
Large-Alphabet Sequence Modelling A Comparative Study

Wen Shao

Supervisory Panel Chair: Marcus Hutter

May, 2014

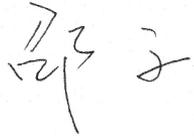
A thesis submitted for the degree of Masters of Philosophy of The Australian National University.



Declaration

This thesis is an original work. None of the work has been previously submitted by me for the purpose of obtaining a degree or diploma in any university or other tertiary education institution. To the best of my knowledge, this thesis does not contain material previously published by another person, except where due reference is made in the text.

Wen Shao

A handwritten signature in Chinese characters, appearing to be '文少' (Wen Shao), written in a cursive style.

11/06/2014.

Acknowledgements

Foremost, I would like to express my sincere gratitude to my supervisor Prof. Marcus Hutter for the continuous support of my Masters study and research, for his patience, motivation, enthusiasm, and immense knowledge. His guidance helped me in all the time of research and writing of this thesis. I could not have imagined having a better supervisor and mentor for my Masters study.

I thank my fellow labmates in Reinforcement Learning Group at the Research School of Computer Science at the ANU: Tor Lattimore, Mayank Daswani, Hadi Afshar, for the stimulating discussions, for the sleepless nights we were working together before deadlines, and for all the fun we have had in the last two years.

Last but not the least, I would like to thank my family and my partner for their support throughout my life.

Abstract

Most raw data is not binary, but over some often large and structured alphabet. Sometimes it is convenient to deal with binarised data sequence, but typically exploiting the original structure of the data significantly improves performance in many practical applications. In this thesis, we study Martin-Löf random sequences that are maximally incompressible and provide a topological view on the size of the set of random sequences. We also investigate the relationship between binary data compression techniques and modelling natural language text with the latter using raw unbinarised data sequence from a large alphabet. We perform an experimental comparative study for them, including an empirical comparison between Kneser-Ney (KN) variants with regular Context Tree Weighting algorithm (CTW) and phase CTW, and with large-alphabet CTW with different estimators. We also apply the idea of Hutter's adaptive sparse Dirichlet-multinomial coding to the KN method and provide a heuristic to make the discounting parameter adaptive. The KN with this adaptive discounting parameter outperforms the traditional KN method on the Large Calgary corpus.

Keywords

Kolmogorov Complexity; minimum description length; online learning; sparse coding; adaptive parameters; Dirichlet process; data compression; universal compression; context tree weighting; small/large alphabet; smoothing;

Contents

| | | |
|----------|--|-----------|
| 1 | Introduction | 11 |
| 2 | Notation and Preliminaries | 13 |
| 3 | Small-Alphabet Sequence Compression Techniques | 18 |
| 3.1 | Kolmogorov Complexity and Martin-Lof Randomness | 19 |
| 3.1.1 | Plain Kolmogorov Complexity | 20 |
| 3.1.2 | Prefix Complexity | 22 |
| 3.1.3 | Randomness and Halting Probability | 23 |
| 3.1.4 | A Topological View on Random Reals | 24 |
| 3.2 | Stationary Memoryless Source | 27 |
| 3.2.1 | Arithmetic Coding | 27 |
| 3.2.2 | Universal Coding | 30 |
| 3.2.3 | Krichevsky-Trofimov Estimator | 34 |
| 3.3 | Stationary Source with Finite Memory | 34 |
| 3.3.1 | Binary Context Tree Weighing Algorithm | 35 |
| 3.3.2 | Model Class and Redundancy | 36 |
| 4 | Large-Alphabet Sequence Compression Techniques | 38 |
| 4.1 | Sparse Sequential Dirichlet Coding | 38 |
| 4.2 | Sparse Adaptive Dirichlet-Multinomial Coding | 39 |
| 5 | Text Modelling Techniques | 41 |
| 5.1 | n-gram Models over Sparse Data | 41 |
| 5.2 | Smoothing Techniques | 41 |
| 5.2.1 | Additive Smoothing | 42 |
| 5.2.2 | Jelinek Mercer Smoothing | 43 |
| 5.2.3 | Absolute Discounting | 43 |
| 5.2.4 | Kneser-Ney Smoothing | 44 |
| 5.2.5 | Modified Kneser-Ney Smoothing | 45 |
| 5.3 | Pitman-Yor Process | 46 |
| 5.3.1 | Dirichlet Distribution and Dirichlet Process | 46 |
| 5.3.2 | Stick Breaking Representation and Chinese Restaurant Process | 47 |
| 5.3.3 | Hierarchical Pitman-Yor Process | 49 |
| 6 | Experiments and Discussion | 51 |
| 6.1 | Kneser-Ney and Binary CTW | 53 |
| 6.1.1 | CTW Implementation Notes | 54 |
| 6.1.2 | Kneser-Ney and Regular CTW | 58 |
| 6.1.3 | Kneser-Ney and Phase CTW | 61 |
| 6.2 | Large-alphabet CTW with Sparse Adaptive Dirichlet-multinomial Coding | 62 |
| 6.3 | Large-alphabet CTW on Artificial Data | 65 |
| 6.4 | Applying SAD to Kneser-Ney | 67 |
| 7 | Conclusion | 72 |

| | | |
|---|-----------------------------------|----|
| A | Explicit Equations for Kneser-Ney | 76 |
| B | Table of Notation | 78 |

1 Introduction

Personal motivation. Compression is of great practical importance. It is widely used in data storage and data transmissions, especially with the advent of the Internet. Loosely speaking, the task of compression is to describe the data coming from a certain source as compactly as possible, which is achieved generally by devising a coding scheme according to a certain predetermined performance criteria, e.g., redundancy. There are two types of compression techniques: lossy and lossless data compression. The former is based on the assumption that some level of information loss can be tolerated and sometimes is necessary for storage purposes. For instance, a lossy compressor may remove non-audible components of an audio signal because humans can only hear a certain range of sound frequencies. In this thesis, we are, however, only interested in lossless compression, in which one could reconstruct the original data entirely and exactly from its encoded version using some computable decoding process.

Beyond its practical importance, compression is also theoretically and philosophically interesting. Science is about learning from the past and predicting the future, at least to a large extent. For example, what is the weather going to be like tomorrow given the weather in the past several years. Many scientific problems can be reformulated as sequence prediction problems. Take the classification problem for example, with the training data being viewed as a sequence of (input, class label) pairs a classification problem can be re-expressed as a problem that given a training sequence and a new input one wants to predict the class label as accurately as possible. However, before being able to predict the future well, as an intermediate step, one needs to understand the past, often by developing compact descriptions/representations of it, and we call such process learning.¹ In light of this, the first step in science is arguably about finding a compact description of things we have observed. We then call such a representation a theory, a law or, in the field of machine learning, a model. As such, a major task for scientists is to represent/describe the observations more compactly and then use the discovered theories/laws/models to predict the data which cannot be observed directly or happens in the future. The problem of finding such a compact description of the past therefore lies at the heart of science and is a fundamental problem.

Although most real-life data is over some large and structured alphabet and may take different forms, for example text data consisting of a sequence of letters or words or a movie made up of images, one of the treatments is to convert them into binary sequences first regardless of its original high level structure. Data can easily be binarised, and in fact is binarised in all modern computers. As we will see in this thesis,

¹We note, however, that induction/learning and prediction can occur together. The learning process may not always explicitly happen prior to prediction. For example, a binary sequence $x_{1:n}$ that is sampled independently from a Bernoulli distribution $Bern(\theta)$ with unknown θ being the probability of $X_i = 1$ for all i . The task is to predict the continuation x_{n+1} as accurately as possible. There are at least two perspectives on this problem. The first one involves an explicit learning process that learns/estimates the unknown θ using the observations $x_{1:n}$, then, as a second step, to predict x_{n+1} using the estimated $\hat{\theta}(x_{1:n})$. However, the prediction can also be done without having explicitly estimated θ . For example, if we have a prior belief on all possible θ 's, denoted as $\pi(\theta)$, then we can construct a distribution $\xi(x) = \int_0^1 \pi(\theta) P(x|\theta) d\theta$, often called Bayes mixture. The conditional probability $\xi(x_{n+1}|x_{1:n})$ can be used directly for prediction. However, the induction/learning process in either case, explicit or implicit, is indispensable to prediction.

sometimes one can safely ignore the original data structure, e.g. for most parts in Algorithmic Information Theory (AIT). Also data compression is often studied for large but theoretically convenient universal classes of binary sequences and as a second step, if at all, adapted to larger alphabet. Many asymptotic compression results are derived for binary alphabet and have strong theoretical guarantees only for small alphabet. However, when it comes to practical data compression or modelling, exploiting the original structure of the data can substantially improve performance. Hence data compression and modelling for concrete tasks are usually done on the original or suitable represented data, as opposed to binary data.

This thesis investigates the relationship between these two paradigms and experimentally compares the approaches used in binary data compression and large-alphabet data modelling. As a prototypical example for the latter, we take here document analysis or more specifically statistical natural language processing.

The outline of this thesis. After establishing notations in Section 2, we focus on compression techniques for binary sequences in Section 3 while large-alphabet compression techniques are discussed in Section 4. Large-alphabet text modelling techniques are surveyed in Section 5. We conduct experiments and present and discuss the results in Section 6. Conclusions are made in Section 7.

2 Notation and Preliminaries

In this section, we introduce notation used in this thesis and some basic preliminaries. We use $\mathbb{N}, \mathbb{N}_0, \mathbb{Z}, \mathbb{Q}, \mathbb{R}$ to denote the sets of natural numbers ($\{1, 2, \dots\}$), natural numbers including zero ($\{0, 1, 2, \dots\}$), integers, rational numbers, and real numbers respectively. \mathcal{S} is used to denote a generic set. The size/cardinality of a set \mathcal{S} is denoted by $|\mathcal{S}|$. $\mathcal{S}_1 \cup \mathcal{S}_2$ and $\mathcal{S}_1 \cap \mathcal{S}_2$ denote the union and intersection of two sets respectively. $\mathcal{S}_1 - \mathcal{S}_2$ denotes the relative complement of \mathcal{S}_2 in \mathcal{S}_1 . In certain settings all sets under discussion are considered to be subsets of a given universal set Ω . In such cases, $\Omega - \mathcal{S}$ is simply called the complement of \mathcal{S} , denoted by \mathcal{S}^c . The power set of a set \mathcal{S} is the set of all subsets of \mathcal{S} , including \mathcal{S} itself and the empty set, denoted by $2^{\mathcal{S}}$. If not explicitly stated otherwise, lower-case letters i, j, k, n are used to denote natural numbers and t represents discrete time steps. We use $\langle \cdot, \cdot \rangle$ to denote some one-to-one mapping from \mathbb{N}^2 to \mathbb{N} , that is, this function associates a unique natural number $\langle x, y \rangle \in \mathbb{N}$ with each pair $(x, y) \in \mathbb{N}^2$. For example, $\langle \cdot, \cdot \rangle$ can be defined as $y + (x + y + 1)(x + y)/2$. Iverson bracket is used in its common sense, that is

$$\mathbb{I}[P] = \begin{cases} 1 & \text{if } P \text{ is true} \\ 0 & \text{otherwise} \end{cases}$$

Inequalities. $<, >, \leq, \geq$ are standard inequalities. Let f, g be real valued functions. We write $f(x) \stackrel{+}{\geq} g(x)$ if there exists a constant c such that $f(x) \geq g(x) + c$ for all x . $f(x) \stackrel{-}{\leq} g(x)$ is defined similarly. $f(x) \stackrel{\pm}{\geq} g(x)$ if $f(x) \stackrel{+}{\geq} g(x)$ and $f(x) \stackrel{-}{\leq} g(x)$. We write $f(x) \stackrel{\times}{\geq} g(x)$ if there exists a constant $c > 0$ such that $f(x) \geq cg(x)$ for all x . $f(x) \stackrel{\times}{\leq} g(x)$ is defined similarly. $f(x) \stackrel{\times}{\geq} g(x)$ if $f(x) \stackrel{\times}{\geq} g(x)$ and $f(x) \stackrel{\times}{\leq} g(x)$.

Finite strings. We are concerned with strings over a non-empty finite set \mathcal{X} of letters or symbols. We use letters and symbols interchangeably to mean an element in \mathcal{X} . We are primarily interested in binary set $\mathbb{B} = \{0, 1\}$ as an example of small alphabet and some generic, but normally large finite alphabet \mathcal{X} . In this thesis, we use ‘string’, ‘word’ and ‘sequence’ almost synonymously; however, we tend to use ‘string’ and ‘word’ to refer to finite strings and ‘sequence’ to refer to infinite ones. If not explicitly stated otherwise, lower-case letters x, y, z are used to denote finite strings (from some \mathcal{X}^*), ϵ to denote the empty string and $\omega = \omega_1 \omega_2 \omega_3 \dots$ infinite sequences (from some \mathcal{X}^∞). The length of a finite string x is denoted by $\ell(x)$. The i^{th} symbol of a string x is denoted by x_i ($0 < i \leq \ell(x)$). \mathcal{X}^n is the set of all strings over \mathcal{X} of length n . The set of all finite string over \mathcal{X} is denoted \mathcal{X}^* . Substrings are denoted $x_{i:j} := x_i x_{i+1} \dots x_j$ where $i, j \in \mathbb{N}$ and $i \leq j$. If $i > j$, then $x_{i:j} = \epsilon$. A useful shorthand is $x_{<t} := x_{1:t-1}$. Strings may be concatenated. Let $x, y \in \mathcal{X}^*$ of length n and m respectively. Then,

$$xy := x_1 x_2 \dots x_{n-1} x_n y_1 y_2 \dots y_{m-1} y_m$$

A string x is called a (proper) prefix of y if there is a $z (\neq \epsilon)$ such that $xz = y$. A set of strings is called a prefix-free set (prefix-code) if no element is a proper prefix of another. In the context of information theory, using prefix-codes, a message can be uniquely decoded and thus can be transmitted as a sequence of concatenated code words

without any out-of-set element to separate the words in the message. For example, modern English uses a white space to separate words because English vocabulary itself is not a prefix-code. For instance, a concatenated string ‘farsidebag’ can be segmented into ‘far sidebag’ or ‘farside bag’ or ‘far side bag’. In many situations, a prefix-code is preferable. For finite binary strings, we can construct a prefix code for a subset $\mathcal{S} \subseteq \mathbb{B}^*$ in the following way: given a string $x \in \mathcal{S}$, its i^{th} order prefix-code word given by the function $E_i: \mathbb{B}^* \rightarrow \mathbb{B}^*$ is defined recursively

$$E_i(x) = \begin{cases} 1^x 0 & \text{for } i=0 \\ E_{i-1}(l(x))x & \text{otherwise} \end{cases} \quad (1)$$

There is a bijective mapping between \mathbb{B}^* and \mathbb{N} , i.e., every element in \mathbb{B}^* can be paired with exactly one element in \mathbb{N} . For example, we get a bijection if we map number $i \in \mathbb{N}$ to $x \in \mathbb{B}^*$ if $i+1$ has a binary expansion of $1x$. The natural number 4 is mapped to the binary string 01. We do not distinguish these two sets in this thesis. In Equation (1), 1^x denotes concatenation of x many 1’s, with x interpreted as a natural number. For example, $E_0(01) = 1111 0$, $E_1(01) = 110 01$ and $E_2(01) = 101 01$. Analogously a string x is called a (proper) suffix of y if there is a $z (\neq \epsilon)$ such that $zx = y$. We use the function $\# : \mathcal{X} \rightarrow \mathbb{N}$ to count the number of occurrence of a certain symbol in some understood corpus and relatedly $\#_t : \mathcal{X} \times \mathcal{X}^{t-1} \rightarrow \mathbb{N}$ to count the number of occurrence of x_t in $x_{1:t-1}$, that is, $\#_t(x_t | x_{1:t-1}) = \sum_{i=1}^{t-1} \mathbb{I}[x_i = x_t]$.

Infinite sequences. The set of infinite sequences is denoted as \mathcal{X}^∞ , which is the infinite product of \mathcal{X} with itself. A point (an element) in \mathcal{X}^∞ is a one-way infinite sequence, normally denoted by $\omega = \omega_{1:\infty} = \omega_1 \omega_2 \omega_3 \dots$ with $\omega_i \in \mathcal{X}$ for all i . We are particularly interested in \mathbb{B}^∞ , which contains many interesting elements, for example the element with $\omega_i = 1$ if i is a prime number and $\omega_i = 0$ otherwise. Finite binary strings may be concatenated with infinite sequences. For a finite binary string $x = x_{1:n} = x_1 x_2 \dots x_n$ and an infinite binary sequence $\omega = \omega_{1:\infty} = \omega_1 \omega_2 \dots$, their concatenation is

$$x\omega = x_1 x_2 \dots x_n \omega_1 \omega_2 \dots$$

The binary expansion of a real number $r \in [0,1]$ establishes a mapping between \mathbb{B}^∞ and $[0,1]$. Given an infinite sequence $\omega_{1:\infty} \in \mathbb{B}^\infty$, a function $f: \mathbb{B}^\infty \rightarrow [0,1]$ maps it to the set of reals in the following way

$$0.\omega := f(\omega_{1:\infty}) = \sum_{n=1}^{\infty} \frac{\omega_n}{2^n}$$

We note, however, that this is not a bijective mapping. In fact, \mathbb{B}^∞ is homomorphic to the Cantor set, and hence is called a Cantor space.

(Un)Countable sets. Mathematically, a set is a collection of elements. A countable set is in some sense a small set that has the same cardinality as some subset of the set of \mathbb{N} . Formally,

Definition 1 (Countable set). A set \mathcal{S} is called countable if there exists an injective function f from \mathcal{S} to the natural numbers \mathbb{N} . If such a function does not exist, then \mathcal{S} is called uncountable.

Remark 2. If f is also surjective and therefore bijective, then \mathcal{S} is called countably infinite.

Probability measures. A (probability) measure characterises the size of a set, more precisely, the size of a subset of some set Ω . If Ω is finite, it is a trivial task to describe the size of any of its subset, for example, just by simple counting (we also call it a counting measure); however, this method is not going to work when Ω is an uncountable set, because otherwise any uncountable (or infinitely countable) subsets of a uncountable set would end up with having equal size, which is not very useful. The concept of measures helps to extend the idea of probability from finite (or countable) sample spaces to continuous ones.

Axiom 3 (Axioms of σ -algebra). *Given a non-empty set Ω , 2^Ω denotes the power set of Ω . $\mathcal{A} \subseteq 2^\Omega$ is a σ -algebra if it satisfies the following.*

1. $\Omega \in \mathcal{A}$.
2. (*\mathcal{A} is closed under countable union*) If A_1, A_2, \dots is a countable sequence of elements in \mathcal{A} , then the union $\bigcup_n A_n \in \mathcal{A}$.
3. (*\mathcal{A} is closed under complement*) If $A \in \mathcal{A}$, its complement $A^c \in \mathcal{A}$.

We call an element in \mathcal{A} a measurable set or an event in a probabilistic setting.

Remark 4. The axioms do not say that all subsets of Ω , denoted by 2^Ω , are measurable. In fact, there are many trivial σ -algebras that obey the axioms, for example the set $\{\Omega, \emptyset\}$ is a valid σ -algebra of Ω .

Axiom 5 (Axioms of probability measure). *A probability measure defined on a σ -algebra \mathcal{A} of Ω is a function $P: \mathcal{A} \rightarrow [0, 1]$ that satisfies*

1. $P(\Omega) = 1$
2. (*Countable additivity*) For any countable sequence $\{A_n\}_{n=1}^\infty$ of pairwise disjoint events (for any $A_i, A_j \in \{A_n\}_{n=1}^\infty$ where $i \neq j$, $A_i \cap A_j = \emptyset$)

$$P\left(\bigcup_{n=1}^{\infty} A_n\right) = \sum_{n=1}^{\infty} P(A_n)$$

The value $P(A)$ is called the probability measure of the event A , or simply the probability of A .

We note, however, these axioms can only be used to verify whether a given function is a valid probability measure and they tell little about how to construct a (non-trivial) probability measure over Ω . In general this is not easy. One can instead define P on some geometrically well shaped sets from which a measure on Ω can be generated. One can formalise this idea and define probability measures on a continuous space \mathcal{X}^∞ for some finite non-empty \mathcal{X} . Consider a continuous space \mathcal{X}^∞ , then a cylinder set is defined as follows

Definition 6 (Cylinder set). A cylinder set is a set $\Gamma_x \subset \mathcal{X}^\infty$ defined by

$$\Gamma_x := \{x\omega : \omega \in \mathcal{X}^\infty\}$$

with $x \in \mathcal{X}^*$.

Geometrically speaking, a cylinder set can be identified with a half open interval in $[0,1)$. For instance, consider the cylinder sets in \mathbb{B}^∞ , which are defined by $\Gamma_x = \{x\omega : \omega \in \mathbb{B}^\infty\}$ with $x \in \mathbb{B}^*$. We can associate Γ_x with a unique half open interval $[0.x, 0.x + 2^{-\ell(x)})$, where $0.x$ is the real number with binary expansion x . The length of the interval is $2^{-\ell(x)}$. Note that x is a (proper) prefix of y iff $\Gamma_y \subseteq (\subset) \Gamma_x$.

Definition 7 (probability measure on \mathcal{X}^∞ and probability distribution on \mathcal{X}^*). Let $\mathcal{G} = \{\Gamma_x : x \in \mathcal{X}^*\}$ be the set of all cylinder sets in \mathcal{X}^∞ . Let \mathcal{A} be the minimal σ -algebra containing \mathcal{G} . A function $\mu : \mathcal{A} \rightarrow \mathbb{R}$ defines a probability measure if

$$\begin{aligned} \mu(\Gamma_\epsilon) &= 1 \\ \mu(\Gamma_x) &= \sum_{y \in \mathcal{X}^*} \mu(\Gamma_{xy}) \end{aligned}$$

A function $P : \mathcal{X}^* \rightarrow [0,1]$ is a probability distribution if $\sum_{x \in \mathcal{X}^*} P(x) = 1$

We write $\mu(x)$ to denote $\mu(\Gamma_x)$. $\mu(x)$ is the μ -probability that a sequence starts with x . $\mu(y|x) := \frac{\mu(xy)}{\mu(x)}$ is the conditional probability of observing $y \in \mathcal{X}$ given that $x \in \mathcal{X}^*$ has already been observed. In particular, we use \mathcal{L} to denote the uniform measure i.e. $\mathcal{L}(x) = |\mathcal{X}|^{-\ell(x)}$ for all $x \in \mathcal{X}^*$.

Information theory. C.E. Shannon in his famous paper [Sha48] laid the foundation of information theory. An important concept in information theory is the entropy, defined as follows

Definition 8 (entropy). Given a finite/countable set Ω with a probability distribution P . Let $P(x)$ be the probability of $x \in \Omega$. The entropy of P is defined by

$$H(P) := \sum_{x \in \Omega} P(x) \log \frac{1}{P(x)}$$

The choice of logarithm base fixes the units used to measure entropy, but is otherwise unimportant. Common choices include base 2 that implies a measurement in bits and base 8 that implies a measurement in bytes.

Information theory is concerned with communicating a message between a sender and a receiver. Mathematically speaking, consider a non-empty finite/countable set Ω of objects and a sender wants to transfer elements in Ω over to a receiver. In the setting of information theory, the element in Ω is normally called a source word. Let $P(x)$ denote the probability of a source word $x \in \Omega$. Additionally, assume for now that the message to be encoded/sent is a concatenation of a source word sampled independently and identically (i.i.d.) according to P . A prefix-code is devised by a coding scheme $\mathbb{C} : \Omega \rightarrow \mathbb{B}^*$ and $\mathbb{C}(x)$ is termed as the code word for $x \in \Omega$. The aim is to minimise the expected code word length under P , that is, one wants to minimise $L_{\mathbb{C},P} = \mathbf{E}(\ell(\mathbb{C}(X))) = \sum_{x \in \Omega} \ell(\mathbb{C}(x))P(x)$. Let $L_P^* = \min_{\mathbb{C}} \{L_{\mathbb{C},P}\}$ be the minimal expected code word length. The following theorem due to C.E. Shannon relates L_P^* with $H(P)$ and asserts that the L_P^* is about the same as the entropy $H(P)$.

Theorem 9 (Shannon coding theorem). *Let L_p^* and P as above. $H(P)$ is the entropy of P , then*

$$H(P) \leq L_p^* \leq H(P) + 1$$

In this regard, for each source word $x \in \Omega$ the quantity $-\log P(x)$ can be thought as its actual information content. We note that the proof of the above theorem is (can be) constructive, that is, when P is known we can construct a prefix-code that has the minimal expected code word length in theory and in practice many coding methods have been devised to approximate the actual information content of each source word. For example, Shannon-Fano coding guarantees that all code word lengths are within one bit of their theoretical ideal. However, Shannon-Fano coding is almost never used. Both arithmetic coding and Huffman coding supersede Shannon-Fano. We will present arithmetic coding in the subsequent chapter.

3 Small-Alphabet Sequence Compression Techniques

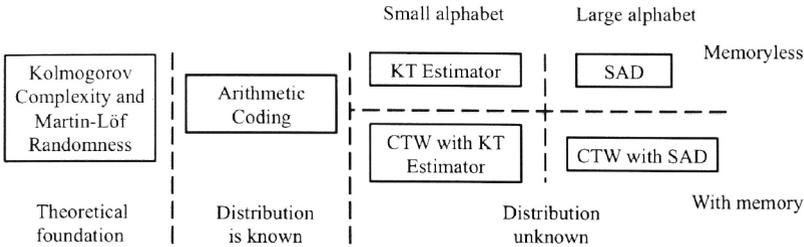


Figure 1: The structure of the discussion on compression in this thesis.

The big picture and a general setting. The discussion on compression theories and techniques is unfolded according to the thread shown in Figure 1. First, we present an overview of Algorithmic Information Theory (AIT) in this section. The purpose of surveying AIT is two-fold. First Kolmogorov complexity, a branch in AIT, can be considered as a theoretical foundation of compression. Secondly, Martin-Löf randomness, another branch in AIT, is an interesting notion, opposite to compressibility. We will introduce Martin-Löf randomness and present our observations on the set of random reals in the unit interval.

We then move to practical compression techniques. In this thesis, we are only concerned with entropy coding methods, which, loosely speaking, is a type of lossless coding to compress data by representing frequently occurring patterns with few bits and rarely occurring patterns with many bits. As such, we always assume that the to-be-compressed data is a sequence of source words from some finite alphabet \mathcal{X} . For the convenience of the discussion later in this thesis and particularly for discussing universal coding, we present here a general setting. In many cases, e.g., the one in arithmetic coding, the setup can be simplified. Assume that there is a generating distribution P on \mathcal{X}^∞ and $P(x_{1:n})$ is the probability of the cylinder set of $x_{1:n}$. In the compression community, P is often termed as a source or a model. We will use these two terms interchangeably. For a fixed n , P induces a probability distribution $P_n(x_{1:n}) = P(x_{1:n})$ on the finite set \mathcal{X}^n . Following P , the induced P_n is also called a source/model (on \mathcal{X}^n). One wants to devise a coding scheme $C_n: \mathcal{X}^n \rightarrow \mathbb{B}^*$ that generates a prefix-code for \mathcal{X}^n , i.e. the co-domain of C_n needs to be a prefix-code. This kind of coding scheme is often termed as Block to Variable (BV) code because the source words in \mathcal{X}^n are of the same length n and the source codes have variable lengths. In this thesis, we only discuss BV codes. The number n is called the delay of the code in some compression literatures. Φ_n is used to denote the set of all possible coding schemes on \mathcal{X}^n .

A performance measure is needed to evaluate the performance of a coding scheme. If P is known, $-\log P(x_{1:n})$ is the optimal code word length for $x_{1:n}$ and any bits needed beyond $-\log P(x_{1:n})$ is considered to be redundant. A common performance measure

used in compression is called expected redundancy and loosely speaking, it is the difference between the expected code word length $\mathbf{E}_{P_n}(\ell(\mathcal{C}_n(x_{1:n})))$ and the optimal expected code length given by the entropy $H(P_n)$, where the expectation \mathbf{E}_{P_n} is taken under P_n .

We start with introducing arithmetic coding as an example of a compression technique when the distribution P is known. In practice, however, the generating distribution P is often unknown but can be assumed to be in a class/set of distributions. One wants to devise a coding scheme that is optimal in some weaker sense. We will introduce the notion of universal coding and investigate the notion of universality in terms of minimax redundancy. This domain can be further divided into four different sub-domains along two different dimensions. One dimension is on the characteristic of P , whether it is memoryless or with finite memory. We will introduce these two terms in this section and will survey both the KT estimator as an example of a coding scheme for memoryless generating distributions and the Context Tree Weighting (CTW) algorithm with a KT estimator as an example for dealing with generating distributions with finite memory. The other dimension we consider is on the size of the alphabet \mathcal{X} . The two methods mentioned above are particularly good for small alphabet, however, are not suitable for large alphabet. We will introduce Hutter's Sparse Adaptive Dirichlet (SAD) coding scheme as an example of a coding scheme designed for large alphabet in the memoryless case. We have also combined the SAD with the CTW algorithm, which will be used as an example for dealing with sources with finite memory over a large alphabet. SAD is introduced in Section 4 and the CTW with SAD is in Section 6.

3.1 Kolmogorov Complexity and Martin-Lof Randomness

A brief introduction. Algorithmic Complexity (AC) or Kolmogorov Complexity is a sub-field of AIT that lays the theoretical foundation of compression, which, loosely speaking, concerns the ultimate compressed version of an object and measures the (in)compressibility of an object. It was actually Solomonoff who first formed the basic ideas about algorithmic complexity and proved the invariance theorem in his long journal paper [Sol64], but it is a tradition to talk about 'Kolmogorov complexity' instead of 'Solomonoff complexity'.

Imagine we want to describe a certain object using a binary string; intuitively the shorter the description the simpler the object. For example, the string $x=0^{1000}$ seems simpler than string $y=01011101010101101010101$ even though the former is much longer than y . This is because we can easily describe x as 'one thousand zeros', whereas there is hardly any shorter description for y than literally writing it down. This is the main idea of Kolmogorov complexity, which measures the ultimate information content in an object. A big issue is that the length of a description depends on the choice of the language used to describe objects. It can be hard to describe a certain thing in one language, but very easy in another. For instance, before coffee was introduced to China there wasn't a word for it, and thus describing 'coffee' in Chinese was much harder than describing it in English. We therefore would like something fair among different languages, i.e. the complexity of a certain object should be invariant with respect to different languages in which we use to describe it.

AIT has its roots in probability theory and information theory. Information theory concerns the expected average information content of a set of objects over which there

is a probability distribution, whereas Kolmogorov complexity looks at the information content of an individual object.

Kolmogorov complexity is naturally connected with another philosophical notion – randomness. Generally speaking, we would intuitively expect random things to be hard to describe and, therefore hard to compress. There has been a long-standing debate as to whether there exist true randomness in the universe or in nature. Some have argued that everything is predetermined; and that seemingly random processes are merely products of our ignorance. By contrast, others have suggested that the world is objectively indeterministic. If we push the second viewpoint to the extreme and assume that there is no pattern at all in nature, then scientists would not be able to compress the observations and thus would not be able to learn (find models/theories/laws). Consequently nothing would be predictable. Philosophically there are two camps as to this issue: one is known as determinism and the other indeterminism.

If we turn our attention from nature to mathematics, we will see that back to the time of Hilbert, mathematicians regard mathematics as absolute truth. Hilbert tried hard to formalise everything in mathematics into a small set of axioms and also proposed that mathematics is so precise that we can have some external machines to check the validity of our proofs. However, some work by Chaitin, e.g. in [GOO63, Cha86], showed a negative result that there exist infinitely many mathematical facts that cannot be effectively compressed into a finite set of axioms. The proof is based on the famous halting probability.

As well as providing philosophical insights, AIT has many applications. For example, Solomonoff’s universal prior ξ [Sol64, Sol75] for induction and Hutter’s universal artificial intelligence [Hut05]. One of the most interesting application is the universal similarity metric [LCL+03], which measures the similarity between string x and y as the length of the shortest program that computes x from y (i.e. $K(x|y)$). By proper normalisation and symmetrisation, this idea yields a universal similarity metric.

3.1.1 Plain Kolmogorov Complexity

The plain Kolmogorov complexity of an object is defined in terms of the information quantity that is required to losslessly describe it, i.e. one should be able to restore the full object just from this description. Among all possible descriptions we choose the shortest one. Formally, the plain Kolmogorov complexity is defined as following [LV08].

Definition 10 ((plain) Kolmogorov complexity). Let x, y, p be finite binary strings. Any partial recursive function $\phi: \mathbb{B}^* \rightarrow \mathbb{B}^*$, together with p and y , such that $\phi(\langle y, p \rangle) = x$, is a description of x given y . The (plain) complexity C_ϕ of x conditioned on y with respect to ϕ is defined by

$$C_\phi(x|y) = \min_p \{ \ell(p) : \phi(\langle y, p \rangle) = x \},$$

where $\ell(p)$ is the length of a program p , and $C_\phi(x|y) = \infty$ if there are no such p . We call p a program to compute x by ϕ , given y .

The requirement of ϕ to be (partial) recursive is natural because we want to reconstruct x from its description p with the help of y . However, the above definition suggests

that the complexity of x given y depends on the partial recursive function ϕ and for different ϕ the complexity of x varies. One would reasonably regard the complexity of a string as its intrinsic property that is independent of the partial recursive function. The following theorem [Gács07, LV08] solves this problem to some degree.

Theorem 11. *There is a universal partial recursive function ϕ_0 such that for any partial recursive function ϕ ,*

$$C_{\phi_0}(x) \leq C_{\phi}(x) + c$$

or simply $C_{\phi_0}(x) \stackrel{+}{\leq} C_{\phi}(x)$, where c is a constant that is independent with x .

That is, the Kolmogorov complexity of any string x with respect to ϕ_0 is no longer than that with respect to any other partial recursive functions up to some constant c that is independent of x . Such a function ϕ_0 is normally termed as an additively optimal universal partial recursive function. As such, fixing such an additively optimal universal ϕ_0 , the subscript of C is discarded and define the (plain) Kolmogorov complexity as $C(x|y) = C_{\phi_0}(x|y)$. The unconditional complexity is defined as $C(x) = C(x|\epsilon)$ where ϵ is the empty string.

Complexity and incompressibility. We can consider p as a compressed version of x , and it is easy to see the length of x is a trivial upper bound for the length of p up to some constant. More formally, there is a constant c such that for all x we have $C(x) \leq \ell(x) + c$; we write $C(x) \stackrel{+}{\leq} \ell(x)$. This holds because we can simply construct a Turing that outputs whatever it is given. On the other hand, by a simple counting argument one can show that most strings cannot be (highly) compressed, that is, most strings are incompressible. The following theorem, which is due to Kolmogorov, reveals this fact [LV08].

Theorem 12 (incompressibility theorem). *Let c be a positive integer. For each fixed y , every non-empty finite set $A \subset \mathbb{B}^*$ of cardinality $m(> 0)$ has at least $m(1 - 2^{-c}) + 1$ elements x with $C(x|y) \geq \log m - c$.*

If we set $c=1$, we can see nearly half of the objects in a finite set whose complexity is almost the logarithm of the size of the set. This tells us that there is no effective way to (highly) compress the majority of objects in a set. This very fact suggests that we should think otherwise: instead trying to compress everything, we should only aim at compressing things that occur often and leave the rest. This idea will be discussed in more detail in the subsequent sections.

The Achilles heel of AC. A problem with AC is that one still needs to choose an additively optimal universal partial recursive function ϕ_0 . Although C is defined on ϕ_0 and therefore additively yields a smaller complexity than any other choices, it is still not an objective measure that reflects the intrinsic complexity of an object in the following sense: for every string x there is an additively optimal recursive function ϕ_x such that $\phi_x(x) = c$ where c is a small constant. Given an additively optimal universal partial recursive function ϕ_0 and a string x , construct another additively optimal universal partial recursive function ϕ'_0 such that $\phi'_0(0) = x$ and $\phi'_0(1p) = \phi_0(p)$. It is clear that

$C_{\phi_0}(x) = 1$. As such, for any string x , there exists an additively optimal universal partial recursive function ϕ_x such that $C_{\phi_x}(x) = 1$. This issue may not show up when studying asymptotic results using Kolmogorov complexity, but may yield dramatically different results when we study the complexity of a specific finite string. Scientists are uncomfortable with such context dependence. Discussions about this issue have been extensive, see for example [Sta01]. It is clear that ϕ_x would not be *natural* as it is engineered towards a specific x , however, how to rigorously formalise the notion of *natural* is a difficult problem. Some researchers [RH11] argue that we should always use some predetermined and universally agreed-upon reference machines to start with before seeing the string we are studying. This argument does not solve the problem entirely as one can always claim that it is possible to accidentally choose a reference machine that yields low complexity for highly complex strings. [LV08] claim that they found a mathematically clean solution to this problem, however, this solution is not widely accepted due to some flaws in their argument. As a result, this problem still remains an open question in this field [Hut09].

Another Achilles heel of AC is that C is necessarily incomputable, moreover, no partial recursive function $\phi(x)$ defined on an infinite set of points can coincide with $C(x)$ over the whole of its domain of definition. Due to its in-computability, it is hard to put AC to practical use. One can, however, approximate $C(x)$ from above, i.e. there is a total recursive function $\psi(x,t)$, monotonically non-increasing in t , such that $\lim_{t \rightarrow \infty} \psi(x,t) = C(x)$.

3.1.2 Prefix Complexity

Problems with plain complexity. Although the idea of plain complexity is groundbreaking and the results regarding it are very fruitful, it suffers from some problems that make it less mathematically beautiful. The most obvious one, as we have already mentioned in the previous subsection, is that plain complexity is not subadditive. The reason for that is that the description itself is not self-delimiting; once two strings are joined together there is no way to tell them apart without additional information. But we would like to have $C(x,y) \leq C(x) + C(y)$, which coincides with our intuition. Unfortunately the plain complexity doesn't enjoy this property. These inconveniences call for another version of complexity, which has better mathematical properties.

Prefix functions and prefix complexity. The idea of prefix complexity was introduced in [Lev74, Gác74, Cha75b]. In order to overcome the aforementioned problems, partial recursive prefix functions are defined as follows:

Definition 13 (partial recursive prefix function). A partial recursive function $\phi: \mathbb{B}^* \rightarrow \mathbb{B}^*$ is a partial recursive prefix function if and only if its domain is a prefix-code.

Analogous to plain complexity, a theorem in [LV08] states that there exists an additively optimal universal partial recursive prefix function ϕ_0 such that for every partial recursive prefix function ϕ there is a constant c_ϕ such that $C_{\phi_0}(x|y) \leq C_\phi(x|y) + c_\phi$ for all $x, y \in \mathbb{N}$. As a result, we fix one additively optimal partial recursive prefix function ϕ_0 as a standard reference machine and we define $K(x|y) = C_{\phi_0}(x|y)$ as the prefix complexity of x . The unconditional prefix complexity is then defined as $K(x) = K(x|\epsilon)$.

3.1.3 Randomness and Halting Probability

There is a natural connection between randomness and Kolmogorov complexity; extensive research has been done in this area, for example, [Lev74, Cha75b, CN97, Sch73]. Martin Lőf randomness, first introduced in [ML66], is usually defined from a measure-theoretic perspective.

Definition 14 (Martin-Lőf random). Let $\{A_i\}_{i=1}^\infty$ be any effectively enumerable infinite sequence of recursively enumerable(r.e.) sets of intervals². Then ω is Martin-Lőf random iff for any such sequence $\{A_i\}_{i=1}^\infty$

$$\forall \{A_i\}_{i=1}^\infty : \forall i, \mathcal{L}(A_i) \leq 2^{-i}, \exists j : \omega \notin A_j$$

where \mathcal{L} is the uniform measure. Equivalent definitions are given by Solovay and Chaitin [Cha75b]. Their equivalence is proved in [GC89].

The basic idea of this definition is that if a sequence is random, then it cannot have any constructive distinguishing features; in other words, it cannot be expressed easily. For instance, a sequence $\omega = (01)^\infty$ is (intuitively) not random, we can easily construct a descending sequence of intervals such that every interval contains this sequence. Also this sequence is very easy to describe, namely ‘repeating 01s’.

A celebrated result is given by Chaitin who showed that for the prefix complexity $K(x)$, random sequences in Martin-Lőf sense with respect to the uniform measure are those sequences for which the complexity of each initial segment is at least its length (up to a constant).

Theorem 15. *An infinite binary sequence ω is Martin-Lőf random with respect to the uniform measure if and only if there is a constant c such that for all n , $K(\omega_{1:n}) \geq n - c$.*

This result explicitly tells us that the random sequences are the complex ones, and there is no effective way to compress them. Indeed, if there exist a c such that $K(\omega_{1:n}) < c$ for all n , then there must be a program p of finite length that generates ω and we will not call it a random sequence because the patterns in it can be described within $\ell(p)$ bits.

Numerous results have been discovered based on this theorem, and one of the most interesting ones is that the halting probability is random in Martin-Lőf sense. The halting probability is the real number $\Omega = \sum_{U(p) < \infty} 2^{-\ell(p)}$, and the sum is taken over all inputs p for which the reference machine U halts. Ω is also known as the number of wisdom. Unfortunately, although this number can be approximated from below, it is random with respect to the uniform measure and thus is maximally unknowable.

Mathematical facts can be random. Hilbert proposed that we could construct some external machines to automatically prove theorems. There are at least two attacks on the thesis of Hilbert. The first one is the well-known ‘halting problem’. We will be concerned with the second one, which is given by Chaitin [Cha71, Cha75a, Cha82, Cha86]. He constructed a sequence of mathematical facts, which are rigorously well defined and which are random, i.e. these facts can’t be compressed into a smaller set of axioms and they are irreducible mathematical information. To see this, we need the definition of Diophantine.

²For a self-contained introduction to Martin-Lőf random, please see [LV08].

Definition 16. (Diophantine) A set $A \subset \mathbb{Z}^n$ is Diophantine, if there exists a polynomial $P(a_1, a_2, \dots, a_n, x_1, x_2, \dots, x_m)$ with integer coefficients such that

$$A = \{(a_1, \dots, a_n) \in \mathbb{Z}^n \mid \exists (x_1, \dots, x_m) \in \mathbb{Z}^m, \text{ s.t. } P(a_1, \dots, a_n, x_1, \dots, x_m) = 0\}$$

There is a well-known theorem that says a set of tuples of positive integers is Diophantine if and only if it is recursively enumerable (r.e.). Now consider a computable increasing sequence of rational numbers $\{r_k\}_{k=1}^{\infty}$ with $\lim_{k \rightarrow \infty} r_k = \Omega$. Construct set A which contains tuple (n, k) such that n^{th} bit of r_k is 1. A is r.e. and thus Diophantine. Hence there exists a polynomial $P'(k, n, x_1, x_2, \dots, x_m)$ which equals 0 if and only if n^{th} bit of r_k is 1. Consequently, the set $D'_n = \{x \mid \exists y_1, y_2, \dots, y_m [P'(x, n, y_1, y_2, \dots, y_m) = 0]\}$ is infinite iff the n^{th} bit of the base-two expansion of Ω is a 1. Now one can easily see the equivalence between whether D'_n is infinite and whether the n^{th} number of Ω is 1. Note that the first is either true or false; however, these mathematical facts can't be compressed into a smaller set of axioms. They are irreducible mathematics information. This can be considered as a quantification of Gödel's incompleteness theorem.

3.1.4 A Topological View on Random Reals

We are interested in characterising how large the set of random sequences is. We study this from three different perspectives: set theoretic, measure theoretic and topological perspective. For simplicity, let $Rand$ denote the set of random infinite binary sequences and let $NonRand = \mathbb{B}^{\infty} - Rand$. The following definitions and remarks are helpful for our discussion.

Definitions. Intuitively, countable sets are smaller than uncountable ones. The set of rational numbers and the set of computable sequences are countable. The set of real numbers and the set of irrational numbers are both uncountable.

Definition 17 (dense). A set $S \subseteq [0, 1]$ is dense in the interval I if S has a nonempty intersection with every subinterval of I ; it is called dense if it is dense in $[0, 1]$.

Density is a topological notion which describes how a set S is distributed. The set of rational numbers in $[0, 1]$ is (everywhere) dense (in $[0, 1]$), but no finite set of real numbers in $[0, 1]$ is (everywhere) dense (in $[0, 1]$).

Definition 18 (nowhere dense). A set S is nowhere dense if it is not dense in any interval, that is, if every interval has a subinterval contained in the complement of S .

Intuitively a nowhere dense set is 'full of holes'. An alternative but equivalent definition is that a set S is nowhere dense if and only if its complement S^c contains a dense open set.

Definition 19 (first and second category). A set S is said to be of the first category or meagre if it can be represented as a countable union of nowhere dense sets. A subset of $[0, 1]$ that cannot be so represented is said to be of second category or non-meagre.

So sets of the second category are 'large' in the following sense: they are so 'large' that a countable union of nowhere dense cannot represent them. Another interpretation

is that sets of second category are denser than sets of first category. It follows that no interval in $[0,1]$ is of first category, the set of rational numbers is first category and Cantor set is of first category (even though it is uncountable).

Lebesgue measure is another way to measure the 'size' of a set from a sampling perspective: The larger the Lebesgue measure of a subset is, the larger the set is.

We also need to introduce two classical lemmas for our discussion, the proofs of which can be found in any mathematics analysis textbooks (e.g. [Zor04] and [RF10]).

Lemma 20. *The Cantor set, defined as*

$$C = [0,1] \setminus \bigcup_{m=1}^{\infty} \bigcup_{k=0}^{3^{m-1}-1} \left(\frac{3k+1}{3^m}, \frac{3k+2}{3^m} \right)$$

is uncountable and nowhere dense in $[0,1]$.

Lemma 21. *Any subset of a set of the first category is of the first category.*

Interesting facts about *Rand*. We study this from three different perspectives: set theoretic, measure theoretic and topological perspective. The known facts are summarised in Table 1.

| Item | Category | Measure | Cardinality | Density | Example |
|------|----------|---------|-------------|---------------|---------------------------------------|
| 1 | first | 1 | countable | dense | not exist |
| 2 | first | 1 | countable | nowhere dense | not exist |
| 3 | first | 1 | uncountable | dense | <i>Rand</i> |
| 4 | first | 1 | uncountable | nowhere dense | not exist |
| 5 | first | 0 | countable | dense | \mathbb{Q} |
| 6 | first | 0 | countable | nowhere dense | single point |
| 7 | first | 0 | uncountable | dense | exist, constructive ^[WH93] |
| 8 | first | 0 | uncountable | nowhere dense | Cantor set |
| 9 | second | 1 | countable | dense | not exist |
| 10 | second | 1 | countable | nowhere dense | not exist |
| 11 | second | 1 | uncountable | dense | $[0,1] - \mathbb{Q}$ |
| 12 | second | 1 | uncountable | nowhere dense | not exist |
| 13 | second | 0 | countable | dense | not exist |
| 14 | second | 0 | countable | nowhere dense | not exist |
| 15 | second | 0 | uncountable | dense | <i>NonRand</i> |
| 16 | second | 0 | uncountable | nowhere dense | not exist |

Table 1: Sixteen combinations of set size characteristic from set theoretic, measure theoretic and topological perspective. Examples are given in the last column.

We have the following observations [Cal02].

1. From a set-theoretic point of view, *Rand* and *NonRand* are both uncountable.

2. From a measure-theoretic point of view, *Rand* has measure 1 in the sense of Lebesgue, whereas *NonRand* has measure 0. This result shows that if sample from \mathbb{B}^∞ according to uniform distribution, one will get a random sequence with probability 1.
3. From a topological point of view, with the ordinary topology on $[0,1]$ both sets are everywhere dense in $[0,1]$, however, *Rand* is of first category, whereas *NonRand* is of second category. This result shows topologically the set of non-random sequences is much denser than the set of random ones.

Cardinality, (Lebesgue) measure and density are ways to measure the size of a set but from different perspectives. Intuitively speaking (though not true strictly), uncountable sets are larger than countable sets; the larger the measure of a set is, the larger the set is; sets of second category are larger than those of first category. The last observation is very counter intuitive especially when compared with the second observation. Even though the set of non-random sequences are much denser than random sequences in $[0,1]$, if one samples randomly then with probability 1 one will get a random one. The third observation doesn't say that the set of non-random sequences is necessarily larger than the set of random ones; it only says the former one is denser than the second one in some sense. One example to best illustrate this involves the Cantor set and the set of rational numbers. The Cantor set is uncountable and thus larger, in a certain sense, than the set of rational numbers which is only countable, but nevertheless the Cantor set is nowhere dense (defined later) and rational numbers are (everywhere) dense in real.

Remark 22. A few remarks on Table 1:

1. Items 1,2,9,10,12,13,14,16 do not exist because: countable sets must have measure zero; nowhere dense sets must be first category; and countable sets must be first category.
2. Item 4 does not exist. More generally, any set that has measure 1 cannot be nowhere dense. However, there exists a nowhere dense set in $[0,1]$ that has positive measure, e.g., the fat Cantor set.

Proposition 23. *The set of nonrandom sequences with respect to the Lebesgue measure is of second category, whereas the set of random ones is of first category [Cal02].*

We provide a different proof from the one in [Cal02].

Proof. Fix an effective enumeration of all rational numbers in $[0,1]$, $\mathbb{Q} = \{q_1, q_2, \dots\}$. Let $I_n = \bigcup_{j=1}^\infty (q_j - 2^{-n-j-1}, q_j + 2^{-n-j-1})$ and $A_i = \bigcap_{n=1}^i I_n$. Then $\{A_i\}$ is an infinite sequence of r.e. sets of intervals. Now we put $A = A_\infty$ and $B = [0,1] \setminus A$. We now want to show (1) every element in A is nonrandom; (2) B is of first category and thus A is of second category and (3) The set of random sequences is a subset of B and thus is of first category. For each A_i , we have

$$\mathcal{L}(A_i) \leq \mathcal{L}(I_i) \leq \sum_{j=1}^\infty \mathcal{L}((q_j - 2^{-i-j-1}, q_j + 2^{-i-j-1})) = \sum_{j=1}^\infty 2^{-i-j} = 2^{-i}$$

For any elements ω in A , we have $\omega \in A_i$ for all i . Hence (1) is true. Next we show B is of first category, observe $B = \bigcup_{n=1}^{\infty} I_n^c$ and I_n^c must be nowhere dense because its complement I_n is a dense open set, so B is a countable union of nowhere dense set and thus of first category. The fact that $A \cup B = [0,1]$ and $A \cap B = \emptyset$ implies A has to be of second category, because $[0,1]$ would be of first category otherwise. (3) is obvious because of the fact that any subset of a set of first category is still of first category. This completes our proof. \square

3.2 Stationary Memoryless Source

Following the setup at the beginning of this section, consider a non-empty alphabet \mathcal{X} with a source P and the induced distributions P_n on \mathcal{X}^n , if $P_n(x_{1:n}) = \prod_{i=1}^n P_1(x_i)$ for all n , then the source P is called a stationary memoryless source and is identified by the distribution P_1 on \mathcal{X} . The term ‘memoryless’ indicates that the conditional probability of the n^{th} symbol given the previous symbols $P(x_n | x_{<n})$ is equal to $P_1(x_n)$ and thus independent of what has been observed in the past. In this subsection, we only consider stationary memoryless source and in this subsection only, we drop the subscript ‘1’ in P_1 and call it the source of the data. For example, we write ‘a data sequence $x_{1:n} \in \mathbb{B}^n$ generated from a Bernoulli distribution’ to mean $x_{1:n}$ is sampled from a stationary memoryless source that is identified with a Bernoulli distribution and $x_i \sim \text{Bern}(\theta)$ for all $i \in \{1, 2, \dots, n\}$.

We first discuss the simplest case where the source P is known before turning to a more realistic situation where we don’t know the source in advance, but can be assumed to be in some general class of sources.

3.2.1 Arithmetic Coding

According to Amir Said [Say03], arithmetic coding is a coding technique that is able to work most efficiently in the largest number of circumstances and purposes and stands out in terms of elegance, effectiveness and versatility. He also lists some of the most desirable features of arithmetic coding.

The basic idea of arithmetic coding is that we fit any sequence into a subinterval of the interval $[0,1]$. The Elias algorithm provided the foundation of all arithmetic codes, an early description of which can be found in [Jel68]. The principle can be traced back to Shannon [Sha48].

Arithmetic coding process. Formally, given an input string $x_{1:n}$, the arithmetic coding process yields a sequence of nested intervals $\{[\alpha_i, \beta_i]\}_{i=1}^n$ where the sequence comes from a stationary memoryless data source with non-empty finite alphabet \mathcal{X} . Without loss of generality, it is assumed that $\mathcal{X} = \{0, 1, 2, \dots, M-1\}$. The probability distribution $p(m) = P(x_i = m)$ with $m \in \mathcal{X}$ and $i = 1, 2, \dots, n$ is assumed to be known in advance. We define $c(m)$ to be the cumulative distribution, i.e. $c(m) = \sum_{j=0}^{m-1} p(j)$ with $m \in \{0, 1, 2, \dots, M\}$. Note that $c(0) \equiv 0$ and $c(M) \equiv 1$. α_k and β_k are real numbers with $0 \leq \alpha_{k-1} \leq \alpha_k \leq \beta_k \leq \beta_{k-1} \leq 1$. To describe arithmetic coding, it is useful to introduce a new notation

$$|b, l) = [\alpha, \beta], \text{ where } b = \alpha \text{ and } l = \beta - \alpha$$

In other words, b is the starting point of an interval whereas l is the length of the interval. The intervals used in arithmetic coding can be described by the set of recursive equations

$$|b_0, l_0\rangle = |0, 1\rangle \tag{2}$$

$$|b_i, l_i\rangle = |b_{i-1} + c(x_i)l_{i-1}, p(x_i)l_{i-1}\rangle, \quad i = 1, 2, \dots, n \tag{3}$$

This process, which was first described in [Jel68], keeps the properties $0 \leq b_i \leq b_{i+1}$ and $0 < l_{i+1} < l_i \leq 1$ and in doing so, we get a sequence of nested intervals. The final step of the coding process is to choose a code value for the sequence. Since given this coding scheme, each real number in the resulting interval $[\alpha_n, \beta_n)$ represents a code (under the known source P) for a string that begins with $x_{1:n}$, we are free to choose any real number from $[0, 1)$ as the code of $x_{1:n}$, provided that we (1) write the number of data symbols (i.e. n) in the compressed file, or (2) add a special symbol signalling the end of message. Naturally we want to choose a real number such that its representation is the shortest with respect to the alphabet, over which the codeword is to be stored or transmitted. For example, if the codeword is to be stored in a physical computer or transmitted over the Internet, one wants to find a real number within the resulting interval whose binary representation is the shortest (trimming off the infinitely many ending zeros).

Arithmetic decoding process. Let $r \in [0, 1)$ be the codeword for $x_{1:n}$. Note that the probability distribution and the length of the original word (i.e. n) is known to the decoder. The decoding process recovers the original word in the same procedure that they were coded and can be expressed using a set of recursive equations

$$\begin{aligned} r_1 &= r \\ x_i &= m, \text{ s.t. } c(m) \leq r_i < c(m+1) \quad i = 1, 2, \dots, n \\ r_{i+1} &= \frac{r_i - c(x_i)}{p(x_i)} \quad i = 1, 2, \dots, n-1 \end{aligned}$$

Generalising the arithmetic coding to non-i.i.d. models. Arithmetic coding can accommodate non-i.i.d models in a very natural way. We have assumed an i.i.d. model in the introduction to arithmetic coding, we now generalise it to non-i.i.d. models, e.g., sources with memory.

The memoryless model is a very simple model and it does not accurately reflect most real world sources, such as language, image, video, which are highly structured and have interesting dependencies between the different symbols in the sequence. Unfortunately the simple i.i.d. model does not account for that. In order to make arithmetic work for real world models, there is a need to the extend arithmetic coding to non-i.i.d models.

Formally, we have assumed that an input string $x_{1:n}$ comes from a memoryless data source with non-empty finite alphabet $\mathcal{X} = \{0, 1, 2, \dots, M-1\}$. The distribution $p(m) = P(x_i = m)$ is known. The joint probability thus can be written as

$$P(x_{1:n}) = \prod_{i=1}^n p(x_i)$$

Now consider an arbitrary known source model given by $Q(x_{1:n})$, which always can be factorised as (under weak regularity conditions)

$$Q(x_{1:n}) = \prod_{i=1}^n q_i(x_i | x_{<i})$$

where

$$q_i(x_i | x_{<i}) = \frac{Q(x_{1:i})}{Q(x_{<i})}$$

is the conditional probability of x_i given all the previous symbols. Granted that these conditional probabilities can be easily computed, there is trivial modification to the arithmetic coding for memoryless models: We define the cumulative distributions $c_i(m) = \sum_{j=0}^{m-1} q(j | x_{<i})$ with $m \in \{0, 1, 2, \dots, M\}$ and modify the recursive process defined in Equation (3) to the following

$$\begin{aligned} |b_0, l_0| &= |0, 1| \\ |b_i, l_i| &= b_{i-1} + c_i(x_i)l_{i-1}, q_i(x_i | x_{<i})l_{i-1}, \quad i = 1, 2, \dots, n \end{aligned}$$

where l_i remains to be the length of the interval. The rest of the process remains the same and the decoding process is modified accordingly.

Performance measure - Expected redundancy. A performance measure is needed to discuss the performance of arithmetic coding and any other coding methods. Many performance measures can serve this purpose from different perspectives, for example,

- **Total or average code word length of a coding scheme.** Formally, for a coding scheme \mathbb{C} and a data sequence $x_{1:n}$, the total (average) code word length is given by $\ell(\mathbb{C}(x_{1:n}))$ ($\ell(\mathbb{C}(x_{1:n}))/n$). It is clear that this measure is not suitable for comparing coding schemes for compressing data sequences generated from different sources, simply because the compressibility of the data sequences varies with the sources. However, this is particular useful when comparing compression techniques on some benchmark corpora and thus widely used in practice. In my experiments, this measure is used for comparing different compression methods.
- **Expected total or average code word of a coding scheme.** To avoid a coding scheme from being engineered towards a particular string $x_{1:n}$, expected total or average code word of a coding scheme can be used, where the expectation is taken with respect to P_n over all $x_{1:n}$.
- **(Expected) Redundancy.** This measure is commonly used in theoretical analysis and many results in the compression field are expressed in terms of redundancy. If a data sequence $x_{1:n}$ is sampled from some distribution P_n , then the theoretical ideal is to code it in $-\log P_n(x_{1:n})$ bits and a lower bound on the expected code word length is the entropy of the source $H(P_n)$. Any bits needed beyond $-\log P_n(x_{1:n})$ are considered to be redundant and the overhead is called redundancy for a particular data sequence $x_{1:n}$, that is, given a coding scheme \mathbb{C} , the individual redundancy is,

$$R(\mathbb{C}_n, P_n, x_{1:n}) = \ell(\mathbb{C}(x_{1:n})) + \log(P_n(x_{1:n}))$$

which takes both positive and negative values. Another reasonable measure is the expected average redundancy, that is,

$$R(\mathbb{C}_n, P_n) = \frac{\mathbf{E}_{P_n}(\ell(\mathbb{C}(x_{1:n}))) - H(P_n)}{n} \quad (4)$$

Shannon coding theorem asserts that this quantity is larger than zero, and one aims to devise a coding scheme that minimises this quantity. In particular, if

$$\lim_{n \rightarrow \infty} R(\mathbb{C}_n, P_n) = 0 \quad (5)$$

then a sequence of coding schemes $\{\mathbb{C}_n\}_{n=1}^\infty$ is called asymptotically optimal. One should note, however, that this optimality notation is rather weak, which will be discussed later in this thesis.

- **Other measures.** Many other measures are also used, for example, regret and perplexity. However, we don't use them in this thesis.

Optimality of arithmetic coding and its limitation. Arithmetic coding has been shown to achieve optimal compression performance when applied to a stationary memoryless source [Sai04], that is, let \mathbb{C}_a represent arithmetic coding, P be a stationary memoryless source over a non-empty finite alphabet \mathcal{X} , then

$$R(\mathbb{C}_a, P_n) = \frac{\mathbf{E}_{P_n}(\ell(\mathbb{C}_a(x_{1:n}))) - H(P_n)}{n} = \frac{\mathbf{E}_{P_n}(\ell(\mathbb{C}_a(x_{1:n})))}{n} - H(P)$$

tends to zero when $n \rightarrow \infty$. In fact, the coding redundancy for any $x_{1:n}$ is at most 2 bits. Despite of its optimality, we need to know a lot of information in advance, namely, the exact probability distribution that generates the sequence. In a real life problem, it is often too much to ask for. What if we don't know the probability distribution in advance? In the next subsection, we deal with the case where we don't know the exact distribution but instead we assume a generic class of distributions, which so large that the true one is (hopefully) in this class.

3.2.2 Universal Coding

Motivation and setup. Arithmetic coding is effective and optimal if we know the underlying distribution $\mu(x_{1:n})$ in advance and code the sequence according to it. Universal coding, on the other hand, deals with the situation where we know little about the underlying distribution and hope we can do nearly (by some reasonable measure) as well as if we knew it. An apt example is given in [Grü07]: consider compressing a sequence $x_{1:n}$, which is generated by a Bernoulli model with unknown parameter θ . Suppose we believe that this sequence is sampled from $Bern(\theta')$ and devise a coding scheme $\mathbb{C}_{\theta'}$ to code this sequence accordingly, that is, let n_0 and n_1 be the number of 0s and 1s in $x_{1:n}$, hence $n_0 + n_1 = n$, then $\mathbb{C}_{\theta'}$ will generate a code word of length $-n_1 \log \theta' - n_0 \log(1 - \theta')$ for $x_{1:n}$. If can be shown that the expected redundancy of $\mathbb{C}_{\theta'}$ for a sequence of length n is

$$R(\mathbb{C}_{\theta'}, Bern(\theta)) = KL(\theta || \theta')$$

where $KL(\theta||\theta') = \theta \log \frac{\theta}{\theta'} + (1-\theta) \log \frac{1-\theta}{1-\theta'}$ is the Kullback-Leibler (KL) divergence, which, though not a proper metric, measures the distance between θ and θ' in the sense that $KL(\theta||\theta') \geq 0$ and only equal to 0 when $\theta = \theta'$. For $\theta' \neq \theta$, the redundancy will not tend to zero, and thus $\mathbb{C}_{\theta'}$ is not optimal. The question is then whether there exists a coding scheme that regardless of the true distribution behaves nearly as well as arithmetic coding when the true distribution is known.

To formalise this question, suppose a data sequence $x_{1:n}$ is generated by a source P , which is unknown but can be assumed to be in a set of sources \mathcal{M} . The aim is to devise a coding scheme that is asymptotically optimal regardless of what the underlying distribution is in \mathcal{M} .

Indifference rule. Laplace, several hundred years ago, pondered the same question, but from a prediction perspective. He asked the following question, ‘What is the probability that the sun will rise tomorrow? (given it has always risen in the past)’. We can formally phrase this question as follows: Given a sequence of $x_{1:n}$ that is generated i.i.d from a $Bern(\theta)$ with unknown θ , what should be $P(x_{n+1} = 1|x_{1:n})$? Now that we have a class of models, namely the family of Bernoulli distributions, each member in this family is uniquely indexed by a $\theta \in \Theta = [0,1]$. Following a Bayesian approach, we express our prior belief on θ with the density function $\pi(\theta)$. To combine the information regarding our prior and our observed data, we use Bayes rule. We define the posterior distribution $\pi(\theta|x_{1:n})$ as

$$\pi(\theta|x_{1:n}) = \frac{P(x_{1:n}|\theta)\pi(\theta)}{\int_{\Theta} P(x_{1:n}|\theta')\pi(\theta')d\theta'} \quad (6)$$

This posterior distribution contains both sources of information that we have about the parameter, namely, our prior beliefs and the observed data. This posterior distribution readily provides changes to our prior belief brought about by the data. The predictive distribution is then given by

$$P(x_{n+1} = 1|x_{1:n}) = \int_{\Theta} P(x_{n+1} = 1|\theta)\pi(\theta|x_{1:n})d\theta \quad (7)$$

If we are indifferent among all possible models (θ), that is, to take a uniform prior distribution over Θ such that $\pi(\theta) = 1$ for all $\theta \in \Theta$, we end up with Laplace rule, that is $P(x_{n+1} = 1|x_{1:n}) = \frac{\#(1)+1}{n+2}$, where $\#(1)$ denotes the number of 1’s in $x_{1:n}$.

A-Priori distribution. We can easily show that this scheme is equivalent to predicting with the following Bayes mixture

$$P(x) = \int_{\Theta} \pi(\theta)P(x|\theta)d\theta \quad (8)$$

This can be interpreted as a mixture of all models ($P(x|\theta)$) weighted according to our prior belief over all possible $\theta \in \Theta$, expressed by $\pi(\theta)$. Assuming the true source is given by $Bern(\theta)$, the expected redundancy of a coding scheme \mathbb{C}_P based on the mixture $P(x)$ in Equation (8) is

$$R(\mathbb{C}_P, Bern(\theta)) = \frac{1}{n} \sum_{x \in \mathbb{B}^n} P(x|\theta) \log \frac{P(x|\theta)}{\int_{\Theta} \pi(\theta)P(x|\theta)d\theta}$$

A well-known theorem provides a bound for this [Grü07]

Theorem 24. *Under weak regularity conditions (see [Grü07]), the following bound holds*

$$\begin{aligned} nR(\mathbb{C}_P, \text{Bern}(\theta)) &= \sum_{x \in \mathbb{B}^n} P(x|\theta) \log \frac{P(x|\theta)}{\int_{\Theta} \pi(\theta) P(x|\theta) d\theta} \\ &\leq \frac{1}{2} \log \frac{n}{2\pi} + \log \pi(\theta)^{-1} + \frac{1}{2} \log(\det \bar{j}(\theta)) + o(1) \end{aligned}$$

where $\det \bar{j}(\theta)$ is the average Fisher information.

For the single-parameter Bernoulli distribution, $\det \bar{j}(\theta) = 1/\theta(1-\theta) = O(1)$ for $\theta \neq 0, 1$. Different versions – weaker than this theorem in some sense, stronger in other senses – can be found in [Bal97, KR95, Sch78, Jef61]. The total redundancy $nR(\mathbb{C}_P, \text{Bern}(\theta))$ grows only logarithmically with n . If $\pi(\theta) > 0$, that is, initially the true model is not eliminated from the set of possible models, then a coding scheme based on P in Equation (8) is asymptotically optimal. This begs the question of what prior distribution one should choose? To answer this question, we will discuss formally the notion of universality and then give a more precise performance measure for redundancy in the context of universal coding. The first results in universal coding are due to B.M. Fitingof [Fit66, Fit67]. There are many papers on this subject, e.g. [KT81] is a very good survey.

Minimax redundancy and universality. Recall the general setup at the beginning of this section, a data sequence is generated from a source P on \mathcal{X}^∞ . P induces a sequence of true distributions P_n on \mathcal{X}^n and $x_{1:n}$ is sampled according to P_n for all n . Denote a set of possible sources on \mathcal{X}^∞ , \mathcal{M} . Let \mathcal{M}_n be the induced distributions on \mathcal{X}^n . A series of block-to-variable coding scheme \mathbb{C}_n is devised on the sequence of sets \mathcal{X}^n , one on each set. Φ_n is used to denote the set of all possible coding schemes on \mathcal{X}^n .

Definition 25 (minimax redundancy). Given the above setting, the following quantity is called the minimax redundancy attainable on \mathcal{M}_n .

$$R_{\text{minimax}}(\Phi_n, \mathcal{M}_n) = \inf_{\mathbb{C}_n \in \Phi_n} \sup_{P_n \in \mathcal{M}_n} R(\mathbb{C}_n, P_n)$$

where $R(\mathbb{C}_n, P_n)$ is the expected average redundancy in Equation (4).

Universality of a coding scheme can be defined with respect to the minimax redundancy.

Definition 26 (universal coding scheme). There is a universal coding scheme for \mathcal{M} if

$$\lim_{n \rightarrow \infty} R_{\text{minimax}}(\Phi_n, \mathcal{M}_n) = 0$$

A sequence of coding schemes $\{\mathbb{C}_n\}_{n=1}^\infty$ is called asymptotically optimal if

$$\lim_{n \rightarrow \infty} \frac{\sup_{P_n \in \mathcal{M}_n} R(\mathbb{C}_n, P_n)}{R_{\text{minimax}}(\Phi_n, \mathcal{M}_n)} = 1 \quad (9)$$

That is, the worst case performance of $\{\mathbb{C}_n\}_{n=1}^\infty$ asymptotically gets close in ratio to the best possible coding scheme for sources in \mathcal{M} .

Remark 27. A few remarks:

1. The optimality notion in Equation (9) is stronger than that in Equation (5).
2. In this thesis, we will not discuss necessary and sufficient conditions for the existence of a universal coding scheme for a model class \mathcal{M} . A in-depth discussion can be found in [Dav73]. Also note there are many notions of universality and there is a hierarchy of universality; a detailed discussion can be found in Chapter 6 of [Grü07].

Performance of universal coding scheme for stationary memoryless sources.

Now we turn to the stationary memoryless sources and answer the question of what prior distribution one should choose. Let \mathcal{M} denote the set of all memoryless sources on \mathcal{X}^∞ where $|\mathcal{X}|$ is the size/cardinality of the non-empty finite alphabet \mathcal{X} . Recall that each source P in \mathcal{M} can be identified by a categorical distribution (generalised Bernoulli distribution) P_0 with vector parameter \mathbf{p} on \mathcal{X} where \mathbf{p} is in $|\mathcal{X}|$ -dimensional simplex $\mathcal{S}_{|\mathcal{X}|-1}$, that is, $\sum_i^{|\mathcal{X}|} p_i = 1$, $p_i \geq 0$ for all i , and $P_1(x_i|\mathbf{p}) = p_i$. [KT81] provides a minimax redundancy bound for \mathcal{M} .

Theorem 28. *Let \mathcal{M} be the set of all stationary memoryless sources on \mathcal{X} and Φ_n are all possible coding schemes on \mathcal{X}^n , then*

$$\lim_{n \rightarrow \infty} \frac{R_{\minimax}(\Phi_n, \mathcal{M})}{\frac{|\mathcal{X}|-1}{2n} \log n} = 1$$

Remark 29. There are three remarks:

1. The total minimax redundancy is $\frac{|\mathcal{X}|-1}{2} \log n$ which is linear in the size of \mathcal{X} and grows logarithmically with n .
2. Surprisingly, the cost for universality is very small. The redundancy decreases as $O(n^{-1} \log n)$ when a source is unknown.
3. An asymptotically optimal code of a source word can be achieved by any Bayes mixture distribution

$$P(x) = \int \dots \int_{\mathcal{S}_{|\mathcal{X}|-1}} \pi(\mathbf{p}) P_0(x|\mathbf{p}) d\mathbf{p}$$

with $\pi(\mathbf{p}) > 0$ (for the true \mathbf{p}). However, we will review a particular one with a Dirichlet distribution with parameter $|\mathcal{X}|$ -dimensional vector $\frac{1}{2}$ as the prior distribution. This particular estimator is called Krichevsky-Trofimov (KT) estimator, which has a distinguishing property that the parameter redundancy can be uniformly bounded. It is impossible to prove a uniform bound for the Laplace estimator. A detailed discussion of Dirichlet distribution is presented in Section 5.3.1.

4. However this bound is only nice in the asymptotic sense, i.e. $|\mathcal{X}|$ is small compared with the length of data to be encoded. However, each element in $|\mathcal{X}|$ contributes $\frac{1}{2} \log n$ redundancy, so the total redundancy grows linearly with $|\mathcal{X}|$. This can be bad: Consider the English text where alphabet is taken to be the set of all English words. It motivate us to consider more sophisticated methods for large-alphabet compression, which we will discuss in Section 4.

3.2.3 Krichevsky-Trofimov Estimator

We begin with a short review of the well-known KT estimator [KT81] for Bernoulli distributions. Let $x_{1:n}$ be a binary string of a zeros and b ones, and x_{n+1} be the next symbol, then the KT estimated probability of x_{n+1} is given by

$$P_{kt}(\epsilon) = 1 \quad (10)$$

$$P_{kt}(x_{n+1}=0|x_{1:n}) = \frac{a+1/2}{a+b+1} \quad (11)$$

$$P_{kt}(x_{n+1}=1|x_{1:n}) = 1 - P_{kt}(x_{n+1}=0|x_{1:n}) \quad (12)$$

From (10) - (12), we can estimate the block probability of a string $x_{1:n}$ using

$$P_{kt}(x_{1:n}) = P_{kt}(x_1|\epsilon)P_{kt}(x_2|x_1)\dots P_{kt}(x_n|x_{<n})$$

A useful property of the KT estimator is that it only depends on the number of zeros and ones in a string, and not on their order. Let $s = 0^a 1^b$ denote a string with a zeros and b ones, and write $P_{kt}(a,b)$ to denote $P_{kt}(s)$, then we can incrementally calculate this quantity as follows:

$$P_{kt}(a+1,b) = \frac{a+1/2}{a+b+1} P_{kt}(a,b)$$

$$P_{kt}(a,b+1) = \frac{b+1/2}{a+b+1} P_{kt}(a,b)$$

with $P_{kt}(0,0)=1$. The KT estimator has a useful theoretical property shown in [WST95] that, for any sequences generated by $Bern(\theta)$ with $\theta \in [0,1]$ for all $a+b \geq 1$, the redundancy is uniformly bounded because

$$nR(\mathbb{C}_{KT}, Bern(\theta)) = \log \frac{(1-\theta)^a \theta^b}{P_{kt}(a,b)} \leq \frac{1}{2} \log(a+b) + 1 = \frac{1}{2} \log(n) + 1 \quad (13)$$

where \mathbb{C}_{KT} is the coding scheme induced by the KT estimator. Note that this is consistent with the optimal bound for a universal code. The alphabet here is the binary set \mathbb{B} of size 2.

3.3 Stationary Source with Finite Memory

Often the probability of the next symbol depends on the previous observed symbols, that is, symbols are not independent of each other. Take English text for example, given a word ‘San’ it is more likely that it is followed by ‘Francisco’ than by ‘web’. Hence, we are interested in modelling the conditional probability of a certain symbol given its context of length n . This is known as the Markov model, and in the text modelling literature also known as n-grams model. In this section, we present a method that deals with variable order Markov models.

3.3.1 Binary Context Tree Weighing Algorithm

The CTW algorithm [WST95, WST97] is a theoretically well-motivated and efficient online binary sequence prediction algorithm. It uses Bayesian model averaging that computes a mixture over all prediction suffix trees [RST96] up to a certain depth, with greater prior weight given to simpler models.

A context tree of depth D is a perfect binary tree of depth D , with the left edges labelled 1 and the right edges labelled 0. Let $s \in \mathbb{B}^{\leq D} = \bigcup_{t=0}^D \mathbb{B}^t$ be a node in the context tree and suppose $x_{1:n}$ is the sequence seen so far and $[x_{1:n}]_s$ is the sequence of bits in $x_{1:n}$ that follow the context s . For example $[00\underline{1}100\underline{0}101]_{|00} = 101$. The counts a_s and b_s corresponding to the number of zeros and ones in $[x_{1:n}]_s$ are stored at each node s and are updated as bits in the input sequence are observed. The KT estimate is calculated at each node s based on the attached counts a_s and b_s . Additionally, we introduce a weighted probability for each node s , which can be recursively calculated by

$$P_w^s(x_{1:n}) = \begin{cases} P_{kt}([x_{1:n}]_s) & \text{if } s \text{ is a leaf node} \\ \frac{1}{2}P_{kt}([x_{1:n}]_s) + \frac{1}{2}P_w^{0s}([x_{1:n}]_{0s})P_w^{1s}([x_{1:n}]_{1s}) & \text{otherwise} \end{cases} \quad (14)$$

The joint probability for the input sequence is then given by the weighted probability $P_{CTW}(x_{1:n}) := P_w^\epsilon(x_{1:n})$ at the root node.

A weighted context tree of a fixed depth D is constructed as follows. Provided with the input sequence $x_{1:n}$, it is first necessary to reserve the initial D bits of the sequence to be used as an initial context (or in practice we pad D zeros in front of $x_{1:n}$). As bits in the input sequence are observed we find the leaf node s of the context tree that corresponds to the current context. The counts a_s and b_s , corresponding to the number of zeros and ones in $[x_{1:n}]_s$, in this node are updated based on the value of the bit observed, then the KT estimate is calculated and a weighted probability is assigned according to Equation (14). We then traverse the tree towards the root, at each node m with left child $0m$ and right child $1m$ updating the counts a_m and b_m as well as the KT-estimate for the node. The new weighted probability is then calculated recursively according to Equation (14). The CTW update is complete after finishing this process for the root node. At this point the weighted probability at the root gives the joint probability for the input sequence observed so far. As the update process only involves a traversal from one leaf node to the root, it can be performed in time linear in D , the depth of the tree.

Prediction with CTW. Having described the process of constructing and updating a weighted context tree, we now consider the prediction aspect of the CTW algorithm. For a weighted context tree that has been updated with an input sequence $x_{1:n}$, the weighted probability at the root gives a probability for the sequence $x_{1:n}$. If $n < D$, then the probability for $x_{n+1} = 1$ is simply set to $\frac{1}{2}$. This is necessary as the input sequence seen so far is insufficient for providing an initial context. To make a prediction for x_{n+1} when $n \geq D$ we use conditional probabilities. Specifically

$$P_{CTW}(x_{n+1} = 1 | x_{1:n}) = \frac{P_w^\epsilon(x_{1:n}1)}{P_w^\epsilon(x_{1:n})}$$

where $x_{1:n}1$ is the sequence $x_{1:n}$ followed by a bit 1, and P_w^ϵ is the weighted probability at the root of the weighted context tree. As $P_w^\epsilon(x_{1:n})$ is already calculated and stored in

the root of the tree, for the purposes of prediction it only remains to calculate $P_w^e(x_{1:n}1)$. This can be done by performing a ‘dummy update’ of the context tree. To perform a dummy update we start at the leaf node of the context tree that corresponds to the current context. Pretending $x_{n+1} = 1$ was observed, the weighted probability of the node is calculated, but is not used to update the node. As this is a leaf node this only requires calculating an updated KT-estimate, and is not dependent on any other nodes in the tree. This new weighted probability is not used to update the tree, but merely propagated back up the tree. At each interior node we also calculate (without updating the node) the KT estimate as if $x_{n+1} = 1$ was observed. Using this value, combined with the weighted probability propagated from the child, allows the new weighted probability for the current node to be calculated. In this manner when the root of the context tree is reached we have a value for $P_w^e(x_{1:n}1)$, without making any changes to the tree. Conditioning on $x_{1:n}$ now gives a probability to be used for prediction. After the actual value of x_{n+1} is observed we update the tree, and the cycle of prediction/observation continues. As with the CTW update process, this propagation up through the tree has computation time linear in the depth of the context tree.

3.3.2 Model Class and Redundancy

Bounded memory tree source. Before we discuss the theoretical insights into the CTW algorithm, we need to introduce a number of concepts. Our discussion in this section is largely drawn from [WST95].

A binary tree source can be described by the notion of a suffix set \mathcal{S} , which is a collection of binary strings s that satisfy the following

- **Properness.** No string in \mathcal{S} is a suffix of any other string in \mathcal{S} .
- **Completeness.** each semi-infinite sequence $\dots x_{t-2}x_{t-1}x_t$ has a suffix that belongs to \mathcal{S} .

A bounded memory tree source of depth D can be identified by a suffix set \mathcal{S} with $\ell(s) \leq D$ for all $s \in \mathcal{S}$. Each suffix $s \in \mathcal{S}$ corresponds to a parameter θ_s , which takes a value in $[0,1]$ and specifies a Bernoulli distribution over \mathbb{B} . The parameter set related to a bounded memory tree source is $\Theta_{\mathcal{S}} := \{\theta_s | s \in \mathcal{S}\}$. The suffix function $\beta_{\mathcal{S}}(\cdot)$ maps semi-infinite sequences onto their unique suffix $s \in \mathcal{S}$. The actual next-symbol probabilities for a bounded memory tree source with suffix set \mathcal{S} and parameter set $\Theta_{\mathcal{S}}$ are

$$P(x_t = 1 | x_{<t}, \mathcal{S}, \Theta_{\mathcal{S}}) = \theta_{\beta_{\mathcal{S}}(x_{<t})}$$

All tree sources with the same suffix set are said to have the same model. The set of all tree models having memory not larger than D is called the model class C_D . The cost of a model \mathcal{S} with respect to model class C_D is defined as

$$\Gamma_D(\mathcal{S}) := |\mathcal{S}| - 1 + |\{s | s \in \mathcal{S}, \ell(s) \neq D\}|$$

Note that tree sources with large depth have greater cost.

Model class that CTW mixes over. The joint probability for the input sequence given by the weighted probability $P_{CTW}(x_{1:n}) := P_w^e(x_{1:n})$ at the root node of a context

tree of depth D can be rewritten as

$$P_{CTW}(x_{1:n}) = \sum_{\mathcal{U} \in C_D} 2^{-\Gamma_D(\mathcal{U})} \prod_{u \in \mathcal{U}} P_{KT}(a_u, b_u)$$

with

$$\sum_{\mathcal{U} \in C_D} 2^{-\Gamma_D(\mathcal{U})} = 1$$

The probability given by the CTW algorithm is essentially a weighted mixture over all tree sources that have memory not larger than D . By a simple counting argument, the number of tree sources with memory not larger than D is double exponential in D . As such, the CTW algorithm is theoretically well-motivated, at the same time that CTW provides an efficient way to calculate a mixture over such a large model class also makes it practically interesting.

Redundancy. Consider a binary string $x_{1:n}$ that is sampled from a bounded memory tree source $\mathcal{S} \in C_D$ ($D < n$) with the actual probability $P_{\mathcal{S}}(x_{1:n})$ and $P_{CTW}(x_{1:n})$ is the estimated probability given by the CTW algorithm. The individual redundancy of $x_{1:n}$, $R(x_{1:n}, P_{CTW}(x_{1:n}), P_{\mathcal{S}}(x_{1:n}))$ can be uniformly (i.e., for all $x_{1:n} \in \mathbb{B}^n$) bounded by [WST95]

$$R(x_{1:n}, P_{CTW}(x_{1:n}), P_{\mathcal{S}}(x_{1:n})) := -\log P_{CTW}(x_{1:n}) + \log P_{\mathcal{S}}(x_{1:n}) \leq \Gamma_D(\mathcal{S}) + |\mathcal{S}| \left(\frac{1}{2} \log \frac{n}{|\mathcal{S}|} + 1 \right)$$

which grows logarithmically with n . The first term $\Gamma_D(\mathcal{S})$ is called the model redundancy, which is the cost for ‘locating’ the true model \mathcal{S} , while the second term $|\mathcal{S}| \left(\frac{1}{2} \log \frac{n}{|\mathcal{S}|} + 1 \right)$ is called the parameter redundancy that stems from the redundancy of the KT estimators for $\theta_{\mathcal{S}} \in \Theta_{\mathcal{S}}$.

4 Large-Alphabet Sequence Compression Techniques

In this section, we look at compression techniques applied directly to large-alphabet sequences, as opposed to binary (or binarised) sequences. We first discuss the challenge that compression has when directly dealing with large alphabet, then we present Veness and Hutter’s sparse sequential Dirichlet coding [VH12] and Hutter’s Sparse Adaptive Dirichlet-multinomial (SAD) coding [Hut13]. Both adaptively allocate/reserve probability for unseen data, however the latter is more theoretically refined. In Section 6, we combine SAD with the CTW and apply CTW-SAD to compress sources with finite memory. One should note that there are many other methods that can be used for compressing sources with large alphabet, for instance, a method [GWT10] based on hierarchical PYP, a method [OSI02] that combines properties of PPM (Prediction by Partial Match [CW84]), CTW, and Kneser-Ney; some of them are discussed in Section 5.

4.1 Sparse Sequential Dirichlet Coding

Given a sequence $x_{1:n} = x_1 \dots x_n$ from some large alphabet \mathcal{X} , the sequential Dirichlet estimator with hyperparameter $\alpha = (\alpha_1, \alpha_2, \dots, \alpha_{|\mathcal{X}|})$ yields the following joint probability

$$\hat{P}_\alpha^{\mathcal{X}}(x_{1:n}) = \prod_{i=1}^n \frac{\#_i(x_i) + \alpha_i}{(i-1) + \sum_i \alpha_i}$$

where $\#_i(x_i)$ is a shorthand for $\#_i(x_i|x_{<i})$ the number of occurrence of x_i in $x_{<i}$. This distribution is called Dirichlet-multinomial distribution, which has been illustrated by the Polya urn scheme in Section 5.3.1. Without any additional knowledge about the structure of \mathcal{X} , it is reasonable to choose an indifferent α such that $\alpha_i = \lambda$ for all i and the resulting joint probability becomes

$$\hat{P}_\lambda^{\mathcal{X}}(x_{1:n}) = \prod_{i=1}^n \frac{\#_i(x_i) + \lambda}{(i-1) + \lambda|\mathcal{X}|}$$

Traditional choices of λ are Laplace indifference rule $\lambda = 1$, the KT estimator $\lambda = \frac{1}{2}$, maximum likelihood $\lambda = 0$ and Perks’ prior $\lambda = \frac{1}{|\mathcal{X}|}$. The first three consider individual elements in \mathcal{X} and pretend that they have appeared λ times a priori. Perks takes a more collective perspective and pretends that all symbols in \mathcal{X} have collectively appeared once a-priori. We study $\hat{P}_\lambda^{\mathcal{X}}$ for some specific λ ’s, for example, the KT estimator ($\lambda = \frac{1}{2}$), the redundancy of which can be uniformly bounded

$$-\log_2 \frac{\hat{P}_{\frac{1}{2}}^{\mathcal{X}}(x_{1:n})}{\mathbb{P}(x_{1:n})} \leq \frac{|\mathcal{X}| - 1}{2} \log_2 n + |\mathcal{X}| - 1$$

This coding technique performs well only if $|\mathcal{X}|$ is intrinsically small or good compression performance is only required asymptotically, where $|\mathcal{X}|$ is fixed while n tends to infinity. However, all the aforementioned choices of λ are problematic for large alphabet \mathcal{X} , which

can be seen from the previous bound for the KT estimator as an example: each element in \mathcal{X} contribute $\frac{1}{2}\log_2 n$ to the total redundancy, which resultantly grows linearly with the size of the alphabet. There is another deeper reason why these choices necessarily perform poorly in the context of large-alphabet sequences. To see this, let \mathcal{A}^n be the unique symbols that have appeared up to time step n , that is $\mathcal{A}^n = \{x_1, x_2, \dots, x_n\}$. Note that the repeating elements get automatically removed in a set. Let $\mathcal{U}^n = \mathcal{X} - \mathcal{A}^n$ and we have

$$\hat{P}_\lambda^{\mathcal{X}}(x_{n+1}|x_{1:n}) = \frac{\#_{n+1}(x_{n+1}) + \lambda}{n + \lambda|\mathcal{X}|}$$

\mathcal{A}^n and \mathcal{U}^n partition the probability space into $\frac{n + \lambda|\mathcal{A}^n|}{n + \lambda|\mathcal{X}|}$ and $\frac{\lambda|\mathcal{U}^n|}{n + \lambda|\mathcal{X}|}$. The probability reserved for \mathcal{U}^n is linear in $|\mathcal{U}^n|$ and with a uniform distribution on \mathcal{U}^n , each unseen symbol has probability $\frac{\lambda}{n + \lambda|\mathcal{X}|}$. However, intuitively we would like the probability of each unseen symbol to be related to the size of \mathcal{A}^n (or \mathcal{U}^n).

Sparse sequential Dirichlet coding. Veness and Hutter [VH12] motivate their approach from a slightly different angle. Consider a sequence of symbols $x_{1:n}$ over some large alphabet \mathcal{X} . This sequence can be seen as being sampled from an unknown (and much smaller) subset $\mathcal{A} \subseteq \mathcal{X}$. If one knew $\mathcal{A} \subseteq \mathcal{X}$, one could code the sequence based on \mathcal{A} , however, we need to deal with the case where we only know \mathcal{X} and at the same time we want to minimise the regret, which is the difference of code length between coding $x_{1:n}$ without knowing \mathcal{A} and knowing \mathcal{A} in hindsight. The idea is that we code the current symbol x_{n+1} using a KT estimator but based on \mathcal{A}^n rather than \mathcal{X} and discount it by a factor $(1 - \beta_{n+1})$, and redistribute the remaining probability uniformly.

$$\xi(x_{n+1}|x_{1:n}) = \begin{cases} \beta_{n+1} \frac{1}{|\mathcal{U}^n|} & \text{if } \#_{n+1}(x_{n+1}) = 0 \\ (1 - \beta_{n+1}) \frac{\#_{n+1}(x_{n+1}) + \frac{1}{2}}{n + \frac{1}{2}|\mathcal{A}^n|} & \text{if } \#_{n+1}(x_{n+1}) > 0 \end{cases}$$

where $\beta_i = \frac{1}{i}$ for $i \in \mathbb{N}$. The parameter β_i controls the total probability reserved for unseen symbol and decays over time, which is based on a reasonable assumption that new symbols become less likely to occur over time. It turns out that the regret (comparing coding with ξ with employing KT estimators on \mathcal{A} without reference to the base alphabet) of this estimator can be bounded by the following

Theorem 30. *Given alphabets \mathcal{X} and \mathcal{A} such that $\mathcal{A} \subset \mathcal{X}$, for all $n \in \mathbb{N}$ and for all $x_{1:n} \in \mathcal{A}^n$, we have*

$$-\log_2 \xi(x_{1:n}) - (-\hat{P}_{\frac{1}{2}}^{\mathcal{A}}(x_{1:n})) \leq \log_2 n + |\mathcal{A}| \log_2 |\mathcal{X}|$$

which results in an overall redundancy bounded by

$$\frac{(|\mathcal{A}| + 1)}{2} \log_2 n + |\mathcal{A}| \log_2 |\mathcal{X}| + |\mathcal{A}| - 1 \quad (15)$$

4.2 Sparse Adaptive Dirichlet-Multinomial Coding

Hutter's model employs the idea of 'escape' symbol and 'escape' probability. The basic idea is simple: When estimating the conditional probability of the next bit, $\hat{P}(x_{n+1}|x_{1:n})$,

for symbols we have seen, we estimate the probability using $\hat{P}(x_{n+1}|x_{1:n}) = \frac{\#_{n+1}(x_{n+1})}{n+\beta_n}$. By adding β_n we effectively reserve probabilities for unseen symbols, and the total amount held out is $\frac{\beta_n}{n+\beta_n}$. We then redistribute this probability among the unseen symbols according to the distribution $\hat{P}(x_{n+1}|x_{1:n}) = \frac{\beta_n w_i^n}{n+\beta_n}$ such that $\sum_{j \in \mathcal{U}^n} w_j^n \leq 1$. For example, w_i^n can be taken as a uniform distribution $\frac{1}{|\mathcal{U}^n|}$. The basic idea is not novel, however, β_n has been treated carelessly. Hutter's main contribution is that β_n is optimised for minimising coding redundancy. He proposes that the optimal β_n should be taken as

$$\beta_n = \frac{|\mathcal{A}^n|}{\ln \frac{n+1}{|\mathcal{A}^n|}} \quad \text{for } n \geq 1 \quad (16)$$

for $n=0$ the value of β_0 does not matter, because this model will automatically reduce to the distribution w_i^0 at the beginning.

Remark 31. Note that β_n depends on $|\mathcal{A}^n|$ that can be intuitively motivated: if $|\mathcal{A}^n|$ is large (compared with $\ln n$), that means a lot of new symbols have appeared in the sequence so far and one would reasonably expect to frequently see new symbols following this momentum. Large $|\mathcal{A}^n|$ leads to large β_n , which means that more probability should be reserved, which corresponds to our intuition. Vice versa, small $|\mathcal{A}^n|$ indicates unlikelihood of seeing new symbols in the future, and hence the reserved probability should be small.

It is shown in [Hut13] that the redundancy of the above \hat{P} with β_n in Equation (16) is bounded by

$$CL(\mathcal{A}) - (|\mathcal{A}| - \frac{1}{2}) \log_2 |\mathcal{A}| + \sum_{j \in \mathcal{A}} \frac{1}{2} \ln n_j - \frac{1}{2} \ln n + |\mathcal{A}| \ln \ln \frac{en}{|\mathcal{A}|} + 0.56|\mathcal{A}| + 0.082 \quad (17)$$

where

$$CL(\mathcal{A}) := \sum_{t \in \text{New}} \ln(1/w_{x_{t+1}}^t)$$

A crude choice for the symbol weights is the uniform distribution $w_i^t = 1/|\mathcal{X}|$, which results in $CL(\mathcal{A}) = |\mathcal{A}| \ln |\mathcal{X}|$. Finer choices are discussed in [Hut13].

Now we compare the redundancies shown in Equation (13) and Equation (15), in which each symbol in $|\mathcal{A}|$ (or even $|\mathcal{X}|$) contributes $\frac{1}{2} \ln n + O(1)$ redundancy. Hutter's model improve upon this in the following significant ways. Now consider the asymptotics $n \rightarrow \infty$ in Equation (17). First, unseen symbols induce zero redundancy. Second, each symbol j in $|\mathcal{A}|$ that only appears finitely often induces only finite redundancy. Only symbols that appear infinitely often have asymptotic redundancy $\frac{1}{2} \ln n + O(1)$.

5 Text Modelling Techniques

Many linguistic applications, such as speech recognition, handwriting recognition, optical character recognition (OCR), and machine translation use probability language models, which help to get rid of ambiguities. To see that, consider two texts consisting of the same sonic features, ‘beautiful flower’ and ‘beautiful flour’. In the lack of comprehensive grammatical knowledge, a probability language model would reveal that $P(\text{beautiful flower}) > P(\text{beautiful flour})$ from a statistical perspective. In this section, we survey the techniques that are used to build such probability language models.

5.1 n-gram Models over Sparse Data

Ultimately we would like to model the (joint) probability distribution over sentences. This can be achieved by modelling the conditional probability of words and stitching them together to form the joint distribution over the entire sentence, that is, assuming a sentence contains l words

$$P(x_{1:l}) = \prod_{i=1}^l P(x_i | x_{<i})$$

The n-gram model aligns with this approach, however, it further assumes that the conditional probability given the entire history $P(x_i | x_{<i})$ can be simplified by considering only the most recent $n-1$ words. That is,

$$P(x_{1:l}) \approx \prod_{i=1}^l P(x_i | x_{\max\{i-n+1, 0\}:i-1})$$

The conditional probability of a word x_i only depends on its context of length $n-1$, which corresponds to a $n-1$ order Markov model. For notational simplicity, we write $P(x_i | x_{i-n+1:i-1}) \equiv P(x_i | x_{\max\{i-n+1, 0\}:i-1})$ and discuss the case where there is not enough context separately. The rationale behind this assumption is that the remote context has negligible influence on determining the probability of the next word. This assumption, though not universally true, is very sensible and goes a long way towards increasing the feasibility of such probability models. In this section, we look at two approaches that are based on n-grams models. The first one uses smoothing methods on the conditional probabilities. The second employs a Pitman-Yor process on n-gram model, resulting in the so-called hierarchical Pitman-Yor process.

Framework. For an n-gram model, the conditional probabilities $P(x_i | x_{i-n+1:i-1})$ are initially unknown and thus need to be learnt. Different from the compression techniques surveyed in this thesis, most text modelling techniques use a standalone training data set to build the model by learning the probabilities. The performance is then evaluated on a separate test data set that is independent of the training set. In this thesis, we use the average code word length as a performance measure.

5.2 Smoothing Techniques

The most straightforward way of estimating $P(x_i | x_{i-n+1:i-1})$ is to simply count how many times $x_{i-n+1:i}$ and $x_{i-n+1:i-1}$ occur in the training set respectively and take the

radio

$$\hat{P}_{ML}(x_i|x_{i-n+1:i-1}) = \frac{\#(x_{i-n+1:i})}{\#(x_{i-n+1:i-1})}$$

as an estimate for $P(x_i|x_{i-n+1:i-1})$. This is normally called the maximum likelihood estimate. In case of $\#(x_{i-n+1:i-1}) = 0$ (and hence $\#(x_{i-n+1:i}) = 0$), define $\hat{P}_{ML}(x_i|x_{i-n+1:i-1}) = 0$.

Unfortunately, this often leads to poor performance in many applications. To see this, consider a 2-gram model and an alphabet of a small size, $|\mathcal{X}| = 20,000$ ³. The training set contains $N = 10,000,000$ words. By simple counting, there are $|\mathcal{X}|^2 = 400,000,000$ possible word pairs, so the training data can only cover 2.5% of them, hence most $P_{ML}(x_i|x_{i-1})$ will be zero. This is not acceptable as this model provides little information to help, for example, a speech recogniser to select the correct transcript. Philosophically maximum likelihood only takes into account the evidence, i.e., the training data set, which only works when the sample size is large compared with the number of parameters that need to be estimated. However, when dealing with large alphabet, the sparseness in training data is inevitable. Few would flatly deny any possibility of a word occurring in the future just because it has not occurred in the training set. Hence, the zero frequency problem needs to be tackled properly.

Smoothing is one technique to address this issue. The idea is to make the maximum likelihood estimate less spiky by bringing low probabilities, e.g. zero frequency, upward and high probabilities downward in a sensible manner. We survey a number of smoothing techniques and analyse their advantages and disadvantages. See [CG96] for a more in-depth survey.

5.2.1 Additive Smoothing

The idea of additive smoothing [Lid20, Joh32, Jef48] is that initially we pretend that each word occurs some $0 \leq \delta \leq 1$ times more than it actually does. This idea yields the following estimate

$$\hat{P}_{add}(x_i|x_{i-n+1:i-1}) = \frac{\delta + \#(x_{i-n+1:i})}{\delta|\mathcal{X}| + \#(x_{i-n+1:i-1})}$$

This equation can be obtained by assuming a $Dir(\delta)$ prior distribution on all parameters. Imagine we have a prior training set of size $\delta|\mathcal{X}|$, and each word occurs uniformly δ times. Thus \hat{P}_{add} is equivalent to the maximum likelihood estimate on this imaginary training set combined with the real training set. To see this more formally, \hat{P}_{add} can be rewritten as

$$\begin{aligned} \hat{P}_{add}(x_i|x_{i-n+1:i-1}) &= \frac{\#(x_{i-n+1:i-1})}{\delta|\mathcal{X}| + \#(x_{i-n+1:i-1})} \frac{\#(x_{i-n+1:i})}{\#(x_{i-n+1:i-1})} + \frac{\delta|\mathcal{X}|}{\delta|\mathcal{X}| + \#(x_{i-n+1:i-1})} \frac{1}{|\mathcal{X}|} \\ &= (1 - \lambda(\delta))\hat{P}_{ML} + \lambda(\delta)\hat{P}_{uniform} \end{aligned}$$

where $\lambda(\delta) = \frac{\delta|\mathcal{X}|}{\delta|\mathcal{X}| + \#(x_{i-n+1:i-1})}$ and $\hat{P}_{uniform} = \frac{1}{|\mathcal{X}|}$. This shows that \hat{P}_{add} is, in fact, a linear combination of maximum likelihood estimator and a uniform distribution. Here $\lambda(\delta)$ controls how much we rely on the initial belief.

³The Carnegie Mellon pronouncing dictionary (0.6) contains 133,737 words.

An immediate advantage of additive smoothing is that it avoids the zero frequency problem. However, there is a major drawback to it: The probability 'held out' for unseen data is linear in the size of the alphabet, which is rough and also counter-intuitive. Because how much probably needs to be held out should be determined by the number of words that actually appeared as opposed to the size of the entire alphabet.

5.2.2 Jelinek Mercer Smoothing

The Jelinek Mercer (JM) method [JM80] is, in some sense, an extension of additive smoothing. In additive smoothing, we have a mixture of n -gram model and 0-gram model (uniform distribution), whereas in JM method, this mixture is taken over all n -gram models up to some fixed n . Information from lower order models is merged into higher order models by linear interpolation and can be defined elegantly in a recursive fashion

$$\hat{P}_{JM}(x_i|x_{i-n+1:i-1}) = \lambda_{x_{i-n+1:i-1}} \hat{P}_{ML}(x_i|x_{i-n+1:i-1}) + (1 - \lambda_{x_{i-n+1:i-1}}) \hat{P}_{JM}(x_i|x_{i-n+2:i-1})$$

where $\lambda_{x_{i-n+1:i-1}} \in (0,1)$ may depend on $x_{i-n+1:i-1}$. The base case, the 0:th order model, of this recursion can be set to be the uniform distribution.

While this equation looks similar to additive smoothing, there are two notable differences. First, in JM method the smoothed n -gram model is defined recursively by a linear interpolation between n :th order maximum likelihood estimate and the smoothed $n-1$:th order model, whereas in additive smoothing, n -gram model is a direct linear interpolation between n :th order maximum likelihood estimate and the uniform distribution. Secondly, in additive smoothing there is one $\lambda(\delta)$ for all context $x_{i-n+1:i-1}$, whereas the JM method allows different $\lambda_{x_{i-n+1:i-1}}$ for different context frequencies, which is more reasonable because how much we should resort to lower order models really depends on how much data we have for $x_{i-n+1:i-1}$. The remaining issue we need to address is how to train the λ 's. Setting all λ 's to the same value results in bad performance [BJM83], while training distinct λ 's is not generally feasible. Jelinek and Mercer [BJM83] chose an intermediate idea that put λ 's into different bins according to $\#(x_{i-n+1:i-1})$ and all λ 's in the same bin are restricted to have the same value.

5.2.3 Absolute Discounting

Jelinek Mercer interpolated higher order models with lower ones in a linear fashion, but there are many other functional structures that can be used. In absolute discounting [NE91, NK94] a nonlinear scheme is adopted. Instead of discounting the actual count by multiplying by a factor $\lambda_{x_{i-n+1:i-1}}$, we subtract from each non-zero count a fixed amount $0 \leq D \leq 1$. That is,

$$\hat{P}_{abs}(x_i|x_{i-n+1:i-1}) = \frac{\max\{\#(x_{i-n+1:i}) - D, 0\}}{\#(x_{i-n+1:i-1})} + (1 - \lambda_{x_{i-n+1:i-1}}) \hat{P}_{abs}(x_i|x_{i-n+2:i-1})$$

Here the factor $(1 - \lambda_{x_{i-n+1:i-1}})$ is chosen such that this distribution sums to 1

$$(1 - \lambda_{x_{i-n+1:i-1}}) = \frac{D}{\#(x_{i-n+1:i-1})} N_{1+(x_{i-n+1:i-1})}$$

The notation $N_{1+}(x_{i-n+1:i-1}\cdot)$ is used to denote the number of unique words in the context of $x_{i-n+1:i-1}$, formally

$$N_{1+}(x_{i-n+1:i-1}\cdot) = |\{x_i : \#(x_{i-n+1:i}) > 0\}|$$

Ney [NK94] has suggested setting

$$D = \frac{n_1}{n_1 + n_2}$$

where n_i is the number of n -grams that occur exactly i times in the training data.

5.2.4 Kneser-Ney Smoothing

A refined lower order distribution has been proposed by Kneser&Ney [KN95]. In previous methods, lower order distributions are taken to be either maximum likelihood or the uniform distribution. We have argued that maximum likelihood is not normally a wise choice. Lower order distributions are especially important when there are insufficient statistics to construct accurate higher order models. Thus, Kneser and Ney argue that lower order distributions should not be used carelessly and need to be optimised to perform well.

Their method is based on the absolute discounting scheme, that is,

$$\hat{P}_{KN}(x_i | x_{i-n+1:i-1}) = \frac{\max\{\#(x_{i-n+1:i}) - D, 0\}}{\#(x_{i-n+1:i-1})} + \frac{DN_{1+}(x_{i-n+1:i-1}\cdot)}{\#(x_{i-n+1:i-1})} \hat{P}_{KN}(x_i | x_{i-n+2:i-1})$$

A difference to absolute discounting is with the base case unigram probability, instead of setting it to be proportional to the number of occurrences of a word, they set it to be the number of distinct words that it follows. Formally

$$P_{KN}(x_i) = \frac{N_{1+}(\cdot x_i)}{N_{1+}(\cdot)}$$

where $N_{1+}(\cdot x_i)$ is the number of different words that x_i follows, $N_{1+}(\cdot x_i) = |\{x_{i-1}, \#(x_{i-1:i}) > 0\}|$ and $N_{1+}(\cdot)$ is the number of unique word pairs, that is, $N_{1+}(\cdot) = |\{(x_{i-1}, x_i), \#(x_{i-1:i}) > 0\}|$.

This strange counting method can be motivated by the following example: Consider an article on the paintings of Vincent Van Gogh. ‘Gogh’ is no doubt a common word, however, it only occurs after the word ‘Van’, which means this word does not have much ‘freedom’. In previous methods, owing to the high frequency of ‘Gogh’ a maximum likelihood estimate would naively assign a relatively high probability to this word, however, intuitively this should not be the case, because ‘Gogh’ only occurs after ‘Van’ and this has already been taken care of by higher order models, e.g., the bigram model. In light of this, this ad hoc way of counting in Kneser-Ney smoothing makes sense, the base case probability is only proportional to the number of unique words that it follows and in some sense it takes into consideration the diversity of histories and measures

genuinely how frequently a word would occur on its own, stripping off what has been characterised by higher order models.

The Kneser Ney smoothing can be defined in the follow recursive fashion

$$\hat{P}_{KN}(x_i|x_{i-n+1:i-1}) = \frac{\max\{N_{1+}(\cdot|x_{i-n+1:i}) - D, 0\}}{N_{1+}(\cdot|x_{i-n+1:i-1})} + \frac{DN_{1+}(x_{i-n+1:i-1})}{N_{1+}(\cdot|x_{i-n+1:i-1})} \hat{P}_{KN}(x_i|x_{i-n+2:i-1})$$

where,

$$N_{1+}(x_{i-n+2:i}) = |\{x_{i-n+1}, \#(x_{i-n+1:i}) > 0\}|$$

$$N_{1+}(x_{i-n+2:i-1}) = |\{(x_{i-n+1}, x_i), \#(x_{i-n+1:i}) > 0\}|$$

with the base case 0:th order distribution being the uniform distribution in order to handle sparseness in the training data.

5.2.5 Modified Kneser-Ney Smoothing

Instead of restricting to having one discount D for all nonzero counts, in a modified Kneser-Ney smoothing [CG96] introduced different parameters D_1, D_2, D_{3+} . The rationale behind this idea is that for different frequency of n -grams different discount should be applied. We have

$$\hat{P}_{KNM}(x_i|x_{i-n+1:i-1}) = \frac{\max\{N_{1+}(\cdot|x_{i-n+1:i}) - D(\#(x_{i-n+1:i})), 0\}}{N_{1+}(\cdot|x_{i-n+1:i-1})} + \lambda(x_{i-n+1:i-1}) \hat{P}_{KNM}(x_i|x_{i-n+2:i-1})$$

with the base case 0:th order model being the uniform distribution and where

$$D(c) = \begin{cases} 0 & \text{if } c=0 \\ D_1 & \text{if } c=1 \\ D_2 & \text{if } c=2 \\ D_{3+} & \text{if } c \geq 3 \end{cases}$$

Again, the purpose of $\lambda(x_{i-n+1:i-1})$ is to make the distribution sum to 1, which is achieved by

$$\lambda(x_{i-n+1:i-1}) = \frac{D_1 N_1(x_{i-n+1:i-1}) + D_2 N_2(x_{i-n+1:i-1}) + D_{3+} N_{3+}(x_{i-n+1:i-1})}{N_{1+}(\cdot|x_{i-n+1:i-1})}$$

where $N_k(x_{i-n+1:i-1}) = |\{x_i, \#(x_{i-n+1:i}) = k\}|$ and $N_{k+}(x_{i-n+1:i-1}) = |\{x_i, \#(x_{i-n+1:i}) \geq k\}|$.

The parameters $D_{1,2,3+}$ are set to

$$D_1 = 1 - 2Y \frac{n_2}{n_1}$$

$$D_2 = 2 - 3Y \frac{n_3}{n_2}$$

$$D_{3+} = 3 - 4Y \frac{n_4}{n_3}$$

where $Y = \frac{n_1}{n_1 + 2n_2}$.

5.3 Pitman-Yor Process

Another approach to discounting is the two parameter Poisson-Dirichlet Process (PDP) [PY97] on n -gram models. PDP, which is also known as the Pitman-Yor process (PYP) in natural language processing literature, is an extension of the Dirichlet Process (DP). Relatedly, PYP/PDP/DP has a stick breaking representation and its marginalisation is known as the Chinese Restaurant Process (CRP), which elegantly demonstrates the power law phenomenon. A Hierarchical PYP applied to n -gram models corresponds to an interpolated version of the Kneser-Ney method. Though we have discussed the intuitive rationale behind Kneser-Ney's ad hoc counting method, the main justification has always remained empirical. An advantage of a PYP-based n -gram model is that it possesses a nice Bayesian interpretation, which we will see through CRP in this section.

This section starts with a brief introduction to the Dirichlet distribution and its non-parametric extension, DP; then we describe two interesting representations, stick breaking and CRP.

5.3.1 Dirichlet Distribution and Dirichlet Process

Dirichlet distribution. First consider a k -dimensional simplex

$$\mathcal{S}_k = \{p = (p_1, p_2, \dots, p_k) : \forall i, p_i \geq 0, \sum_i p_i = 1\}$$

This simplex has one-to-one correspondence with the set of all measures on $\mathcal{X} = \{1, 2, \dots, k\}$, $\mathcal{M}(\mathcal{X})$. Now consider a prior distribution on $\mathcal{M}(\mathcal{X})$ [GR03].

Definition 32 (Dirichlet Distribution). Let $\alpha = (\alpha_1, \alpha_2, \dots, \alpha_k)$ with $\alpha_i > 0$ for all $i = 1, 2, \dots, k$. $p = (p_1, p_2, \dots, p_k)$ is said to have Dirichlet distribution with parameter α , $p \sim \text{Dir}(\alpha)$ if p has the following distribution

$$P(p) = \frac{\Gamma(\sum_{i=1}^k \alpha_i)}{\prod_{i=1}^k \Gamma(\alpha_i)} \prod_{i=1}^k p_i^{\alpha_i - 1} \quad \forall p \in \mathcal{S}_k$$

where $\Gamma(\cdot)$ is the Gamma function, $\Gamma(x) = \int_0^\infty t^{x-1} e^{-t} dt$.

Remark 33. There are two things about the Dirichlet distribution worth noting

1. For a fixed sample space \mathcal{X} , the Dirichlet distribution can be considered to be a distribution of measures, $\mathcal{M}(\mathcal{X})$.
2. In the case where any $\alpha_i = 0$, by convention we interpret it as a downgraded $n-1$ dimensional Dirichlet distribution.

Polya urn interpretation. There is an interesting and useful view of the Dirichlet distribution known as the Polya urn scheme, which goes as follows.

Consider a Polya urn (named after Hungarian Jewish mathematician George Pólya) that contains $\alpha_0 := \sum_{i=1}^k \alpha_i$ balls, of which α_i balls are of color i , $i = 1, 2, \dots, k$. Now pick

one ball from the urn uniformly at random and replace it with two balls of the same colour. Set $X_m = i$ if the m :th ball drawn from the urn has colour i . In general, we have

$$P(X_{n+1} = j | X_{1:n}) = \frac{\alpha_j + \sum_{i=1}^n \mathbb{I}(X_i = j)}{\alpha_0 + n}$$

We can rewrite this as the following to provide more insight

$$\begin{aligned} P(X_{n+1} = j | X_{1:n}) &= \frac{n}{\alpha_0 + n} \frac{\sum_{i=1}^n \mathbb{I}(X_i = j)}{n} + \frac{\alpha_0}{\alpha_0 + n} \frac{\alpha_j}{\alpha_0} \\ &= \lambda(\alpha_0) \frac{\sum_{i=1}^n \mathbb{I}(X_i = j)}{n} + (1 - \lambda(\alpha_0)) \frac{\alpha_j}{\alpha_0} \end{aligned}$$

where $\lambda(\alpha_0) = n / (\alpha_0 + n)$. Just as before, this is a weighted combination of an empirical frequency estimate combined with a prior belief. The value α_0 is called the concentration or strength parameter because it measures the strength of the prior belief.

Dirichlet Process. The Dirichlet distribution $p \sim \text{Dir}(\alpha)$ can be interpreted as a prior distribution on the simplex \mathcal{S}_k , which 1-1 corresponds to all the measures on $\mathcal{X} = \{1, 2, \dots, k\}$. We want to generalise this idea and consider the case where $\mathcal{X} = \mathbb{R}$. Just as Gaussian processes that effectively put a distribution over all functions on \mathbb{R} , a Dirichlet process places a measure over all measures on \mathbb{R} or more generally on any measurable space. The definition for Dirichlet processes is as follows [GR03].

Definition 34. Consider a sample space Ω and a probability measure G_0 on Ω and let α_0 be a positive real number. A Dirichlet process is the distribution of a random probability measure G over Ω such that, for any finite measurable partition A_1, \dots, A_k of Ω , $(G(A_1), \dots, G(A_k))$ is distributed according to a k -dimensional Dirichlet distribution, that is,

$$(G(A_1), \dots, G(A_k)) \sim \text{Dir}(\alpha_0 G_0(A_1), \dots, \alpha_0 G_0(A_k))$$

Then we write $G \sim DP(\alpha, G_0)$ if G is a Dirichlet process. We also call G_0 the base measure of G and call α_0 the concentration parameter.

Remark 35. A few remarks:

1. DP is an extension of a Dirichlet distribution on finite spaces. In fact, when the base measure G_0 is a categorical distribution over a finite space, the Dirichlet process can be reduced to a Dirichlet distribution [BH10].
2. In some other definitions of DP, G_0 does not need to be a probability measure on Ω provided that $G_0(\Omega)$ is finite. However, one can always normalise it to derive a probability measure. For simplicity, we will stick to G_0 being a probability measure.

5.3.2 Stick Breaking Representation and Chinese Restaurant Process

The definition of DP is not intuitive. We are going to see two different representations of DP, which provide more insight. The first is called the stick breaking representation: a Dirichlet process can be defined in terms of an impulse mixture model [BH10]

Definition 36 (Impulse mixture model). Given a probability distribution $H(\cdot)$ on a measurable space \mathcal{X} , assume that values $\theta_k \in \mathcal{M}(\mathcal{X})$ are i.i.d. according to $H(\cdot)$ for $k = 1, 2, \dots$. Also an infinite dimensional probability vector p is sampled from a distribution $Q(\cdot)$ independently of each θ_k so that $p_k \in [0, 1]$ and $\sum_{k=1}^{\infty} p_k = 1$. Then

$$G(\cdot) = \sum_{k=1}^{\infty} p_k \mathbb{I}[\cdot = \theta_k]$$

is an impulse mixture model with base distribution $H(\cdot)$.

Definition 37 (Dirichlet process). Following the definition of impulse mixture model, if

$$p_k = \beta_k \prod_{j=1}^{k-1} (1 - \beta_j)$$

and

$$\beta_k \sim \text{Beta}(1, \alpha_0)$$

then the impulse mixture model is a Dirichlet process with concentration parameter α_0 and base measure $H(\cdot)$.

It is easy to verify that $p_k \in [0, 1]$ and $\sum_{k=1}^{\infty} p_k = 1$ with probability 1.

More generally, when each β_k in the definition above is drawn from $\text{Beta}(1 - d_0, \alpha_0 + kd_0)$, we call the resulting mixture model $G(\cdot)$ a Pitman-Yor process (PYP) with concentration parameter α_0 and discount parameter d_0 with a base measure $H(\cdot)$.

Chinese Restaurant Process. Now we consider the Pitman-Yor process and present an interesting and useful representation known as the Chinese Restaurant Process (CRP). Suppose a random probability measure G on a sample space Ω is sampled from a PYP, that is, $G \sim \text{PYP}(\alpha_0, G_0, d_0)$ where α_0 is the concentration parameter, d_0 the discount parameter and G_0 the base measure. θ is then sampled according to G , $\theta \sim G$. CRP provides a nice generative interpretation with G being marginalised out. Later we will talk about how to fit CRP into the text modelling context.

Now imagine a Chinese restaurant with infinitely many tables, $t = 1, 2, \dots$. Customers are indexed by $i = 1, 2, \dots$ with values x_i ⁴. Once a customer is seated at a table t , we say that table t is occupied and c_t is the number of customers seated at table t . Let T be the total number of occupied tables so far and c to be the total number of customers

⁴Or pretending that each customer will order a dish x_i

so far. Tables have values θ_t that are sampled from G_0 .

Algorithm 1: Sampling from a $CRP(\alpha_0, G, d_0)$

```

customer 1 enters the restaurant and sits at table 1.
 $x_1 = \theta_1$ , where  $\theta_1 \sim G_0$ ,  $c = 1$ ,  $c_1 = 1$ ,  $T = 1$ 
for  $i = 1, 2, \dots$  do
    customer  $i$  sits at table  $t$  with probability  $\frac{c_t - d_0}{\alpha_0 + c}$  or he sits at a new table  $T + 1$ 
    with probability  $\frac{\alpha_0 + d_0 T}{\alpha_0 + c}$ 
    if new table was chosen then
         $T = T + 1$ 
         $\theta_{T+1} \sim G_0$ 
    end
    Put  $x_i = \theta_t$ , where  $t$  is the table at which customer  $i$  sits
     $c_t = c_t + 1$ 
end

```

There are two interesting phenomena that can be observed from a $CRP(\alpha_0, G, d_0)$. First, a $CRP(\alpha_0, G, d_0)$ displays a clustering effect, or simply put ‘the-rich-gets-richer’ effect. To see that, note that the probability of a customer to sit at a certain table t is proportional to how popular this table already is ($\propto n_t$): the more customers sit at a table, the more likely it becomes that a new customer will join them. Secondly, a $CRP(\alpha_0, G, d_0)$ exhibits a diversity effect. Note that the probability that a customer will sit at a new table is determined by the concentration parameter α_0 and the total number of occupied tables T . For a fixed α_0 , the more occupied tables there are, the more likely a new customer will start at a new table. There are many other phenomenon that can be modelled by a CRP . Consider a freshman who wants to join a club at a university, that student is more likely to join a popular club that has already a lot of members and less likely to join an unpopular one. With a certain probability, that student will decide to start his own club. The number of existing clubs reflects how much the university encourage students to create clubs on their own (assuming there can be infinitely many possible clubs). As such, the more clubs there are, the more likely the student will start his own.

CRP can also be used for modelling text. Think of customers as observed words in a text and tables as vocabularies. Some words are commonly used, e.g. ‘the’, ‘a’, ‘to’, and thus one is more likely to observe them in a text. The observed word ‘to’ sits at a vocabulary table ‘to’ and the more popular the table is, the more likely the next word will sit at it, i.e. the next word will be ‘to’. The total number of occupied tables T can be interpreted as how active one’s vocabulary is. The more new words the author has used, the more likely he will use a new word in the future.

5.3.3 Hierarchical Pitman-Yor Process

Now we put things together and present an n -gram model based on a hierarchical extension of the PYP [Teh06]. An n -gram model represents the conditional probability of x_i given its context $x_{i-n+1:i-1}$ of length $n-1$. Now assume that this probability is a

Pitman-Yor process that takes the following specific form,

$$P(\cdot|x_{i-n+1:i-1}) \sim \text{PYP}(\alpha_{n-1}, P(\cdot|x_{i-n+2:i-1}), d_{n-1})$$

Note that both the concentration parameter and the discount parameter depend only on the length of the context while the base measure is the distribution of the current word given all but the earliest words in the context. In the lack of any knowledge about $P(\cdot|x_{i-n+2:i})$, it is defined recursively as another PYP with concentration parameter α_{n-2} , discounting parameter d_{n-2} and the base measure $P(\cdot|x_{i-n+3:i})$. This process ends with the empty context $P(\cdot|x_{i:i-1}) = P(\cdot|\epsilon)$, which is chosen to be a PYP with uniform distribution as its base measure,

$$P(\cdot|\epsilon) \sim \text{PYP}(\alpha_0, \mathcal{L}, d_0)$$

where $\mathcal{L}(x_i) = \frac{1}{|\mathcal{X}|}$ for all $x_i \in \mathcal{X}$.

6 Experiments and Discussion

In this section, we perform an experimental comparative study for investigating the relationship between binary data compression techniques and natural language modelling techniques, including an empirical comparison between Kneser-Ney (KN) variants with regular Context Tree Weighting algorithm (CTW) and phase CTW, and with large-alphabet CTW with different estimators. We also apply the idea of Hutter's adaptive sparse Dirichlet-multinomial coding to the KN method and provide a heuristic to make the discounting parameter adaptive.

Choice of corpus and vocabulary. For the experiments in this section, we have chosen the following three corpora:

1. Australian Broadcasting Commission 2006 (abc)
 - Source: <http://www.abc.net.au/>
 - Number of words: 766,813
 - Number of sentences: 29,404
 - Average number of words per sentence: 26.079

2. Brown Corpus (brown)
 - Source: <http://www.hit.uib.no/icame/brown/bcm.html>
 - Number of words: 1,161,192
 - Number of sentences: 57,340
 - Average number of words per sentence: 20.251

3. CONLL 2000 Chunking Corpus (conll2000)
 - Source: <http://www.cnts.ua.ac.be/conll2000/chunking/>
 - Number of words: 259,104
 - Number of sentences: 10,948
 - Average number of words per sentence: 23.667

We used the Carnegie Mellon pronouncing dictionary (0.6) as our dictionary, which contains 133737 words ($\log_2 133737 \approx 17.029$). We have also used the Calgary Corpus for compression. Descriptions of each file in Calgary Corpus and their size is shown in Table 2. Table 3 shows the file sizes under conventional Lempel-Ziv compression, implemented by gzip (version Apple gzip 2). There are 1 to 9 different compression levels with 1 being the fastest yet worst compression and 9 being the best yet slowest compression.

| Size (bytes) | File name | Description |
|--------------|-----------|--|
| 111,261 | BIB | ASCII text in UNIX 'refer' format – 725 bibliographic references. |
| 768,771 | BOOK1 | unformatted ASCII text – Thomas Hardy: Far from the Madding Crowd. |
| 610,856 | BOOK2 | ASCII text in UNIX 'troff' format – Witten: Principles of Computer Speech. |
| 102,400 | GEO | 32 bit numbers in IBM floating point format – seismic data. |
| 377,109 | NEWS | ASCII text – USENET batch file on a variety of topics. |
| 21,504 | OBJ1 | VAX executable program – compilation of PROGP. |
| 246,814 | OBJ2 | Macintosh executable program – 'Knowledge Support System'. |
| 53,161 | PAPER1 | Witten, Neal, Cleary: Arithmetic Coding for Data Compression. |
| 82,199 | PAPER2 | Witten: Computer (in)security. |
| 513,216 | PIC | 1728 x 2376 bitmap image (MSB first): text in French and line diagrams. |
| 39,611 | PROGC | Source code in C – UNIX compress v4.0. |
| 71,646 | PROGL | Source code in Lisp – system software. |
| 49,379 | PROGP | Source code in Pascal – program to evaluate PPM compression. |
| 93,695 | TRANS | ASCII and control characters – transcript of a terminal session. |

Table 2: Calgary Corpus.

| File name | Original size (bytes) | gzip compressed (best) | gzip compressed (worst) |
|-----------|-----------------------|------------------------|-------------------------|
| BIB | 111,261 | 35,087 | 43,898 |
| BOOK1 | 768,771 | 312,504 | 365,274 |
| BOOK2 | 610,856 | 206,153 | 248,935 |
| GEO | 102,400 | 68,377 | 69,882 |
| NEWS | 377,109 | 144,497 | 164,305 |
| OBJ1 | 21,504 | 10,334 | 10,707 |
| OBJ2 | 246,814 | 81,032 | 93,486 |
| PAPER1 | 53,161 | 18,543 | 21,612 |
| PAPER2 | 82,199 | 29,684 | 35,099 |
| PIC | 513,216 | 52,237 | 65,575 |
| PROGC | 39,611 | 13,348 | 15,455 |
| PROGL | 71,646 | 16,164 | 20,038 |
| PROGP | 49,379 | 11,186 | 13,382 |
| TRANS | 93,695 | 18,934 | 23,966 |

Table 3: Calgary Corpus under conventional Lempel-Ziv compression, implemented by gzip.

6.1 Kneser-Ney and Binary CTW

We compare the Kneser-Ney method for large-alphabet sequences with the CTW algorithm for binary (or binarised) sequences.

Kneser-Ney smoothing implementation. We implemented 3-gram and 2-gram Kneser-Ney, however, there are many variations of the Kneser-Ney method. In the experiment, we implemented three different variations: the normal Kneser-Ney method, the modified Kneser-Ney and the modified Kneser-Ney with normal counting. Explicit equations are in the appendix.

Depth of binary CTW. For fair comparisons, we need to set the depth of binary CTW to be commensurate with the ‘depth’ of Kneser-Ney. On average, there is 4.5 letters in an English word [Pie12]. Thus, to match a 3-gram Kneser-Ney model we need 108 ($4.5 \times 8 \times 3$) bits depth binary context trees. In our experiments, we have tested binary CTW for depth 2,4,16,32,48,64,68,72,96,108. A discussion of the depth of CTW in relation to its performance is presented later.

Preprocessing and CTW alphabet coding. Punctuation is useful for Kneser-Ney to separate words, but the Kneser-Ney method does not model punctuations directly. Also the Kneser-Ney method does not distinguish between upper and lower case letters, i.e. ‘What’ and ‘what’ are treated as the same word. Consequently, we need to do some preprocessing by removing all punctuations and lower-casing every letter.

We have a lot flexibility when binarise after preprocessing. The way we binaries hugely influences the performance of CTW so we want to choose a way of binarising that does not favour either method. A naive way of doing this is to take the ASCII code of each letter and simply use the 8-bit binary representation of ASCII codes. For example the word ‘am’ has ASCII codes 97 and 109 and their 8-bit binary representations are 01100001 and 01101101. We thus code ‘am’ using the concatenation of these two binary code resulting in 0110000101101101.

However, since we remove all the punctuations and lower-case all letters, we do not use ASCII code to its full capacity and in fact we only use 26 symbols out of the possible 256 codes. It would not be fair to code each lower-case letter using ASCII code because it forces CTW to make extra effort to recognise the symbols that we never use. In light of this and also to make the comparison fair, we redesigned the coding for each letter. For a 26 symbol alphabet, $\lceil \log_2 26 \rceil = 5$ bits is enough. As a result, we uniformly code each letter using 5 bits, see Table 4.

However, adopting a 5-bit uniform coding still doesn’t make the comparison completely fair. Consider the following example: Define the following new language called randEnglish: take each legitimate English word w_i and append it with a random sequence of letters of length 100, r_i . Each word in the alphabet of randEnglish is the resulting word $w_i r_i$. Note that this operation preserves the size of English alphabet, that is, the size of alphabet of the new language randEnglish is the same of that of English. Now we compare Kneser-Ney method with binary CTW. Since Kneser-Ney works on the level of words, the experimental results per word remain the same as for English, which will result in a much smaller (≈ 100 times smaller) code length per byte (each word in randEnglish is at least 100 byte). However, we would reasonably expect equal or larger code length per byte from CTW because it works at the binary level.

| char | ASCII code | uniform code | char | ASCII code | uniform code |
|------|------------|--------------|------|------------|--------------|
| ' ' | 100000 | 00000 | 'n' | 1101110 | 01110 |
| 'a' | 1100001 | 00001 | 'o' | 1101111 | 01111 |
| 'b' | 1100010 | 00010 | 'p' | 1110000 | 10000 |
| 'c' | 1100011 | 00011 | 'q' | 1110001 | 10001 |
| 'd' | 1100100 | 00100 | 'r' | 1110010 | 10010 |
| 'e' | 1100101 | 00101 | 's' | 1110011 | 10011 |
| 'f' | 1100110 | 00110 | 't' | 1110100 | 10100 |
| 'g' | 1100111 | 00111 | 'u' | 1110101 | 10101 |
| 'h' | 1101000 | 01000 | 'v' | 1110110 | 10110 |
| 'i' | 1101001 | 01001 | 'w' | 1110111 | 10111 |
| 'j' | 1101010 | 01010 | 'x' | 1111000 | 11000 |
| 'k' | 1101011 | 01011 | 'y' | 1111001 | 11001 |
| 'l' | 1101100 | 01100 | 'z' | 1111010 | 11010 |
| 'm' | 1101101 | 01101 | | | |

Table 4: ASCII code and 5-bit uniform code for white space and English letters . The second column is their ASCII codes. The third column is their 5-bit codes.

This example suggests that we should devise codes on words rather than letters in order to match the inner workings of Kneser-Ney. We devised three different variations. The first one is the binary representation of alphabetical index of a word in the dictionary. The other two are Huffman codes based on the length of a word and the frequency of a word in a certain corpus respectively.

Tradeoff between universality and domain-specific knowledge involvement.

We should note that the two compression techniques being compared are fundamentally different in the sense that CTW is a universal compressor whilst Kneser-Ney is used mainly for text compression/modelling. No a-priori knowledge is needed for CTW while Kneser-Ney uses a lot of text-specific knowledge. The process of binarising text demonstrates the tradeoff between universality and domain-specific knowledge involvement. One extreme is to use ASCII code for each letter, where there is nearly no domain-specific knowledge fed into CTW, on the other end of the spectrum we use Huffman code for each word based on their frequency in a certain corpus, where in this case CTW does not need to worry about the spelling of a word; even more we have in some sense cheated by giving CTW corpus specific knowledge. As we will see later in this section, the more domain knowledge we give to CTW, the better it performs in general. However, it comes with the price of losing its universality.

6.1.1 CTW Implementation Notes

Lazy instantiation. A big challenge we had in implementing CTW is that it is very memory intensive, especially for large depth. The naive CTW implementation introduced in Section 3.3.1 would use up 4GB memory on a typical personal computer very quickly for depth larger than 48. However, for a fair comparison with the Kneser-Ney method there is a need to make CTW feasible for a large depth, e.g. 108. Note that we sometimes don't need to extend a branch of the tree to its maximum depth and

create all nodes along the path, particularly when a path does not have any branches. For example, consider a context tree with depth 108. For the very first bit in a file (assuming we have a dummy history containing 108 zeros), we know the conditional probability of that bit is a half because essentially there is no ‘mixing’ happening down the path. As such, we don’t need to create these 108 nodes to work out a probability that we could figure out otherwise. As such, the idea is to create a node only when we have to create it. This principle in program design is normally termed ‘lazy instantiation’ or ‘tree pruning’ [WT97].

There are two variations of lazy instantiation considered in [WT97]. We have implemented the more advanced one, called strict unique path pruning. The pseudo-code of this algorithm is presented in Algorithm 2.

The improvement is shown in Table 5. The savings on nodes expanded (and thus memory) is nearly 90% for deep trees. The amount of memory saved increases as the depth gets larger. A graphical illustration is presented in Figure 2. The working implementation of CTW uses lazy instantiation, which allows us to run CTW with large depth.

| D | bib(111KB) | | | book1 (769KB) | | | obj2 (247KB) | | |
|-----|------------|------------|--------|---------------|------------|--------|--------------|------------|--------|
| | lazy | normal | saving | lazy | normal | saving | lazy | normal | saving |
| 2 | 7 | 7 | 0.00% | 7 | 7 | 0.00% | 7 | 7 | 0.00% |
| 4 | 31 | 31 | 0.00% | 31 | 31 | 0.00% | 31 | 31 | 0.00% |
| 16 | 29,303 | 39,771 | 26.32% | 35,669 | 45936 | 68.18% | 101,114 | 112,079 | 9.78% |
| 32 | 548,853 | 1,376,810 | 60.14% | 1,469,818 | 2,943,039 | 69.93% | 1,525,153 | 4,888,073 | 68.80% |
| 48 | 1,446,469 | 5,281,829 | 72.61% | 8,239,332 | 21,994,201 | 48.94% | 3,629,199 | 16,135,477 | 77.51% |
| 64 | 2,659,482 | 11,433,687 | 76.74% | 19,712,624 | 68,957,098 | 38.06% | 5,922,041 | 31,827,109 | 81.39% |
| 68 | 2,961,038 | 13,266,662 | 77.68% | 22,781,135 | N/A | N/A | 6,480,135 | 36,260,831 | 82.13% |
| 72 | 3,263,137 | 15,196,830 | 78.53% | 25,779,819 | N/A | N/A | 7,047,995 | 40,854,227 | 82.75% |
| 96 | 5,031,138 | 28,514,343 | 82.36% | 40,234,161 | N/A | N/A | 10,386,466 | N/A | N/A |
| 108 | 5,886,568 | 36,103,173 | 83.70% | 44,729,343 | N/A | N/A | 12,041,406 | N/A | N/A |
| 128 | 7,317,818 | 49,770,953 | 85.30% | 48,974,028 | N/A | N/A | 14,737,162 | N/A | N/A |
| 192 | 10,877,602 | 98,497,107 | 88.96% | 52,309,612 | N/A | N/A | 22,200,996 | N/A | N/A |
| 256 | 13,183,011 | N/A | N/A | 52,649,428 | N/A | N/A | 28,596,777 | N/A | N/A |

Table 5: Number of nodes expanded in normal CTW and CTW with lazy instantiation. Those that normal CTW with large depths used up all memory and didn’t finish compression within 10 minutes are marked N/A.

Multi-threading CTW. Since multiple cores are now ubiquitous, I have implemented a multithreading CTW.

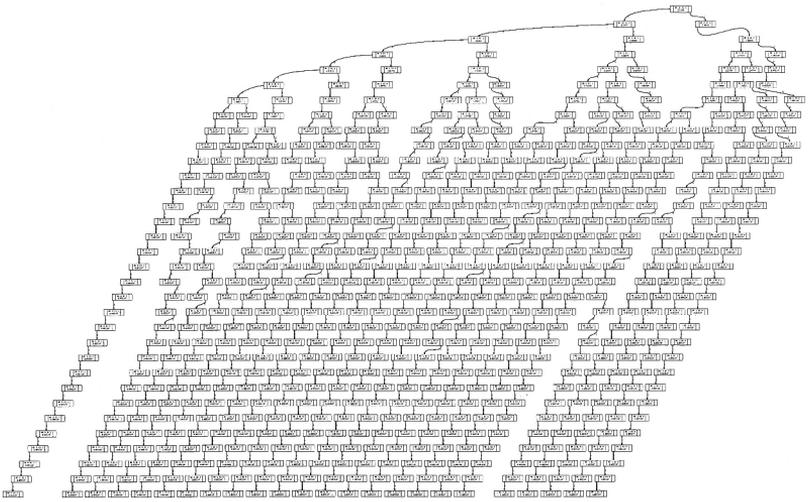
The inner working of CTW can be divided into two separate phases: (1) collecting nodes along the context and (2) updating all the nodes on the path in reverse order. With special care, these two phases can be done by two different threads. As such, the node collection operation and updating operation can be done in parallel using the producer and consumer scheme. The producer and consumer problem is a classical problem in multithread programming and there are a number of frameworks available. As to this particular implementation, there is a product buffer that holds path stacks. The buffer is implemented as a queue (FIFO). The producer thread collects relevant nodes, capsulates them as a stack and puts them in the product buffer, while the

Algorithm 2: CTW lazy instantiation

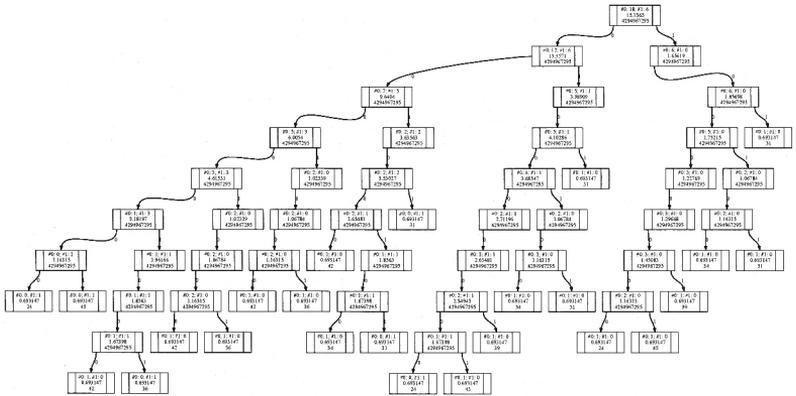
```
current node = root
path.push(current node)
current context index = size of file buffer
for  $i=0:depth:1$  do
  if child corresponding to current context does not exist then
    current node = child
    path.push(current node)
    current context index-
  end
  else
    if current node file index is empty then
      current node = child = new node
      current node.pruned context = current context index - 1
      path.push(current node)
      break
    end
    else if pruned context != current context then
      child of pruned context = copy(current node)
      child of pruned context.pruned context = current node.pruned
      context - 1
      child of current context = new node
      child of current context = current context - 1
      set current node.pruned context to empty
      path.push(child of current context)
    end
    else
      child of current context = copy(current node)
      child of current context.pruned context = current node.pruned
      context - 1
      current node = child of current context
      current context index-
      path.push(current node)
    end
  end
end
end
```

consumer thread constantly checks the product buffer and once it is non-empty, pops the front element from the buffer and consumes it (update all the nodes in the stack). Special care has been taken to handle race conditions.

The performance is shown in Table 6. We use the single thread CTW later for comparison because firstly the time saving of two threads CTW is marginal and also because debugging a multi-threading program is much harder than debugging a single thread program.



(a) Naive CTW implementation



(b) Naive CTW implementation with lazy instantiation.

Figure 2: A graphical illustration of the memory improvement of CTW with lazy instantiation. Both CTW are of depth 32, after processing the first 24 bits of the bib file in the Calgary corpus.

It is worth noting that [ST01] adopted a different and more sophisticated approach to the parallel CTW implementation. They split the tree over a given number of processors such that every processor works independently on symbols whose context

| | single thread | two threads | time saving |
|-----------------|---------------|-------------|-------------|
| bib depth 196 | 41s | 35s | 14.6% |
| bib depth 256 | 46s | 39s | 15.2% |
| book1 depth 128 | 4m39s | 4m3s | 12.9% |

Table 6: Two-thread CTW performance

falls into a particular subtree. Speed gains from a factor of 2 to 7 were achieved with their implementation.

6.1.2 Kneser-Ney and Regular CTW

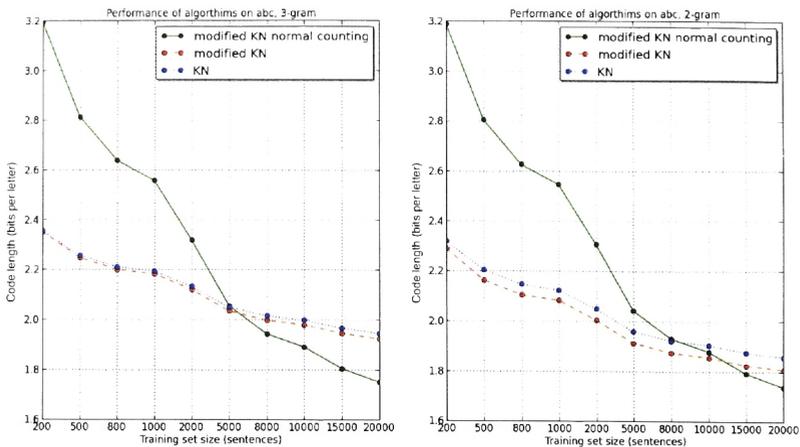
Results. Experimental results of Kneser-Ney variants on their own are shown in Figure 3a to Figure 3c, the comparison with regular CTW is shown in Table 7.

| Corpus | KN | D | ASCII | 5 bits uniform | alphabetical | Huffman (len) | Huffman (freq) |
|-----------|-------------|-------------|-------------|----------------|--------------|---------------|----------------|
| abc | 1.94 | 2 | 9.375929704 | 5.747319104 | 5.077420912 | 5.763001652 | 2.141370455 |
| | | 4 | 8.976426167 | 5.689325598 | 4.848616234 | 5.30841402 | 2.141162723 |
| | | 16 | 3.715387898 | 3.31082257 | 3.044632157 | 2.702719353 | 2.111372354 |
| | | 32 | 2.439632608 | 2.244485751 | 2.421701278 | 2.074434783 | 2.088172015 |
| | | 48 | 2.211723286 | 2.226114853 | 2.182635132 | 2.062969508 | 2.088171622 |
| | | 64 | 2.194516202 | 2.224705186 | 2.180646019 | 2.060129822 | 2.088171622 |
| | | 72 | 2.193144816 | 2.224588231 | 2.179250532 | 2.060129764 | 2.088171622 |
| | | 96 | 2.191879964 | 2.224556668 | 2.179194317 | 2.060042847 | 2.088171622 |
| | | 108 | 2.191829914 | 2.22454415 | 2.179109195 | 2.060042847 | 2.088171622 |
| brown | 2.2 | 2 | 9.417994045 | 5.759628191 | 5.208435087 | 5.900170496 | 2.231358058 |
| | | 4 | 9.003493245 | 5.702626191 | 4.965836405 | 5.436051867 | 2.231279619 |
| | | 16 | 3.71343232 | 3.423379792 | 3.137747665 | 2.805508529 | 2.216659651 |
| | | 32 | 2.574312174 | 2.398197687 | 2.521619048 | 2.199189223 | 2.210365901 |
| | | 48 | 2.357993016 | 2.384292341 | 2.320540037 | 2.189209113 | 2.210365941 |
| | | 64 | 2.343900951 | 2.383813016 | 2.31956709 | 2.188570879 | 2.210365941 |
| conll2000 | 1.98 | 2 | 9.331813477 | 5.732065629 | 4.969715227 | 5.630713072 | 2.085574222 |
| | | 4 | 8.951352437 | 5.679587256 | 4.739264211 | 5.191496668 | 2.085560668 |
| | | 16 | 3.743421183 | 3.358213413 | 3.044148275 | 2.73413459 | 2.071582332 |
| | | 32 | 2.543277403 | 2.468912229 | 2.502505565 | 2.247259704 | 2.061505984 |
| | | 48 | 2.398660361 | 2.46109411 | 2.365711752 | 2.243357326 | 2.061505675 |
| | | 64 | 2.39186783 | 2.46067117 | 2.364371056 | 2.241961493 | 2.061505675 |
| 108 | 2.391141942 | 2.460629683 | 2.363863803 | 2.241921652 | 2.061505675 | | |

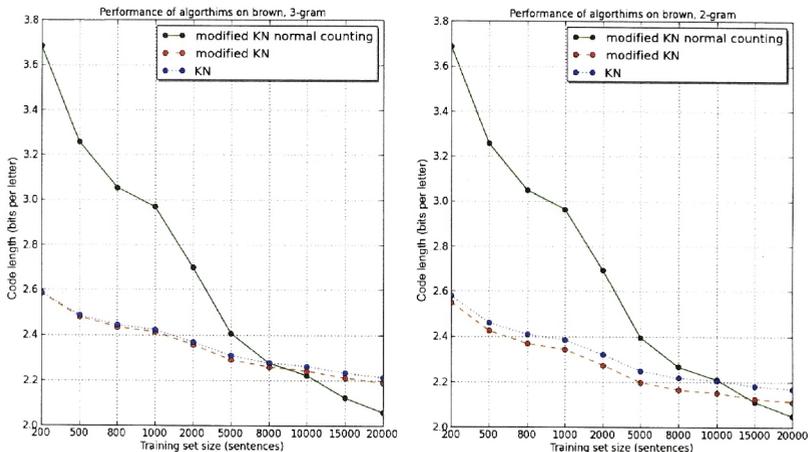
Table 7: Experimental results of regular CTW compared to baseline Kneser-Ney. We have chosen modified Kneser-Ney on 3-gram model with maximum training sentences of each corpus. The number in the Kneser-Ney (KN) column is the average code length in bits per letter. We have used five different ways of binarising each corpus. ASCII represents a 8-bit ASCII coding for each letter, 5 bits uniform uses a 5-bits uniform coding as shown in Table 4. The remaining three code words instead of letters. ‘lexicographical’ uses the binary representation of lexicographical index of a word in the alphabet set. Huffman (len) and Huffman (freq) are Huffman codes for each word based on their length and frequency in each corpus respectively. The column D represents different depths of CTW. The numbers shown in columns ASCII, 5 bits uniform, alphabetical, Huffman (len), Huffman (freq) are actual code length. The smaller the number is, the better the performance.

Discussion. A few observations can be made from the results.

1. **Kneser-Ney.** Firstly, for all variants of Kneser-Ney methods, the larger the training set, the better the performance. This coincides with our expectation. Secondly it is interesting to see that the performance of Kneser-Ney does not vary too much between 2-gram and 3-gram models. This may be due to the following two reasons: (1) the severe sparsity in training data for 3-gram model; (2) the distribution of a word largely depends on the most recent word, but only marginally on words prior to that. Thirdly, both Kneser-Ney and modified Kneser-Ney outperform modified Kneser-Ney with normal counting when the training set is relatively small and modified Kneser-Ney slightly outperforms Kneser-Ney. However, when the size of the training set is large, it seems that both modified Kneser-Ney and Kneser-Ney with ad hoc counting cannot effectively take full advantage of the training set and benefit from its increasing size, while modified Kneser-Ney with normal counting excels when the training set size is relatively large. This phenomenon is consistent across three different corpora.
2. **CTW.** We can see that the performance of CTW improves as the depth is increased. It is also clear that the improvement tends to zero as the depth gets larger. This phenomenon is independent of how the text file is binarised. On the one hand, the larger the depth, the more models CTW is mixing over, so asymptotically the performance should not get worse as the depth grows. On the other hand, the ‘true’ model of text dependency may be close to a Markov model with small depth/order, so it is reasonable to expect the performance to converge as the depth grows.
3. **CTW and Kneser-Ney.** It is obvious from Table 7 that Kneser-Ney outperforms CTW regardless of the used binarising method we use. Except for ASCII coding, we have essentially helped CTW by precompressing the text. This goes to the extreme when we use Huffman coding on the frequency of the words, in which case CTW achieves the closest performance to Kneser-Ney. Even though



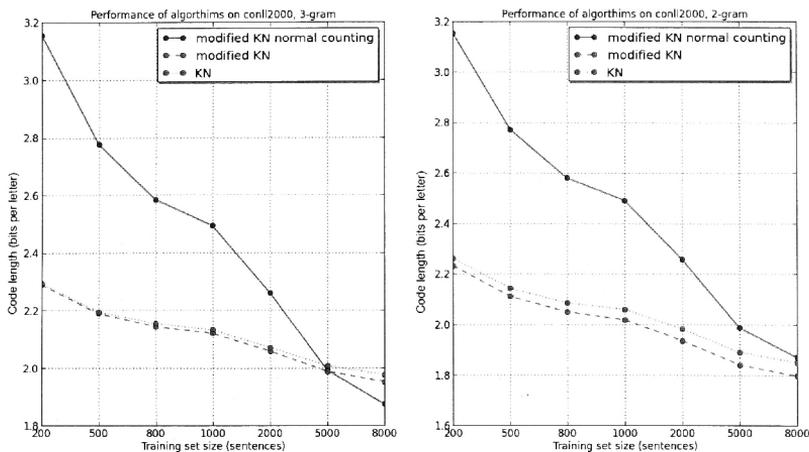
(a) Variants of Kneser-Ney on corpus abc.



(b) Variants of Kneser-Ney on corpus brown.

we have advantaged CTW by pre-compressing the file with Huffman coding on word frequency, it still doesn't outperform Kneser-Ney. This is partly because Kneser-Ney treats each work as an atomic object, however, CTW needs to take care of the inner dependence of a binary string. It seems, however, that Huffman coding on word length makes a fair comparison because the length of a word is its intrinsic property that is independent of the corpus it is in.

Interestingly enough, we would expect that CTW with 5 bits uniform coding



(c) Variants of Kneser-Ney on corpus conll2000.

Figure 3: Variants of Kneser-Ney methods on different corpora. 3-gram models are on the left, 2-gram models are on the right. The vertical axis is the average code length per letter in bits. The horizontal axis is the size of the training set in sentences.

should outperform CTW with plain ASCII coding, however, this is not the case for large depth, i.e. CTW with ASCII coding actually outperforms CTW with 5 bits uniform coding for depth larger than 48 and this is consistent across all corpora. We will see that this phenomenon is reversed in phase CTW.

6.1.3 Kneser-Ney and Phase CTW

Phase CTW. For a fixed-length coding of letters, each bit in the code is called a phase. In the previous chapter, we have used a single CTW for all bits in a code. In this section, we consider separating each code into bits and use separate context trees for each phase. The context of each bit consists of the previous bits from the complete string up to some depths d . For example, for ASCII coding we use 8 separate context trees, each for one position/phase in the ASCII code.

Results. The results of phase CTW compared to Kneser-Ney are shown in Table 8.

| Corpus | KN | D | ASCII (regular) | 5 bits (regular) | ASCII (phase) | 5 bits (phase) |
|--------|------|----|-----------------|------------------|---------------|----------------|
| abc | 1.94 | 2 | 7.435929704 | 3.807319104 | 4.11086164 | 3.354424684 |
| | | 4 | 7.036426167 | 3.749325598 | 3.369493906 | 2.921444207 |
| | | 16 | 1.775387898 | 1.37082257 | 1.728137512 | 0.925975908 |
| | | 32 | 0.499632608 | 0.304485751 | 0.497851983 | 0.235185756 |
| | | 48 | 0.271723286 | 0.286114853 | 0.269947772 | 0.217032691 |
| | | 64 | 0.254516202 | 0.284705186 | 0.252740676 | 0.215628662 |

| | | | | | | |
|-----------|------|-----|-------------|-------------|-------------|-------------|
| | | 72 | 0.253144816 | 0.284588231 | 0.251369289 | 0.215511786 |
| | | 96 | 0.251879964 | 0.284556668 | 0.250104436 | 0.215480224 |
| | | 108 | 0.251829914 | 0.284544415 | 0.250054386 | 0.215467705 |
| brown | 2.20 | 2 | 7.217994045 | 3.559628191 | 3.879133147 | 3.097215086 |
| | | 4 | 6.803493245 | 3.502626191 | 3.127894451 | 2.676870517 |
| | | 16 | 1.51343232 | 1.223379792 | 1.469065993 | 0.766950376 |
| | | 32 | 0.374312174 | 0.198197687 | 0.373068039 | 0.129623313 |
| | | 48 | 0.157993016 | 0.184292341 | 0.156753103 | 0.115938397 |
| | | 64 | 0.143900951 | 0.183813016 | 0.142661028 | 0.115462064 |
| | | 108 | 0.142582803 | 0.183802438 | 0.141342879 | 0.115451482 |
| conll2000 | 1.98 | 2 | 7.351813477 | 3.752065629 | 4.047244468 | 3.297517366 |
| | | 4 | 6.971352437 | 3.699587256 | 3.336167894 | 2.890452517 |
| | | 16 | 1.763421183 | 1.378213413 | 1.714945978 | 0.922824739 |
| | | 32 | 0.563277403 | 0.488912229 | 0.559342484 | 0.372916214 |
| | | 48 | 0.418660361 | 0.48109411 | 0.414738018 | 0.365208717 |
| | | 64 | 0.41186783 | 0.48067117 | 0.407945498 | 0.364789 |
| | | 108 | 0.411141942 | 0.480629683 | 0.407219609 | 0.364747513 |

Table 8: Experimental results of phase CTW compared to baseline Kneser-Ney and regular CTW. We have chosen modified Kneser-Ney on 3-gram model with maximum training sentences of each corpus. The number in the Kneser-Ney (KN) column is the average code length in bits per letter. We have reported regular and phase CTW with ASCII coding and 5 bits uniform coding. The column D is the varying depth of CTW and the numbers in the last four columns represent the code length difference to Kneser-Ney. The smaller the number, the better the performance.

Discussion. With increasing depth, the performance of phase CTW improves and tends to converge at large depths. Phase CTW outperforms normal CTW. This is consistent across different corpora and different codings. Separating context trees for each bit makes it easier for CTW to capture the characteristic of each bit in the code and thus yield smaller code length. Among the four different variations, phase CTW with 5 bit uniform coding is the closest to Kneser-Ney, but the code length difference varies among different corpora, with the smallest one being 0.181 and the largest one being 0.429. What is interesting is that for regular CTW, plain ASCII coding outperforms 5 bits uniform coding, which is different from what we might have expected. However, for phase CTW the opposite can be observed.

6.2 Large-alphabet CTW with Sparse Adaptive Dirichlet-multinomial Coding

In this section we consider large-alphabet CTW. Rather than working directly on the bit level, each node in a large-alphabet CTW represents a letter (byte). As we have discussed in Section 4.1, one of the problems we need to address is how much probability needs to be reserved for unseen symbols. We have implemented CTW with Sparse Adaptive Dirichlet-multinomial (SAD) coding and compared it to CTW with the Laplace and KT estimator. In CTW with SAD, each node has a different β_n^* , which is calculated based on the statistics associated with the node.

We have experimented with large-alphabet CTW with various depths and different alphabet sizes. In order to make a fair comparison with Kneser-Ney and considering the average number of letters in an English word, we have tested various CTW depths up to 16. We note, however, the actual symbols that occur in our text are lower-case English letters, i.e., only 26 symbols actually appear in the text.

Results. Experimental results of large-alphabet CTW are shown in Tables 9 to 11.

| Depth | Alphabet size | Hutter (base-line) | Laplace | KT |
|-------|---------------|--------------------|-------------|-------------|
| 0 | 32 | 4.168427607 | 4.168123713 | 4.168110285 |
| 0 | 64 | 4.168439692 | 4.168282165 | 4.168194513 |
| 0 | 128 | 4.168449116 | 4.168579219 | 4.168352965 |
| 0 | 256 | 4.168457745 | 4.169133785 | 4.16865002 |
| 0 | 512 | 4.168466043 | 4.170163986 | 4.169204585 |
| 0 | 1024 | 4.168474189 | 4.172066697 | 4.170234786 |
| 1 | 32 | 3.644612734 | 3.640724338 | 3.640356651 |
| 1 | 64 | 3.644910921 | 3.643414741 | 3.641831663 |
| 1 | 128 | 3.645145985 | 3.648281285 | 3.644522066 |
| 1 | 256 | 3.64536186 | 3.656993432 | 3.64938861 |
| 1 | 512 | 3.645569721 | 3.672393279 | 3.658100757 |
| 1 | 1024 | 3.645773885 | 3.699196603 | 3.673500604 |
| 2 | 32 | 3.251017127 | 3.233358373 | 3.226647963 |
| 2 | 256 | 3.261460342 | 3.380167084 | 3.316824955 |
| 2 | 1024 | 3.267512902 | 3.608676058 | 3.484202472 |
| 4 | 32 | 2.588403535 | 2.65190733 | 2.567903744 |
| 4 | 256 | 2.742268043 | 3.283311777 | 3.115748212 |
| 4 | 1024 | 2.81576277 | 3.605782129 | 3.466789663 |
| 6 | 32 | 2.442142941 | 2.607649666 | 2.500812885 |
| 6 | 256 | 2.674927488 | 3.282770517 | 3.113200867 |
| 6 | 1024 | 2.770630007 | 3.605782273 | 3.4668005 |
| 8 | 32 | 2.434400699 | 2.607268896 | 2.500044646 |
| 8 | 256 | 2.673701575 | 3.282777561 | 3.113212848 |
| 8 | 1024 | 2.769943823 | 3.605782316 | 3.46680068 |
| 16 | 32 | 2.433121507 | 2.607266642 | 2.49999745 |
| 16 | 256 | 2.673732105 | 3.282778342 | 3.113215264 |
| 16 | 1024 | 2.769974428 | 3.605782318 | 3.466800696 |

Table 9: Experimental results of large-alphabet CTW on corpus abc. SAD, Laplace and KT estimators are implemented and used as underlying estimators. Different base alphabet sizes are considered. The numbers in the last three columns are the code length per letter. Smaller numbers imply better performance.

| Depth | Alphabet size | Hutter (base-line) | Laplace | KT |
|-------|---------------|--------------------|-------------|-------------|
| 0 | 32 | 4.171498521 | 4.171285334 | 4.171275776 |
| 0 | 64 | 4.171506868 | 4.171398397 | 4.171335762 |
| 0 | 128 | 4.171513376 | 4.171610816 | 4.171448826 |
| 0 | 256 | 4.171519335 | 4.172008342 | 4.171661244 |
| 0 | 512 | 4.171525066 | 4.172748881 | 4.17205877 |
| 0 | 1024 | 4.171530692 | 4.174121048 | 4.172799309 |

| | | | | |
|----|------|-------------|-------------|-------------|
| 1 | 32 | 3.63035764 | 3.627584277 | 3.627318156 |
| 1 | 64 | 3.630565441 | 3.629540111 | 3.628385786 |
| 1 | 128 | 3.630729137 | 3.633096123 | 3.63034162 |
| 1 | 256 | 3.630879417 | 3.639501155 | 3.633897633 |
| 1 | 512 | 3.631024095 | 3.650905958 | 3.640302665 |
| 1 | 1024 | 3.631166188 | 3.670930278 | 3.651707468 |
| 2 | 32 | 3.262561668 | 3.247793357 | 3.242846405 |
| 2 | 256 | 3.270209336 | 3.359738806 | 3.31134109 |
| 2 | 1024 | 3.27462424 | 3.549625722 | 3.445676775 |
| 4 | 32 | 2.72274913 | 2.746707182 | 2.677021519 |
| 4 | 256 | 2.845097474 | 3.271198727 | 3.133259272 |
| 4 | 1024 | 2.902269253 | 3.547139791 | 3.426493011 |
| 6 | 32 | 2.598980311 | 2.708036537 | 2.617089186 |
| 6 | 256 | 2.791648382 | 3.271030676 | 3.132182057 |
| 6 | 1024 | 2.867963557 | 3.547139929 | 3.42649914 |
| 8 | 32 | 2.593458404 | 2.70787415 | 2.616583004 |
| 8 | 256 | 2.790694332 | 3.271032914 | 3.132195892 |
| 8 | 1024 | 2.867456985 | 3.547139968 | 3.426499271 |
| 16 | 32 | 2.593105057 | 2.707913672 | 2.616650502 |
| 16 | 256 | 2.790747185 | 3.271033122 | 3.132196818 |
| 16 | 1024 | 2.867488428 | 3.547139968 | 3.426499283 |

Table 10: Experiment results of large-alphabet CTW on corpus brown. The setting is the same as in Table 9

| Depth | Alphabet size | Hutter (base-line) | Laplace | KT |
|-------|---------------|--------------------|-------------|-------------|
| 0 | 32 | 4.181539507 | 4.180682931 | 4.180647642 |
| 0 | 64 | 4.181575831 | 4.181112172 | 4.180877294 |
| 0 | 128 | 4.181604155 | 4.181910998 | 4.181306535 |
| 0 | 256 | 4.181630088 | 4.183389811 | 4.182105361 |
| 0 | 512 | 4.181655028 | 4.186110255 | 4.183584174 |
| 0 | 1024 | 4.18167951 | 4.191077361 | 4.186304617 |
| 1 | 32 | 3.664996405 | 3.654973827 | 3.654005529 |
| 1 | 64 | 3.665871721 | 3.661884426 | 3.657849976 |
| 1 | 128 | 3.666564016 | 3.674167773 | 3.664760575 |
| 1 | 256 | 3.667200406 | 3.695695414 | 3.677043922 |
| 1 | 512 | 3.667813414 | 3.732772722 | 3.698571563 |
| 1 | 1024 | 3.668415629 | 3.795278358 | 3.73564887 |
| 2 | 32 | 3.297432769 | 3.259424593 | 3.244554243 |
| 2 | 256 | 3.323582513 | 3.534250334 | 3.43357441 |
| 2 | 1024 | 3.33832068 | 3.792593005 | 3.690184039 |
| 4 | 32 | 2.747128368 | 2.847442282 | 2.73969143 |
| 4 | 256 | 2.978419162 | 3.520120794 | 3.371902547 |
| 4 | 1024 | 3.075131872 | 3.792616128 | 3.689417577 |
| 6 | 32 | 2.636693247 | 2.832323649 | 2.711207616 |
| 6 | 256 | 2.941238375 | 3.520155989 | 3.37199363 |
| 6 | 1024 | 3.055851171 | 3.792616128 | 3.689418372 |
| 8 | 32 | 2.630823692 | 2.83265242 | 2.711383613 |
| 8 | 256 | 2.941093683 | 3.520157614 | 3.372001115 |
| 8 | 1024 | 3.05584711 | 3.792616128 | 3.689418488 |

| | | | | |
|----|------|-------------|-------------|-------------|
| 16 | 32 | 2.629544094 | 2.832695192 | 2.711421709 |
| 16 | 256 | 2.94114957 | 3.520157694 | 3.372001818 |
| 16 | 1024 | 3.055888147 | 3.792616128 | 3.689418489 |

Table 11: Experiment results of large-alphabet CTW on corpus conll2000. The setting is the same as in Table 9

Discussion. For all three methods, the code lengths decrease as depths increases, while the code lengths increase with the base alphabet size. These two observations coincide with our expectation. SAD outperforms Laplace and KT estimators on every tested depth, alphabet size and corpus. However, Kneser-Ney method still outperforms SAD even at depth 26 with alphabet size 26. SAD adapts to the alphabet that actually appears in the text and is not influenced too much by the size of the base alphabet size. For example, for corpus abc, CTW with depth 16, SAD has a 0.38 difference in the code length when the alphabet size goes from 26 to 1024, while Laplace has a 1.07 difference and KT 1.03.

6.3 Large-alphabet CTW on Artificial Data

We now investigate the behaviour of large-alphabet CTW with various estimators on artificial data.

Data generation. The alphabet size D is set to be $2^{13} = 8192$. We have generated three kinds of artificial data. In the first, we sampled $\theta_1, \dots, \theta_m$ uniformly from the $m-1$ -dimensional probability simplex and set θ_{m+1} up to θ_D to zero. So effectively we limit the actual alphabet size to m . We then sample $x_{1:n}$ i.i.d from $\theta = (\theta_1, \dots, \theta_D)$ with n set to $2^{13} = 8192$. We should note, however, unless $n \gg m$ or $D \gg n$, we usually don't have a sequence with exactly m distinct symbols.

In the second experiment, we choose θ to be Zipf-distributed. The motivation of Zipf's distribution is that it mimics some corpus of natural language, i.e., the frequency of any word is inversely proportional to its rank in the frequency table. We set $\theta_i \propto i^{-\gamma}$ (with proper normalisation) with varying Zipf exponent $\gamma > 0$. Large γ means θ will get discounted quickly, which results in smaller used alphabet.

As a special case, we have set $\gamma=0$, which results in a uniform distribution over the whole alphabet.

Results. The experimental results are shown in Tables 12 and 13. Another related experiment is carried out and the results are shown in Table 14.

| $m=2^i$ | SAD | Laplace | KT |
|---------|---------|---------|---------|
| 2^1 | 1480.03 | 13180.5 | 9653.96 |
| 2^2 | 6975.76 | 19040.2 | 15521.3 |
| 2^3 | 15875.8 | 27049.4 | 23545.7 |
| 2^4 | 20929.5 | 31874.8 | 28395.6 |

| | | | |
|-------|---------|---------|---------|
| 2^5 | 27420.7 | 37769.7 | 34335.5 |
| 2^6 | 34069.9 | 43128.5 | 39772 |
| 2^7 | 43770.1 | 49438.9 | 46227.4 |
| 2^8 | 54289.6 | 54326.8 | 51339.7 |
| 2^9 | 67324.6 | 59490.3 | 56871.8 |

Table 12: Large-alphabet CTW performance with different estimators. The sequence length is fixed to $n=2^{13}=8192$ and total alphabet size $D=2^{13}=8192$. The results are shown for varying actual used alphabet. The numbers in the last three columns are the code length of different estimators. The depth of the tree is set to 16.

| γ | SAD | Laplace | KT |
|----------|---------|---------|---------|
| 0 | 74410 | 73817.9 | 73817.9 |
| 0.1 | 74420.6 | 73818.3 | 73818.6 |
| 0.2 | 74420.6 | 73817.2 | 73816.6 |
| 0.3 | 74393.9 | 73815.5 | 73813.9 |
| 0.4 | 74334 | 73774 | 73799.4 |
| 0.5 | 74229 | 73121.4 | 73527.8 |
| 0.6 | 72888.4 | 71492.1 | 71612.4 |
| 0.7 | 71752.2 | 70151.8 | 70028.6 |
| 0.8 | 69144.8 | 67825.2 | 67411.9 |
| 0.9 | 65030.5 | 63277.3 | 62258.6 |
| 1 | 59323 | 58649.2 | 57238.8 |
| 1.1 | 54583.3 | 54710 | 52879.4 |
| 1.2 | 45806.5 | 47285.8 | 44858.5 |
| 1.3 | 39191.9 | 43086.1 | 40529.3 |
| 1.4 | 32018.6 | 37468.1 | 34521.8 |
| 1.5 | 27733.7 | 34323.1 | 31240.5 |
| 1.6 | 24160 | 31916.8 | 28765.1 |
| 1.7 | 22328.5 | 30440.7 | 27238.9 |
| 1.8 | 18759.1 | 28114 | 24836.6 |
| 1.9 | 16894 | 26559.2 | 23230.4 |
| 2 | 14463.3 | 24609.3 | 21245.9 |

Table 13: Large-alphabet CTW performance with different estimators. The sequence length is fixed to $n=2^{13}=8192$ and total alphabet size $D=2^{13}=8192$. The results are shown for varying γ . The numbers in the last three columns are the code length of different estimators. The depth of the tree is set to 16.

| file | unique byte | SAD | Laplace | KT |
|------|-------------|-----|---------|----|
|------|-------------|-----|---------|----|

| | | | | |
|--------|-----|-------------|-------------|-------------|
| bib | 81 | 2.554339692 | 3.463086182 | 3.287594622 |
| book1 | 82 | 2.692666277 | 3.103120309 | 2.976561147 |
| book2 | 96 | 2.468006573 | 3.243156993 | 3.061018727 |
| ge | 256 | 5.024632854 | 4.987565302 | 4.854468696 |
| obj1 | 256 | 4.034621057 | 5.0187149 | 4.769524821 |
| obj2 | 256 | 2.894387533 | 4.017523458 | 3.773957657 |
| paper1 | 95 | 3.125416738 | 3.833884697 | 3.653470425 |
| paper2 | 91 | 3.059909001 | 3.531632247 | 3.400612638 |
| pic | 159 | 0.891418006 | 0.907320272 | 0.882832975 |
| prog | 92 | 2.878394366 | 3.78047352 | 3.564002641 |
| progl | 87 | 2.012269231 | 3.132896197 | 2.93340523 |
| progp | 89 | 1.818428076 | 3.225695784 | 2.929146573 |
| trans | 99 | 1.802350099 | 3.44992225 | 3.201895669 |

Table 14: The performance of large-alphabet CTW with different estimators. The unique byte column shows the number of unique bytes in the files. For ASCII text files, this number is relatively low, because they only contain letters and punctuations, while for binary files, e.g. executable files and pictures, the number of unique bytes is high. The total alphabet size is set to 256. The numbers in the last three columns are actual code length per byte. The depth of the tree is set to 16.

Discussion. We find that SAD performs well compared with Laplace and KT when the actual used alphabet is small compared to the total alphabet, e.g., when m is small and γ large. On the other hand, when the actual used alphabet is close to the total alphabet, e.g. when m is large and γ closes to 0, we find that the performance of SAD is comparable to Laplace and KT. To further investigate this, we tested large-alphabet CTW with these three estimators on the Calgary corpus. We calculate the number of unique bytes in the files and report the performance of these estimators in Table 14. The total alphabet size is set to 256. The results are consistent with what we find in artificial data: SAD outperforms Laplace and KT when the actual used alphabet is small, for example, for file ‘progl’, SAD outperforms Laplace and KT by 1 bit per byte. However, when the number of unique bytes is relatively high, e.g. as in file ‘pic’ and ‘ge’, the performance of SAD is comparable to the other two. These results are consistent with the results in [Hut13].

6.4 Applying SAD to Kneser-Ney

We have seen the power of SAD and wish to bring this idea to Kneser-Ney’s method. Given a sequence $x_{1:n} := x_1x_2\dots x_n$ from some large alphabet \mathcal{X} , we study the back off version of Nyer-Ney method applied to a unigram model, which is given by the following

$$P(x_{t+1}|x_{1:t}) = \begin{cases} \frac{\#_{t+1}(x_{t+1})-d}{dm^t q_t^{\dagger}(x_{t+1})} & \#_{t+1}(x_{t+1}) > 0 \\ \frac{d}{dm^t q_t^{\dagger}(x_{t+1})} & \#_{t+1}(x_{t+1}) = 0 \end{cases}$$

where $0 < d < 1$ is the discounting parameter and $m^t := |\mathcal{A}^t| = |\{x_1, x_2, \dots, x_t\}|$ is the number of unique symbols appeared up to time step t . $\#_{t+1}(x_{t+1})$ is the number of occurrence of x_{t+1} in $x_{<t+1}$. $q(\cdot)$ is some distribution over \mathcal{X} such that $\sum_{x_{t+1}} q(x_{t+1}) \leq 1$. We also use the following notations in our discussion. $m = |\mathcal{A}| = |\{x_1, x_2, \dots, x_n\}|$ denotes the total of number of unique symbols in $x_{1:n}$; n_i denotes the nubmer of occurrence of symbol i in $x_{1:n}$. \mathcal{A}_i (\mathcal{A}_{i+}) is the set of symbols, in which each symbol appears exactly (larger or equal to) i times. Relatedly $m_i = |\mathcal{A}_i|$ and $m_{i+} = |\mathcal{A}_{i+}|$. We are interested in having an adaptive discounting parameter that depends on how many symbols we have seen. Such an optimal adaptive discounting parameter can be obtained by maximising the following joint probability

$$\begin{aligned} P_{KN}(x_{2:n}) &= \prod_{t=1}^{n-1} P_{KN}(x_{t+1}|x_{1:t}) \\ &= \prod_{t=1}^{n-1} \frac{1}{t} \prod_{t \in Old} (n_{x_{t+1}} - d) \prod_{t \in New} dm^t q(x_{t+1}) \end{aligned}$$

where $New := \{t = 0 \dots n-1 | x_{t+1} \notin \mathcal{A}^t\}$ and $Old := \{t = 0 \dots n-1 | x_{t+1} \in \mathcal{A}^t\}$. The middle term, for a fixed $n_j \geq 2$, is a product from $(1-d)$ up to (n_j-1-d) . Since

$$\frac{\Gamma(z+n)}{\Gamma(z)} = \prod_{k=0}^{n-1} (z+k), \quad n \in \mathbb{N}$$

We can rewrite the joint probability in terms of Gamma function with $z = 1-d$

$$P_{KN}(x_{2:n}) = \frac{1}{\Gamma(n)} \prod_{j \in \mathcal{A}_{2+}} \frac{\Gamma(z+n_j-1)}{\Gamma(z)} \prod_{j \in \mathcal{A}} (1-z)m^t q(x_{t+1})$$

The total code length function is then

$$CL_{KN} = -\ln \frac{1}{\Gamma(n)} - \sum_{j \in \mathcal{A}_{2+}} \ln \frac{\Gamma(z+n_j-1)}{\Gamma(z)} - \sum_{j \in \mathcal{A}} \ln(1-z)m^t q(x_{t+1})$$

Now take derivative of CL_{KN} with respect to z , setting it to 0, we get

$$\sum_{j \in \mathcal{A}_{2+}} \Psi(z+n_j-1) - \sum_{j \in \mathcal{A}_{2+}} \Psi(z) = \frac{m}{1-z}$$

where $\Psi(\cdot)$ is the diGamma function. We can approximate this by

$$\begin{aligned} \sum_{j \in \mathcal{A}_{2+}} \Psi\left(z + \frac{n_j-1}{z}\right) - \Psi(z) &= \frac{m}{1-z} \\ \sum_{j \in \mathcal{A}_{2+}} \ln\left(z + \frac{n_j-1}{z}\right) - \Psi(z) &= \frac{m}{1-z} \\ \sum_{j \in \mathcal{A}_{2+}} \ln\left(1 + \frac{n_j-1}{z}\right) + \ln z - \Psi(z) &= \frac{m}{1-z} \end{aligned}$$

where we have used the approximation $\Psi(\cdot) \approx \ln(\cdot)$. We set $\ln z - \Psi(z)$ to zero for approximation. Then we have

$$\sum_{j \in \mathcal{A}_{2+}} \ln\left(1 + \frac{n_j - 1}{1 - d}\right) = \frac{m}{d}$$

Now we further assume that for all $j \in \mathcal{A}$, j is either of the following types

1. $j \in \mathcal{A}_1$, i.e. j just appears once.
2. if $j \in \mathcal{A}_{2+}$, then $j \in \mathcal{A}_{n^*}$, i.e. if j appear more than once, it must appear n^* times.
We also have $n^* = \frac{n - m_1}{m_{2+}}$

Then we can further simplify this equation to

$$m^{2+} \ln\left(1 + \frac{n^* - 1}{1 - d}\right) = \frac{m}{d} \quad (18)$$

This problem has been simplified into a standard yet nontrivial mathematical equation that we need to solve for d .

Heuristic. Although I'm not able to work out a closed form solution, I have a heuristic for Equation (18).

$$d = \frac{m}{(m - m_1) \ln \frac{n - m_1}{m - m_1}} \quad (19)$$

Two other heuristics below that were proposed by others in my research group are used for comparison.

$$d' = \frac{m}{2m - m_1 + 1} \quad d'' = \frac{m}{n + 1} \quad d_{KN} = \frac{m_1}{m_1 + 2m_2}$$

where d_{KN} is the discounting parameter suggested by Ney.

Justification of my heuristic. First we consider several intuitions, i.e. what we expect d to be under different asymptotic behaviours of m , m_1 and n . We would reasonably hope that a good heuristic would exhibit all the intuitions.

1. When n is much larger than m , we would want to reserve less probability for unseen symbols, i.e. we would expect d to be small. My heuristic does yield a small d when n is much larger than m (fix m and m_1 and let n tend to infinity). d'' has the property, whereas d' does not depend on n . A refined version of this intuition will be provided later.
2. If m_1 is large compared to $m_{2+} = m - m_1$, then we would like to reserve more probability for unseen symbol, the reason being that the first time we encounter a new symbol, we redistribute the held-out probabilities. On the other hand, when m_{2+} is large we would like to reserve less probability, because from the second time we come across a symbol, we have used a discounted maximum likelihood estimator. My heuristic captures this intuition, however, d'' does not.

We now examine some asymptotic cases

- When $m_1=0$, my heuristic becomes

$$d = \frac{1}{\ln \frac{n}{m}}$$

which roughly reduces to β^* used by SAD, considering β^* is the collective discount and d is individual discount.

- If $n \rightarrow \infty$, $m < \infty$ ($\Rightarrow m_1 < \infty$ and $m_{2+} < \infty$), then $d \rightarrow 0$ with $O(1/\ln n)$, which is what we want.
- If $n \rightarrow \infty$, $m \propto n$ and $m_{2+} < \infty$ ($\Rightarrow m_1 \rightarrow \infty$), then $d \rightarrow \infty^5$, which is also what we want.
- If $n \rightarrow \infty$, $m \propto n$ ($m=cn$), $m_1 < \infty$, $m_{2+} \propto n$ ($m_{2+}=c'n$) then $d = \frac{c}{c' \ln \frac{n}{c}}$ (somewhere between 0 and 1, depending on c and c').

Empirical justification. We have tested the aforementioned discounting parameters on the Calgary Corpus, and the experimental results are shown in Table 15.

| file | d | d' | d'' | d_{KN} |
|--------|--------------------|-------------|-------------|--------------------|
| bib | 579447.5244 | 579595.8113 | 580339.0914 | 579461.8551 |
| book1 | 3481222.115 | 3481390.373 | 3482605.951 | 3481693.784 |
| book2 | 2928631.723 | 2928853.657 | 2930261.568 | 2930094.358 |
| geo | 579812.6773 | 580165.424 | 582177.8465 | 583858.7544 |
| news | 1958105.619 | 1958342.537 | 1959525.687 | 1959600.619 |
| obj1 | 129124.8454 | 129294.5952 | 130887.9617 | 129296.9900 |
| obj2 | 1546946.369 | 1547382.815 | 1550275.673 | 1550947.687 |
| paper1 | 265718.6174 | 265828.1258 | 266621.9505 | 265710.0736 |
| paper2 | 379018.4999 | 379112.7036 | 380004.6417 | 379119.0332 |
| pic | 622094.4247 | 622096.3187 | 625676.3681 | 622223.0408 |
| progc | 206759.1905 | 206885.1354 | 207527.1794 | 206888.6851 |
| progl | 342539.8018 | 342651.3373 | 343669.9613 | 343899.483 |
| progp | 241196.9393 | 241303.7778 | 242047.1719 | 241213.6001 |
| trans | 519319.2058 | 519479.9265 | 520368.7756 | 519329.9752 |

Table 15: Performance of different discounting parameters on the Calgary Corpus. The numbers in this table are the total code length. Smaller is better. The best performance among the four has been bolded.

⁵The domain of the discounting parameter d is $(0,1)$. So in practice, d is set to a number on a physical computer (subject to the finite precision of IEEE arithmetic) in $(0,1)$ that is the closest to the calculated value given by Equation (19).

The experimental results show that d outperforms other three discounting parameters for all but one files. Moreover, d_{KN} does only slightly better than d on file 'paper1', which is the single file that d_{KN} outperforms d .

7 Conclusion

Most raw data is not binary, but over some often large and structured alphabet. Sometimes it is convenient to deal with binarised data sequences, but typically exploiting the original structure of the data significantly improves performance in many practical applications. The main focus of this thesis was to investigate the relationship between binary data compression techniques and natural language modelling techniques.

As a foundation of compression, we surveyed Kolmogorov complexity and Martin-Löf randomness. We studied the size of the set of random and nonrandom reals in $[0,1]$ from three different perspectives: set theoretic, measure theoretic and topological perspective. We summarised sixteen possible combinations of characterising the size of a set and provided examples.

We conducted an empirical comparison between Kneser-Ney (KN) variants with regular Context Tree Weighting algorithm (CTW) and phase CTW, and with large-alphabet CTW with different estimators. We found that both Kneser-Ney and modified Kneser-Ney outperformed modified Kneser-Ney with normal counting when the training set is relatively small and modified Kneser-Ney slightly outperforms Kneser-Ney. However, when the size of the training set is large, it seems that both modified Kneser-Ney and Kneser-Ney with ad hoc counting cannot effectively take full advantage of the training set and benefit from its increasing size, while modified Kneser-Ney with normal counting excels when the training set size is relatively large.

We also apply the idea of Hutter's adaptive sparse Dirichlet-multinomial coding to the KN method and provide a heuristic to make the discounting parameter adaptive. The KN with this adaptive discounting parameter outperforms the traditional KN method on the Large Calgary corpus.

References

- [Bal97] Vijay Balasubramanian. Statistical inference, occam's razor, and statistical mechanics on the space of probability distributions. *Neural computation*, 9(2):349–368, 1997.
- [BH10] Wray Buntine and Marcus Hutter. A bayesian view of the poisson-dirichlet process. *arXiv preprint arXiv:1007.0296*, 2010.
- [BJM83] Lalit R. Bahl, Frederick Jelinek, and Robert L. Mercer. A maximum likelihood approach to continuous speech recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 5(2):179–190, 1983.
- [Cal02] Cristian Calude. *Information and randomness: an algorithmic perspective*. Springer, 2002.
- [CG96] S. F. Chen and J. Goodman. An empirical study of smoothing techniques for language modelling. In *Proceedings of the Thirty-Fourth Annual Meeting of the Association for Computational Linguistics*, pages 310–318, San Francisco, 1996. Morgan Kaufmann.
- [Cha71] Gregory J. Chaitin. Computational complexity and Gödel's incompleteness theorem. *ACM SIGACT News*, 9:11–12, April 1971.
- [Cha75a] Gregory J. Chaitin. Randomness and mathematical proof. *Scientific American*, 232:47–52, 1975.
- [Cha75b] Gregory J. Chaitin. A theory of program size formally identical to information theory. *Journal of the Association for Computing Machinery*, 22:329–340, 1975.
- [Cha82] Gregory J. Chaitin. Gödel's theorem and information. *International Journal of Theoretical Physics*, 22:941–954, 1982.
- [Cha86] Gregory J. Chaitin. Randomness and Gödel's theorem. *Mondes en Développement*, 14(54-55):125–128, 1986.
- [CN97] Cristian S. Calude and André Nies. Chaitin Omega numbers and strong reducibilities. *Journal of Universal Computer Science*, 3:1162–1166, 1997.
- [CW84] John G. Cleary and Ian H. Witten. Data compression using adaptive coding and partial string matching. *IEEE Transactions on Communications*, 32:396–402, 1984.
- [Dav73] L. D. Davisson. Universal noiseless coding. *IEEE Transactions on Information Theory*, 19(6):783–795, 1973.
- [Fit66] B. M. Fitingof. Optimal encoding for unknown and changing statistics of messages. *Problems of Information Transmission*, 2(2):3–11, 1966.
- [Fit67] B. M. Fitingof. The compression of discrete information. *Problems of Information Transmission*, 3(3):28–36, 1967.
- [Gác74] Péter Gács. On the symmetry of algorithmic information. *Soviet Mathematics Doklady*, 15:1477–1480, 1974.
- [Gác07] Péter Gács. Lecture notes on descriptonal complexity and randomness. Technical report, Computer Science Department Boston University, 1988–2007.
- [GC89] Péter Gács and Gregory J. Chaitin. Algorithmic information theory. *Journal of Symbolic Logic*, 54:624–627, 1989.
- [GOO63] RL GOODSTEIN. On formally undecidable propositions of principia mathematica and related systems. *Philosophical Books*, 4(1):17–18, 1963.
- [GR03] J.K. Ghosh and R.V. Ramamoorthi. *Bayesian nonparametrics*. New York : Springer, 2003.
- [Grü07] Peter D. Grünwald. *The Minimum Description Length Principle*. The MIT Press, 2007.
- [GWT10] Jan Gasthaus, Frank Wood, and Yee Whye Teh. Lossless compression based on the sequence memoizer. In *Data Compression Conference (DCC), 2010*, pages 337–345. IEEE, 2010.
- [Hut05] Marcus Hutter. *Universal Artificial Intelligence: Sequential Decisions based on Algorithmic Probability*. Springer, Berlin, 2005.
- [Hut09] Marcus Hutter. Open problems in universal induction & intelligence. *Algorithms*, 3(2):879–906, 2009.

- [Hut13] Marcus Hutter. Sparse adaptive dirichlet-multinomial-like processes. *arXiv preprint arXiv:1305.3671*, 2013.
- [Jef48] H Jeffreys. *Theory of Probability*. Clarendon Press, Oxford, second edition edition, 1948.
- [Jef61] Harold Jeffreys. *Theory of probability*, 1961.
- [Jel68] Frederick Jelinek. *Probabilistic Information Theory*. McGraw Hill, New York, 1968.
- [JM80] Frederick Jelinek and Robert L. Mercer. Interpolated estimation of markov source parameters from sparse data. In *Proceedings of the Work-shop on Pattern Recognition in Practice*, May 1980.
- [Joh32] W.E Johnson. Probability: deductive and inductive problems. *Mind*, 41:421–423, 1932.
- [KN95] R. Kneser and H. Ney. Improved backing-off for m-gram language modeling. In *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 181–184, 1995.
- [KR95] Robert E Kass and Adrian E Raftery. Bayes factors. *Journal of the american statistical association*, 90(430):773–795, 1995.
- [KT81] Rafail E. Krichevsky and V. K. Trofimov. The performance of universal encoding. *IEEE Transactions on Information Theory*, 27(2):199–207, 1981.
- [LCL+03] Ming Li, Xin Chen, Xin Li, Bin Ma, and Paul M. B. Vitányi. The similarity metric. In *SODA '03: Proceedings of the fourteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 863–872, 2003.
- [Lev74] Leonid A. Levin. Laws of information conservation (non-growth) and aspects of the foundation of information theory. *Problems of Information Transmission*, 10(3):206–210, 1974.
- [Lid20] G.J. Lidstone. Note on the general case of the bayes-laplace formula for inductive or a posteriori probabilities. *Transactions of the Faculty of Actuaries*, 8:182–192, 1920.
- [LV08] Ming Li and Paul M.B. Vitányi. *An Introduction to Kolmogorov Complexity and Its Applications*. Springer, 3rd edition, 2008.
- [ML66] Per Erik Rutger Martin-Löf. The definition of random sequences. *Information and Control*, 9:602–619, 1966.
- [NE91] H. Ney and U. Essen. On smoothing techniques for bigram-based natural language modelling. In *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 825–829, 1991.
- [NK94] Essen U. Ney, H. and R Kneser. On structuring probabilistic dependences in stochastic language modeling. *Computer Speech and Language*, 8:1–38, 1994.
- [OSI02] Takumi Okazaki, Kunihiro Sadakane, and Hiroshi Imai. Data compression method combining properties of ppm and ctw. In *Progress in Discovery Science*, pages 268–283. Springer, 2002.
- [Pie12] John Robinson Pierce. *An introduction to information theory: symbols, signals and noise*. DoverPublications. com, 2012.
- [PY97] Jim Pitman and Marc Yor. The two-parameter poisson-dirichlet distribution derived from a stable subordinator. *The Annals of Probability*, 25(2):855–900, 1997.
- [RF10] H.L. Royden and P.M. Fitzpatrick. *Real Analysis*. Pearson, 4th edition, 2010.
- [RH11] Samuel Rathmanner and Marcus Hutter. A philosophical treatise of universal induction. *Entropy*, 13(6):1076–1136, 2011.
- [RST96] Dana Ron, Yoram Singer, and Naftali Tishby. The power of amnesia: learning probabilistic automata with variable memory length. *Machine Learning*, pages 117–150, 1996.
- [Sai04] Amir Said. *Introduction to arithmetic coding - theory and practice*, chapter 5, pages 101–152. Academic Press, 2004.
- [Say03] Khalid Sayood. *Lossless Compression Handbook*. Academic Press, 2003.
- [Sch73] C.P. Schnorr. Process complexity and effective random tests. *Journal of Computer and System Sciences*, 7:376–388, 1973.

- [Sch78] Gideon Schwarz. Estimating the dimension of a model. *The annals of statistics*, 6(2):461–464, 1978.
- [Sha48] Claude E. Shannon. A mathematical theory of communication. *The Bell System Technical Journal*, 27:379–423, 1948.
- [Sol64] Ray J. Solomonoff. A formal theory of inductive inference: parts 1 and 2. *Information and Control*, 7:1–22, 1964.
- [Sol75] Ray J. Solomonoff. Inductive inference theory—a unified approach to problems in pattern recognition and artificial intelligence. In *4th International Conference on Artificial Intelligence*, pages 274–280, 1975.
- [ST01] M Stassen and T Tjalkens. A parallel implementation of the ctw compression algorithm. In *22nd Symposium on Information Theory in the Benelux*, pages 85–92, 2001.
- [Sta01] Russell K. Standish. On complexity and emergence. *Complexity International*, 9, 2001.
- [Teh06] Yee Whye Teh. A hierarchical bayesian language model based on pitman-yor processes. In *Proceedings of the 21st International Conference on Computational Linguistics and the 44th annual meeting of the Association for Computational Linguistics*, pages 985–992. Association for Computational Linguistics, 2006.
- [VH12] Joel Veness and Marcus Hutter. Sparse sequential dirichlet coding. *arXiv preprint arXiv:1206.3618*, 2012.
- [WH93] Gary L Wise and Eric B Hall. *Counterexamples in probability and real analysis*. Oxford University Press Oxford, 1993.
- [WST95] Frans M. J. Willems, Yuri M. Shtarkov, and Tjalling Tjalkens. The context tree weighting method: Basic properties. *IEEE Transactions on Information Theory*, 41:653–664, 1995.
- [WST97] Frans M. J. Willems, Yuri M. Shtarkov, and Tjalling Tjalkens. Reflections on "the context-tree weighting method: Basic properties". *Newsletter of the IEEE Information Theory Society*, 1997.
- [WT97] Frans MJ Willems and Tjalling Jan Tjalkens. *Complexity reduction of the context-tree weighting algorithm: A study for KPN research*. Euler Institute of Discrete Mathematics and its Applications, 1997.
- [Zor04] Vladimir A. Zorich. *Mathematical Analysis I*. Springer, 2004.

A Explicit Equations for Kneser-Ney

Here are a list of explicit equations used in our experiments. We implemented basic Kneser-Ney, and modified Kneser-Ney and modified Kneser-Ney with normal counting. There are many variations of Kneser-Ney method, and different researchers deal with the base case differently. For clarity, we have specified each algorithm by writing out the recursion formulae. we have also included 3,2-gram Kneser-Ney with normal counting in the following, though we include it in the experiments.

3,2-gram Kneser-Ney (KN2, KN3).

$$P_{KN_3}(x_i|x_{i-2:i-1}) = \begin{cases} \frac{\max\{N_{1+(\cdot x_{i-2:i-1})-D^{(3)},0}\}}{N_{1+(\cdot x_{i-2:i-1})}} \\ + \frac{D^{(3)}N_{1+(\cdot x_{i-2:i-1})}}{N_{1+(\cdot x_{i-2:i-1})}} P_{KN_2}(x_i|x_{i-1}) & N_{1+(\cdot x_{i-2:i-1})} > 0 \\ P_{KN_2}(x_i|x_{i-1}) & \text{otherwise} \end{cases}$$

$$P_{KN_2}(x_i|x_{i-1}) = \begin{cases} \frac{\max\{N_{1+(\cdot x_{i-1})-D^{(2)},0}\}}{N_{1+(\cdot x_{i-1})}} \\ + \frac{D^{(2)}N_{1+(\cdot x_{i-1})}}{N_{1+(\cdot x_{i-1})}} P_{KN_1}(x_i) & N_{1+(\cdot x_{i-1})} > 0 \\ P_{KN_1}(x_i) & \text{otherwise} \end{cases}$$

$$P_{KN_1}(x_i) = \begin{cases} \frac{\max\{N_{1+(\cdot x_i)-D^{(1)},0}\}}{N_{1+(\cdot)}} \\ + \frac{D^{(1)}N_{1+(\cdot)}}{N_{1+(\cdot)}} \frac{1}{|\mathcal{X}|} & N_{1+(\cdot)} > 0 \\ \frac{1}{|\mathcal{X}|} & \text{otherwise} \end{cases}$$

where

$$D^{(i)} = \frac{n_1^{(i)}}{n_1^{(i)} + 2n_2^{(i)}}$$

with $n_j^{(i)}$ the number of i -grams that occurs exactly j times.

3,2-gram Kneser-Ney with normal counting (KNNC2, KNNC3).

$$P_{KNNC_3}(x_i|x_{i-2:i-1}) = \begin{cases} \frac{\max\{\#(x_{i-2:i-1})-D^{(3)},0\}}{\#(x_{i-2:i-1})} \\ + \frac{D^{(3)}N_{1+(x_{i-2:i-1})}}{\#(x_{i-2:i-1})} P_{KNNC_2}(x_i|x_{i-1}) & \#(x_{i-2:i-1}) > 0 \\ P_{KNNC_2}(x_i|x_{i-1}) & \text{otherwise} \end{cases}$$

$$P_{KNNC_2}(x_i|x_{i-1}) = \begin{cases} \frac{\max\{\#(x_{i-1})-D^{(2)},0\}}{\#(x_{i-1})} \\ + \frac{D^{(2)}N_{1+(x_{i-1})}}{\#(x_{i-1})} P_{KNNC_1}(x_i) & \#(x_{i-1}) > 0 \\ P_{KNNC_1}(x_i) & \text{otherwise} \end{cases}$$

$$P_{KNNC_1}(x_i) = \frac{\max\{\#(x_i)-D^{(1)},0\}}{|\mathcal{X}|} + \frac{D^{(1)}N_{1+(\cdot)}}{|\mathcal{X}|} \frac{1}{|\mathcal{X}|}$$

3,2-gram modified Kneser-Ney (modKN2, modKN3).

$$P_{\text{modKN}_3}(x_i|x_{i-2:i-1}) = \begin{cases} \frac{N_{1+(x_i-2:i)} - D^{(3)}(N_{1+(x_i-2:i)})}{N_{1+(x_i-2:i-1)}} + \\ \gamma(x_{i-2:i-1}) P_{\text{modKN}_2}(x_i|x_{i-1}) & N_{1+(x_{i-2:i-1})} > 0 \\ P_{\text{modKN}_2}(x_i|x_{i-1}) & \text{otherwise} \end{cases}$$

where

$$\gamma(x_{i-2:i-1}) = \frac{D_1^{(3)}N_1(x_{i-2:i-1}) + D_2^{(3)}N_2(x_{i-2:i-1}) + D_{3+}^{(3)}N_{3+}(x_{i-2:i-1})}{N_{1+(x_{i-2:i-1})}}$$

$$P_{\text{modKN}_2}(x_i|x_{i-1}) = \begin{cases} \frac{N_{1+(x_{i-1:i})} - D^{(2)}(N_{1+(x_{i-1:i})})}{N_{1+(x_{i-1})}} + \\ \gamma(x_{i-1}) P_{\text{modKN}_1}(x_i) & N_{1+(x_{i-1})} > 0 \\ P_{\text{modKN}_1}(x_i) & \text{otherwise} \end{cases}$$

where

$$\gamma(x_{i-1}) = \frac{D_1^{(2)}N_1(x_{i-1}) + D_2^{(2)}N_2(x_{i-1}) + D_{3+}^{(2)}N_{3+}(x_{i-1})}{N_{1+(x_{i-1})}}$$

$$P_{\text{modKN}_1}(x_i) = \begin{cases} \frac{N_{1+(x_i)} - D^{(1)}(N_{1+(x_i)})}{N_{1+(\cdot)}} + \\ \gamma(\epsilon) \frac{1}{|\mathcal{X}|} & N_{1+(\cdot)} > 0 \\ \frac{1}{|\mathcal{X}|} & \text{otherwise} \end{cases}$$

where

$$\gamma(\epsilon) = \frac{D_1^{(1)}N_1(\cdot) + D_2^{(1)}N_2(\cdot) + D_{3+}^{(1)}N_{3+}(\cdot)}{N_{1+(\cdot)}}$$

Overall

$$D^{(i)}(c) = \begin{cases} 0 & \text{if } c=0 \\ D_1^{(i)} & \text{if } c=1 \\ D_2^{(i)} & \text{if } c=2 \\ D_{3+}^{(i)} & \text{if } c \geq 3 \end{cases}$$

where

$$D_1^{(i)} = 1 - 2Y^{(i)} \frac{n_2^{(i)}}{n_1^{(i)}}$$

$$D_2^{(i)} = 2 - 3Y^{(i)} \frac{n_3^{(i)}}{n_2^{(i)}}$$

$$D_{3+}^{(i)} = 3 - 4Y^{(i)} \frac{n_4^{(i)}}{n_3^{(i)}}$$

where $Y^{(i)} = \frac{n_1^{(i)}}{n_1^{(i)} + 2n_2^{(i)}}$.

3,2-gram modified Kneser-Ney with normal counting (modKNNC2, modKNNC3).

$$P_{\text{modKNNC}_3}(x_i|x_{i-2:i-1}) = \begin{cases} \frac{\#(x_{i-2:i}) - D^{(3)}(\#(x_{i-2:i}))}{\#(x_{i-2:i-1})} + \\ \gamma(x_{i-2:i-1})P_{\text{modKNNC}_2}(x_i|x_{i-1}) & \#(x_{i-2:i-1}) > 0 \\ P_{\text{modKNNC}_2}(x_i|x_{i-1}) & \text{otherwise} \end{cases}$$

where

$$\gamma(x_{i-2:i-1}) = \frac{D_1^{(3)}N_1(x_{i-2:i-1}\cdot) + D_2^{(3)}N_2(x_{i-2:i-1}\cdot) + D_{3+}^{(3)}N_{3+}(x_{i-2:i-1}\cdot)}{\#(x_{i-2:i-1})}$$

$$P_{\text{modKNNC}_2}(x_i|x_{i-1}) = \begin{cases} \frac{\#(x_{i-1:i}) - D^{(2)}(\#(x_{i-1:i}))}{\#(x_{i-1})} + \\ \gamma(x_{i-1})P_{\text{modKNNC}_1}(x_i) & \#(x_{i-1}) > 0 \\ P_{\text{modKNNC}_1}(x_i) & \text{otherwise} \end{cases}$$

where

$$\gamma(x_{i-1}) = \frac{D_1^{(2)}N_1(x_{i-1}\cdot) + D_2^{(2)}N_2(x_{i-1}\cdot) + D_{3+}^{(2)}N_{3+}(x_{i-1}\cdot)}{\#(x_{i-1})}$$

$$P_{\text{modKNNC}_1}(x_i) = \frac{\#(x_i) - D^{(1)}(\#(x_i))}{|\mathcal{X}|} + \gamma(\epsilon) \frac{1}{|\mathcal{X}|}$$

where

$$\gamma(\epsilon) = \frac{D_1^{(1)}N_1(\cdot) + D_2^{(1)}N_2(\cdot) + D_{3+}^{(1)}N_{3+}(\cdot)}{|\mathcal{X}|}$$

$D^{(n)}(k)$ and $D_k^{(n)}$ as above.

B Table of Notation

The following is a list of commonly used notations. The first column is the symbol itself, the second column is its corresponding name and/or explanation. Some notation is formally defined/introduced in this thesis, e.g. $C(x)$ and $K(x)$, while some standard ones are used without being formally defined, e.g. \mathbb{R} .

| Symbol | Explanation |
|---------------------------|--|
| Sets and alphabets | |
| \mathbb{R} | set of real numbers |
| \mathbb{R}^+ | set of nonnegative real numbers |
| \mathbb{N} | set of natural numbers $\mathbb{N} = \{1, 2, 3, \dots\}$ |
| \mathbb{N}_0 | set of natural numbers including zero |

| | |
|---------------------------------|--|
| \mathbb{Z} | set of integers $\mathbb{Z} = \{\dots, -2, -1, 0, 1, 2, \dots\}$ |
| \mathbb{Q} | set of rational numbers $\mathbb{Q} = \{\frac{p}{d}\}, n \in \mathbb{Z}, d \in \mathbb{N}$ |
| \mathbb{B} | binary alphabet/set $\mathbb{B} = \{0, 1\}$ |
| \mathcal{Y}, \mathcal{S} | generic sets |
| \mathcal{X} | generic alphabet, normally large |
| Standard abbreviations | |
| [hutter05] | paper, book or other reference |
| (5.3) | label/reference for a formula/theorem/definition/(sub)section |
| ∞ | infinity |
| $\{a, b, \dots, z\}$ | set containing elements a, b, \dots, z . \emptyset is the empty set |
| $[a, b)$ | interval on real line, closed at a (includes a) and open at b (excludes b) |
| $\forall, \exists, \Rightarrow$ | for all, there exists, implies |
| \square | q.e.d (Latin), which was to be demonstrated |
| i, k, n, t | natural numbers, t is normally interpreted as discrete time step |
| $O(\cdot), o(\cdot)$ | big and small oh-notation |
| $\mathbb{I}[P]$ | Iverson bracket, $\mathbb{I}[P] = 1$ only when P is true |
| e.g. | exempli gratia (Latin), for example |
| i.e. | id est (Latin), that is |
| etc. | et cetera (Latin), and so forth |
| i.i.d. | independently identically distributed |
| iff | if and only if |
| s.t. | such that |
| e | constant $e = 2.71828\dots$ |
| π | constant $\pi = 3.14159\dots$ |
| ϵ | some small positive real number |
| E | expectation value |
| Var | variance |
| P | probability distribution |
| \mathcal{L} | the uniform measure |
| μ | probability measure |
| Ω | sample space or Chaitin's number |
| Elementary operations | |
| $+, -, (\times), /$ | standard arithmetic operations: sum, difference, product, ratio |
| $\sqrt{\quad}$ | square root |
| $ \mathcal{S} , a $ | size/cardinality of set \mathcal{S} , absolute value of a |
| $\sum_{i=1}^n$ | summation from $i = 1$ to n |
| $\prod_{i=1}^n$ | product from $i = 1$ to n |
| min/max | min-/maximal element of a set: $\min_{x \in \mathcal{X}}$ |
| arg min | $\arg \min_x f(x)$ is the x minimising $f(x)$ |
| arg max | $\arg \max_x f(x)$ is the x maximising $f(x)$ |
| log | logarithm to some basis |
| \log_b | logarithm to basis b |
| ln | natural logarithm to basis $e = 2.71828\dots$ |
| $\lim_{n \rightarrow \infty}$ | limiting value of argument for n tending to infinity |
| (In)equalities | |
| $=, \neq, \approx, \equiv$ | equal to, not equal to, approximately equal to, equivalent to |
| $\leq, \geq, <, >$ | standard inequalities |
| \subset, \subseteq | proper subset, subset |

| | |
|------------------------------------|--|
| $\leq, \geq, =$ | less/greater/equal within an additive constant |
| $\times, \leq, \geq, =$ | less/greater/equal within a multiplicative constant |
| Strings | |
| ϵ | empty string |
| $x_{1:n}$ | $x_1x_2\dots x_n$, string of length n |
| $x_{<t}$ | $x_1x_2\dots x_{t-1}$, string of length $t-1$ |
| x_i | i :th letter in string x |
| x, y, z | finite strings |
| ω | infinite sequence, elementary event |
| $\ell(x)$ | length of string x |
| $\#(x)$ | the number of string x that occurs in some corpus |
| $\#_t(x_t x_{t-1})$ or $\#_t(x_t)$ | the number of occurrence of x_t in $x_{<t}$ |
| \mathcal{M} | set of models/sources, model class |
| \mathbb{C} | a coding scheme |
| BV code | block to variable code |
| AIT-related | |
| AIT | algorithmic information theory |
| AR | algorithmic ‘Martin-Löf’ randomness |
| AC | algorithmic ‘Kolmogorov’ complexity |
| $\Gamma_{x_{1:n}}$ | $\{\omega x_{1:n}\} \subset \mathbb{B}^\infty$, cylinder set of $x_{1:n}$ |
| $C(x)$ | plain Kolmogorov complexity of string x |
| $C(x y)$ | plain Kolmogorov complexity of string x , given y |
| $K(x)$ | prefix Kolmogorov complexity of string x |
| $K(x y)$ | prefix Kolmogorov complexity of string x , given y |
| $\langle \cdot \rangle$ | a recursive bijective pairing function $\mathbb{N} \times \mathbb{N} \rightarrow \mathbb{N}$ |