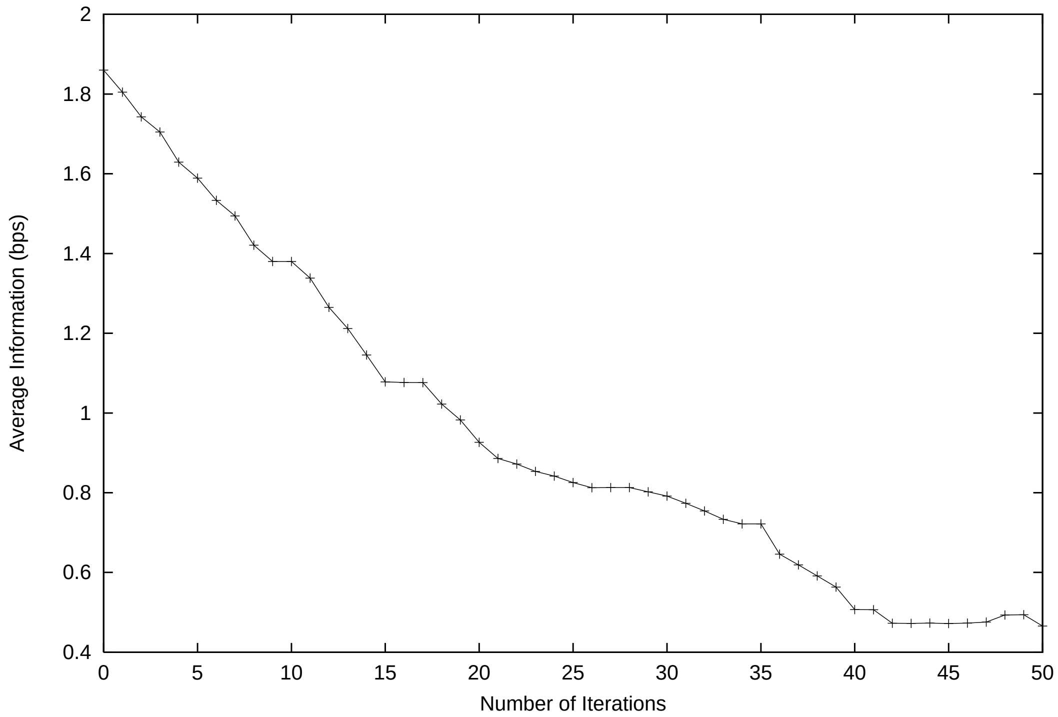

FIGURE 6.22: The performance of the UpWrite Predictor at the end of each iteration as evaluated on modified data generated by the phrase-structure grammar of figure 6.20.

7	⟨HATED⟩
10	⟨DAVID⟩
13	⟨JOHN⟩
16	[DAVID|JOHN]
17	⟨FAST⟩
21	⟨RUN⟩
22	⟨MARY⟩
53	⟨[DAVID|JOHN]LOVESSUSAN⟩
54	⟨[DAVID|JOHN]LOVESMARY⟩
55	⟨SUSAN⟩
56	⟨SLOWLY⟩
57	⟨[DAVID|JOHN]HATEDSUSAN⟩
58	⟨[DAVID|JOHN]HATEDMARY⟩
63	⟨YOUWALKFAST⟩
64	⟨YOUWALKSLOWLY⟩
65	⟨YOURUNSLOWLY⟩
67	⟨YOURUNFAST⟩
68	⟨WERUNFAST⟩
69	⟨WERUNSLOWLY⟩

TABLE 6.11: A list of the structure found by the UpWrite Predictor on modified data generated by the phrase-structure grammar of figure 6.20, shown in the order of acquisition.

be repeated any number of times whenever the symbol class (2) is used in a production. We generated a training corpus 150485 bytes in length and a testing corpus 14809 bytes in length from this grammar.

$$
\begin{array}{lll}
\# & \mapsto & (1)(2)(3)(4)|(5)(6)(7) \\
1 & \mapsto & \texttt{A}|\texttt{THE} \\
2 & \mapsto & \texttt{VERY}|\texttt{VERY}(2) \\
3 & \mapsto & \texttt{FAST}|\texttt{SLOW} \\
4 & \mapsto & \texttt{CAR}|\texttt{SHIP} \\
5 & \mapsto & \texttt{SOME}|\texttt{FEW} \\
6 & \mapsto & \texttt{LARGE}|\texttt{SMALL} \\
7 & \mapsto & \texttt{MEN}|\texttt{BOOKS}
\end{array}
$$

Sample: `SOMESMALLMENAVERYVERYSLOWSHIPTHEVERYFASTCAR` ...

FIGURE 6.23: The phrase-structure grammar used to generate Text 7.

The plot of the performance of the UpWrite Predictor shown in figure 6.24 exhibits a rapid improvement over the first twenty iterations of the algorithm, after which performance begins to slow, beginning to level off after the fiftieth iteration.

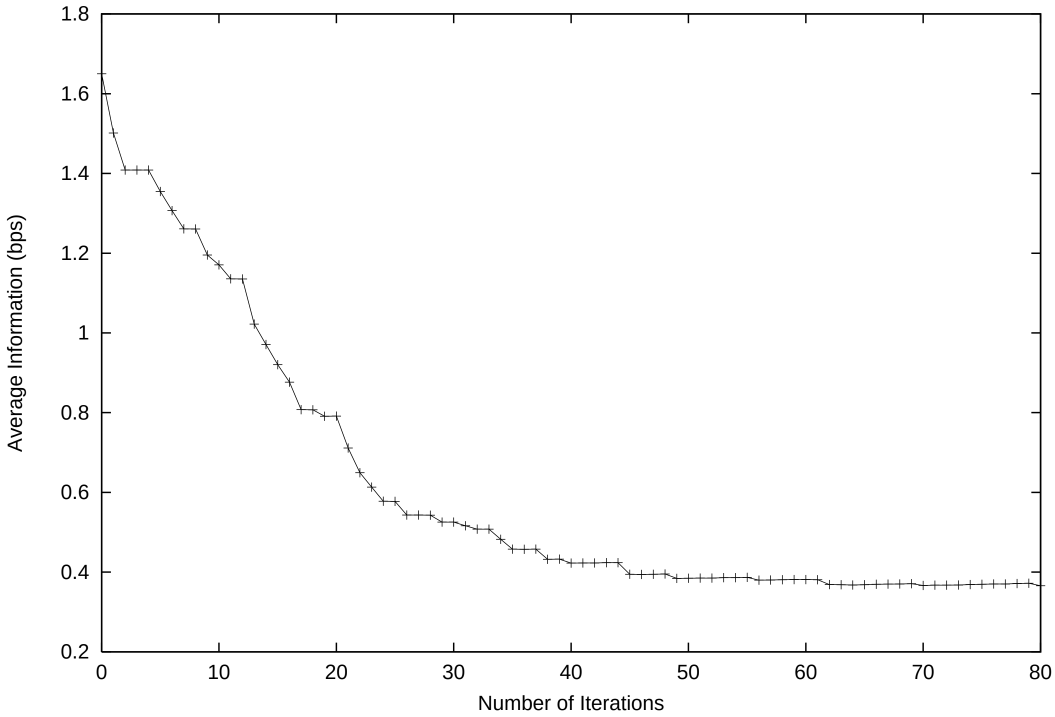

FIGURE 6.24: The performance of the UpWrite Predictor at the end of each iteration as evaluated on data generated by the phrase-structure grammar of figure 6.23.

Table 6.12 lists some of the structure found by the UpWrite Predictor in this data. The symbol sequence `VERY` is found first, and this is due to the fact that it is the most frequently occurring symbol sequence in the data. In fact, the symbol sequence `VERYVERY` is formed before the majority of the remaining symbol sequences in the data are discovered by the algorithm.

3	⟨VERY⟩
8	⟨BOOKS⟩
11	⟨CAR⟩
18	⟨MEN⟩
20	⟨VERYVERY⟩
23	⟨FAST⟩
26	⟨THE⟩
28	⟨FEW⟩
30	⟨FEWSMALL⟩
33	⟨FEWLARGE⟩
36	⟨LARGE⟩
37	⟨VERYFAST⟩
39	⟨VERYSLOW⟩
41	⟨VERYFASTSHIP⟩
42	⟨VERYFASTCAR⟩
43	⟨AVERYVERY⟩
47	⟨THEVERYVERY⟩
48	⟨VERYSLOWCAR⟩
50	⟨VERYSLOWSHIP⟩
51	⟨FEWLARGEMEN⟩
52	⟨FEWLARGEBOOKS⟩
55	⟨FASTCAR⟩
57	⟨FASTSHIP⟩
60	⟨FEWSMALLBOOKS⟩
61	⟨FEWSMALLMEN⟩
65	⟨THEVERYFASTSHIP⟩
66	⟨THEVERYFASTCAR⟩
67	⟨THEVERYSLOWSHIP⟩
68	⟨THEVERYSLOWCAR⟩

TABLE 6.12: A list of the structure found by the UpWrite Predictor on data generated by the phrase-structure grammar of figure 6.23, shown in the order of acquisition.

The UpWrite Predictor finds no symbol classes whatsoever in the data of this example, and forms many phrases instead. This is due to the fact that forming a phrase which contains the symbol sequence `VERY` is advantageous, as it provides extra context to the UpWrite Predictor, enabling it to better predict which symbols are coming next. The fact that the UpWrite Predictor was designed to discover hierarchical structure in data, and not recursive structure, explains its poor performance in this example. However, we should note that recursive structure of a sort may be discovered by the UpWrite Predictor by forming symbol classes out of symbol sequences of different lengths, each of which consists of a repetition of the same lower level symbol.

6.6.8 Discussion

The UpWrite Predictor performs reasonably well on the seven texts used by Wolff to evaluate the performance of his SNPR algorithm. Lower level symbol sequences, corresponding to words, and higher level symbol sequences, corresponding to phrases, are found by the algorithm is all cases, while symbol classes tend to be found more readily when the grammar used to generate the data is not ambiguous, and when sufficient training data is available. This is the performance we expect to see, based upon the nature of the agglomeration algorithm which is used in the UpWrite Predictor to form symbol classes. The performance of the UpWrite Predictor would be improved in these circumstances if contextually dependent symbol classes were found by the algorithm, or if methods based upon the insights into the classification problem given in the previous chapter were incorporated. We would be interested in pursuing such work in the future.

The performance of the UpWrite Predictor was occasionally hindered by the fact that structure formed early on in the process, such as the [L|B] symbol class shown in table 6.7 and the ⟨JOHNLIKES⟩ symbol sequence shown in table 6.8, prevented the acquisition of more useful structure at a later stage. The performance of the UpWrite Predictor would be improved in such situations by introducing a more sophisticated method of using the feedback mechanism to detect and correct erroneous UpWrites.

We have deliberately avoided a direct comparison with the results of Wolff in this section, as in the absence of an objective performance criterion such comparisons would be futile.

6.7 Performance on Natural Language Text

Unfortunately we have not had the opportunity to infer an UpWrite Predictor from a large corpus of natural language text—the inference process is far too expensive in terms of both memory requirements and processing requirements to enable a thorough examination of its performance on natural language. However, we have performed experiments with small natural language corpora, and we present the results of applying the UpWrite technique to natural language text in this section.

A testing corpus was created from the first 99983 bytes of the Sherlock corpus, and a training corpus was created from the 1000034 bytes which followed. We plot the performance of the UpWrite Predictor inferred from the training corpus and evaluated on the testing corpus in figure 6.25, together with the performance of 2^{nd}– and 3^{rd}–order Markov models inferred from and evaluated on the same data.

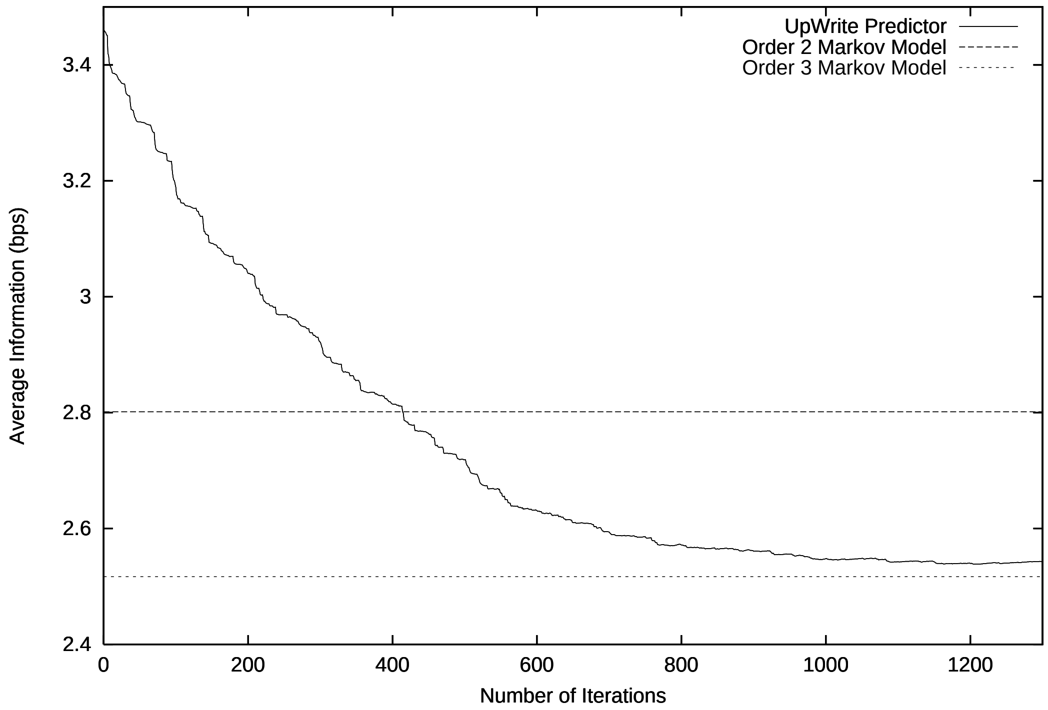

FIGURE 6.25: The performance of the UpWrite Predictor at the end of each iteration as evaluated on natural language text.

After 1205 iterations, the performance of the UpWrite Predictor was 27% better than its performance prior to the first iteration, at which stage it was equivalent to a 1^{st}–order Markov model. At iteration 415 of the algorithm, the performance of the 1^{st}–order UpWrite Predictor exceeded that of the 2^{nd}–order Markov model, and by iteration 1205, the performance of the UpWrite Predictor was a mere 0.8% worse than the performance of the 3^{rd}–order Markov model.

By this stage of the process, the UpWrite Predictor had discovered symbol sequences such as ⟨`Sherlock`∧`Holmes`⟩, ⟨`Watson`⟩, ⟨`,"`∧`said`∧`Holmes.`⟩ and ⟨`and`∧`the`⟩, as well as symbol sequences corresponding to frequently occurring English words. A small number of symbol classes, such as [`This`|`Then`|`But`] were also discovered by the algorithm, but these tended to be in the minority. We have generally found that symbol classes are only formed once no common symbol sequences remain to be found, and the algorithm was not iterated sufficiently to uncover all common symbol sequences in the data.

The fact that a 1^{st}–order Markov model which makes predictions about an UpWritten version of natural language text almost reaches the performance of a standard 3^{rd}–order

Markov model suggests that the UpWrite is beneficial, and that the structure found by the UpWrite Predictor is useful in practice.

6.8 UpWriting and DownWriting

The UpWrite Predictor constructs a hierarchical representation of the data which it is inferred from, and this high level representation may be DownWritten to produce an approximation to the original data. This ability to DownWrite the data may be necessary in some applications, such as that of data compression, and we also find that the DownWritten form of the data provides a useful indication of the performance of the UpWrite Predictor, as it enables us to examine the structure extracted from the original data by eye.

A word-level 1^{st}–order UpWrite Predictor was inferred from the training corpus of the previous example. Symbols were formed from words rather than characters to expedite the formation of symbol classes. The UpWritten form of the testing corpus created by the UpWrite Predictor was DownWritten, after 42 iterations of the algorithm, in five different ways.[45] Five different DownWritten versions of the first two sentences of the testing corpus are shown in figures 6.26 to 6.30. The original versions of these two sentences may be seen in figure 1.1, as they are identical to the first two sentences of the `Test` portion of the Sherlock corpus.

Figure 6.26 shows the DownWritten version of the data which was assigned the highest probability by the UpWrite Predictor. It bears a strong resemblance to the original data, with the exception that the symbols `she`, `is`, `him` and `under` have been replaced with the symbols `he`, `was`, `me` and `on`. This changes the meaning of the original data significantly, and corresponds to the output which we would expect to see if a lossy data compressor was implemented using the UpWrite Predictor.

Figure 6.27 shows the least probable DownWritten version of the original data, according to the UpWrite Predictor. The resulting data is ungrammatical, although the resemblance to the original data is still obvious, as many of the words have remained unchanged. We also note that the data is considerably longer than the most probable DownWrite, and this is due to the fact that symbols such as `the` have been replaced with symbol sequences such as `to from of the`.

A similar DownWritten form of the original data is shown in figure 6.28, in which we illustrate the longest DownWrite. This was produced by selecting, at each stage of the DownWrite, the longest possible lower level symbol, and it results in data which is highly ungrammatical due to the fact that the UpWritten versions of single words have been DownWritten to long symbol sequences.

The opposite effect may be created by selecting, at each stage of the DownWrite, the lower level symbols which are shorter than any other, and the results of doing this are shown in figure 6.29. This DownWritten version of the original data is very similar to

> To Sherlock Holmes he was always the woman. I have seldom heard me mention her on any other name.

FIGURE 6.26: The most probable DownWritten form of the first two sentences of the Sherlock corpus.

> To Sherlock Holmes who it is always to from of the woman but I had seldom heard myself mention her under any other name.

FIGURE 6.27: The least probable DownWritten form of the first two sentences of the Sherlock corpus.

> To Sherlock Holmes she it was always to with of the woman, and I have seldom heard myself mention her through any other name.

FIGURE 6.28: The longest DownWritten form of the first two sentences of the Sherlock corpus.

> To Sherlock Holmes he is always a woman. I had seldom heard me mention her on any other name.

FIGURE 6.29: The shortest DownWritten form of the first two sentences of the Sherlock corpus.

> To Sherlock Holmes she was always to from the woman, and I have seldom heard us mention her by any other name.

FIGURE 6.30: A random DownWritten form of the first two sentences of the Sherlock corpus.

the most probable DownWrite, and is quite grammatical, even though the meaning of the original data has been drastically altered in the process.

One of the many hundreds of possible DownWritten forms of the data are shown in figure 6.30. The process used to form this data may be thought of, in a very imprecise way, as one of abstracting the original data, and then using the UpWrite Predictor to generate data at random after constraining this generation with the abstracted form of the original data.

One property of all of the five DownWritten forms shown in this section is that they all have the same higher level representation with respect to the UpWrite Predictor. This property of the UpWrite Predictor may be of use in may different natural language applications, such as that of providing a natural language front-end to an online search engine, as it enables us to abstract the exact form of the text entered by the user, and, ideally, it would enable us to cluster these high level forms into groups of queries which represent the same *concept*.

6.9 Generations

One way of getting an intuitive feel for the performance of a predictive model is to use it generatively, and *eye-ball* the data which results. In figures 6.31 to 6.35 we give examples of data generated by character-level Markov models of orders 0, 1, 2, 3 and 8. In all cases, the Markov models were inferred from the entire Sherlock corpus. Note that the resemblance to English text increases as the order of the model is increased, with many common English words appearing in the generated data of figure 6.34, and many valid English phrases appearing in the generated data of figure 6.35.

Other authors, including Claude Shannon in his classic paper "The Mathematical Theory of Communication" [2], and Timothy Bell, John Cleary and Ian Witten in their book on text compression [1], have generated data in similar ways in order to illustrate the ability of finite context models to model English text.

In the next chapter we shall be showing how the predictions made by Markov models of orders 0, 1, 2 and 3 may be smoothed together, and used in a data compression system. The fact that such a system works may be attributed to the ability of the 3^{rd}–order Markov model to capture some salient features of English text, and this much is obvious following an examination of the generated data of figure 6.34.

The UpWrite Predictor is based on a 1^{st}–order Markov model, and the data generated by the UpWrite Predictor prior to incorporating any higher level structure is similar to that shown in figure 6.32. As the UpWrite Predictor is iterated, however, the data which it generates will change in form, to reflect the higher level structure which has been found by the algorithm and used to UpWrite the data from which the predictive model is inferred. In figures 6.36 to 6.40 we give examples of data generated by an UpWrite Predictor inferred from the entire Sherlock corpus.

itndnf nItinmkttncgta atb ttu riodqiiensthef hce,t iI ti ol-Wdet.ynelbs.adstet rwnt su kn myyltitrtin t elTocienmahaa"liey olee.h f tpdv eieetqaM e gevgl i.h ee b"eotWod iehrhosgie tfleo r rnstbuunTbt lt"tsalw aoeed asbinhno d p ehm tkesuto shneoereqdhleIht jietromsDfesekhslaeh?y tt" y.tdoon eh di ioa,etf,n sa utiets ee tvdiia etie hrit loes i wmsd oheilau l i n enb. s e mtrdaw nttot bsctne ep dtauar P

FIGURE 6.31: A portion of data generated by a Markov model of order 0.

thitwarder asy hen ld aling mat Jomabal f thitheseng Yer. fek I ikitrevo a Wis thadis t h l me impesougist moums taye hand's micass y isingrnde y. t int allw s inkeve w heathooriten athatir en deanours. mappstre. blllllavinghan yof aid fr ts k cerrendico qun uratisa wa my th d t ng Pr winemeearaneveng Thongexpr."Golulinamanouimak s bow whevare nd tuthen lie lerppachie tared Hea s way pir g nlitheneanorehiase Mr s bup

FIGURE 6.32: A portion of data generated by a Markov model of order 1.

sprow in my upontlight the moose go. "hed pappen therfe, my yeat shishis fromen wity ell wo aftersearledaylithes." atte king the whis get reary as lifin ey st res, a light-pece mooralwas St. Ther. I es, ack he a wourrivis fored your wassures yound the my a the hich hinforrind yone soureas opply rone," henight thise kend then ally he dow ot you not morknowlebblew ennotervers. "He noccia wor wou?" "So could lingreflo

FIGURE 6.33: A portion of data generated by a Markov model of order 2.

"What need-ever to descover here this hall ther Chard to Know," safe," said his of the his? Mould he sunk the is on the way booke narrible whichmenter ching," head a get in that hiddless?" "Take as I had plas no and him of the bar, there corone have you neast to seal said he ming three he savague! Do you have asked my line one heap at end, hough. And so is connel absolumblack expectorman paulty, throught the othese

FIGURE 6.34: A portion of data generated by a Markov model of order 3.

Most of the unused ticket has been married forward of that Colonel Upwood in conclusions to the detectives from the bedroom. For myself, I thought arranged for a movements. I had to take advantage of emaciated figure was a wild suspicious death. She had sat down the papers in my delight. "You may remember, Monday last year?" "Two thousand-pound box of which I wish you simply impossible!" cried the impression

FIGURE 6.35: A portion of data generated by a Markov model of order 8.

> tesanck ay sl had beeere setles. There if I have dl omearame wousthe ut we the fenl omed mitcovering te-sts nd s." Sho vachar man. He had hapeth!" searit, before he make teventittepanoof the pld may der, the ft here." That is tolly chancex." "Only for blopterame, acesthe houseseeching which pof the scl centhing which hapon where in his hale th where had sounotia owashould sshareco forwhend man lake acts.

FIGURE 6.36: Data generated by the UpWrite Predictor after 200 iterations.

> Lorse tane ngid was ces." "Mean her sargesatotid ever m, w gaked ltaled had sped by ale weven a a my fait." "Insisawitapums doing ave it." "Yes; but nenden. Her Ke, you do tsut thesere nof aft olandevile sen capaiangind bet, Midealo dark a interesthes my o art this sllaso the pt scover ste gink by the fas. What o ure the nhaps mar, as you sthing sit il ulteles could ns anthing mosan; form o that he id that toupe in the

FIGURE 6.37: Data generated by the UpWrite Predictor after 400 iterations.

> so ple, and with his ainanger a sonild me have not heard the bore ctthing which was the w paremore that had ts rn?" ""t, and he twraplas, and sor Steverto lathing which one ok tight think that you of them it horing. I pplar of Ked his cion, side, for lean and dow, alowo od sant teperhapply 22d whot ame. STul, behining befadeare seen you," said the dily this mored. It is thy pill. I bring mbefore your clike a dage ft was the sus.

FIGURE 6.38: Data generated by the UpWrite Predictor after 600 iterations.

> He tat dear sed me for a bookince were am I had givant. "It was sonsile"I save am dur whoth, as I inted Roberghrough tris, to say than you." "I ach to pustill. The Hall you, ther in the with an inted they care to prgnatime ffels cy of moment in the man, make yet you an - would kinshe tlene soctioseconves reds my more than Sherlock Holmes casions to wopetrousene of wn hour from Mrs. Hencentre and wondeferend

FIGURE 6.39: Data generated by the UpWrite Predictor after 800 iterations.

> do not in the eletely at the first note in is nothing of For in your hair, Watson! Godget of the tlances Mr. He's ink-cl?" "The ses muchying! This about Grun to she would hardly be hoprightly visized glookso son that up in the doingle in my find-gun to resh he could find that you but unny our very kind. This could reater which led er in his beach was no silebject the cowr." Se, when the young Murn only one by this ushed

FIGURE 6.40: Data generated by the UpWrite Predictor after 1000 iterations.

In figure 6.36 we observe that the generated data contains many frequently occurring English words, with the result that the generated data appears to be more similar to that of a 2^{nd}–order Markov model than that of a 1^{st}–order Markov model. Sequences such as `Sherlock Holmes` appear in the generated data of figure 6.39, and this is indicative of the fact that long symbol sequences have been discovered by the UpWrite Predictor. After 1000 iterations of the algorithm, the generated data, as shown in figure 6.40, appears to be very similar to that generated by a 3^{rd}–order Markov model. This is to be expected, as we have previously shown that the performance of a 1^{st}–order UpWrite Predictor approaches that of a 3^{rd}–order Markov model as the number of iterations of the algorithm is increased.

6.10 Summary and Conclusion

In this chapter we have performed various experiments with the UpWrite Predictor, both on artificial data generated by simple phrase-structure grammars and on relatively small natural language corpora. It has been demonstrated that the UpWrite Predictor is capable of acquiring symbol sequences and symbol classes which correspond very closely to those inherent in the data generated by simple phrase-structure grammars, particularly in situations where ambiguous symbol classes are not present in the grammar. Performance on natural language text is also improved by the incorporation of higher level, and examination of the data generated by DownWriting and generating reveals that the structure discovered by the UpWrite Predictor in natural language text seem to be quite reasonable.

Notes

[42] The symbol sequence `XX` is only ever followed by the symbol `X` in the UpWritten data.

[43] This is due to the fact that whatever follows the symbol `X` in the data will also follow the symbol sequence `XXX` in the UpWritten data.

[44] These symbol sequences are ambiguous because the symbol sequences `IT` and `HE` appear in generations which contain the ambiguous symbol sequence `THEM`.

[45] Recall that a stochastic symbol class is DownWritten by selecting one of its members at random, according to the probability distribution over the class.

References

[1] Timothy C. Bell, John G. Cleary, and Ian H. Witten. *Text Compression*. Prentice Hall, 1990.

[2] Claude E. Shannon and Warren Weaver. *The Mathematical theory of Communication.* University of Illinois Press, 1949.

[3] J. G. Wolff. An algorithm for the segmentation of an artificial language analogue. *British Journal of Psychology*, 66(1):79–90, 1975.

[4] J. Gerard Wolff. Language acquisition, data compression and generalization. *Language & Communication*, 2(1):57–89, 1982.

Chapter 7

Data Compression

> "Take the message also, Watson, that we may check each other. A single flash—that is A, surely. Now, then. How many did you make it? Twenty. So did I. That should mean T. AT—that's intelligible enough! Another T. Surely this is the beginning of a second word."
>
> *His Last Bow*
> SIR ARTHUR CONAN DOYLE

7.1 Introduction

A *data compressor*[46] is an algorithm for encoding data such that the encoded representation of the data occupies less space, and may therefore be transmitted more efficiently.[47] Of course, it is not possible to design a data compressor which is capable of compressing arbitrary data; all data compressors are equally bad when results are averaged over all possible data. Rather, it is desirable to develop data compressors which perform well on a particular type of data, such as natural language text or image files.

The "modern paradigm" of data compression involves separating the compression process into two components: modelling and coding. Adaptive predictive models are used to make a prediction about the next symbol in the data being compressed, and arithmetic coding is then applied to encode the symbol which actually occurs next with respect to this prediction. Designing good data compressors therefore becomes a problem of designing good predictive models. The fact that a data compressor must necessarily make few assumptions about the data, and must perform well across all the data in some domain, means that data compression is in fact an ideal test-bed for predictive modelling.

7.1.1 Overview

In this chapter we give a brief history of the field of data compression, during which we describe the process of arithmetic coding. The data compression method of Prediction by Partial Matching, or PPM, is then presented, and we proceed to introduce various novel modifications and additions to the *ad hoc* techniques typically used by PPM data compressors to improve their compression performance. Finally, we describe how the UpWrite Predictor may be applied to the data compression problem.

7.2 A Compressed History

It seems that no matter how inexpensive storage space becomes, people will always be interested in using data compressors to make better use of that space. The growing popularity of the Internet has meant that data compression has become more important than ever—modern modems which operate over a telephone connection have low bandwidth, and downloading any reasonably sized file over such a channel takes a considerable amount of time. Data compression is therefore an important area of research, and is as old as data storage and transmission themselves.

7.2.1 Origins of Compression

The Zipfian relationship between word length and word frequency in natural languages is evidently a result of a compression process during language evolution [33]. We have a tendency to favour acronyms and clichés for similar reasons; doing so expedites the transmission of ideas from our brain to someone else's.

Text compression has been used whenever storage space is expensive, as was the case in ancient Greece, where precious papyrus was made best use of by writing texts devoid of punctuation and whitespace, with the result that the time taken to read the texts was increased [26].

Louis Braille developed a system in the 1820's which allowed blind people to read by passing their fingers across a sheet of paper embossed with a sequence of cells, each of which contained a pattern constructed from a small grid of six dots. Because each of these cells required the same space as ten printed characters, and because a grid of six dots gives 64 possible cell patterns, the patterns which remained after the letters of the alphabet and the digits had been encoded were used to represent common words and letter groups [5].

In 1835 Samuel Morse developed the Morse code, which makes use of dots and dashes to represent letters, numerals and punctuation. He designed the code such that frequent letters, such as 'e', were given less code space than infrequent letters, such as 'x'. In Morse code, an 'e' is encoded with a single dot, while an 'x' is encoded as a dash, followed by two dots and another dash, meaning that it takes eight times longer to transmit an 'x' as it does an 'e', as the length of a dash is equivalent to the length of three dots [5].

Modern compression systems emerged with the development of the digital computer. Perhaps the most well-known of these is the algorithm devised by Ziv and Lempel, variants of which are used in computer programs such as `zip` and `compress`, and as part of the *GIF* file format [34, 35].

7.2.2 Statistical Compression Versus Dictionary Compression

Approaches to text compression may be roughly divided into two classes: dictionary techniques and statistical techniques.

Dictionary techniques work by substituting a small code for a longer sequence which occurs frequently in the data being compressed. The Braille system is an example of a dictionary technique; each letter is allotted the same code length, but frequently occurring combinations of letters are allotted smaller codes. Dictionary compressors tend to be quite fast.

Statistical techniques work slightly differently; each symbol is allotted a code length such that the code length of any symbol multiplied by its frequency is roughly constant. Morse code uses statistical compression by assigning a shorter code to the more frequent letters.

Bell, Cleary and Witten have shown that dictionary methods cannot be more powerful than statistical methods. They demonstrated this by specifying an algorithm which is able to convert an arbitrary dictionary encoder to a statistical encoder which achieves exactly the same compression [5].

7.2.3 Static, Semi-Adaptive and Adaptive Compression

Compression techniques are only of value if the decompressor is aware of the coding system used. This necessitates transmission of the coding system, and this overhead may outweigh the advantages of compressing the data to begin with. There are two ways of surmounting this problem: either develop a coding system which is independent of the data being transmitted, or make the coding system implicit in the encoded data itself. We can therefore divide compression techniques into three classes: static, semi-adaptive and adaptive.

Static techniques work by establishing a coding system and sticking to it, no matter how inappropriate it is to the data being encoded. Both the Braille system and Morse code are examples of static techniques.

Semi-Adaptive techniques adapt the coding system to the data being compressed, and transmit the coding system prior to transmitting the data. In many cases the cost of transmitting the coding system means that the semi-adaptive technique is not worthwhile.

Adaptive techniques work by encoding the data on the fly, even as it is being transmitted. In these techniques, the transmitter and the receiver must agree on a system for determining what the codes are from the data transmitted thus far.

It has been shown that adaptive modelling sacrifices little in the way of compression by restricting itself to only the symbols transmitted thus far, rather than the entire data [5].

7.2.4 Lossy or Lossless Compression

Data compression may be *lossless*, meaning that the original data is restored perfectly by the decompressor, or *lossy*, in which case a reasonable approximation to the data is produced. What constitutes a reasonable approximation is obviously determined by the consumer of the data; it is difficult to imagine anyone being satisfied with a reasonable approximation to a novel which they wish to read if this meant that the protagonist's name was varied at random,[48] while a reasonable approximation to an image which contains a lot of skin-tones and little else may be perfectly acceptable to some, especially if it halves the download time.

7.2.5 Huffman Coding

Statistical data compressors work by estimating the probability of each symbol in the data being compressed, and then encoding these symbols as efficiently as possible. In 1952, shortly after Shannon's work on Information Theory, David Huffman devised a method for constructing efficient codes [17], and the *Huffman coding* algorithm proceeds as follows.

- Create a set of all possible messages,[49] and associate a probability with each of the messages in the set;
- Remove the two least probable messages from this set, choosing arbitrarily if there are more than two such messages, and add the result of concatenating this pair to the set as a new message, with a probability equal to the sum of the probabilities of the two original messages;
- Continue this process until a single message remains in the set, and represent the sequence of operations used to form this set as a binary tree;
- Assign codewords to each node of the binary tree such that the shortest codewords are assigned to the most probable messages, and that the codewords satisfy the property that no codeword is a prefix of any other. This later requirement ensures that any given string of codewords is uniquely decodable.

The Huffman coding algorithm creates a prefix code for each message in the original set by traversing the binary tree from the root to the node which corresponds to the message being coded, emitting a `0` if a left branch is taken and a `1` if a right branch is taken.[50]

Example 7.1. *Consider the set of messages* $\mathcal{L} = \{A, B, C, D, E\}$ *with associated probabilities* 0.25, 0.25, 0.2, 0.15 *and* 0.15. *The binary tree formed by the Huffman coding algorithm is illustrated in figure 7.1, and the resulting codewords are shown in table 7.1. The average code length is* 2.30 *bits per message, slightly more than the theoretical optimal length of* 2.29 *bits per message.*

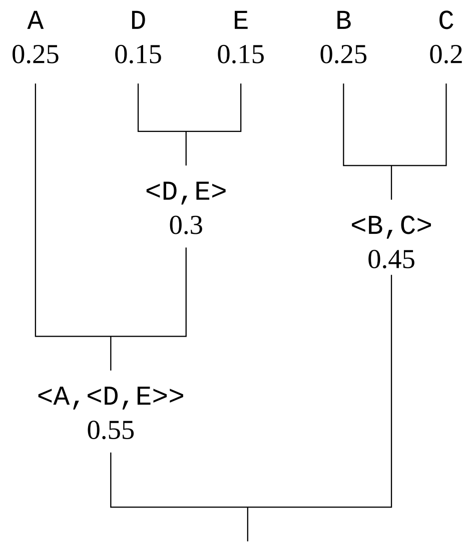

FIGURE 7.1: The binary tree formed from the messages of example 7.1 by the Huffman coding algorithm.

Message	Codeword
A	00
B	10
C	11
D	010
E	011

TABLE 7.1: The codewords assigned to the messages of example 7.1 by the Huffman coding algorithm.

Encoding a message using Huffman coding would appear to require an enumeration of all possible messages in order to determine the code for the particular message we are interested in—a seemingly impossible task. Traditionally, therefore, Huffman coding has been used to encode each symbol in the message individually, and the fact that each symbol x_i cannot be encoded in exactly $-\log_2 P(x_i)$ bits, unless $P(x_i)$ is a power of $\frac{1}{2}$, means that the entire message is encoded sub-optimally.

This problem can be alleviated somewhat by expressing the message as a symbolic time series of blocks b_i, each of which is a substring of n symbols, such that $b_i = \langle x_{ni-n+1}, \dots, x_{ni} \rangle$ and $P(b_i) = \prod_{j=ni-n+1}^{ni} P(x_j)$, and encoding each block using the Huffman coding algorithm. This approach, known as *block coding*, is optimal, and equivalent to performing Huffman coding on the entire message by enumerating all messages, as $n \to \infty$.

7.2.6 Arithmetic Coding

Arithmetic coding emerged in the late 1970's as a result of work by Mauro Guazzo, Frank Rubin and Rissanen and Langdon [15, 21, 22], the origins of which, according to Bell, Cleary and Witten [5], lie in work by Peter Elias in the 1960's [5]. Arithmetic coding solves the encoding problem; it is a technique which enables a message to be encoded very nearly optimally, in a way which is theoretically similar to enumerative Huffman coding, its main power being a method of sequentially constructing the codeword without having to enumerate all possible messages first. Many authors have published efficient computational implementations of the algorithm; Alistair Moffat, Radford Neal and Ian Witten's implementation being one of the most recent [20].

In theory, arithmetic coding operates by partitioning the unit interval $[0, 1)$, assigning every possible message m_i to a sub-interval such that the size of the sub-interval corresponding to message m_i is equal to $P(m_i)$. The message m_i may then be encoded by expressing the corresponding sub-interval in an unambiguous way, allowing the decoder to locate it after partitioning the unit interval in an identical process. Compression is achieved when the sub-interval is specified with the minimum necessary precision to disambiguate it from other sub-intervals.

Example 7.2. *Consider the diagram of figure 7.2 which illustrates the partitioning of the unit interval in accordance with the probabilities of the set of messages of the previous example. Message* **C** *corresponds to the sub-interval* $[0.5, 0.7)$, *and may be encoded by identifying this sub-interval by emitting any number which falls within it;* 0.6 *for example.*

FIGURE 7.2: The unit interval is partitioned into one sub-interval for each message, with the size of a sub-interval determined by the probability of the message.

When encoding symbolic time series the unit interval is partitioned iteratively according to the following steps.

1. Make the unit interval the currently selected interval.

2. Partition the currently selected interval into sub-intervals, each of which corresponds to some symbol from $\mathcal{A}$, such that the size of each sub-interval is proportional to the probability of the symbol to which it corresponds.

3. Make the sub-interval which corresponds to the current symbol x_i the currently selected sub-interval.

4. Repeat from step 2 until all the symbols x_i in the data have been used.

5. Emit a codeword which uniquely identifies the currently selected sub-interval.

Example 7.3. *Figure 7.3 shows the progressive division of the unit interval via the arithmetic coding algorithm according to the message* `CAB`*, with the symbol probabilities given in example 7.1. The message itself may be encoded as any number in the range* $[0.5125, 0.525)$*, and* 0.52 *would be a sensible choice, owing to the fact that arithmetic coding achieves compression by specifying the sub-interval which corresponds to the message with the minimal acceptable precision. In this case, the message may be encoded using two decimal digits (the zero and the decimal point are redundant as each possible message will be encoded with a number* $\in [0, 1)$*), resulting in compression.*

We refrain from discussing the computational implementation of arithmetic coding, suffice it to say that techniques do exist for progressively transmitting the sub-interval which corresponds to the message being encoded using fixed-precision arithmetic. Also, it is important for us to point out that we do not actually use arithmetic coding in the experiments which follow, as it is apparent that the average information supplied to the predictive model by the data being compressed, as defined in equation 2.6, sufficiently approximates the compression performance which would be achieved by a system which incorporated an arithmetic coder.

7.2.7 Ziv-Lempel Compression

Jacob Ziv and Abraham Lempel introduced what is now known as the *Ziv-Lempel family of data compressors* in two landmark papers published in 1977 and 1978 [34,35]. Ziv-Lempel

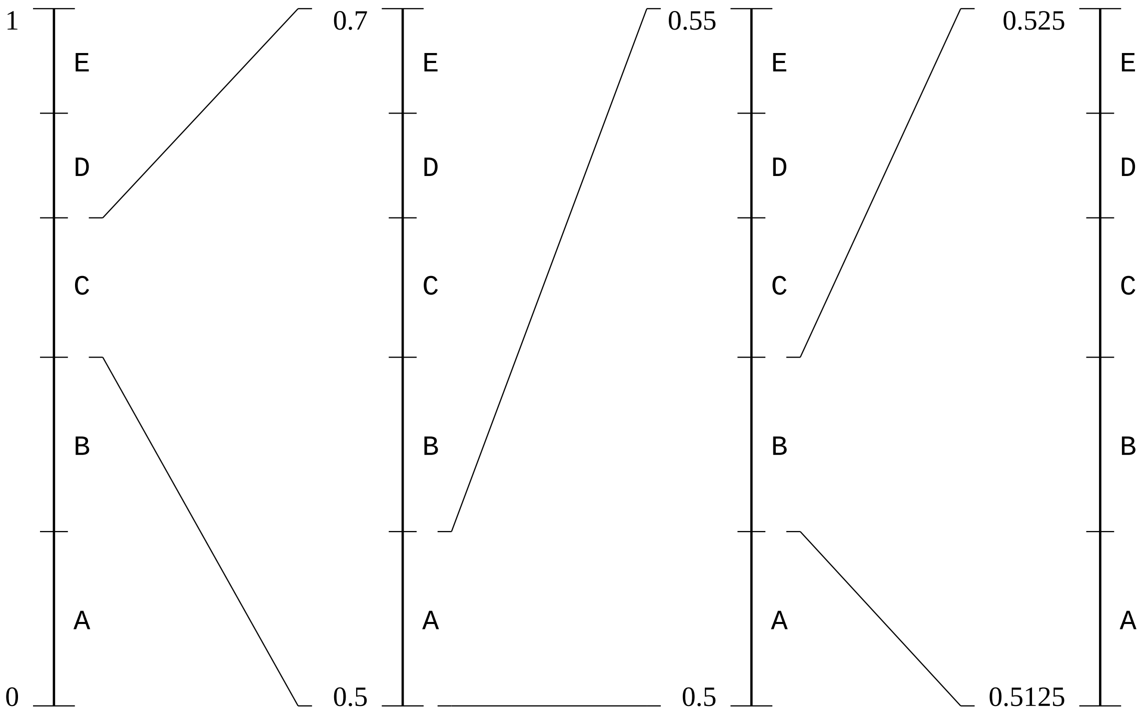

FIGURE 7.3: The unit interval is progressively partitioned according to the symbol sequence C,A,B.

compression works by replacing repeated substrings in the data being compressed with a pointer into an adaptive dictionary. Compression is achieved when the encoded versions of these pointers occupy less space than the strings which they reference. Decompression is trivial; each pointer is simply replaced with the string which it references.

Many variations of the original Ziv-Lempel algorithm exist, which is why we refer to a family of data compressors, and these variants differ mainly in the way the adaptive dictionary is specified, and the way the pointer into the dictionary is encoded. Well-known members of the Ziv-Lempel family of compressors are LZ77, LZ78 and LZW; these and many more are discussed in "Text Compression", by Bell, Cleary and Witten [5]. We shall present the LZ77 algorithm in this section.

LZ77 works by maintaining a sliding window over the data being encoded. This window, which contains N symbols, is divided into a look-ahead buffer of F symbols which are yet to be encoded, and a history buffer of $N - F$ symbols which have already been encoded. A commonly used value of N is 8192 symbols. The adaptive dictionary consists of all possible substrings in the sliding window which are no more than F symbols in length, and which begin in the history buffer. An interesting property of the LZ77 compressor is that these substrings may overlap the look-ahead buffer. At the beginning of the compression process, the history buffer is initialised with some predetermined symbol sequence.

Example 7.4. *Consider figure 7.4, which shows a sliding window of length $N = 22$, with a look-ahead buffer of length $N = 7$.*

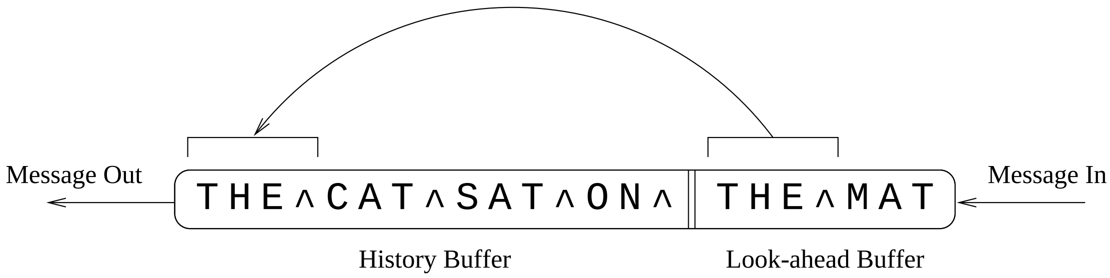

FIGURE 7.4: A Ziv-Lempel data compressor replaces a repeated substring with a pointer into a recent history buffer.

Encoding proceeds by finding the longest string in the adaptive dictionary which matches the prefix string of the look-ahead buffer—this is equivalent to finding the longest matching string in the sliding window which begins in the history buffer—and encoding the prefix string as a triple $\langle i, j, a \rangle$, where i is an offset from the beginning of the look-ahead buffer back into the history buffer to the beginning of the matched string, j is the length of the matched string, and a is the first character in the history buffer which was not matched. The inclusion of a non-matching character in this triple ensures that encoding is possible in situations where no matching string was found. After the prefix string has been encoded, the sliding window is advanced by the number of characters encoded, and the process is repeated.

Example 7.5. *In figure 7.4, the longest string in the history buffer which matches the prefix string of the look-ahead buffer is THE∧, and the prefix string of the look-ahead buffer is therefore encoded as the triple $\langle 15, 4, M \rangle$.*

Ziv-Lempel compressors work because [5]

- common words and morphemes occur with regularity in natural language text;
- specialist words, such as proper names in newspaper articles, tend to occur in sudden bursts;
- less common words are often made up of fragments of more common words; and
- run-length encoding is handled implicitly due to the fact that the matching substring may overlap the look-ahead buffer.

Ziv-Lempel compressors exhibit moderate compression performance, and their main advantage is that they are very quick. Many popular data compression programs, such as *zip* and *compress*, use variants of the Ziv-Lempel technique.

7.2.8 Burrows-Wheeler Compression

The block-sorting data compression algorithm was discovered by David Wheeler in 1983, and was introduced by Burrows and Wheeler in 1994 [9]. It achieves compression performance comparable with the best statistical data compressors at speeds comparable with the Ziv-Lempel family of data compressors.

Burrows-Wheeler compression works by applying a reversible transformation to the data being compressed such that the transformed version of the data exhibits the property that a particular symbol is likely to reappear if it has been observed recently, and is unlikely to appear otherwise. This property makes the transformed data perfectly suited to *move-to-front encoding*, and compression is achieved when the output of the move-to-front encoder is encoded via Huffman coding or arithmetic coding.

The block-sorting transformation works by forming a $z \times z$ matrix of all possible cyclic shifts of the data s_z, sorted lexigraphically. The transformed version of s_z is equal to the sequence of suffix symbols in column z of this matrix.

Example 7.6. *Consider figure 7.5, which shows the matrix formed by the block-sorting algorithm for the data* $s_z =$ ***THE∧CAT∧SAT∧ON∧THE∧MAT****. Row* 20 *of the matrix, indicated with arrows in the figure, contains the original data* s_z*. The transformed version of* s_z *corresponds to the sequence of symbols* ***EETTNSCM∧HHTT∧O∧∧AAT∧A*** *in column* z *of this matrix.*

```
   ∧CAT∧SAT∧ON∧THE∧MATTHE
   ∧MATTHE∧CAT∧SAT∧ON∧THE
   ∧ON∧THE∧MATTHE∧CAT∧SAT
   ∧SAT∧ON∧THE∧MATTHE∧CAT
   ∧THE∧MATTHE∧CAT∧SAT∧ON
   AT∧ON∧THE∧MATTHE∧CAT∧S
   AT∧SAT∧ON∧THE∧MATTHE∧C
   ATTHE∧CAT∧SAT∧ON∧THE∧M
   CAT∧SAT∧ON∧THE∧MATTHE∧
   E∧CAT∧SAT∧ON∧THE∧MATTH
   E∧MATTHE∧CAT∧SAT∧ON∧TH
   HE∧CAT∧SAT∧ON∧THE∧MATT
   HE∧MATTHE∧CAT∧SAT∧ON∧T
   MATTHE∧CAT∧SAT∧ON∧THE∧
   N∧THE∧MATTHE∧CAT∧SAT∧O
   ON∧THE∧MATTHE∧CAT∧SAT∧
   SAT∧ON∧THE∧MATTHE∧CAT∧
   T∧ON∧THE∧MATTHE∧CAT∧SA
   T∧SAT∧ON∧THE∧MATTHE∧CA
→  THE∧CAT∧SAT∧ON∧THE∧MAT  ←
   THE∧MATTHE∧CAT∧SAT∧ON∧
   TTHE∧CAT∧SAT∧ON∧THE∧MA
```

FIGURE 7.5: The block-sorting transformation matrix.

A fascinating property of the block-sorting transformation is that the reverse transformation can be performed as long as the row of the matrix in which s_z appears is known. The reverse transformation proceeds as follows. The symbols $x_1, \ldots, x_z$, which appear in the transformed version of s_z received and decoded by the decompressor, are sorted alphabetically. Since the transformed version of s_z corresponds to column z of the matrix, and the sorted version of the transformed version of s_z corresponds to the first column of the matrix, we have recovered the first and last columns of the matrix. The remaining columns are recovered as follows. Consider a row in the matrix $x_1, \ldots, x_z$, where x_1 and x_z are known, but the symbols $x_2, \ldots, x_{z-1}$ are unknown, and where this row is the i^{th} row of the matrix which begins with the symbol x_1. The identity of x_2 may be recovered by finding the i^{th} row of the matrix $y_1, \ldots, y_z$ which ends with the symbol $y_z = x_1$, and then setting $x_2 = y_1$. This process may be generalised in order to reconstruct the entire matrix, after which original data s_z may be recovered, as it is known in which row of the matrix it appears.

Example 7.7. *Consider the transformed version of s_z of the previous example. The sorted version of this is* $\wedge\wedge\wedge\wedge\wedge$`AAACEEHHMNOSTTTTT`*, and is apparent that this is equal to the first column of the matrix of figure 7.5. Row 20 of the matrix, marked with arrows in the figure, begins with the* `T` *symbol, and is the third row of the matrix to do so. Row 12 of the matrix is the third row which ends with a* `T`*, and it begins with an* `H`*. Therefore the* `H` *symbol follows the* `T` *symbol of row 20.*

We have shown that the block-sorting transformation is reversible. We shall now consider how the transformed string may be encoded efficiently. The transformed string has the property that recently occurring symbols tend to recur, and this property is ideally suited to move-to-front encoding.

Move-to-front encoding maintains an adaptive list of symbols, which begins in some predetermined state. A symbol is encoded efficiently by its position in the list, allocating less code-space to symbols which are near the front of the list, and Huffman coding may be used to achieve this. After a symbol has been encoded, it is moved to the front of the list, so that it will be encoded efficiently should it recur. Peter Fenwick has pointed out that the output of the move-to-front encoder is equivalent to the coding of sentences given in Shannon's 1951 paper [13, 23]. That is, move-to-front encoding is equivalent to the Shannon Game, in that the order of symbols in the adaptive list maintained by the encoder may be considered to be an ordered list of predictions of what the next symbol in the data is going to be.

Block-sorting data compression performs almost as well as less computationally efficient statistical data compressors, and it has been shown that block-sorting is equivalent to a PPM model with unbounded context lengths [10]. However, in the case of block-sorting data compression, the contexts *follow* the symbol being compressed. The algorithm effectively predicts which symbol is coming *before* the current context. The fact that performance is very nearly equivalent to that of PPM compressors suggests that the Markovian assumption made by the Markov models used in PPM compressors is as arbitrary as was suggested in chapter 3.

7.2.9 Statistical Compression

The so-called "modern paradigm" of data compression is based around a predictive model, and avoids the overhead associated with transmitting this model explicitly by inferring it adaptively from the data being compressed, ensuring that the decompressor is able to construct an identical model as it decodes the data it receives [5].

Figure 7.6 is a block diagram of such a system. This diagram is conceptually similar to that given by Shannon in his 1951 paper [23].

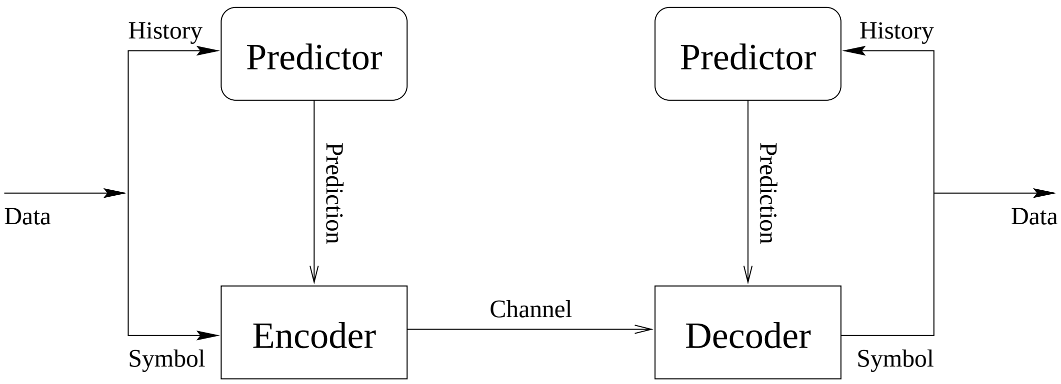

FIGURE 7.6: A block diagram of a modern adaptive statistical data compressor.

The data to be compressed is fed into the system symbol-by-symbol, the predictor making a prediction about the next symbol in the data in the form of a probability distribution over the alphabet of symbols. This prediction is fed into an encoder, which efficiently encodes the symbol which actually did occur next in the sequence with respect to this probability distribution. Once a symbol has been encoded, it is fed into the predictive model, which is then able to update its statistics.

Decompression is the inverse process. An identical predictive model makes an identical prediction about the next symbol in the sequence, and this prediction is used by the decoder to determine what the next symbol actually is, from the encoded representation of the symbol it received. Once the symbol has been decoded, it is fed into the predictive model, which updates its statistics accordingly.

This procedure guarantees that the predictive models at the receiver and the transmitter evolve in lock-step, ensuring that they are identical at all stages of the process. As long as the models begin in a well-defined state, and update their statistics from the symbols encoded (or decoded) thus far, this follows naturally.

Adaptive statistical data compressors achieve the best known compression performance on natural language text, often representing the text in two bits-per-character, corresponding to a compressed representation which is 25% of the original file size.

7.2.10 Learning as Compression

We would now like to acknowledge the work of Gerry Wolff, and point out the similarities between Wolff's work and our own [28–32]. We discussed Wolff's segmentation algorithm in section 5.3.2, and we used six artificial grammars of Wolff's in the experiments of chapter 6.

Wolff proposes that the notion of "cognitive economy" may be applied to the language acquisition process by considering certain aspects of language acquisition to be manifestations of data compression principles which have the net effect of striking a balance between the cost of a cognitive operation (in terms of computational complexity or storage requirements) and the effectiveness of that operation. Wolff writes that [28]

> ... the systems being investigated, and indeed brain structures and functions generally, are the products of evolutionary processes of natural selection and they are, in consequence, likely to be governed in some sense by principles of efficiency.

One way of achieving this balance is to reduce or eliminate any redundancies in the data, and this is precisely the goal of data compression. Wolff cites as evidence in support of his thesis the fact that biological nervous systems tend to respond to changes in the environment, and he considers this behaviour to be a form of difference coding.

Wolff states that segmental, or syntagmatic, and disjunctive, or paradigmatic, groupings of linguistic elements describe two of the most fundamental types of structure which are prominent, and which may in fact be universal, in natural languages. These two types of structure are precisely equivalent to the sub-objects and quotient-objects which the UpWrite Predictor is able to discover.

Wolff developed a computer program called SNPR which constructs a non-stochastic phrase-structure grammar from data using processes similar to those presented in section 5.3.2. The program searches for and adds to an evolving grammar both syntagmatic and paradigmatic groupings of symbols in a process which aims to maximise the "compression value" of the grammar. The measure of "compression value" is simply a ratio between the percentage reduction of the data when it is expressed in terms of the grammar, and the size of the grammar as determined by the number of bits required to specify it [28]. Heuristic techniques are used to simplify the inferred grammar and ensure that it is able to generalise to strings which were not observed during inference, and a necessary feature of SNPR is its ability to correct any over-generalisations made when forming paradigmatic groupings.

We refrain from giving a detailed description of Wolff's SNPR algorithm. We merely wish to draw parallels between SNPR and the UpWrite Predictor. Both algorithms construct language models by progressively augmenting simple models with structure discovered from the data, and both algorithms are concerned with data compression. The differences between the two approaches are as follows.

- SNPR constructs a non-stochastic phrase-structure grammar while the UpWrite Predictor constructs a predictive model. The latter is arguably of more use in language related applications such as data compression and speech recognition.

- SNPR makes many passes over a corpus of data in the process of inferring a grammar. The UpWrite Predictor is not restricted to a multi-pass approach—it may be used adaptively, in which case it makes a single pass over the data. This ability, of course, depends on the exact implementation.

- The techniques used by SNPR to discover structure in data are heuristic, while the UpWrite Predictor discovers structure by analysing the sequence of predictions made by a simple predictive model—that is, structure is found *with respect to the model.*

- The UpWrite Predictor is based on a syntactic pattern recognition framework which was inspired in part by cognitive processes in the human brain and visual system, and which has a proven track record in the field of image recognition.

- SNPR needs to parse the data in terms of the grammar it has constructed. The UpWrite Predictor avoids the parsing problem altogether, as the process of discovering structure and UpWriting this structure to produce a representation of the data at a higher level is equivalent to parsing, and unambiguous when both steps are performed concurrently. This is apparent from figure 5.1, which clearly shows that the data is UpWritten *as structure is being discovered.*

- Heuristic techniques are used by SNPR in order to ensure that the grammar which it constructs is capable of generalising to unseen data. The UpWrite Predictor is able to generalise immediately due to the requirement that the predictions it makes must be smoothed.

- SNPR uses techniques to correct over-generalisations made during the inference process, while the UpWrite Predictor tends to avoid such over-generalisations due to the fact that symbol classes are added to the alphabet in a 'skeptical' manner.

- The "compression value" measure is used to evaluate the performance of a candidate grammar inferred by SNPR, and this is roughly equivalent to treating SNPR as a semi-adaptive data compressor. The UpWrite Predictor, when used in an adaptive data compression system, may be evaluated in a similar way, but in this case the storage requirements of the model and the representation of the data with respect to the model are inseparable, and comparisons with other data compression algorithms are more readily made.

Wolff's SNPR algorithm and our UpWrite Predictor were developed for different reasons, and approach the data compression problem from opposite directions. Wolff considers data compression to be central to the process of learning, while we discovered that data compression provides a good way of evaluating predictive models designed with other goals in mind. It is therefore interesting that the two methods exhibit certain similarities. It is our belief that Wolff's algorithm may offer greater insights into the language acquisition process in human beings (an area of study in which we are still very much wet behind the ears), while our UpWrite Predictor has more application in language-related applications, such as speech recognition, due to the fact that it is based around a standard predictive model. We also believe that the process used in the UpWrite Predictor to discover structure is more general, and that the fact that it is based on the notion of using measures from Information Theory to analyse the predictions made by a simple predictive model is an important innovation.

7.3 Prediction by Partial Matching

The advent of arithmetic coding inspired John Cleary and Ian Witten to develop the Prediction by Partial Matching data compressor, which is commonly referred to as PPM [11]. PPM incorporates a predictive model which makes a prediction about the next symbol x_i in the data being compressed by combining the predictions made by Markov models of various orders, from 0 up to some maximum m, along with the prediction made by a special predictive model of "order -1"[51], which makes the uniform prediction, and which is present to ensure that each symbol is assigned a non-zero probability even when no observations have been made. A PPM predictor of order 3, therefore, makes a prediction which is based on the predictions made by 5 separate predictive models.

In figures 6.31 to 6.34 we gave examples of data generated by character-level Markov models of orders 0, 1, 2 and 3. These examples give us an intuitive feel for the amount of structure the predictive models have extracted from the data, and it is evident that a 3^{rd}–order Markov model inferred from natural language text is capable of generating random data which bears a passing resemblance to the text the model was inferred from. The fact that the PPM compressor performs so well may be attributed to the ability of the 3^{rd}–order Markov model to embody the statistics of natural language text sufficiently well.[52]

The predictions made by the various predictive models in the PPM data compressor are usually combined using one of two possible processes—*blending* or *escape*—which are roughly equivalent to the methods of smoothing and fallback which were discussed in chapter 3.

7.3.1 The Escape Mechanism

Blending the predictions made by various predictive models together is computationally expensive, and PPM typically abandons blending in favour of an escape mechanism. A special *escape symbol* is introduced to the alphabet, and we choose to denote this escape symbol by φ, and each of the Markov models in the PPM compressor are required to guarantee that this symbol is assigned a non-zero probability so that, no matter what data is encountered, it will always be possible to encode the escape symbol.

A particular symbol x_i is then encoded as a sequence of escape symbols followed by the symbol itself, with the number of escape symbols in the sequence indicating the degree of fallback required to reach a predictive model which assigns a non-zero probability to x_i. Markov models of various orders, from 0 to some maximum m, together with the model of "order -1", make predictions about the symbol x_i based upon the context in which it occurs. The prediction made by the highest-order predictive model $\mathcal{M}_m$, the predictive model of order m, is used to encode x_i if $P(x_i|\mathcal{M}_m, s_{i-1}) > 0$, and is used to encode φ otherwise. If x_i was encoded, the process is complete, and moves on to encode the next symbol x_{i+1} in the data. If, however, φ was encoded, the prediction made by $\mathcal{M}_{m-1}$ is consulted, and the encoding process iterates. In the worst case, $\mathcal{M}_{-1}$ will be used to encode x_i.

Because PPM is an adaptive technique, $\mathcal{M}_{-1}$ will be used frequently at the beginning of the compression process, when many symbols are being observed for the first time. As the PPM compressor begins to adapt to the data, the predictions made by higher order Markov models will come in to play, and compression performance will improve.

Various techniques exist for estimating $P(\varphi|\mathcal{M}_n, s_{i-1})$. The problem of estimating this *escape probability* is usually framed as a problem of estimating the probability of observing a novel symbol in a particular context. We shall discuss three techniques for estimating this probability; these are referred to in the literature as Method A, Method B and Method C, and the PPM data compressors which use these methods are referred to as PPMA, PPMB and PPMC.

Method A

Method A was introduced by Cleary and Witten in their 1984 paper, and works by allocating a count of one to the escape symbol [11]. The probability of the escape symbol is then calculated by dividing its count by the total count of all symbols seen in the context, as in equation 7.1, where $\mathcal{M}_n$ denotes the predictive model of order n, s_{i-1} denotes the history $x_1, \dots, x_{i-1}$, and $C(s)$ is a counting function which returns the number of occurrences of the string s in the observed data.

$$P(\varphi|\mathcal{M}_n, s_{i-1}) \approx \frac{1}{C(x_{i-n}, \dots, x_{i-1}) + 1} \tag{7.1}$$

$$P(x_i|\mathcal{M}_n, s_{i-1}) \approx \frac{C(x_{i-n}, \dots, x_i)}{C(x_{i-n}, \dots, x_{i-1}) + 1} \tag{7.2}$$

Because the escape symbol consumes part of the probability distribution, the probability estimate of the observed symbol $x_i \in \mathcal{A}$ is reduced somewhat, and this is shown in equation 7.2.

Method B

Method B was also introduced by Cleary and Witten [11]. It makes the assumption that the first occurrence of a particular symbol in a particular context may be taken as evidence of a novel symbol appearing in that context, and therefore does not contribute towards the estimate of the probability of the symbol which occurred. Method B estimates the probability of the escape symbol as in equation 7.3, where q denotes the number of different symbols seen in the context so far.

$$P(\varphi|\mathcal{M}_n, s_{i-1}) \approx \frac{q}{C(x_{i-n}, \ldots, x_{i-1})} \tag{7.3}$$

$$P(x_i|\mathcal{M}_n, s_{i-1}) \approx \frac{C(x_{i-n}, \ldots, x_i) - 1}{C(x_{i-n}, \ldots, x_{i-1})} \tag{7.4}$$

As was the case with Method A, the fact that the special escape symbol is assigned a non-zero probability means that the probability estimates of the symbols in the alphabet need to be scaled down accordingly. Method B achieves this by subtracting 1 from each symbol count, as in equation 7.4, meaning that a symbol needs to be observed twice in order to be assigned a non-zero probability by the predictive model.

Method C

Alistair Moffat introduced Method C in a paper which presented an efficient computational implementation of the PPM data compressor [19]. Method C is similar to Method B, with the distinction that the first observation of a particular symbol in a particular context also contributes towards the probability estimate of the symbol itself. That is, the first occurrence of a symbol is considered to give two observations; one of the escape symbol and one of the symbol itself. Method C estimates the escape symbol probability as in equation 7.5.

$$P(\varphi|\mathcal{M}_n, s_{i-1}) \approx \frac{q}{C(x_{i-n}, \ldots, x_{i-1}) + q} \tag{7.5}$$

$$P(x_i|\mathcal{M}_n, s_{i-1}) \approx \frac{C(x_{i-n}, \ldots, x_i)}{C(x_{i-n}, \ldots, x_{i-1}) + q} \tag{7.6}$$

As with the other methods, Method C scales down the probability estimate assigned to the symbols, and this is reflected by equation 7.6.

Other Methods

Out of the three methods of estimating the escape probability we have presented, Method C consistently gives the best results. Its performance remains close to the state of the art, and it is typically used as a benchmark when evaluating new compression algorithms. Other methods have been presented in the literature, and we acknowledge the work of Jan Aberg, Yuri Shtarkov and B.J.M. Smeets [3], Suzanne Bunton [6–8], Ross Neal Williams [26], Paul Glor Howard [16] and William. J. Teahan [24].

7.3.2 Exclusion

The escape mechanism encodes a particular symbol x_i as a sequence of escape symbols followed by the symbol itself, each of which has been assigned a non-zero probability by some predictive model. We may therefore calculate $P(x_i)$ by taking the product of these individual symbol probabilities, as in equation 7.7, where k denotes the highest order predictive model which assigns a non-zero probability to x_i, m denotes the highest order predictive model in the family of predictive models which is able to make a prediction for the specified context, and $\mathcal{M}$ represents the combined PPM predictive model.

$$P(x_i|\mathcal{M}, s_{i-1}) \approx P(x_i|\mathcal{M}_k, s_{i-1}) \prod_{j=k+1}^{m} P(\varphi|\mathcal{M}_j, s_{i-1}) \tag{7.7}$$

We would expect $\sum_{x_i \in \mathcal{A}} P(x_i|\mathcal{M}, s_{i-1}) = 1$, but this turns out not to be the case. This is due to the fact that the highest order predictive model which assigns a non-zero probability to x_i is used to encode it, and the probabilities assigned to this symbol by lower order predictive models are effectively discarded, with the result that some proportion of the probability mass is wasted.

A process known as *exclusion* was introduced by Moffat, and was found by him to improve the performance of the PPMC data compressor, but Moffat did not make explicit the fact that exclusion is *required* to ensure that the predictions made by the PPM predictive model are valid probability distributions when the escape technique is used [19]. The process of exclusion functions by temporarily setting $C(x_{i-j}, \ldots, x_i)$ to zero, where $j \neq k$, during the calculation of $P(x_i|\mathcal{M}, s_{i-1})$.

7.3.3 Blending

The escape mechanism is used in PPM predictive models mainly due to the fact that it is computationally efficient. A process of *blending* could equally be used to combine the predictions made by the predictive models of various orders into a single prediction, as in equation 7.8, where the blending weights $\lambda_j(s_{i-1}) > 0$ are functions of the largest context, and $\sum_{j=-1}^{m} \lambda_j(s_{i-1}) = 1$.

$$P(x_i|\mathcal{M}, s_{i-1}) \approx \sum_{j=-1}^{m} \lambda_j(s_{i-1}) P(x_i|\mathcal{M}_j, s_{i-1}) \tag{7.8}$$

If the blending mechanism is to be used in preference to the escape mechanism, the question of estimating the values of the blending weights arises. It is evident that the blending weights may be expressed as functions of the escape probabilities, as in equation 7.9.

$$\lambda_k(s_{i-1}) \approx [1 - P(\varphi|\mathcal{M}_k, s_{i-1})] \prod_{j=k+1}^{m} P(\varphi|\mathcal{M}_j, s_{i-1}) \tag{7.9}$$

A PPM data compressor which uses blending, and which estimates the values of the blending weights in this way, will exhibit an identical compression performance to that of a PPM data compressor which uses the escape mechanism together with the process of exclusion. However, if the PPM data compressor using the blending mechanism does not make use of exclusion, its performance will differ, and in practice is often superior.

7.3.4 Update Exclusion

The technique of *update exclusion* was introduced by Moffat [19]. This technique only updates $C(x_{i-j}, \ldots, x_i)$ where $j \geq k$—that is, it only updates the count assigned to the symbol x_i in the highest-order predictive model which assigned it a non-zero probability, and in all of the predictive models at higher levels than that.

Update exclusion seems to be motivated by the fact that if exclusion is to be used, the counts assigned to x_i by predictive models of order $j < k$ will be set to zero anyway. Update exclusion allows for a more computationally efficient implementation of the PPM data compressor, and is also found to improve its compression performance.

7.3.5 Recency Scaling

Moffat also introduced a method known as *recency scaling* [19]. This is a process which serves to put an upper bound on the value which the frequency count $C(x_{i-n}, \ldots, x_{i-1})$ can take, and which has the side-effect of causing the PPM predictive model to adapt more rapidly to changes in the data being compressed.

Update exclusion works by monitoring the value of $C(x_{i-n}, \ldots, x_{i-1})$. Whenever it exceeds some predetermined scaling threshold, the counts of all symbols which can occur in the context $\langle x_{i-n}, \ldots, x_{i-1} \rangle$ are halved such that they are guaranteed to remain non-zero. That is, $C(x_{i-n}, \ldots, x_i)' = \lceil \frac{1}{2} C(x_{i-n}, \ldots, x_i) + \frac{1}{2} \rceil \; \forall \, x_i \in \mathcal{A}$. The count $C(x_{i-n}, \ldots, x_{i-1})$ is then normalised by setting $C(x_{i-n}, \ldots, x_{i-1}) = \sum_{x_i \in \mathcal{A}} C(x_{i-n}, \ldots, x_i)'$.

Computational implementations of the PPM data compressor store these counts using integer variables, and therefore they are usually incremented by an amount greater than 1 in order to maintain precision when the recency scaling process halves their values, with the result that infrequent events can be allotted a smaller portion of probability mass than would otherwise be the case. Moffat found that incrementing counts by 8, and setting the scaling threshold to 512, produced good results.

7.4 Corpora for Evaluation of Compression Performance

In order to evaluate the performance of a data compressor, and to enable comparison between it and a wide range of other compression algorithms, it is necessary to establish a standard suite of data files and make them widely available. Bell, Cleary and Witten devised the Calgary corpus for this purpose. The Calgary corpus consists of a rather esoteric cross-section of English text in different writing styles, computer program source code, executable files, geophysical data and a binary image bitmap [5]. A brief overview of the Calgary corpus is given in table 7.2.

File	Size	Description
bib	111261 bytes	A bibliography file in the Unix 'refer' format
book1	768771 bytes	Thomas Hardy's "Far from the Madding Crowd"
book2	610856 bytes	Ian Witten's "Principles of Computer Speech"
geo	102400 bytes	Geophysical data of seismic activity
news	377109 bytes	A variety of postings various newsgroups on USENET
obj1	21504 bytes	A VAX executable of program 'progp'
obj2	246814 bytes	A Macintosh executable of a "knowledge support system"
paper1	53161 bytes	A technical paper on arithmetic coding
paper2	82199 bytes	A technical paper on computer security
paper3	46526 bytes	A technical paper on computational autonomy
paper4	13286 bytes	A technical paper on programming by example
paper5	11954 bytes	A technical paper on arithmetic in logic programming
paper6	38105 bytes	A technical paper on memory efficient hash tables
pic	513216 bytes	A facsimile bitmap, CCITT test picture 5
progc	39611 bytes	C source code to the Unix `compress` program
progl	71646 bytes	LISP source code
progp	49379 bytes	Pascal source code of a PPM evaluation program
trans	93695 bytes	Transcript of an EMACS session

TABLE 7.2: An overview of the files in the Calgary corpus.

Ross Arnold and Timothy Bell later developed the Canterbury corpus in an attempt to address their concern that many researchers were fine-tuning their algorithms to the one corpus, and to introduce a few new file types appropriate to the current popularity of the Internet [4].[53] The Canterbury corpus consists of English prose, English poetry, English plays, facsimile images, source code, spreadsheet files, executable files, technical documents and HTML code. An overview of the Canterbury corpus is shown in table 7.3.

Both the Calgary and Canterbury corpora are available on the World Wide Web, and may be downloaded by following the appropriate links from the Web site of this dissertation [1].

File	Size	Description
`alice29.txt`	152089 bytes	Lewis Carroll's "Alice in Wonderland"
`asyoulik.txt`	125179 bytes	William Shakespeare's "As You Like It"
`cp.html`	24603 bytes	An HTML file of a web page on compression
`fields.c`	11150 bytes	C source code
`grammar.lsp`	3721 bytes	LISP source code for a simple parser
`kennedy.xls`	1029744 bytes	Excel spreadsheet files
`lcet10.txt`	426754 bytes	Proceedings of a workshop on electronic texts
`plrabn12.txt`	481861 bytes	John Milton's "Paradise Lost"
`ptt5`	513216 bytes	A facsimile bitmap, CCITT test picture 5
`sum`	38240 bytes	A Sparc executable
`xargs.1`	4227 bytes	The GNU manual page for the xargs command

TABLE 7.3: An overview of the files in the Canterbury corpus.

In this chapter we shall be giving results of experiments performed using various data compression techniques over both the Calgary and Canterbury corpora. Results are expressed in *average bits-per-symbol*, which we shall abbreviate to *bps*, and which is calculated either by dividing the length of the compressed version of the data by the length of the uncompressed version and multiplying by eight to express this ratio in bits-per-symbol, or by calculating the average information supplied to the predictive model by the data, as in equation 2.6. In both cases it is implicit that a symbol and a byte are the same thing, and we shall assume that an alphabet containing the 256 bytes is appropriate for all files in both corpora unless otherwise stated.

7.5 Analysis of the Performance of Various PPM Models

7.5.1 The 'Optimal' Model

We would like to determine the maximum possible performance attainable by the PPM data compressor, irrespective of the various techniques applied. In order to achieve this, we introduce the notion of an 'optimal' PPM data compressor.

An 'optimal' PPM data compressor is one which calculates $P(x_i|\mathcal{M}, s_{i-1})$ as in equation 7.10. This is equivalent to an escape mechanism which assigns $P(\varphi|\mathcal{M}_j, s_{i-1}) = 1 \; \forall \; j > k$, or a blending mechanism which assigns $\lambda_k(s_{i-1}) = 1$ and $\lambda_j(s_{i-1}) = 0 \; \forall \; j \neq k$, where $\mathcal{M}_k$ is the model which assigns the highest probability to x_i.

$$P(x_i|\mathcal{M}, s_{i-1}) = arg \max_{\mathcal{M}_j} P(x_i|\mathcal{M}_j, s_{i-1}) \tag{7.10}$$

Clearly the 'optimal' PPM compressor cannot exist in practice, as it implies the existence of an oracle which is capable of determining the predictive model which assigns the highest probability to x_i, the symbol which occurs next in the data being compressed, and, if such an oracle did exist, compression would be rendered moot. However, it is possible to determine what the performance of the 'optimal' PPM compressor would be if such an oracle existed, because the encoder can be made to look-ahead and see which symbol is going to occur next, at the expense of making decompression impossible.

In table 7.4 we show the compression performance of 'optimal' PPM predictive models of various orders. Results are given across across all of the files in the Calgary and Canterbury corpora. From these results it is apparent that the file `pic` from the Calgary corpus and the file `ptt5` from the Canterbury corpus are identical.

File	Order 0	Order 1	Order 2	Order 3	Order 4	Order 5
`bib`	5.16bps	3.27bps	2.25bps	1.58bps	1.34bps	1.23bps
`book1`	4.46bps	3.40bps	2.61bps	2.06bps	1.74bps	1.58bps
`book2`	4.69bps	3.54bps	2.55bps	1.84bps	1.49bps	1.34bps
`geo`	5.31bps	4.24bps	3.92bps	3.76bps	3.73bps	3.73bps
`news`	5.09bps	3.87bps	2.79bps	2.02bps	1.68bps	1.56bps
`obj1`	5.51bps	3.74bps	3.04bps	2.84bps	2.79bps	2.78bps
`obj2`	5.96bps	3.71bps	2.57bps	2.12bps	1.82bps	1.71bps
`paper1`	4.88bps	3.51bps	2.39bps	1.78bps	1.55bps	1.48bps
`paper2`	4.52bps	3.37bps	2.43bps	1.84bps	1.57bps	1.47bps
`paper3`	4.60bps	3.42bps	2.55bps	1.94bps	1.70bps	1.62bps
`paper4`	4.62bps	3.44bps	2.51bps	2.02bps	1.89bps	1.85bps
`paper5`	4.86bps	3.54bps	2.53bps	2.09bps	1.97bps	1.93bps
`paper6`	4.91bps	3.51bps	2.39bps	1.80bps	1.60bps	1.53bps
`pic`	1.11bps	0.74bps	0.66bps	0.62bps	0.60bps	0.58bps
`progc`	5.12bps	3.51bps	2.30bps	1.77bps	1.58bps	1.52bps
`progl`	4.67bps	3.07bps	1.99bps	1.39bps	1.17bps	1.07bps
`progp`	4.81bps	3.05bps	1.83bps	1.31bps	1.17bps	1.10bps
`trans`	5.48bps	3.26bps	1.97bps	1.29bps	1.04bps	0.95bps
Average	4.76bps	3.34bps	2.41bps	1.89bps	1.69bps	1.61bps
`alice29.txt`	4.51bps	3.26bps	2.37bps	1.80bps	1.55bps	1.43bps
`asyoulik.txt`	4.76bps	3.28bps	2.41bps	1.94bps	1.70bps	1.59bps
`cp.html`	5.16bps	3.37bps	2.08bps	1.66bps	1.56bps	1.53bps
`fields.c`	4.94bps	3.00bps	1.82bps	1.46bps	1.34bps	1.29bps
`grammar.lsp`	4.60bps	2.89bps	1.93bps	1.65bps	1.56bps	1.55bps
`kennedy.xls`	3.19bps	2.33bps	1.53bps	1.42bps	1.41bps	1.34bps
`lcet10.txt`	4.59bps	3.32bps	2.44bps	1.77bps	1.45bps	1.31bps
`plrabn12.txt`	4.48bps	3.21bps	2.53bps	2.02bps	1.74bps	1.59bps
`ptt5`	1.11bps	0.74bps	0.66bps	0.62bps	0.60bps	0.58bps
`sum`	5.07bps	3.27bps	2.36bps	2.01bps	1.89bps	1.82bps
`xargs.1`	4.89bps	3.37bps	2.36bps	2.02bps	1.95bps	1.93bps
Average	4.30bps	2.91bps	2.04bps	1.67bps	1.52bps	1.45bps

TABLE 7.4: Results of 'optimal' PPM compression using various maximum orders of Markov model, over both the Calgary and Canterbury corpora

A plot of the average compression performance versus the order of the 'optimal' PPM compressor is shown in figure 7.7.

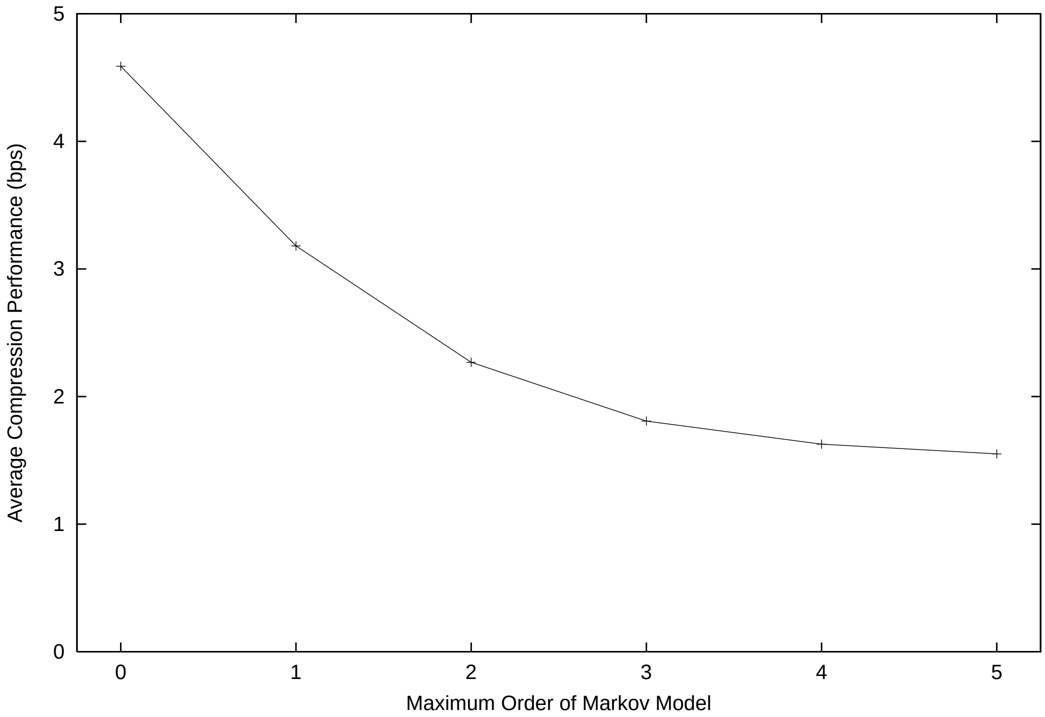

FIGURE 7.7: A plot of the average compression performance over both the Calgary and Canterbury corpora for 'optimal' PPM compressors of differing orders.

It is evident from this plot that the compression performance monotonically increases with the order of the PPM compressor, but that the law of diminishing returns prevails. This, combined with the fact that higher order models have increased memory requirements, and that the process of estimating escape probabilities or blending weights becomes increasingly difficult as more predictive models are added to the mix, justifies the traditional choice of setting the order of the PPM compressor to 3. All of our subsequent experiments shall assume this, unless otherwise stated.

In table 7.5 we show the compression performance of 'optimal' PPM compressors which use various combinations of the *ad hoc* techniques of exclusion, update exclusion and recency scaling, across all of the files in the Calgary and Canterbury corpora. We have named the various PPMO compressors using E to indicate exclusion, U to indicate update exclusion and R to indicate recency scaling. Recency scaling was performed using a threshold of 512 and by updating counts with a value of 8 rather than 1.

It is interesting to note that the best performance is achieved when update exclusion and recency scaling are used together, and that exclusion does not improve results whatsoever. It is also interesting to note that difference between the performance of the best predictive model and the worst is a mere 5%, while the performance difference between

Model	Calgary	Canterbury	Average
PPMO	1.89bps	1.67bps	1.81bps
PPMO-U	1.86bps	1.63bps	1.77bps
PPMO-R	1.85bps	1.62bps	1.76bps
PPMO-E	1.93bps	1.70bps	1.84bps
PPMO-UR	1.84bps	1.60bps	1.75bps
PPMO-UE	1.87bps	1.65bps	1.79bps
PPMO-RE	1.91bps	1.66bps	1.82bps
PPMO-URE	1.97bps	1.63bps	1.84bps

TABLE 7.5: Results of 'optimal' compression using various combinations of *ad hoc* techniques for improving the performance of PPM.

the worst performing 'optimal' PPM compressor and the standard PPMC compressor, the results of which are given in the next section, is 23%. This suggests that great gains are still to be had by improving the methods used to estimate escape probabilities and blending weights.

7.5.2 The Standard Methods

We shall now present the performance of our implementations of the three standard PPM compressors: PPMA, PPMB and PPMC.[54] The results of all subsequent experiments in this chapter will be contrasted with the performance of the PPMC compressor.

In table 7.6 we show the compression performance of the PPMA, PPMB and PPMC compressors, using both the escape and blending mechanisms. We use B to denote the compressors which used blending rather than escape. In all cases, exclusion, update exclusion and recency scaling were also used. It is clear that PPMC-B performs the best overall, but it is interesting to note that PPMB out-performs all other PPM compressors on the files `geo`, `pic`, `kennedy.xls` and `ptt5`.[55] This suggests that the PPMB compressor is able to generalise across a wider range of data, at the expense of decreased performance on natural language text. Method B estimates the probability of a novel symbol in a process whereby a symbol needs to be seen twice to seen believed, and it seems that this produces better results on files which may be subject to random noise, as the four files mentioned may very well be.

7.5.3 Other Data Compression Algorithms

In order to enable comparison between PPM data compressors and other data compressors, we shall now present results of compressing the files in the Calgary and Canterbury corpora using four popular data compression programs.

`compress` is based on a variation of the LZ78 algorithm, and is still used today in many Unix systems. Version 4.2.4-9 was installed under Linux from the Debian package, and was executed with the `-b16` argument.

File	PPMA	PPMB	PPMC	PPMA-B	PPMB-B	PPMC-B
`bib`	2.17bps	2.20bps	2.11bps	2.26bps	2.44bps	2.07bps
`book1`	2.51bps	2.51bps	2.50bps	2.56bps	2.64bps	2.48bps
`book2`	2.28bps	2.29bps	2.24bps	2.33bps	2.43bps	2.22bps
`geo`	6.25bps	4.70bps	4.82bps	7.07bps	4.99bps	5.04bps
`news`	2.79bps	2.76bps	2.66bps	2.89bps	3.02bps	2.63bps
`obj1`	4.47bps	3.80bps	3.66bps	4.70bps	4.15bps	3.73bps
`obj2`	2.85bps	2.69bps	2.56bps	2.95bps	2.95bps	2.53bps
`paper1`	2.57bps	2.64bps	2.48bps	2.69bps	2.98bps	2.42bps
`paper2`	2.50bps	2.53bps	2.45bps	2.60bps	2.81bps	2.40bps
`paper3`	2.79bps	2.82bps	2.70bps	2.94bps	3.18bps	2.65bps
`paper4`	3.11bps	3.14bps	2.92bps	3.30bps	3.58bps	2.89bps
`paper5`	3.23bps	3.26bps	3.00bps	3.41bps	3.71bps	2.97bps
`paper6`	2.64bps	2.70bps	2.52bps	2.76bps	3.07bps	2.46bps
`pic`	1.03bps	0.91bps	0.97bps	1.07bps	0.98bps	0.97bps
`progc`	2.66bps	2.68bps	2.48bps	2.78bps	3.06bps	2.44bps
`progl`	1.92bps	1.99bps	1.87bps	1.97bps	2.24bps	1.82bps
`progp`	1.90bps	1.98bps	1.82bps	1.95bps	2.25bps	1.76bps
`trans`	1.80bps	1.93bps	1.74bps	1.82bps	2.18bps	1.68bps
Average	2.75bps	2.64bps	2.53bps	2.89bps	2.93bps	2.51bps
`alice29.txt`	2.32bps	2.35bps	2.30bps	2.40bps	2.57bps	2.25bps
`asyoulik.txt`	2.55bps	2.57bps	2.51bps	2.65bps	2.82bps	2.46bps
`cp.html`	2.57bps	2.54bps	2.35bps	2.71bps	2.86bps	2.34bps
`fields.c`	2.26bps	2.41bps	2.13bps	2.31bps	2.77bps	2.05bps
`grammar.lsp`	2.62bps	2.77bps	2.40bps	2.72bps	3.19bps	2.34bps
`kennedy.xls`	1.59bps	1.10bps	1.10bps	2.30bps	1.14bps	1.35bps
`lcet10.txt`	2.19bps	2.22bps	2.18bps	2.24bps	2.36bps	2.15bps
`plrabn12.txt`	2.45bps	2.47bps	2.45bps	2.50bps	2.60bps	2.42bps
`ptt5`	1.03bps	0.91bps	0.97bps	1.07bps	0.98bps	0.97bps
`sum`	3.09bps	2.90bps	2.69bps	3.33bps	3.28bps	2.71bps
`xargs.1`	3.23bps	3.30bps	2.98bps	3.38bps	3.78bps	2.94bps
Average	2.36bps	2.32bps	2.19bps	2.51bps	2.58bps	2.18bps

TABLE 7.6: Results of PPM compression using the standard methods of estimating escape probabilities, including performance for PPM predictive models which use blending rather than escape.

`zip` is a widely used on Windows-based computers to archive and compress collections of files, and it is based on the LZ77 algorithm. Version 2.20-2 was installed under Linux from the Debian package, and was executed with the `-9` argument.

`gzip` is widely used on Linux-based computers to archive and compress collections of files, and is also based on the LZ77 algorithm. Version 1.2.4-28 was installed under Linux from the Debian package, and was executed with the `-9` argument.

`bzip2` is a relatively new data compression program which is based on Burrows-Wheeler block-sorting compression with Huffman coding. Version 0.9.0c-2 was installed under Linux from the Debian package, and was executed with the `-9 --repetitive-best` arguments.

Results of compressing the Calgary and Canterbury corpora using each of these programs are summarised in table 7.7. We note that the three algorithms based on Ziv-Lempel compression perform poorly, with `zip` and `gzip` recording similar results, as is to be expected. The standard PPMC compressor performs 11% better than `gzip`, and performs slightly *worse* than `bzip2`, which is consistent with our earlier statement that the Burrows-Wheeler compressor achieves results competitive with statistical methods.

Algorithm	Calgary	Canterbury	Average
`compress`	3.73bps	3.30bps	3.57bps
`zip`	2.81bps	2.59bps	2.73bps
`gzip`	2.79bps	2.55bps	2.70bps
`bzip2`	2.49bps	2.22bps	2.39bps

TABLE 7.7: Results of other well-known compression algorithms.

Jeff Gilchrist keeps up-to-date statistics about the performance of various compression algorithms on the World Wide Web [14]. As of July 1999, according to Mr. Gilchrist's statistics, the algorithm which performed the best on the Calgary corpus was Ian Sutton's BOA compressor, which achieved an average compression performance of 1.91 bps, while Igor Pavlov's 777 compression algorithm achieved the best result on the Canterbury corpus with an average compression performance of 1.26 bps. Both compression algorithms are based on the PPM technique, and give results which are either equal to or greater than the best possible theoretical performance of a PPM compressor of order 3. This is evidence that they use higher order Markov models, or do not use Markov models at all, but the fact that they are proprietary means that we are unfortunately unable to analyse these programs in detail.

7.6 Some Modifications of and Additions to Standard PPM

We are now in a position to consider some modifications and additions to the standard PPM compressor. Ultimately we wish to apply the UpWrite Predictor to the data compression problem, but we would like to explore a few *ad hoc* approaches of our own along the way. We have developed seven possible extensions to PPM, and these are summarised below.

- The escape mechanism used by PPM possesses some defects, including the fact that the escape probability will be non-zero in situations where escape is impossible (such as when all symbols in the alphabet have already been assigned a non-zero probability). We shall explore some escape methods of our own.

- It has been shown that blending may result in superior performance. We would therefore like to explore some methods for estimating the blending weights which are independent of the estimated escape probabilities.

- One shortcoming of adaptive data compressors is the fact that the encoder and decoder must start with an empty slate, meaning that compression performance is poor at the beginning of the process. We shall explore the pre-transmission of *some* statistics about the data.

- The methods of exclusion and update exclusion tend to improve compression performance, but are not very well defined. We shall consider a more general exclusion mechanism which *discounts* symbol counts from the current predictive model if they have been used in higher order predictive models.

- PPM uses an escape mechanism to progressively fall back from high order predictive models to the highest order predictive model which assigns a non-zero probability to x_i. The ordering of the predictions made by the predictive models is implicit; it seems obvious that the predictions of a Markov model of order n should be consulted before those of a Markov model of order $n-1$. Even so, we would like to explore mechanisms for reordering the precedence assigned to the predictive models on the fly. This is of particular use in situations where the order of the predictive models is not obvious.

- The PPM compressor is based around Markov models of various orders. We shall explore the introduction of predictive models which use equivalence classifications other than that dictated by the Markovian assumption.

- A major drawback with Markov models is that they only gather statistics about the data within a small context window, and discard long-range information as a result. We shall consider incorporating long-range statistics into the PPM data compressor via a novel *goal-oriented* predictive model of our own design.

7.6.1 Alternative Escape Mechanisms

All else being equal, the performance of a PPM data compressor is dependent on the escape mechanism used. In fact, we have shown that there is a significant amount of potential improvement to be gained in this area, as the 'optimal' PPM compressor significantly out-performs PPMC.

Method F

An escape mechanism should assign a high probability to the escape symbol in predictive models at a higher order than the one which assigns the highest probability to x_i, and a low probability to the escape symbol in this model and all lower order predictive models.

We introduce Method F which finds the predictive model which assigned the highest probability to the symbol x_i *after it has been encoded*, and updates its estimate of the escape probability for the predictive models of higher orders as in equation 7.11, where $C_F(\mathcal{M}_n, s_{i-1})$ counts the number of times that a lower order predictive model made a better prediction than the model $\mathcal{M}_n$. The probability of the symbol x_i is calculated by Method F as in equation 7.12.

$$P(\varphi|\mathcal{M}_n, s_{i-1}) \approx \frac{C_F(\mathcal{M}_n, s_{i-1}) + 1}{C(x_{i-n}, \dots, x_{i-1}) + C_F(\mathcal{M}_n, s_{i-1}) + 1} \tag{7.11}$$

$$P(x_i|\mathcal{M}_n, s_{i-1}) \approx \frac{C(x_{i-n}, \dots, x_i)}{C(x_{i-n}, \dots, x_{i-1}) + C_F(\mathcal{M}_n, s_{i-1}) + 1} \tag{7.12}$$

Method G

Current methods of estimating the escape probability fail when no observations have been made, and when each symbol in the alphabet has been observed at least once. In the former case, the estimates of $P(\varphi)$ determined by Method B and Method C are undefined, whereas $P(\varphi) = 1$ would be appropriate. In the latter case, all of the escape methods assign a non-zero value to $P(\varphi)$; a waste considering that the escape symbol will never be encoded.

We would therefore like to develop a method of calculating the escape probability which satisfies these boundary constraints. We introduce Method G which satisfies both of these constraints by calculating the escape probability as in equation 7.13, with the probability of the symbol x_i being calculated as in equation 7.14. In these equations $\theta > 0$ represents a scaling constant which determines the weighting given to the escape probability.

$$P(\varphi|\mathcal{M}_n, s_{i-1}) \approx \frac{\theta(|\mathcal{A}| - q)}{C(x_{i-n}, \dots, x_{i-1}) + \theta(|\mathcal{A}| - q)} \tag{7.13}$$

$$P(x_i|\mathcal{M}_n, s_{i-1}) \approx \frac{C(x_{i-n}, \ldots, x_i)}{C(x_{i-n}, \ldots, x_{i-1}) + \theta(|\mathcal{A}| - q)} \tag{7.14}$$

Method G is nothing more than an *ad hoc* escape method developed solely to satisfy the boundary conditions, and we do not expect it to perform particularly well. It also has the disadvantage of introducing yet another parameter which needs to be adjusted in order to maximise compression performance.

Method H

Our estimate of the escape probability should also be related to the distribution of observations. Method C, for example, makes the same estimate for particular values of q and $C(x_{i-n}, \ldots, x_i)$, whereas it seems sensible that, for a given value of q, the escape probability estimate given when the non-zero counts $C(x_{i-n}, \ldots, x_i)$ are fairly uniform should be different from the escape probability estimate given when a particular count is an order of magnitude greater than the other counts.

The entropy of the probability distribution is a good indication of how the counts are distributed. It does not indicate, however, the magnitude of the counts themselves; if only a single symbol has been observed, the entropy of the probability distribution will be zero, regardless of the number of times it has been observed. We therefore propose that the entropy be calculated from a modified probability distribution formed by applying *Laplace's Law of Succession*, as in equation 7.15, whereby each count is incremented by 1 so that unobserved symbols are assigned a non-zero probability.

$$P_{Laplace}(x_i|\mathcal{M}_n, s_{i-1}) \approx \frac{C(x_{i-n}, \ldots, x_i) + 1}{C(x_{i-n}, \ldots, x_{i-1}) + |\mathcal{A}|} \tag{7.15}$$

The ratio between the entropy of a probability distribution and the maximum possible entropy can then be used as a measure of the likelihood of observing a novel symbol, as shown in equation 7.16. We shall refer to this method of estimating the escape probability as Method H. The calculation of $P(x_i|\mathcal{M}_n, s_z)$ is rather more complicated with Method H, and is performed as in equation 7.17.

$$P(\varphi|\mathcal{M}_n, s_{i-1}) \approx \frac{H_{Laplace}(\mathcal{M}_n|s_{i-1})}{\log_2 |\mathcal{A}|} \tag{7.16}$$

$$P(x_i|\mathcal{M}_n, s_z) \approx [1 - P(\varphi|\mathcal{M}_n, s_z)] \frac{C(x_{i-n}, \ldots, x_i)}{C(x_{i-n}, \ldots, x_{i-1})} \tag{7.17}$$

Note that Method G and Method H both require knowledge of the value of $|\mathcal{A}|$. If we assume that the alphabet may only ever contain a maximum of 256 symbols, which is a reasonable assumption when compression is being performed on the character level, then transmission of the alphabet size results in an overhead of a single byte, so the inclusion of $|\mathcal{A}|$ in these calculations is of little consequence.

Experimental Results

In table 7.8 we show a summary of the results obtained when using Method F, Method G and Method H in a PPM data compression system. These results were generated by replacing Method C in a standard PPMC compressor which incorporates the methods of exclusion, update exclusion and recency scaling. In the case of Method F, the results shown are for $\theta = 0.1$, which was found to give the best compression performance.

Algorithm	**Calgary**	**Canterbury**	**Average**
PPMF	2.56bps	2.22bps	2.43bps
PPMG	2.89bps	2.57bps	2.77bps
PPMH	4.18bps	3.77bps	4.03bps

TABLE 7.8: Results of using Method F, Method G and Method H to estimate the probability of a novel event.

This experiment reveals that PPMF performs almost as well as PPMC, which is encouraging, while PPMG and PPMH perform considerably worse; PPMH is outperformed even by the `compress` program! The performance of PPMG is 0.53bps worse than PPMC on the file `geo`, and 0.14bps better on the file `pic`. This suggests that the escape mechanism is used much more frequently on the former file than the latter.

7.6.2 Alternative Blending Mechanisms

Experimental results indicate that the compression performance of a PPM compressor is usually superior if the blending mechanism is used instead of the escape mechanism. We have shown that blending weights may be expressed as functions of the escape probabilities, but would like to explore methods for estimating the blending weights which are not based on the escape mechanism.

Method I

Method I calculates the blending weights $\lambda_n(s_i)$ in a way which was inspired by Method F—the predictive model which made the best prediction may be determined in an *a posteriori* fashion, and the weight assigned to that predictive model may then be updated accordingly.

Method I estimates the blending weights as in equation 7.18, where $C_I(\mathcal{M}_n, s_{i-1})$ counts the number of times $\mathcal{M}_n$ assigned the greatest probability to x_i, and m is the highest order predictive model used in the PPM predictor. Each blending weight is determined by counting the number of times the corresponding predictive model made the best prediction, adding 1 to this count to ensure that all blending weights are non-zero, then normalising by dividing the result by the number of predictions made plus the $m+1$ extra counts added to ensure that all weights are non-zero.

$$\lambda_n(s_{i-1}) \approx \frac{C_I(\mathcal{M}_n, s_{i-1}) + 1}{C(x_{i-m}, \ldots, x_{i-1}) + m + 1} \tag{7.18}$$

The weight $\lambda_n(s_{i-1})$ as calculated by Method I may be regarded as an estimate of the probability that predictive model $\mathcal{M}_n$ will assign the highest probability to the symbol x_i which follows the history s_{i-1}.

Method J

Another way of calculating the blending weights is to consider them to constitute a probability distribution over the family of predictive models used by the PPM compressor, and to initialise this distribution using the maximum entropy prior.[56] In this case, we choose to use a single set of smoothing weights, rather than making the weights functions of the context.

Baye's rule may be applied in order to recalculate this estimated probability distribution over the predictive models as data is observed, as in equation 7.19, where λ_n is the *a priori* estimate that predictive model $\mathcal{M}_n$ is the correct one. The modified Laplacian probability estimate is used when calculating the probability of the symbol x_i with respect to the predictive model $\mathcal{M}_n$ to ensure that all blending weights remain non-zero.

$$\lambda'_n \approx \frac{P_{Laplace}(x_i|\mathcal{M}_n, s_{i-1})\lambda_n}{\sum_j \lambda_j} \tag{7.19}$$

Experience has shown that this method of re-calculating the blending weights at each iteration sometimes causes one or more of the weights to tend to zero, and because the computational implementation of the algorithm uses fixed precision variables to store the weights, this has the effect of causing some weights to become zero; an event which has disastrous consequences. We therefore introduce a normalisation process which guarantees that no blending weight falls below 0.001.

Experimental Results

In table 7.9 we show the results of experiments performed using these two methods for calculating the blending weights. The results given for PPMI were obtained simply by using Method I in a standard PPMC compressor which incorporated update exclusion and recency scaling, and applying blending in place of escape. Update exclusion and recency scaling were both found to adversely effect the performance of the PPMJ compressor, however, and therefore the results shown for Method J were obtained for a compressor which incorporated neither of these techniques.

Algorithm	Calgary	Canterbury	Average
PPMI	2.59bps	2.29bps	2.48bps
PPMJ	2.63bps	2.33bps	2.52bps

TABLE 7.9: Results of using Method I and Method J to estimate the values of the blending weights.

PPMI performs similarly, but slightly worse, than PPMF, and we find it curious that the escape mechanism out-performs the blending mechanism in this particular instance, given that PPMJ and PPMF are based on similar assumptions. The performance of PPMJ is worse, and roughly equivalent to that of PPMB. As with PPMB, we note that PPMJ performs significantly better than PPMC on files which do not contain natural language text. Furthermore, the fact that PPMJ performed reasonably well given that the blending weights were *not* functions of the context should be noted.

7.6.3 Pre-Transmission of Statistics

Adaptive data compression is successful due to the fact that it avoids the need to transmit a complete predictive model prior to the data which is encoded with respect to this model by constructing the predictive model on the fly from the data which has already been encoded. This process requires that the predictive model begins in some well-defined state, and the usual approach of beginning with a clean slate may be a case of throwing the baby out with the bath water. We shall consider two techniques of pre-transmitting some minimal amount of information about the data being compressed, the result being a data compressor which lies somewhere between the two extremes of semi-adaptive and adaptive.

Pre-Transmitting the Alphabet

The PPM data compressor makes the implicit assumption that the data consists of a sequence of 8-bit symbols. In reality this is almost never the case; most data, particularly natural language text, is represented in a smaller alphabet. One rather *naïvé* way of giving the decoder some information about the data prior to encoding it is to pre-transmit the alphabet that the data will be represented in.

The overhead which this method introduces amounts to $|\mathcal{A}| + 1$ bytes, as the alphabet may be transmitted as a byte indicating the number of symbols in the alphabet followed by the symbols themselves. In the worst case scenario the compressed file will be 257 bytes larger than it would have been otherwise, but we hope that the resulting improvement in compression performance will outweigh this overhead.

Knowledge of the alphabet that the data is represented in may be used to improve compression performance by limiting $\mathcal{M}_{-1}$ to making predictions about the symbols which are known to occur in the data, and is essential if either Method G or Method H, which both need to know $|\mathcal{A}|$ in order to calculate escape probabilities, are to be used.

Pre-Transmitting the Order–0 Model

An extension to pre-transmitting the alphabet is to pre-transmit the entire Markov model of order 0. In this case a count needs to be transmitted along with each symbol. Assuming that the counts are four bytes in length, and that no attempt is made to compress the predictive model for transmission, this will result in an overhead of $5|\mathcal{A}| + 1$ bytes—the worst case scenario being a compressed file which is 1281 bytes larger than it otherwise would have been. This penalty is small enough to make exploration of this technique worthwhile.

Knowledge of $\mathcal{M}_0$ means that $\mathcal{M}_{-1}$ may be discarded entirely, since the PPM predictive model will never escape from $\mathcal{M}_0$. Predictive models of higher orders may be inferred adaptively as usual, but the question arises as to whether $\mathcal{M}_0$ should be updated during compression. It seems worthwhile to *decrement* the counts of $\mathcal{M}_0$ as symbols are observed, effectively *discarding* statistics due to symbols which have already been encoded. This updating process renders recency scaling unusable, but has the advantage that symbol counts can be excluded from all higher order predictive models *permanently* whenever the count of the symbol in $\mathcal{M}_0$ falls to zero.

Experimental Results

In table 7.10 we summarise the results of compressing the files in the standard testing corpora using each of these algorithms, which we refer to as PPMC-A and PPMC-M, where A stands for *alphabet* and M stands for *model*. PPMC-A made use of exclusion, update exclusion and recency scaling as usual, while PPMC-M necessarily used exclusion only.

Algorithm	Calgary	Canterbury	Average
PPMC-A	2.53bps	2.18bps	2.40bps
PPMC-M	2.59bps	2.27bps	2.47bps

TABLE 7.10: Results of pre-transmitting the alphabet and pre-transmitting $\mathcal{M}_0$.

Results indicate that PPMC-A performs slightly better than standard PPMC, but the improvement is so small as to be insignificant. The performance of PPMC-M is almost indistinguishable from that of a PPMC compressor which uses exclusion only, and we conclude that pre-transmitting $\mathcal{M}_0$ doesn't have any positive effects—the advantages in the PPM compressor adapting to the data more quickly are almost exactly balanced by the overhead of pre-transmitting $\mathcal{M}_0$.

7.6.4 Equivalence Exclusion

The methods of exclusion and update exclusion introduced by Moffat are found to improve the performance of the PPM compressor, and exclusion is absolutely necessary when the escape mechanism is being used in order to ensure that the predictions made by the PPM compressors are valid probability distributions. We would like to introduce a third type of exclusion, called *equivalence exclusion*, which can be used in place of update exclusion when the blending mechanism is being used.

Equivalence exclusion is based upon the notion that each observation made should have an equal contribution to the final probability distribution. This is not usually the case, as a single observation may influence the probability estimates of several predictive models. Equivalence exclusion works by progressively subtracting counts of observations made by higher order predictive models from the counts of lower order predictive models, as in equation 7.20.

$$C(x_{i-n},\dots,x_i)' = C(x_{i-n},\dots,x_i) - C(x_{i-n+1},\dots,x_i)' \tag{7.20}$$

The fact that equivalence exclusion temporarily modifies the counts of a particular predictive model based upon the counts in the predictive models of higher order means that recency scaling cannot be used, as it would not make sense to perform equivalence exclusion if the counts of some of the predictive models had been scaled, and the counts of others had not.

Experimental Results

In table 7.11 we show results for using equivalence exclusion in an 'optimal' PPM compressor and in a PPMC compressor which used blending only. We refer to these two models as PPMO-Q and PPMC-Q, where Q is used to denote *equivalence*.

Algorithm	Calgary	Canterbury	Average
PPMO-Q	1.86bps	1.63bps	1.77bps
PPMC-Q	2.59bps	2.29bps	2.48bps

TABLE 7.11: Results of applying equivalence exclusion to a standard PPM model.

These results indicate that PPMO-Q performs better than PPMO-UE, which justifies our intuition that equivalence exclusion is a more theoretically sound method of discounting the probability estimates made by lower order predictive models. PPMC-Q performs slightly better than standard PPMC which incorporates blending only, and out-performs PPMA, PPMB and PPMC on the files `book1`, `pic` and `plrabn12.txt`. It is significantly worse on the file `kennedy.xls`.

7.6.5 Re-Determining Model Precedence

The escape mechanism used by the PPM data compressor makes the implicit assumption that the family of predictive models are ordered in some way, so that the prediction process has a well-defined state at which to begin, and can gradually escape to other predictive models until the symbol which is to be encoded is assigned a non-zero probability. This is not a problem when Markov models of various orders are used, as they have a natural precedence, beginning at the highest order Markov model. If other predictive models are to be added to the PPM data compressor, it may not be so obvious how they should be ordered. A technique for determining the precedence of the predictive models based upon the predictions they make would be useful in this instance.

Entropic Precedence

The entropy of the prediction made by each predictive model provides an indication of the uncertainty that the predictive model has about the identity of the next symbol, and it seems natural to order the predictive models by their levels of uncertainty. In order to ensure that all models are assigned a non-zero uncertainty, which is necessary to avoid the problem whereby a predictive model which has only made one observation is determined to be completely certain, and to ensure that a predictive model is assigned maximum uncertainty if it has made no observations, we use the Laplacian entropy $H_{Laplace}$, which is calculated with respect to modified symbol probabilities as defined previously in equation 7.15.

Probabilistic Precedence

A second method of ordering the predictive models is to estimate the probability that a particular predictive model will assign the highest probability to x_i. Method I calculates blending weights in a way which is equivalent to estimating this probability, and we may therefore order the predictive models based upon the values of the blending weights assigned by Method I. Even though Method I is used to determine the precedence of the predictive models, it is not used to blend the predictions made by the various predictive models together, as re-determining the precedence of the predictive models is only useful if an escape mechanism is used.

Experimental Results

In table 7.12 we summarise the results of applying these two re-ordering techniques to the family of predictive models used in the standard PPMC data compressor. We refer to the two algorithms as PPMC-H and PPMC-P, where H is used to denote *entropy* and P stands for *probability*.

Both of these methods result in slightly impaired performance, and this is not surprising given the fact that the Markov models used in the PPMC compressor already

Algorithm	Calgary	Canterbury	Average
PPMC-H	2.55bps	2.25bps	2.44bps
PPMC-P	2.63bps	2.28bps	2.50bps

TABLE 7.12: Results of re-ordering the precedence of the predictive models by the methods of entropic precedence and probabilistic precedence.

have a natural precedence. The technique of re-determining model precedence may be of use if predictive models other than standard Markov models are introduced to the PPM compressor.

7.6.6 Alternative Equivalence Classifications

The PPM data compressor implicitly uses the Markovian equivalence classification. It is possible that alternative equivalence classifications will provide more accurate statistics about the data, and we are interested in exploring this.

Consider the context $\langle x_{i-2}, x_{i-1}\rangle$ which is used to make a prediction about x_i, the next symbol in the sequence s_z. If the symbol x_i has never been in this context observed before, the PPM compressor will escape, and the context $\langle \star, x_{i-1}\rangle$ will be used, where $\star$ denotes a special *wildcard symbol* which matches every possible symbol which has ever immediately preceded x_{i-1}. If x_i has never been in this context either, the PPM compressor escapes once more and uses the context $\langle \star, \star\rangle$ to make a prediction. These three contexts correspond exactly to the equivalence classifications used by Markov models of orders 2, 1 and 0 respectively.

We note that the context $\langle x_{i-2}, \star\rangle$ is never used, even though it seems quite reasonable to do so. This context occurs within the same finite window of symbols as the other contexts, and would be particularly useful in situations where x_{i-2} had been observed much more frequently than x_{i-1}. We introduce *wildcard equivalence classification* in an attempt to incorporate predictive models which use these *wildcard contexts* into the PPM data compressor.

The question arises as to how these new equivalence classes should be used, and we consider three possibilities.

- Add a new predictive model corresponding to each of the new equivalence classifications. Both the blending and escape mechanisms become more complicated in this instance, as there are many more predictive models to choose from (a PPM data compressor of order 3 uses a family of 5 predictive models, but a wildcard PPM data compressor of the same order would use a family of 9 predictive models). The precedence of the wildcard predictive models is not very well defined, meaning that we would need to re-determine the precedence of these predictive models, via methods previously discussed, prior to using them to make a prediction if the escape mechanism was to be used.

- Combine the statistics of all predictive models whose contexts contain an equal number of wildcard symbols. This is equivalent to defining another level of equivalence classification, and may be achieved easily by performing equivalence exclusion on the models individually, and then summing the counts of the predictive models in each meta-equivalence class.

- Whenever the escape mechanism requires a predictive model of order k, select a predictive model with $m-k$ wildcard symbols in its context from all such predictive models using some sort of selection criterion.

Of these, the second method seems to be the most practical, and will be the one for which we present experimental results in this section. This method is equivalent to estimating a probability distribution for the symbols which can follow the context $\langle x, y\rangle$ by counting the symbols which can follow the context $\langle x, \overline{y} \vee \overline{x}, y\rangle$, where $\overline{x}$ indicates every possible symbol apart from x. as calculated by performing equivalence exclusion on the context $\langle \star, y\rangle$, $\overline{y}$ indicates every possible symbol apart from y, as calculated by performing equivalence exclusion on the context $\langle x, \star\rangle$, and $\vee$ is the logical OR operator, denoting a sum of the two contexts.

Experimental Results

In table 7.13 we summarise experimental results for the wildcard PPM predictive model, for both the 'optimal' PPM compressor and the standard PPMC compressor, which we refer to as PPMO-W and PPMC-W respectively, where W stands for *wildcard*.

Algorithm	Calgary	Canterbury	Average
PPMO-W	1.94bps	1.75bps	1.87bps
PPMC-W	2.57bps	2.31bps	2.48bps

TABLE 7.13: Results of using wildcard equivalence models and class-based models in the PPM data compression system.

It is indeed disheartening to discover that PPMO-W performs no better than the other 'optimal' PPM compressors which were presented in table 7.5. We put this down to the fact that the wildcard model is of most use in situations where particular contexts have not been observed very often, or have not been observed at all, and that low order character-level Markov models tend not to suffer significantly from such problems. It is likely that the wildcard model may be of use in other situations; a word-level language model may benefit from its application, for instance.

Given the performance of PPMO-W, it is hardly surprising that PPMC-W performs slightly worse than the equivalent PPMC compressor.

7.6.7 Incorporating Long-Range Statistics

One drawback of using Markov models in the PPM data compressor is the fact that Markov models only take into account a short local context when estimating the probability of the next symbol in the sequence. The incorporation of long-range statistics has the potential to improve results in situations where the value of the next symbol is constrained by a symbol in the distant past. John Cleary, William Teahan and Ian Witten applied a variable-length context model to their PPM compressor, reporting an average compression performance of 2.34bps over the Calgary corpus [10]. However, Suzanne Bunton has since pointed out that Moffat had reported results superior to this when using a PPMC compressor of order 5 [6,19].

We shall take a different approach when incorporating long-range statistics into the PPM compressor; an approach which transmits the data out-of-order in a way which allows standard Markov models to constrain their predictions based upon events which are known to occur in the future. We call this method *goal-oriented modelling.*

Our work on goal-oriented modelling originated in the rather trivial pastime of generating random sentences for use in a conversation simulator which we were developing for entry into the annual Loebner Contest in Artificial Intelligence [18], which is an instantiation of the Turing Test [25]. We required a technique of generating a sentence using a Markov model (which we thought desirable, because generations made by Markov models tend to be original, in that they do not occur in the training corpus, are occasionally amusing, and are grounded in real-world data) while constraining the generation using one or more key-words, in order to make the generated sentence seem more appropriate to the sentences the human judge had previously typed to the conversation simulator.

The 'Fractal' Language Model

Our first solution to this problem was to develop a *'fractal' language model* which collects statistics about the words which occur anywhere between two specified words which are separated by an arbitrary distance within the same sentence. Generation of a novel sentence begins with a template which contains a number of target words, and is a matter of "filling in the gaps" between these words by using the 'fractal' language model to generate words which may appear between them. This generation procedure was implemented recursively, in a process vaguely reminiscent of the algorithm used to generate two-dimensional fractal 'landscapes'.

The 'fractal' language model estimates the probability of a word x_j occurring between a pair of words x_i, x_k, which are separated by an arbitrary number of intervening words within the same sentence, as in equation 7.21.

$$P(x_j|\mathcal{M}_{fractal}, x_i, x_k) \approx \frac{C(x_i, \dots, x_j, \dots, x_k)}{C(x_i, \dots, x_k)} \tag{7.21}$$

A major problem with this algorithm is that the stopping-criterion used during generation is not very well-defined, with the result that many generations tend to be quite lengthy.[57] Even so, the generated sentences did exhibit long-distance dependencies, a property which is desirable in sentence generation, but unattainable using standard Markov models.

We cannot resist giving some example generations made by the 'fractal' language model, in order to illustrate its ability to capture long-distance dependencies. Prior to doing this, we show some generations made by a 1^{st}–order Markov model, to enable a comparison to be made. Figure 7.8 shows four sentences generated by a 1^{st}–order Markov model which was inferred from 174 sentences taken from the Probert E-Text Encyclopaedia, Edition 10.0 [2]. This corpus is available from the World Wide Web site of this dissertation [1].

> The hard skeletal part of attraction between the end of smell.
>
> An alcoholic drink of modified skin glands of chalk and living substance of vision.
>
> Ecology is the organ of the fruit of the tendon is another name for the arm.
>
> The carcass of the Kiwi is another country or by the bladder is a missile is a small portable musical instrument with a baker is a German composer.

FIGURE 7.8: Four sentences generated by a 1^{st}–order Markov model.

In figure 7.9 we show three sentences generated by a 'fractal' language model inferred from the same corpus. These sentences were generated by forming a template which contained a pair of target words which had been previously observed together in at least one of the training sentences, and "filling in the gaps" between these target words. Note that these sentences do exhibit long-distance dependencies. For example, in the second sentence the words *Dance* and *movement* are clearly related, even though they are widely separated, and these two words were, in fact, the target words used when generating this sentence.

The 1^{st}–order Markov model is capable of generating the same sentences as the 'fractal' language model, but in practice it is considerably more difficult to constrain the generations of a Markov model in this way. Furthermore, the probability assigned to generations such as these by the Markov model will be less than the probability assigned to them by the 'fractal' language model, due to the fact that the Markov model is less constrained.

> A sword is an offensive weapon designed chiefly for the sale and consumption of deep unconsciousness.
>
> Dance is a person who suffers from the carcass of movement.
>
> Cement is information, especially that stored in a small, usually tree dwelling primate.

FIGURE 7.9: Three sentences generated by the 'fractal' language model.

The Goal-Oriented Language Model

We were concerned with the fact that the 'fractal' language model suffered a major problem when generating sentences, due to a rather lax stopping criterion for generation, and we therefore concentrated on constraining this language model further. This was ultimately achieved by updating the probability estimate made by the model as in equation 7.22—that is, the model collects statistics about symbols x_j which occur not anywhere between the contextual symbol pair x_i, x_k as before, but immediately after the symbol x_i. It soon became clear that this model was equivalent to a family of 1^{st}–order Markov models, each of which is constrained by a different symbol x_k which occurs somewhere in the future.

$$P(x_j|\mathcal{M}_{goal}, x_i, x_k) \approx \frac{C(x_i, x_j, \ldots, x_k)}{C(x_i, \ldots, x_k)} \tag{7.22}$$

This insight prompted the development of a general goal-oriented language model which consists of a family of Markov models of various orders, each of which is constrained by a different symbol. We refer to such models as $\mathcal{M}_n^{x_k}$, where n indicates the order of the model, and x_k indicates the constraining symbol, which may be thought of as the goal which the model will head towards when used generatively.[58] We also introduce the model denoted $\mathcal{M}_n^{\star}$, which is unconstrained, and is therefore equivalent to a standard n^{th}–order Markov model.

The goal-oriented language model will obviously require considerably more storage space than a standard Markov model of the same order. In fact, because the constraining future symbol may occur *anywhere* between the next symbol and the end of the sentence, it will require more storage space than a traditional Markov model of order $n + 1$.

Definition 7.1. *We define a string s_k to be a* prefix string *of a string s_n if the first k words in each of the two strings match, and $n \geq k$.*

Inference of an n^{th}–order goal-oriented language model is a simple process. For each prefix string s_k of the training data s_z, we update the statistics of model $\mathcal{M}_n^{x_k}$ by performing inference on the string s_{k-1} using regular Maximum Likelihood techniques. The goal-oriented model may be used as a predictor by selecting a goal symbol x_k, and combining the predictions made by the goal-oriented models $\mathcal{M}_n^{x_k}$ for various n via the standard PPM approach.

In figure 7.10 we show five amusing generations made by a 1^{st}–order goal-oriented model which was inferred from the same training corpus as before, and where the generations were seeded with target words in the same manner. These generations exhibit the same long-distance dependencies as those produced by the 'fractal' language model with the advantage that all generations are of a reasonable length (*i.e.* the average length of a generated sentence is approximately the same as the average length of a sentence in the training data).

> An acronym is a curved wooden weapon of other words.
>
> A symbol is a suspended brass disk which represents something else.
>
> Bone is a hollow shell filled with the external coating of an animal.
>
> Leonardo da Vinci was an Italian artist and expert in Kung Fu who popularised the martial arts in unpublished note books.
>
> The nose is an English naturalist. He published his theory of smell.

FIGURE 7.10: Five sentences generated by the goal-oriented model.

Data Compression with the Goal-Oriented Language Model

We are now in a position to consider the application of the goal-oriented model to the data compression problem. The goal-oriented model can be used to predict the value of x_i as described previously, but this requires knowledge of a target symbol x_k, where $k > i$. Data compression may be achieved by identifying the target symbol x_k first, encoding x_k with respect to $\mathcal{M}_0^\star$, escaping to the model of order -1 if necessary, so that the decoder is aware of the value of x_k, and then encoding the string $x_i, \ldots, x_{k-1}$ by applying standard PPM techniques to the family of models $\mathcal{M}_n^{x_k}$ for various n. Once the target symbol is reached, a new target can be selected, and this process iterated until the entire sequence has been compressed.

We have considered two ways of determining the identity of the target symbol.

- Require that the encoder and decoder decide on the target symbols *a priori*. For example, it may be decided that every tenth symbol in the data being compressed is a target symbol.

- Get the encoder to look-ahead in order to find a target symbol which will constrain the goal-oriented model the most, resulting in the best compression of the intervening symbols. This method requires that additional information be transmitted to the decoder—we may need to transmit the target symbol twice, for instance, or the number of intervening symbols, so that the decoder is able to determine when the target symbol has been reached.

Experimental Results

For the purposes of this experiment, we chose to implement the first of the two methods of determining the target symbol, introducing a new parameter to the PPM model which determines the number of intervening symbols which occur between the target symbols. When this parameter is set to zero, compression performance is equivalent to that of standard PPM. The best performing non-zero value of this parameter was found to be 10. Table 7.14 summarises the performance of an optimal goal-oriented model over both the Calgary and Canterbury corpora.

Algorithm	Calgary	Canterbury	Average
PPMO-G	2.90bps	2.71bps	2.83bps

TABLE 7.14: Results of applying the goal-oriented model to a PPM compressor.

The performance of the goal-oriented model, when used as part of an 'optimal' PPM data compressor, is incredibly poor. This is due to the fact that goal-oriented modelling is most effective when a large amount of training data is available; the constraints placed on the model effectively extends, by several orders of magnitude, the time it takes to adapt to the data being compressed. However, when a goal-oriented model is used in the standard PPMC compressor, we find that performance of the file `geo` is improved by 20%, and performance on the file `pic` is improved by 28%. This suggests that these two files contain constraints which are not captured by the traditional low order Markov models, and that the alphabet of symbols is sufficiently small to enable the goal-oriented model to adapt to the data rapidly.

We speculate that the goal-oriented language model may be of use in other natural language applications, due to its ability to capture long distance dependencies, and due to the fact that it may be easily used to generate random sentences which are constrained by a number of target words.

7.6.8 Discussion

In this section we have presented various novel additions and modifications to the techniques used by the standard PPM data compressor, and these were divided into seven categories. None of the techniques we introduced significantly improved the performance of the PPM data compressor, and some of them adversely affected its performance. It is our belief, however, that all of the techniques introduced were worth investigating, and that some of them, notably equivalence exclusion, the wildcard predictive model and the goal-oriented language model, may prove to be useful in other domains.

7.7 The UpWrite Compressor

We would like to conclude this chapter with a look at how the UpWrite Predictor may be applied to the data compression problem.

Such a task is fraught with difficulties; not the least of which is due to the fact that the UpWrite Predictor must be made adaptive in order to be useful in this domain. However, it is our strong belief that adaptive data compression provides a suitable way of testing the performance of a predictive model, because it encodes the model implicitly in the compressed representation of the data, and therefore embodies the principle of *Occam's razor* which dictates that, all else being equal, simpler models are preferable over complex ones.

Our solution to this dilemma is as follows. Compression begins with a preprocessing stage, during which the UpWrite Predictor constructs a hierarchical representation of the data being compressed, with respect to the sub-objects and quotient-objects it discovers in the data. The higher level alphabet of symbol sequences and symbol classes is then encoded for transmission, followed by the UpWritten data itself, which is compressed using the standard PPMC compressor.

The stopping criterion used during the UpWrite phase of this process is to calculate, at the end of each iteration, the size of the encoded representation of the higher level alphabet, and to halt the UpWrite process as soon as this size exceeds some predetermined threshold value.

The motivations behind approaching the problem in this manner are twofold.

- The UpWritten version of the data should be abstracted in such as way that the standard PPMC compressor can extract more redundancy from it, and that the benefits of this will outweigh the overheads of transmitting the higher level alphabet of symbol classes and symbol sequences;

- Although this approach is computationally expensive, it is much more efficient than implementing a truly adaptive version of the UpWrite Predictor.

Experimental Results

Table 7.15 summarises the results of using the UpWrite Predictor to preprocess the data which is subsequently compressed by the standard PPMC compressor. Results are shown for three different sizes of the higher level alphabet which contains the symbol classes and symbol classes discovered by the UpWrite Predictor: 256 bytes, 1024 bytes and 2048 bytes.

File	PPMC	PPMU-256	PPMU-1024	PPMU-2048
bib	2.11bps	2.06bps	2.01bps	1.99bps
book1	2.50bps	2.48bps	2.44bps	2.44bps
book2	2.24bps	2.20bps	2.17bps	2.11bps
geo	4.82bps	5.02bps	4.98bps	4.94bps
news	2.66bps	2.61bps	2.59bps	2.56bps
obj1	3.66bps	3.70bps	3.73bps	3.75bps
obj2	2.56bps	2.42bps	2.32bps	2.31bps
paper1	2.48bps	2.42bps	2.44bps	2.47bps
paper2	2.45bps	2.42bps	2.44bps	2.47bps
paper3	2.70bps	2.69bps	2.75bps	2.81bps
paper4	2.92bps	3.00bps	3.14bps	3.23bps
paper5	3.00bps	3.09bps	3.17bps	3.26bps
paper6	2.52bps	2.49bps	2.50bps	2.54bps
pic	0.97bps	0.78bps	0.77bps	0.77bps
progc	2.48bps	2.43bps	2.43bps	2.45bps
progl	1.87bps	1.80bps	1.71bps	1.66bps
progp	1.82bps	1.69bps	1.62bps	1.61bps
trans	1.74bps	1.62bps	1.54bps	1.46bps
Average	2.53bps	2.49bps	2.49bps	2.49bps
alice29.txt	2.30bps	2.27bps	2.30bps	2.33bps
asyoulik.txt	2.51bps	2.53bps	2.58bps	2.63bps
cp.html	2.35bps	2.34bps	2.40bps	2.44bps
fields.c	2.13bps	2.00bps	2.01bps	2.08bps
grammar.lsp	2.40bps	2.38bps	2.56bps	2.74bps
kennedy.xls	1.10bps	1.37bps	1.22bps	1.11bps
lcet10.txt	2.18bps	2.15bps	2.13bps	2.05bps
plrabn12.txt	2.45bps	2.43bps	2.45bps	2.47bps
ptt5	0.98bps	0.78bps	0.77bps	0.77bps
sum	2.70bps	2.52bps	2.47bps	2.47bps
xargs.1	2.98bps	3.06bps	3.23bps	3.40bps
Average	2.19bps	2.16bps	2.19bps	2.23bps

TABLE 7.15: Results of compressing an UpWritten version of the data, with the specified maximum size of the alphabet of sub-objects and quotient-objects, together with the results of standard PPMC for comparison.

From these results, it is evident that compressing an UpWritten version of the data results in better performance, even when the overheads associated with transmitting the higher level alphabet are taken into account. The best performing UpWrite compressor overall is the one which limited the size of the higher level alphabet to 256 bytes, but the performance of the UpWrite compressor on individual files tends to vary both with the file size and the type of data which it contains.

We see that the performance of the UpWrite compressor on reasonably large natural language files, such as `book1` and `book2`, tends to improve as the maximum size of the higher level alphabet is increased. This suggests that the structure found by the UpWrite Predictor improves compression, and that the overheads associated with transmitting the UpWritten alphabet are outweighed by the advantages of UpWriting the data prior to compressing it.

Files which contain quasi-natural language data, but which are more constrained, such as `bib`, `progl` and `progp`, tend to be compressed rather well by the UpWrite compressor, even for large sizes of the UpWritten alphabet. This seems to be due to the extra constraints inherent in such data. The fact that the two files which contain C source code do not compress as well as the files which contain source code in other programming languages suggests that the syntax of the C language is less constrained, and that the C source files contain more natural language comments. This is indeed the case.

Smaller natural language files, such as the six files `paper1` to `paper6`, do not compress as well. This is due to the fact that the advantages gained by UpWriting the data are quickly outweighed by the overhead of transmitting the higher level alphabet, as the size of this alphabet is a significant proportion of the size of the file being compressed.

Files which contain little, if any, language like structure, such as `geo`, `obj1` and `kennedy.xls`, tend to exhibit a worse compression performance under the UpWrite compressor than the standard PPMC compressor. This is indicative of the fact that the UpWrite Predictor incorporates language-like structure into the hierarchical representation of the data which it constructs, and files which do not contain this structure suffer due to the overheads of transmitting the higher level alphabet. The file `pic` is compressed particularly well because long sequences which correspond to empty space in the image are discovered by the UpWrite Predictor.

Overall the UpWrite Predictor results in an improvement in the performance of the PPMC compressor. This is sufficient to illustrate that the structure found by the UpWrite Predictor is useful, in that it enables us to better compress the data with respect to the model. However, the UpWrite compressor is computationally inefficient, and its performance varies too greatly on individual files for it to be used as part of a genuine data compressor.

7.8 Summary and Conclusion

In this chapter we have presented an overview of the field of data compression, concentrating on adaptive statistical data compression which provides state-of-the-art performance. Results of experiments performed on the application of many novel improvements and additions to the standard PPM data compressor were presented, and it was concluded that although the suggested modifications to PPM did not significantly improve the performance of the compressor, the individual techniques may be worth pursuing in other problem domains. The UpWrite Predictor was then successfully applied to the data compression problem, and this served to give an example of the fact that the UpWrite Predictor may be of use in applications which require a predictive model.

Notes

[46] Data compression is more properly known as *source coding.* We have deliberately chosen to use the term data compression throughout this dissertation.

[47] Although we talk about transmission of the encoded representation of the data, data compression may also be used to reduce the storage space required. We consider transmission and storage to be the same operation, and tend to prefer the term transmission due to its origins in Information Theory.

[48] Other researchers have experimented with the notion of lossy text compression in the past [27], and stenography is an example of real-life lossy text compression, as it requires human intervention to produce an approximation to the original text [12].

[49] *Messages* and *data* are synonymous.

[50] Or vice-versa—the order does not matter as long as it is consistent.

[51] This unfortunate terminology is pervasive throughout the data compression literature.

[52] Applications other than data compression may regard the generated data shown in figure 6.34 to be far from sufficient!

[53] The introduction of the Canterbury corpus evidently means that researchers are now fine-tuning their algorithms to two corpora!

[54] It should be noted the the performance of our implementations of these data compressors diverges slightly from the performance reported in the literature, and this seems to be due to implementation-specific factors which we have not been successful in locating.

[55] Recall that the files `pic` and `ptt5` are identical.

[56] This simply means that the probability distribution is initially uniform.

[57] The stopping criterion used is to continue generation until all adjacent word-pairs in the generated sentence have been observed at least once in the training corpus. The problem is that word pairs such as `the-the` are never adjacent, and that the word `the` occurs between this pair with a high probability, and is therefore likely to be generated by the model.

[58] Traditional Markov models generate sentences in a random-walk fashion, stumbling upon the end of the sentence by chance. Goal-oriented models, however, generate sentences via a constrained random walk; they are aware of certain target words through which they must pass on their way to the end of the sentence.

References

[1] Jason Hutchens' PhD web site. Available at:
`http://ciips.ee.uwa.edu.au/~hutch/phd/`

[2] The Probert e-text encyclopaedia. Available at:
`http://www.pins.co.uk/upages/probertm/`

[3] J. Aberg, Yu. M. Shtarkov, and B. J. M. Smeets. Towards understanding and improving escape probabilities in PPM. In James A. Storer and Martin Cohn, editors, *Data Compression Conference (DCC '97)*, pages 22–31, 1997.

[4] Ross Arnold and Tim Bell. A corpus for the evaluation of lossless compression algorithms. In James A. Storer and Martin Cohn, editors, *Data Compression Conference (DCC '97)*, pages 201–210, 1997.

[5] Timothy C. Bell, John G. Cleary, and Ian H. Witten. *Text Compression*. Prentice Hall, 1990.

[6] Suzanne Bunton. *On-Line Stochastic Processes in Data Compression*. PhD thesis, University of Washington, 1996.

[7] Suzanne Bunton. An executable taxonomy of on-line modeling algorithms. In James A. Storer and Martin Cohn, editors, *Data Compression Conference (DCC '97)*, pages 42–51, 1997.

[8] Suzanne Bunton. A generalization and improvement to PPM's "blending". Technical report, The University of Washington, 1997. Technical Report UW-CSE-97-01-10.

[9] M. Burrows and D. J. Wheeler. A block-sorting lossless data compression algorithm. Technical Report 124, Digital Systems Research Center, 130 Lytton Ave., Palo Alto, California 94301, May 1994.

[10] John G. Cleary, W.J. Teahan, and Ian H. Witten. Unbounded context lengths for PPM. In *Data Compression Conference (DCC '95)*, pages 52–61, 1995.

[11] John G. Cleary and Ian H. Witten. Data compression using adaptive coding and partial string matching. *IEEE Transactions on Communications*, COM-32(4):396–402, April 1984.

[12] David Crystal. *The Cambridge Encyclopedia of Language.* Cambridge University Press, 1987.

[13] Peter Fenwick. Block sorting text compression. In *19th Australasian Computer Science Conference*, January 1996.

[14] Jeff Gilchrist. The archive compression test. Available on the World Wide Web at `http://act.by.net/`

[15] Mauro Guazzo. A general minimum-redundancy source-coding algorithm. *IEEE Transactions on Information Theory*, 26(1):15–25, January 1980.

[16] Paul Glor Howard. *The Design and Analysis of Efficient Lossless Data Compression Systems.* PhD thesis, Brown University, 1993.

[17] David A. Huffman. A method for the construction of minimum-redundancy codes. *Proceedings of the I.R.E. (now known as Proceedings of the IEEE)*, 40:1098–1101, September 1952.

[18] Jason L. Hutchens. Introducing MegaHAL. In David M. W. Powers, editor, *NeMLaP3 / CoNLL98 Workshop on Human-Computer Conversation, ACL*, pages 271–274, January 1998.

[19] Alistair Moffat. Implementing the PPM data compression scheme. *IEEE Transactions on Communications*, 38(11):1917–1921, November 1990.

[20] Alistair Moffat, Radford Neal, and Ian H. Witten. Arithmetic coding revisited. *ACM Transactions on Information Systems*, 16(3):256–294, July 1998.

[21] J. Rissanen and G. G. Langdon. Arithmetic coding. *IBM Journal of Research and Development*, 23(2):149–162, March 1979.

[22] Frank Rubin. Arithmetic stream coding using fixed precision registers. *IEEE Transactions on Information Theory*, 25(6):672–675, November 1979.

[23] C.E. Shannon. Prediction and entropy of printed English. *The Bell System Technical Journal*, XXX(1):50–64, January 1951.

[24] W.J. Teahan and John G. Cleary. Models of English text. In *Data Compression Conference (DCC '97)*, pages 12–21, 1997.

[25] A.M. Turing. Computing machinery and intelligence. In D.C. Ince, editor, *Collected works of A.M. Turing: Mechanical Intelligence*, chapter 5, pages 133–160. Elsevier Science Publishers, 1992.

[26] Ross Neil Williams. *Adaptive Data Compression*. PhD thesis, University of Adelaide, 1989.

[27] Ian H. Witten, Timothy C. Bell, Alistair Moffat, Craig G. Nevill-Manning, Tony C. Smith, and Harold Thimbleby. Semantic and generative models for lossy text compression. *The Computer Journal*, 37(2):83–87, 1994.

[28] J. Gerard Wolff. Language acquisition, data compression and generalization. *Language & Communication*, 2(1):57–89, 1982.

[29] J. Gerard Wolff. Learning syntax and meanings through optimization and distributional analysis. In Y. Levy, I.M. Schlesinger, and M.D.S Braine, editors, *Categories and Processes in Language Acquisition*, chapter 7, pages 179–215. Lawrence Erlbaum, 1988.

[30] J. Gerard Wolff. Computing as compression: An overview of the SP theory and system. *New Generation Computing*, 13:187–214, 1995.

[31] J. Gerard Wolff. Computing as compression: SP20. *New Generation Computing*, 13:215–241, 1995.

[32] J. Gerard Wolff. Learning and reasoning as information compression by multiple assignment, unification and search. In A. Gammerman, editor, *Computational Learning and Probabilistic Reasoning*, pages 67–85. Chichester: Wiley, 1996.

[33] George Kingsley Zipf. *Human Behavior and the Principle of Least Effort: An Introduction to Human Ecology*. Addison Wesley, 1949.

[34] J. Ziv and A. Lempel. A universal algorithm for sequential data compression. *IEEE Transactions on Information Theory*, 23(3):337–343, May 1977.

[35] J. Ziv and A. Lempel. Compression of individual sequences via variable rate coding. *IEEE Transactions on Information Theory*, 24(5):530–535, September 1978.

Chapter 8

Conclusion

"I don't mean to deny that the evidence is in some ways very strongly in favour of your theory," said he. "I only wish to point out that there are other theories possible. As you say, the future will decide. Good-morning!"

The Return of Sherlock Holmes
SIR ARTHUR CONAN DOYLE

8.1 Introduction

In this dissertation we have developed and presented the UpWrite Predictor, a general framework for the modelling of symbolic time series, with applications in natural language processing and data compression. We have shown that two types of structure, sub-objects and quotient-objects, may be found in data from the sequence of predictions made about the data by a fairly simple predictive model, and that this structure may be used to abstract the data, allowing the model to generalise to unseen data.

The first three chapters of this dissertation provided necessary background material on Information Theory, the inference of predictive models, and the UpWrite. These three domains were brought together in chapter 5, which saw the development of the UpWrite Predictor. Various techniques for finding symbol sequences and symbol classes in symbolic time series were explored, and the novel methods of agglutination and agglomeration were developed and used in the version of the UpWrite Predictor which was subsequently implemented. Experiments were performed on this version of the UpWrite Predictor in chapter 6, and results of these experiments showed that the UpWrite Predictor is successful in extracting both types of structure from data. We then introduced the data compression problem, presenting several novel additions and modifications to the standard data compression techniques before showing that the UpWrite Predictor may be used to improve the performance of a traditional PPMC data compressor by UpWriting the data prior to compression taking place.

8.2 Future Work

Our work on the UpWrite Predictor marks the beginning of a new methodology for the inference of predictive models. As such, a large amount of potential work remains.

- In this dissertation we developed a particular implementation of the UpWrite Predictor which found structure in data using the two techniques of agglutination and agglomeration. Exploration of other techniques for finding structure in data may be worthwhile.
- An insight into the problem of syntactic category acquisition was made in chapter 5, and the ramifications of this insight need to be explored. This may result in new, more successful algorithms for the automatic extraction of ambiguous symbol classes from data.
- The UpWrite Predictor uses a greedy process to incorporate structure found from the data into the predictive model, and this may result in the premature inclusion of higher level structure to the detriment of overall performance. Examination of various techniques for the detection and correction of errors made during the UpWrite needs to be performed.
- Performance of the UpWrite Predictor needs to be evaluated on very large corpora of natural language text. This has proved difficult due to the computationally expensive nature of our implementation of the UpWrite Predictor, but alternative implementations may make a detailed exploration of performance on natural language text feasible.
- The UpWrite Predictor is a general predictive model, and we would like to test its performance on a wide variety of real-world applications which use predictive models.
- The performance of the wildcard language model introduced in this dissertation, together with the technique of equivalence exclusion necessary to blend the predictions made by several wildcard contexts together, needs to be evaluated in other domains.
- Similarly, the goal-oriented language model introduced in this dissertation needs to be more completely evaluated.

We give as references for this concluding chapter a list of the conference papers, technical reports, research reports *et cetera* which were produced during the course of our research, and details of technical presentations which have been given.

References

[1] Jason Hutchens' PhD web site. Available at:
`http://ciips.ee.uwa.edu.au/~hutch/phd/`

[2] Jason L. Hutchens. Adding a dynamic dictionary to PPM. Unpublished research report. Available at:
`http://ciips.ee.uwa.edu.au/~hutch/phd/report4.ps.gz`

[3] Jason L. Hutchens. The chunk compressor. Unpublished research report. Available at:
`http://ciips.ee.uwa.edu.au/~hutch/phd/report6.ps.gz`

[4] Jason L. Hutchens. Does chunking work? Unpublished research report. Available at:
`http://ciips.ee.uwa.edu.au/~hutch/phd/report5.ps.gz`

[5] Jason L. Hutchens. Extensions to the PPM data compression scheme. Unpublished technical report. Email `hutch@ciips.ee.uwa.edu.au` for details.

[6] Jason L. Hutchens. Finding chunks in symbolic time series. Technical presentation given as part of the CIIPS seminar series. Available at:
`http://ciips.ee.uwa.edu.au/~hutch/phd/talk5.ps.gz`

[7] Jason L. Hutchens. Finding chunks in symbolic time series. Unpublished technical report. Available at:
`http://ciips.ee.uwa.edu.au/~hutch/phd/research1.ps.gz`

[8] Jason L. Hutchens. Future directions. Unpublished research report. Available at:
`http://ciips.ee.uwa.edu.au/~hutch/phd/report7.ps.gz`

[9] Jason L. Hutchens. A goal-oriented language model. Unpublished technical report. Available at:
`http://ciips.ee.uwa.edu.au/~hutch/phd/research5.ps.gz`

[10] Jason L. Hutchens. Grammatical inference - a magical mystery tour. Technical presentation given as part of the CIIPS seminar series. Available at:
`http://ciips.ee.uwa.edu.au/~hutch/phd/talk1.ps.gz`

[11] Jason L. Hutchens. Grammatical inference and the UpWrite Predictor. Unpublished research report. Available at:
`http://ciips.ee.uwa.edu.au/~hutch/phd/report1.ps.gz`

[12] Jason L. Hutchens. Grammatical inference (pure and simple). Technical presentation given as part of the CIIPS seminar series. Available at:
`http://ciips.ee.uwa.edu.au/~hutch/phd/talk4.ps.gz`

[13] Jason L. Hutchens. Grammatical inference: What I think. Unpublished research report. Available at:
`http://ciips.ee.uwa.edu.au/~hutch/phd/report2.ps.gz`

[14] Jason L. Hutchens. A hybrid PPM/LZ data compression scheme. Unpublished technical report. Available at:
`http://ciips.ee.uwa.edu.au/~hutch/phd/research2.ps.gz`

[15] Jason L. Hutchens. Insights into the problem of syntactic category acquisition. Unpublished technical report. Email `hutch@ciips.ee.uwa.edu.au` for details.

[16] Jason L. Hutchens. NonI: A non-intelligent conversation simulator. Technical presentation given as part of the CIIPS seminar series. Available at:
`http://ciips.ee.uwa.edu.au/~hutch/phd/talk6.ps.gz`

[17] Jason L. Hutchens. The NonI conversation simulator. Unpublished technical report. Available at:
`http://ciips.ee.uwa.edu.au/~hutch/phd/research9.ps.gz`

[18] Jason L. Hutchens. Results of online language experiments. Unpublished technical report. Email `hutch@ciips.ee.uwa.edu.au` for details.

[19] Jason L. Hutchens. Text compression and language modelling. Technical presentation given as part of the CIIPS seminar series. Available at:
`http://ciips.ee.uwa.edu.au/~hutch/phd/talk3.ps.gz`

[20] Jason L. Hutchens. Why doesn't my computer understand me? Technical presentation given as part of the CIIPS seminar series. Available at:
`http://ciips.ee.uwa.edu.au/~hutch/phd/talk7.ps.gz`

[21] Jason L. Hutchens. Natural language grammatical inference. Honours thesis, Department of Electrical & Electronic Engineering, The University of Western Australia, Australia 6907, 1994.

[22] Jason L. Hutchens. How to pass the Turing test by cheating. Technical Report TR97-05, The Centre for Intelligent Information Processing Systems, Department of Electrical & Electronic Engineering, The University of Western Australia, Australia 6907, 1997.

[23] Jason L. Hutchens. Language acquisition and data compression. In Sarah Boyd, editor, *1997 Australasian Natural Language Processing Summer Workshop*, pages 39–49, February 1997.

[24] Jason L. Hutchens. Finding structure via compression. In David M. W. Powers, editor, *NeMLaP3 / CoNLL98: New Methods in Language Processing and Computational Language Learning, ACL*, pages 79–82, January 1998.

[25] Jason L. Hutchens. Introducing MegaHAL. In David M. W. Powers, editor, *NeMLaP3 / CoNLL98 Workshop on Human-Computer Conversation, ACL*, pages 271–274, January 1998.

Epilogue

"But do you mean to say," I said, "that without leaving your room you can unravel some knot which other men can make nothing of, although they have seen every detail for themselves?"

"Quite so. I have a kind of intuition that way. Now and again a case turns up which is a little more complex. Then I have to bustle about and see things with my own eyes. You see I have a lot of special knowledge which I apply to the problem, and which facilitates matters wonderfully. Those rules of deduction laid down in that article which aroused your scorn are invaluable to me in practical work. Observation with me is second nature. You appeared to be surprised when I told you, on our first meeting, that you had come from Afghanistan."

"You were told, no doubt."

"Nothing of the sort. I knew you came from Afghanistan. From long habit the train of thoughts ran so swiftly through my mind that I arrived at the conclusion without being conscious of intermediate steps. There were such steps, however. The train of reasoning ran, 'Here is a gentleman of a medical type, but with the air of a military man. Clearly an army doctor, then. He has just come from the tropics, for his face is dark, and that is not the natural tint of his skin, for his wrists are fair. He has undergone hardship and sickness, as his haggard face says clearly. His left arm has been injured. He holds it in a stiff and unnatural manner. Where in the tropics could an English army doctor have seen much hardship and got his arm wounded? Clearly in Afghanistan.' The whole train of thought did not occupy a second. I then remarked that you came from Afghanistan, and you were astonished."

A Study in Scarlet
Sir Arthur Conan Doyle

Complete Bibliography

This is a complete bibliography of all of the papers, theses, books, technical reports and so on which were read during the production of this dissertation.

J. Aberg, Yu. M. Shtarkov, and B. J. M. Smeets. Towards understanding and improving escape probabilities in PPM. In James A. Storer and Martin Cohn, editors, *Data Compression Conference (DCC '97)*, pages 22–31, 1997.

Michael Alder, Christopher deSilva, and Yianni Attikiouzel. Automatic knowledge acquisition. Technical Report TR90-14, The Centre for Intelligent Information Processing Systems, Department of Electrical & Electronic Engineering, The University of Western Australia, Australia 6907, 1990.

Michael Alder, Christopher deSilva, and Yianni Attikiouzel. On the automatic generation of higher levels of description. Technical Report TR90-15, The Centre for Intelligent Information Processing Systems, Department of Electrical & Electronic Engineering, The University of Western Australia, Australia 6907, 1990.

Michael D. Alder. Inference of syntax for point sets. In E. S. Gelsema and L. N. Kanal, editors, *Pattern Recognition in Practice IV: Multiple Paradigms, Comparative Studies and Hybrid Systems*, number 16 in Machine Intelligence and Pattern Recognition, pages 45–58. Elsevier Science B.V., June 1994.

Michael D. Alder. Principles of pattern classification: Statistical, neural net and syntactic methods of getting robots to see and hear. Unpublished book. Available at: `ftp://ciips.ee.uwa.edu.au/pub/syntactic/book`, 1994.

Michael D. Alder and Christopher deSilva. Topological stochastic grammars. In *IEEE International Symposium on Information Theory*, July 1994.

Michael D. Alder, Gek Lim, and Christopher J.S. deSilva. The syntax of images. Technical Report TR95-01, The Centre for Intelligent Information Processing Systems, Department of Electrical & Electronic Engineering, The University of Western Australia, Australia 6907, 1995.

Mike Alder. Stochastic grammatical inference. Master's thesis, Department of Mathematics, The University of Western Australia, Australia 6907, 1988.

Ross Arnold and Tim Bell. A corpus for the evaluation of lossless compression algorithms. In James A. Storer and Martin Cohn, editors, *Data Compression Conference (DCC '97)*, pages 201–210, 1997.

Fred Attneave. *Applications of Information Theory to psychology.* Henry Holt, 1959.

A. Averbuch, L. Bahl, R. Bakis, P. Brown, A. Cole, G. Daggett, S. Das, K. Davies, S. Degennaro, P. de Souza, E. Epstein, D. Fraleigh, F. Jelinek, S. Katz, B. Lewis, R. Mercer, A. Nadas, D. Nahamoo, M. Picheny, G. Schichman, and P. Spinelli. An IBM PC based large-vocabulary isolated-utterance speech recognizer. Technical report, IBM T.J. Watson Research Center, January 1986.

A. Averbuch, L. Bahl, R. Bakis, P. Brown, G. Daggett, S. Das, K. Davies, S. Degennaro, P. de Souza, E. Epstein, D. Fraleigh, F. Jelinek, B. Lewis, R. Mercer, J. Moorhead, A. Nadas, D. Nahamoo, M. Picheny, G. Schichman, P. Spinelli, D. Van Compernolle, and H. Wilkens. Experiments with the Tangora 20,000 word speech recognizer. Technical report, IBM T.J. Watson Research Center, 1986.

L. Bahl, R. Bakis, J. Bellegarda, P. Brown, D. Burshtein, S. Das, P. de Souza, P. Gopalakrishnan, F. Jelinek, D. Kanevsky, R. Mercer, A. Nadas, D. Nahamoo, and M. Picheny. Large vocabulary natural language continuous speech recognition. Technical report, IBM T.J. Watson Research Center.

L. Bahl, R. Bakis, P. de Souza, and R. Mercer. Polling: A quick way to obtain a short list of candidate words in speech recognition. Technical report, IBM T.J. Watson Research Center, September 1987.

L. Bahl, P. Brown, P. de Souza, and R. Mercer. A tree-based statistical language model for natural language speech recognition. Technical report, IBM T.J. Watson Research Center, September 1987.

L. Bahl, P. Brown, P. de Souza, R. Mercer, and M. Picheny. A method for the construction of acoustic Markov models for words. Technical report, IBM T.J. Watson Research Center, September 1987.

L. Bahl, F. Jelinek, and R. Mercer. A maximum likelihood approach to continuous speech recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-5(2), March 1983.

Yehoshua Bar-Hillel, editor. *Language and Information: Selected Essays on their theory and application.* Addison Wesley, 1964.

Doug Beeferman, Adam Berger, and John Lafferty. Text segmentation using exponential models. In *Proceedings of the Second Conference on Emperical Methods in Natural Language Processing*, 1997.

Timothy C. Bell, John G. Cleary, and Ian H. Witten. *Text Compression.* Prentice Hall, 1990.

Timothy C. Bell, Ian H. Witten, and Alistair Moffat. *Managing Gigabytes.* Van Nostrand Reinhold, 1994.

Frederic Bimbot, Roberto Pieraccini, Esther Levin, and Bishnu Atal. Variable-length sequence modeling: Multigrams. *IEEE Signal Processing Letters*, 2(6):111–113, June 1995.

Thorsten Brants. Better language models with model merging. In *Conference on Empirical Methods in NLP*, 1996.

Michael R. Brent. Advances in the computational study of language acquisition. *Cognition*, 61:1–38, 1996.

Ted Briscoe. Grammatical acquisition: Coevolution of language and of the language acquisition device. Unpublished manuscript. Available at: `http://www.cl.cam.ac.uk/users/ejb/bioprogram.ps.gz`, 1998.

P. Brown, J. Cocke, S. Della Pietra, V. Della Pietra, F. Jelinek, J. Lafferty, R. Mercer, and P. Roossin. A statistical approach to machine translation. Technical report, IBM T.J. Watson Research Center, October 1988.

P. Brown, J. Cocke, S. Della Pietra, V. Della Pietra, F. Jelinek, R. Mercer, and P. Roossin. A statistical approach to French/English translation. Technical report, IBM T.J. Watson Research Center.

P. Brown, F. Jelenik, and R.L. Mercer. Basic methods of probabilistic context free grammars. Technical report, IBM T.J. Watson Research Center.

Peter F. Brown, John Cocke, Stephen A. Della Pietra, Vincent J. Della Pietra, Frederick Jelinek, John D. Lafferty, Robert L. Mercer, and Paul S. Roossin. A statistical approach to machine translation. *Computational Linguistics*, 16(2):79–85, 1990.

Peter F. Brown, Stephen A. Della Pietra, and Vincent J. Della Pietra Robert L. Mercer. The mathematics of statistical machine translation: Parameter estimation. *Computational Linguistics*, 19(2):263–311, 1993.

Peter F. Brown, Stephen A. Della Pietra, Vincent J. Della Pietra, Jennifer C. Lai, and Robert L. Mercer. An estimate of an upper bound for the entropy of English. *Computational Linguistics*, 18(1):31–40, 1992.

Peter F. Brown, Vincent J. Della Pietra, Peter V. deSouza, Jennifer C. Lai, and Robert L. Mercer. Class-based n-gram models of natural language. *Computational Linguistics*, 18(4):467–479, 1992.

Roger Brown. *A First Language.* George Allen & Unwin Ltd., 1973.

Suzanne Bunton. *On-Line Stochastic Processes in Data Compression.* PhD thesis, University of Washington, 1996.

Suzanne Bunton. An executable taxonomy of on-line modeling algorithms. In James A. Storer and Martin Cohn, editors, *Data Compression Conference (DCC '97)*, pages 42–51, 1997.

Suzanne Bunton. A generalization and improvement to PPM's "blending". Technical report, The University of Washington, 1997. Technical Report UW-CSE-97-01-10.

Suzanne Bunton. A percolating state selector for suffix-tree context models. In James A. Storer and Martin Cohn, editors, *Data Compression Conference (DCC '97)*, pages 32–41, 1997.

M. Burrows and D. J. Wheeler. A block-sorting lossless data compression algorithm. Technical Report 124, Digital Systems Research Center, 130 Lytton Ave., Palo Alto, California 94301, May 1994.

Jeremy C Campbell. *Grammatical Man: Information, Entropy, Language and Life.* Pelican Books, 1984.

Rafael C. Carrasco and Jose Oncina. Learning stochastic regular grammars by means of a state merging method. In *International Colloquium on Grammatical Inference*, pages 139–152, 1994.

John B. Carroll, editor. *Language, thought and Reality: Selected Writings of Benjamin Lee Whorf.* The MIT Press, 1956.

Timothy A. Cartwright and Michael R. Brent. Distributional regularity and phonotactic constraints are useful for segmentation. *Cognition*, 61:93–125, 1996.

Timothy A. Cartwright and Michael R. Brent. Syntactic categorization in early language acquisition: Formalizing the role of distributional analysis. *Cognition*, 62:121–170, 1997.

Timothy Andrew Cartwright and Michael R. Brent. Segmenting speech without a lexicon: The roles of phonotactics and speech source. In *Proceedings of the First Meeting of the ACL Special Interest Group in Computational Phonology*, pages 83–90, 1994.

Francisco Casacuberta. Statistical estimation of stochastic context-free grammars using the inside-outside algorithm and a transformation on grammars. In *International Colloquium on Grammatical Inference*, pages 119–129, 1994.

Eugene Charniak. *Statistical Language Learning.* MIT Press, 1993.

Stanley F. Chen. *Building Probabilistic Models for Natural Language.* PhD thesis, Harvard University, 1996.

Stanley F. Chen and Joshua Goodman. An empirical study of smoothing techniques for language modeling. In *Proceedings of the* 34^{th} *Annual Meeting of the Association of Computational Linguistics*, pages 310–318, 1996.

Tung-Hui Chiang, Yi-Chung Lin, and Keh-Yih Su. Robust learning, smoothing, and parameter tying on syntactic ambiguity resolution. *Computational Linguistics*, 21(3):321–359, 1995.

Noam Chomsky. Aspects of the theory of syntax. Special Technical Report of the Research Laboratory of Electronics 11, The Massachusetts Institute of Technology, 1965.

Noam Chomsky. *The Logical Structure of Linguistic Theory.* Plenum Press, 1975.

Noam Chomsky. *Syntactic Structures.* Mouton, 1975.

John G. Cleary and W.J. Teahan. Experiments on the zero frequency problem. In James A. Storer and Martin Cohn, editors, *Data Compression Conference (DCC '95)*, pages 52–61, 1995.

John G. Cleary, W.J. Teahan, and Ian H. Witten. Unbounded context lengths for PPM. In *Data Compression Conference (DCC '95)*, pages 52–61, 1995.

John G. Cleary and Ian H. Witten. Data compression using adaptive coding and partial string matching. *IEEE Transactions on Communications*, COM-32(4):396–402, April 1984.

Daniel I. A. Cohen. *Introduction to Computer Theory.* John Wiley & Sons, Inc., 1991.

Jordan R. Cohen. Application of an auditory model to speech recognition. *Journal of the Acoustical Society of America*, 85(6), June 1989.

Thomas M. Cover and Joy A. Thomas. *Elements of Information Theory.* John Wiley & Sons, 1991.

Richard T. Cox. *The Algebra of Probable Inference.* Johns Hopkins Press, 1961.

David Crystal. *The Cambridge Encyclopedia of Language.* Cambridge University Press, 1987.

Richard Timon Daly. *Applications of the Mathematical Theory of Linguistics.* Mouton & Co., 1974.

Frederick J. Damerau. *Markov models and linguistic theory.* Mouton, 1971.

Carl G. de Marcken. *Unsupervised Language Acquisition.* PhD thesis, Massachusetts Institute of Technology, 1996.

Sabine Deligne and Frederic Bimbot. Language modeling by variable length sequences: Theoretical formulation and foundation of multigrams. In *ICASSP 1995 International Conference on Acoustics, Speech and Signal Processing*, pages 169–172, 1995.

Evangelos Dermatas and George Kokkinakis. Automatic stochastic tagging of natural language texts. *Computational Linguistics*, 21(2):137–163, 1995.

Steven J. DeRose. Grammatical category disambiguation by statistical optimization. *Computational Linguistics*, 14(1):31–39, 1988.

Christopher deSilva, Michael Alder, and Yianni Attikiouzel. Complexity, gratuitious and otherwise. Technical Report TR91-19, The Centre for Intelligent Information Processing Systems, Department of Electrical & Electronic Engineering, The University of Western Australia, Australia 6907, 1991.

C.J.S. deSilva, M.D. Alder, and Y. Attikiouzel. Automating knowledge engineering. Technical Report TR90-03, The Centre for Intelligent Information Processing Systems, Department of Electrical & Electronic Engineering, The University of Western Australia, Australia 6907, 1990.

G. Dunn and B.S. Everitt. *An Introduction to Mathematical Taxonomy.* Cambridge University Press, 1982.

Ted Dunning. Accurate methods for the statistics of surprise and coincidence. *Computational Linguistics*, 19(1):61–74, 1993.

Phillip Dunstan, Gek Lim, and Michael D. Alder. Real-time head detection. In *Proceedings of the IEEE International Conference on Neural Networks*, volume 5, pages 2217–2221, November 1995.

Umberto Eco. *Semiotics and the Philosophy of Language.* Macmillan Press, 1984.

Jeffrey L. Elman. Distributed representations, simple recurrent networks, and grammatical structure. *Machine Learning*, 7, 1991.

Ute Essen and Volker Steinbiss. Cooccurrence smoothing for stochastic language modeling. In *ICASSP 1990 International Conference on Acoustics, Speech and Signal Processing*, pages 161–164, 1990.

Martin Farach, Michiel Noordewier, Serap Savari, Larry Shepp, Abraham Wyner, and Jacob Ziv. On the entropy of DNA: Algorithms and measurements based on memory and rapid convergence. In *Proceedings of the 6th Annual Symposium on Discrete Algorithms*, pages 48–57, 1995.

Peter Fenwick. Block sorting text compression. In *19th Australasian Computer Science Conference*, January 1996.

Steve Finch, Nick Chater, and Martin Redington. Acquiring syntactic information from distributional statistics. In J. Levy, D. Bairaktaris, J.A. Bullinaria, and P. Cairns, editors, *Connectionist Models of Memory and Language*, pages 229–242. UCL Press.

Steven Paul Finch. *Finding Structure In Language.* PhD thesis, University of Edinburgh, 1993.

King Sun Fu. *Syntactic methods in pattern recognition.* Academic Press, 1974.

King Sun Fu, editor. *Syntactic pattern recognition, applications.* Springer-Verlag, 1977.

King-Sun Fu and Taylor L. Booth. Grammatical inference: Introduction and survey - part I. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-8(3), May 1986.

King-Sun Fu and Taylor L. Booth. Grammatical inference: Introduction and survey - part II. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-8(3), May 1986.

Charles M. Goldie and Richard G.E. Pinch. *Communication Theory.* Cambridge University Press, 1991.

I.J. Good. *The Estimation of Probabilities.* MIT Press, 1965.

Ralph Grishman. *Computational linguistics: An introduction.* Cambridge University Press, 1986.

Mauro Guazzo. A general minimum-redundancy source-coding algorithm. *IEEE Transactions on Information Theory*, 26(1):15–25, January 1980.

Zellig Harris. *Language and Information.* Columbia University Press, 1988.

Zellig Harris. *A Theory of Language and Information: A Mathematical Approach.* Oxford University Press, 1991.

Zellig S. Harris. From phoneme to morpheme. *Language*, 31:190–222, 1955.

Zellig S. Harris. Distributional structure. In Henry Hiz, editor, *Papers on Syntax*, pages 3–22. Kluwer Boston, 1981.

John A. Hartigan. *Clustering Algorithms.* John Wiley & Sons, 1975.

R.V.L. Hartley. Transmission of information. *The Bell System Technical Journal*, VII(3):535–563, July 1928.

John R. Hayes, editor. *Cognition and the Development of Language.* John Wiley & Sons, 1970.

John R. Hayes and Herbert H. Clark. Experiments on the segmentation of an artificial speech analogue. In John R. Hayes, editor, *Cognition and the Development of Language*, pages 221–234. John Wiley & Sons, Inc., 1970.

Peter A. Heeman and James F. Allen. Intonational boundaries, speech repairs and discourse markers: Modeling spoken dialog. In *Proceedings of the* 35^{th} *Annual Meeting of the Association of Computational Linguistics*, 1997.

Patrick Chisan Hew. *Pixels to Strokes to Digits.* PhD thesis, Department of Mathematics, The University of Western Australia, Australia 6907, 1999.

Charles F. Hockett. *The State of the Art.* Mouton, 1968.

John E. Hopcroft and Jeffrey D. Ullman. *Introduction to automata theory, languages, and computation.* Addison-Wesley Publishing Company, Inc., 1979.

Paul G. Howard and Jeffrey Scott Vitter. Design and analysis of fast text compression based on quasi-arithmetic coding. In James A. Storer and Martin Cohn, editors, *Data Compression Conference (DCC '93)*, pages 98–107, 1993.

Paul Glor Howard. *The Design and Analysis of Efficient Lossless Data Compression Systems.* PhD thesis, Brown University, 1993.

David A. Huffman. A method for the construction of minimum-redundancy codes. *Proceedings of the I.R.E. (now known as Proceedings of the IEEE)*, 40:1098–1101, September 1952.

Jason L. Hutchens. Natural language grammatical inference. Honours thesis, Department of Electrical & Electronic Engineering, The University of Western Australia, Australia 6907, 1994.

Jason L. Hutchens. Language acquisition and data compression. In Sarah Boyd, editor, *1997 Australasian Natural Language Processing Summer Workshop*, pages 39–49, February 1997.

Jason L. Hutchens. Finding structure via compression. In David M. W. Powers, editor, *NeMLaP3 / CoNLL98: New Methods in Language Processing and Computational Language Learning, ACL*, pages 79–82, January 1998.

David J. Hutches. *Data Structures and Algorithms for the Efficient Representation and Retrieval of Incremental Lexical Information.* PhD thesis, University of California, San Diego, 1993.

Arie S. Issar. *From Primeval Chaos to Infinite Intelligence: On Information as a Dimension and on Entropy as a Field of Force.* Avebury, 1995.

Ajay N. Jain and Alex H. Waibel. Robust connectionist parsing of spoken language. In *ICASSP 1990 International Conference on Acoustics, Speech and Signal Processing*, pages 593–596, 1990.

Nicholas Jardine and Robin Sibson. *Mathematical Taxonomy.* John Wiley & Sons, 1971.

Michèle Jardino and Gilles Adda. Automatic determination of a stochastic bi-gram class language model. In *International Colloquium on Grammatical Inference*, pages 57–65, 1994.

Frederick Jelinek. Principles of lexical language modeling for speech recognition. Technical report, IBM T.J. Watson Research Center.

Frederick Jelinek. Self-organized language modeling for speech recognition. Technical report, IBM T.J. Watson Research Center.

Frederick Jelinek. *Probabilistic Information Theory: Discrete and Memoryless Models.* McGraw-Hill, 1968.

Frederick Jelinek. The development of an experimental discrete dictation recognizer. *Proceedings of the IEEE*, 73(11), November 1985.

Frederick Jelinek. Markov source modeling of text generation. Technical report, IBM T.J. Watson Research Center, 1986.

Frederick Jelinek. Computation of the probability of initial substring generation by stochastic context free grammars. Technical report, IBM T.J. Watson Research Center, April 1990.

Frederick Jelinek. *Statistical Methods for Speech Recognition.* MIT Press, 1997.

Frederick Jelinek and John D. Lafferty. Computation of the probability of initial substring generation by stochastic context free grammars. *Computational Linguistics*, 17(3):315–323, 1991.

Philip Johnson-Laird. *The Computer and the Mind: An Introduction to Cognitive Science.* Harvard University Press, 1988.

G.J.F Jones, H. Lloyd-Thomas, and J.H. Wright. Adaptive statistical and grammar models of language for application to speech recognition. In *International Colloquium on Grammatical Inference*, pages 25/1–8, April 1993.

Karen Sparck Jones. How much has Information Technology contributed to linguistics? Prepared for a British Academy Symposium on Information Technology and Scholarly Disciplines. Available at:
`http://xxx.lanl.gov/ps/cmp-lg/9702011`

Slava M. Katz. Estimation of probabilities from sparse data for the language model component of a speech recognizer. *IEEE Transactions on Acoustics, Speech and Signal Processing*, ASSP-35(3), March 1987.

G.R. Kiss. Grammatical word classes: A learning process and its simulation. *Psychology of Learning and Motivation*, 7:1–41, 1973.

Hideki Kozima. Text segmentation based on similarity between words. In *Proceedings of the* 31^{st} *Annual Meeting of the Association of Computational Linguistics*, pages 286–288, 1993.

Brigitte Krenn and Christer Samuelsson. The linguist's guide to statistics. Unpublished book. Available at:
`http://www.coli.uni-sb.de/~krenn/stat_nlp.ps`

Julian Kupiec. Robust part-of-speech tagging using a hidden Markov model. *Computer Speech and Language*, 6(3), July 1992.

Marta Kutas and Steven A. Hillyard. Reading senseless sentences: Brain potential reflects semantic incongruity. *Science*, 207(11):203–205, January 1980.

Mark Lauer. *Designing Statistical Language Learners; Experiments on Noun Compounds.* PhD thesis, Macquarie University, 1995.

W.J.M. Levelt. Hierarchical chunking in sentence processing. *Perception & Psychophysics*, 8:99–103, 1970.

Hang Li and Naoki Abe. Word clustering and disambiguation based on co-occurrence data. In *Proceedings of the* 36^{th} *Annual Meeting of the Association for Computational Linguistics and the* 17^{th} *International Conference on Computational Linguistics*, volume 2, pages 749–755, 1998.

Gek Lim and Michael D. Alder. Calvin and Hobbes. Technical Report TR95-02, The Centre for Intelligent Information Processing Systems, Department of Electrical & Electronic Engineering, The University of Western Australia, Australia 6907, 1995.

Sok Gek Lim. *Visual Object Shape Recognition Using Hierarchical Syntax Extraction.* PhD thesis, Department of Electrical & Electronic Engineering, The University of Western Australia, Australia 6907, 1997.

Dr. S.M. Lucas. Algebraic grammatical inference. In *International Colloquium on Grammatical Inference*, pages 1–10, 1993.

Simon Lucas. Structuring chromosomes for context-free grammar evolution. In *Proc. IEEE Int. Conf. on Evolutionary Computation*, pages 130–135, 1994.

S.M. Lucas. New directions in grammatical inference. In *International Colloquium on Grammatical Inference*, pages 1/1–7, April 1993.

David J.C. MacKay. Optimisation of probabilities over probabilities—with applications in language modelling. Draft copy obtained directly from author.

David J.C. MacKay. *Bayesian Methods for Adaptive Models.* PhD thesis, California Institute of Technology, 1992.

David J.C. MacKay. Bayesian neural networks and density networks. *Nuclear Instruments and Methods in Physics Research A*, (354):73–80, 1995.

David J.C. MacKay. Density networks and their application to protein modelling. In J. Skilling and S. Sibisi, editors, *Maximum Entropy and Bayesian Methods*, pages 259–268. Kluwer Academic Publishers, 1996.

David J.C. MacKay. Information Theory, inference and learning algorithms. Unpublished book, version 2.0.0. Available at:
`http://wol.ra.phy.cam.ac.uk/mackay/itprnn/book.ps.gz`, 1999.

David J.C. MacKay and Linda C. Bauman Peto. A hierarchical Dirichlet language model. *Natural Language Engineering*, 1(3):289–307, 1995.

David M. Magerman. A review of Charniak's "Statistical language learning". *Computational Linguistics*, 21(1):103–111, 1995.

David Marr. *Vision.* W.H. Freeman and Company, 1982.

James L. Massey. Applied digital Information Theory. Lecture notes, 1992.

Robert J. McEliece. *The Theory of Information and Coding.* Addison Wesley, 1977.

Robert A. McLaughlin. *Intelligent algorithms for finding curves and surfaces in real world data.* PhD thesis, Department of Electrical & Electronic Engineering, The University of Western Australia, Australia 6907, 1999.

Robert A. McLaughlin and Michael D. Alder. Syntactic pattern recognition of simple shapes. In *Proceedings of the Australian and New Zealand Conference on Intelligent Information Systems*, pages 5–9, 1993.

Robert A. McLaughlin and Michael D. Alder. Recognising aircraft: Automatic extraction of structure by layers of quadratic neural nets. In *Proceedings of the IEEE International Conference on Neural Networks*, volume 7, pages 4288–4293, June 1994.

Robert A. McLaughlin and Michael D. Alder. Recognising cubes in images. In E. S. Gelsema and L. N. Kanal, editors, *Pattern Recognition in Practice IV: Multiple Paradigms, Comparative Studies and Hybrid Systems*, number 16, pages 45–58. Elsevier Science B.V., June 1994.

Robert A. McLaughlin and Michael D. Alder. The Hough Transform and the UpWrite: a comparison. In *Proceedings of the IEEE International Conference on Neural Networks*, volume 1, pages 146–151, 1995.

Robert A. McLaughlin and Michael D. Alder. Recognition of infra red images of aircraft rotated in three dimensions. In *Proceedings of the Australian and New Zealand Conference on Intelligent Information Systems*, pages 82–87, November 1995.

Robert A. McLaughlin and Michael D. Alder. The Hough Transform versus the UpWrite. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(4):396–400, April 1998.

Robert A. McLaughlin, Michael D. Alder, and Christopher J. S. deSilva. Inference of structure: hands. *Pattern Recognition Letters*, 15(10):957–962, October 1994.

Bernard Merialdo. Tagging English text with a probabilistic model. *Computational Linguistics*, 20(2):155–171, 1994.

Floyd Merrell. *A Semiotic Theory of Texts.* Mouton, 1985.

George A. Miller. The magical number seven, plus or minus two: Some limits on our capacity for processing information. *The Psychological Review*, 63(2):81–97, March 1956.

Don C. Mitchell, Fernando Cuetos, Martin M.B. Corley, and Marc Brysbaert. Exposure-based models of human parsing: Evidence for the use of coarse-grained (nonlexical) statistical records. *Journal of Psycholinguistic Research*, 24(6):469–489, 1995.

Alistair Moffat. Implementing the PPM data compression scheme. *IEEE Transactions on Communications*, 38(11):1917–1921, November 1990.

Alistair Moffat, Radford Neal, and Ian H. Witten. Arithmetic coding revisited. *ACM Transactions on Information Systems*, 16(3):256–294, July 1998.

Doede Nauta. *The Meaning of Information.* Mouton, 1972.

Mark Nelson and Jean-Loup Gailly. *The Data Compression Book.* M&T Books, second edition, 1996.

Craig G. Nevill-Manning. *Inferring Sequential Structure.* PhD thesis, University of Waikato, 1996.

Craig G. Nevill-Manning and Ian H. Witten. Identifying hierarchical structure in sequences: A linear-time algorithm. *Journal of Artificial Intelligence Research*, 7:67–82, 1997.

Craig G. Nevill-Manning, Ian H. Witten, and David L. Maulsby. Compression by induction of hierarchical grammars. In James A. Storer and Martin Cohn, editors, *Data Compression Conference (DCC '94)*, pages 244–253, 1994.

Hermann Ney, Ute Essen, and Reinhard Kneser. On the estimation of 'small' probabilities by leaving-one-out. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 17(12):1202–1212, December 1995.

M. Ostendorf and N. Veilleux. A hierarchical stochastic model for automatic prediction of prosodic boundary location. *Computational Linguistics*, 20(1):27–54, 1994.

Fernando Pereira, Naftali Tishby, and Lillian Lee. Distributional clustering of English words. In *International Colloquium on Grammatical Inference*, pages 5/1–3, April 1993.

Fernando C. Pereira and Yoram Singer. Beyond word n-grams. Revised version of a paper in the Proceedings of the Third Workshop on Very Large Corpora. Available at: `http://xxx.lanl.gov/ps/cmp-lg/9607016`

S. Della Pietra, V. Della Pietra, J. Gillett, J. Lafferty, H. Printz, and L. Ureš. Inference and estimation of a long-range trigram model. In *International Colloquium on Grammatical Inference*, pages 57–65, 1994.

Steven Pinker. *The Language Instinct.* The Penguin Press, 1995.

A. Radford. *Transformational Grammar, 2nd Ed.* Cambridge University Press, 1988.

Allan Ramsay. Inference in language processing. In *International Colloquium on Grammatical Inference*, pages 5/1–3, April 1993.

Martin Redington and Nick Chater. The guessing game: A paradigm for artificial grammar learning. In A. Ram and K. Eiselt, editors, *Proceedings of the Sixteenth Annual Conference of the Cognitive Science Society*, pages 745–749, 1994.

Martin Redington and Nick Chater. Transfer in artificial grammar learning: A reevaluation. *Journal of Experimental Psychology: General*, 125:123–138, 1996.

Martin Redington, Nick Chater, and Steven Finch. Distributional information and the acquisition of linguistic categories: A statistical approach. In *Proceedings of the Fifteenth Annual Conference of the Cognitive Science Society*, pages 848–853, 1993.

Martin Redington, Nick Chater, and Steven Finch. Distributional information: A powerful cue for acquiring syntactic categories. *Cognitive Science*, 22(4):425–469, 1998.

Jeffrey C. Reynar and Adwait Ratnaparkhi. A maximum entropy approach to identifying sentence boundaries. In *Proceedings of the Fifth Conference on Applied Natural Language Processing*, March 1997.

G. Riccardi, E. Bocchieri, and R. Pieraccini. Non deterministic stochastic language models for speech recognition. In *ICASSP 1995 International Conference on Acoustics, Speech and Signal Processing*, pages 237–240, 1995.

J. Rissanen and G. G. Langdon. Arithmetic coding. *IBM Journal of Research and Development*, 23(2):149–162, March 1979.

Frank Rubin. Arithmetic stream coding using fixed precision registers. *IEEE Transactions on Information Theory*, 25(6):672–675, November 1979.

Jenny R. Saffran, Elissa L. Newport, and Richard N. Aslin. Word segmentation: The role of distributional cues. *Journal of Memory and Language*, 35:606–621, 1996.

Christer Samuelsson. Grammar specialization through entropy thresholds. In *Proceedings of the* 32^{nd} *Annual Meeting of the Association for Computational Linguistics*, pages 188–195, 1994.

Christer Samuelsson. Relating Turing's formula and Zipf's law. In *Proceedings of the First International Workshop on Very Large Corpora*, pages 70–78, 1996.

Juan Andrés Sánchez and José Miguel Benedí. Statistical inductive inference of regular formal languages. In *International Colloquium on Grammatical Inference*, pages 130–138, 1994.

W.W. Schuhmacher. *Cybernetic Aspects of Language.* Mouton, 1972.

Hinrich Schütze. Part-of-speech induction from scratch. In *Proceedings of the* 31^{st} *Annual Meeting of the Association for Computational Linguistics*, pages 251–258, 1993.

Hinrich Schütze and Yoram Singer. Part-of-speech tagging using a variable memory Markov model. In *Proceedings of the* 32^{nd} *Annual Meeting of the Association for Computational Linguistics*, pages 181–187, 1994.

C.E. Shannon. Prediction and entropy of printed English. *The Bell System Technical Journal*, XXX(1):50–64, January 1951.

Claude E. Shannon and Warren Weaver. *The Mathematical theory of Communication.* University of Illinois Press, 1949.

Raoul N. Smith. *Probabalistic Performance Models of Language.* Mouton & Co., 1973.

Peter H.A. Sneath and Robert R. Sokal. *Numerical Taxonomy.* W.H. Freeman and Company, 1973.

V. Steinbiss, A. Noll, A. Paeseler, H. Ney, H. Bergmann, C. Dugast, H. H. Hamer, H. Piotrowski, H. Tomaschewski, and A. Zielinski. A 10,000 word continuous-speech recognition system. In *ICASSP 1990 International Conference on Acoustics, Speech and Signal Processing*, pages 57–60, 1990.

Andreas Stolcke. An efficient probabilistic context-free parsing algorithm that computes prefix probabilities. *Computational Linguistics*, 21(2):165–201, 1995.

Andreas Stolcke and Stephen Omohundro. Inducing probabilistic grammars by Bayesian model merging. In *International Colloquium on Grammatical Inference*, pages 106–118, 1994.

Walter Stolz. A probabilistic procedure for grouping words into phrases. *Language and Speech*, 8:219–235, 1965.

W. J. Teahan. *Modelling English Text.* PhD thesis, The University of Waikato, 1998.

W.J. Teahan. Probability estimation for PPM. Paper submitted to NZCSRSC'95. Available at:
`http://www.cs.waikato.ac.nz/~wjt/papers/NZCSRSC.ps.gz`

W.J. Teahan and John G. Cleary. Models of English text. In *Data Compression Conference (DCC '97)*, pages 12–21, 1997.

A.M. Turing. Computing machinery and intelligence. In D.C. Ince, editor, *Collected works of A.M. Turing: Mechanical Intelligence*, chapter 5, pages 133–160. Elsevier Science Publishers, 1992.

A.M. Turing. Digital computers applied to games. In D.C. Ince, editor, *Collected works of A.M. Turing: Mechanical Intelligence*, chapter 6, pages 161–185. Elsevier Science Publishers, 1992.

A.M. Turing. Intelligent machinery. In D.C. Ince, editor, *Collected works of A.M. Turing: Mechanical Intelligence*, chapter 3, pages 107–127. Elsevier Science Publishers, 1992.

A.M. Turing. Solvable and unsolvable problems. In D.C. Ince, editor, *Collected works of A.M. Turing: Mechanical Intelligence*, chapter 7, pages 187–203. Elsevier Science Publishers, 1992.

J.P. Ueberla. Domain adaption with clustered language models. Unpublished paper. Available at:
`http://xxx.lanl.gov/ps/cmp-lg/9703001`

J.P. Ueberla. Analysis of a simple bipos language model - attempt at a strategy to improve language models for speech recognition. In *International Colloquium on Grammatical Inference*, pages 10/1–8, April 1993.

Satosi Watanabe. *Knowing and Guessing.* John Wiley & Sons, Inc., 1969.

Ralph Weischedel, Marie Meteer, Richard Schwartz, Lance Ramshaw, and Jeff Palmucci. Coping with ambiguity and unknown words through probabilistic models. *Computational Linguistics*, 19(2):359–382, 1993.

Norbert Wiener. *Cybernetics, or Control and Communication in the Animal and the Machine.* John Wiley & Sons, 1961.

Paul S. Williams. *The automatic hierarchical decomposition of images into sub-images for use in image recognition and classification.* PhD thesis, Department of Electrical & Electronic Engineering, The University of Western Australia, Australia 6907, 1999.

Ross Neil Williams. *Adaptive Data Compression.* PhD thesis, University of Adelaide, 1989.

Terry Winograd. *Understanding Natural Language.* Academic Press, 1972.

Ian H. Witten and Timothy C. Bell. The zero-frequency problem: Estimating the probabilities of novel events in adaptive text compression. *IEEE Transactions on Information Theory*, 37(4):1085–1094, July 1991.

Ian H. Witten, Timothy C. Bell, Alistair Moffat, Craig G. Nevill-Manning, Tony C. Smith, and Harold Thimbleby. Semantic and generative models for lossy text compression. *The Computer Journal*, 37(2):83–87, 1994.

J. G. Wolff. An algorithm for the segmentation of an artificial language analogue. *British Journal of Psychology*, 66(1):79–90, 1975.

J. G. Wolff. The discovery of segments in natural language. *British Journal of Psychology*, 68:97–106, 1977.

J. Gerard Wolff. Language acquisition, data compression and generalization. *Language & Communication*, 2(1):57–89, 1982.

J. Gerard Wolff. Learning syntax and meanings through optimization and distributional analysis. In Y. Levy, I.M. Schlesinger, and M.D.S Braine, editors, *Categories and Processes in Language Acquisition*, chapter 7, pages 179–215. Lawrence Erlbaum, 1988.

J. Gerard Wolff. A scaleable technique for best-match retrieval of sequential information using metrics-guided search. *Journal of Information Science*, 20(1):16–28, 1994.

J. Gerard Wolff. Computing as compression: An overview of the SP theory and system. *New Generation Computing*, 13:187–214, 1995.

J. Gerard Wolff. Computing as compression: SP20. *New Generation Computing*, 13:215–241, 1995.

J. Gerard Wolff. Learning and reasoning as information compression by multiple assignment, unification and search. In A. Gammerman, editor, *Computational Learning and Probabilistic Reasoning*, pages 67–85. Chichester: Wiley, 1996.

J.H. Wright, G.J.F. Jones, and H. Lloyd-Thomas. Training and application of integrated grammar/bigram language models. In *International Colloquium on Grammatical Inference*, pages 246–259, 1994.

Ave Wrigley. Parse tree n-grams for spoken language modeling. In *International Colloquium on Grammatical Inference*, pages 26/1–6, April 1993.

George Kingsley Zipf. *Human Behavior and the Principle of Least Effort: An Introduction to Human Ecology.* Addison Wesley, 1949.

J. Ziv and A. Lempel. A universal algorithm for sequential data compression. *IEEE Transactions on Information Theory*, 23(3):337–343, May 1977.

J. Ziv and A. Lempel. Compression of individual sequences via variable rate coding. *IEEE Transactions on Information Theory*, 24(5):530–535, September 1978.

Index