DOCTOR OF PHILOSOPHY

Grammar-based preprocessing for PPM compression and classification

Aljehane, Nojood Obaidallah M

Award date:
2018

Awarding institution:
Bangor University

PRIFYSGOL
BANGOR
UNIVERSITY
School of Computer Science

GRAMMAR-BASED PREPROCESSING FOR PPM COMPRESSION AND CLASSIFICATION

by

Nojood Aljehane

A thesis submitted for the degree of Doctor of Philosophy

March 2018

ABSTRACT

The aim of this study is to investigate the efficiency of novel methods using context-free grammars and Prediction by Partial Matching (PPM) in order to build and evaluate the quality of compression models for text files such as English, Arabic, Persian, Welsh and Chinese. A further aim is then to apply these models to the problem of the classification of text to see how well they perform at this application.

We apply grammar-based pre-processing prior to using the PPM compression algorithm. The methods achieve significantly better compression for different natural language texts compared with other well-known compression methods. Our method first generates a grammar based on the most common two-character sequences (bigraphs) or three-character sequences (trigraphs) in the text being compressed and then substitutes these sequences using the respective non-terminal symbols defined by the grammar in a pre-processing phase prior to compression.

We describe further improvements using a two-pass scheme where grammar-based pre-processing is applied again in a second pass through the text. We then apply the algorithms to the files in the Calgary Corpus and also achieve significantly improved results in compression when compared with other compression algorithms, including a grammar-based approach known as the Sequitur algorithm.

Despite the advances of the PPM method in predicting upcoming symbols or words in the English language, more research is required to devise better compression methods for other languages, such as Arabic due to, for example, the rich morphological nature of Arabic text, in which a single word can take many different forms. In this dissertation, we propose a new method that achieves the best compression rates not only for

Arabic language text but also for other languages that use Arabic script in their writing systems, such as Persian. Our word-based method (GRW-PPM) constructs a context-free grammar for the text; this grammar is then encoded using PPM to achieve excellent compression rates.

Finally, we investigate the classification of genre in English and Arabic text by using our new character-based text compression scheme (GRB-PPM). Experimental results on a parallel Arabic and English corpus show that our new method is very effective compared with traditional compression-based classification methods. We have also confirmed that good compression leads to good classification.

Acknowledgements

First, I would like to express my deep and sincere gratitude to my supervisor Dr. William J. Teahan, whom I am greatly indebted to for the continuous guidance and advice during this research. I appreciate his support, patience and encouragement, in which this research would not be completed without. I am very grateful for him for giving me the opportunity to do and achieve this PhD. Dr William J. Teahan is an outstanding supervisor and I have been very fortunate to be a PhD student under his supervision.

Many thanks to my dear parents and my beloved husband Hassan for their love and support, for having faith in me, and being there for me at all times. Many thanks to my precious kids Layan, Wael and Joory for their patience, great support and encouragement, for cheering me up and putting a smile on my face when I most needed, and for being there for me in the good and the hard times through this rough journey. Last but not least, a big thank you to my dear brother Mohammed, for helping me gain a strong faith and confidence in myself, for his continuous and immense support.

I would like dedicate my dissertation to my beloveds my parent, my husband Hassan and my lovely kids Layan, Wael and Joory…

CONTENTS

Bibliography 136

ABBREVIATIONS

PPM	Prediction by Partial Matching
GR-PPM	Grammar based pre-processing for PPM
GRB-PPM	Single-Pass Grammar Bigraphs for PPM
GRT-PPM	Single-Pass Grammar Trigraphs for PPM
GRBB-PPM	Two-Pass Grammar Bigraphs for PPM
GRTT-PPM	Two-Pass Grammar Trigraphs for PPM
GRW-PPM	Word-based grammars for PPM

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

INTRODUCTION

Contents

1.1 Background & Motivation

The volume of data being stored, transferred and used by computers users is rapidly growing. In 1995, the number of internet users in the world was less than 1% of the total world population (Internet Live, 2016). Figure 1.1 shows the the growth of the number of internet users, which has risen from 1995 to 2015. Therefore, the need for processing natural languages using computers has never been more urgent. The goal of Natural Language Processing (NLP) is to help computers better understand human language in order to help improve interaction between computers and humans. Text compression, text classification, machine translation and speech recognition are just some examples of NLP research areas.

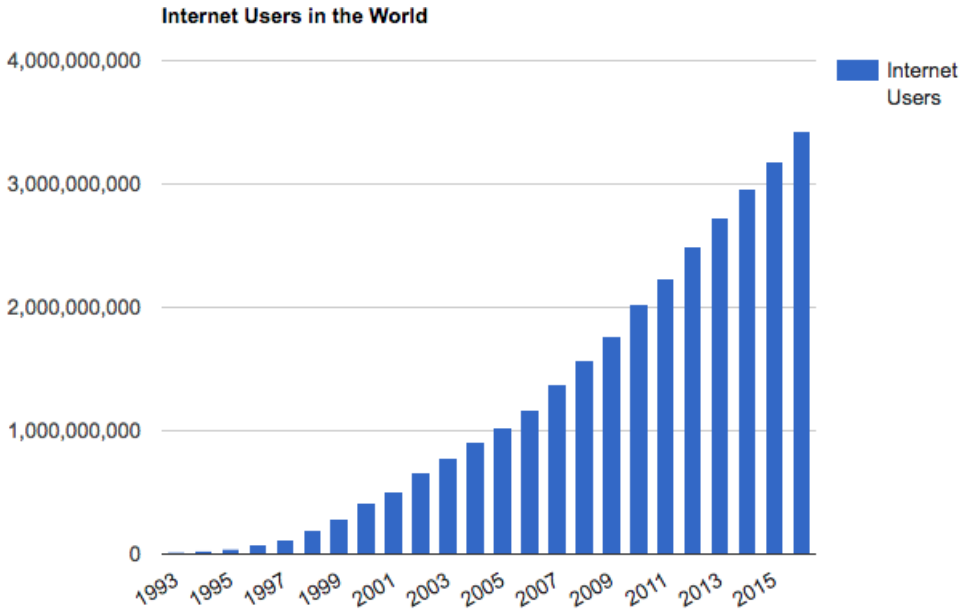


FIGURE 1.1: Internet users around the world (Internet Live, 2016)

The motivation of this study is to apply compression-based techniques in order to understand better the nature of natural language. Also, due to the growth of digital data in recent years, it has become vitally important to improve the mechanisms used in digital data compression to enhance savings in data storage and speed. Although the growth in data may not be a major concern for everyday users, it is an important issue for medium- and large-sized service providers. Therefore, as technology advances, it is important to develop compression methodologies that will enable the growth of digital data to be sustained.

Text compression reduces the space needed to store data by using different encoding techniques. Text compression is also used for encryption, error correction and error detection, among many others applications (Bell et al., 1990). The basic aim of all text compression is to save resources and media space.

There are two basic schemes that text compression models use for compressing text. One approach is dictionary-based, and the other approach is statistical (Bell et al., 1990). Experiments from different studies show that, although dictionary schemes offer fast execution times, their compression rate performance is less effective when compared with statistical approaches. In 1994, Burrows and Wheeler introduced a new method that combines high speed and better compression rates by using the Burrows-Wheeler Transform (BWT) (Blelloch, 2001). Another adaptive approach for text compression is grammar-based compression algorithms. Research shows that grammar-based methods outperform other compression methods for some applications, such as text compression for natural language (Sayood, 2017). In this approach, a Context-Free Grammar (CFG) is used to help compress the text file.

PPM is one of the most effective statistical techniques, having achieved excellent results (Cleary and Witten, 1984). PPM dynamically structures and upgrades a language model of the text depending on previous characters or symbols (such as words) in the input stream.

Text classification is another NLP application that assigns one or more categories or classes to a text. In recent years, various studies have shown that using a compression-based approach based on the PPM scheme for text classification can outperform traditional text classification methods for different languages.

In this thesis we combine two approaches (grammar-based processing and PPM compression) to attain the best compression rate in terms of performance for different languages and then apply this new approach to classify text to obtain good results based on experiment with standard text corpora.

1.2 Aim and Objectives

The primary aim of this study is to investigate the efficiency of novel methods using CFGs and PPM to build and evaluate improved methods for compressing text files and

then apply those methods to the problem of the classification of text to see how well they perform at this application. Therefore, the objectives of this study are as follows:

• Design and implement novel grammar-based methods for compression based on characters using PPM (see chapter 4).

• Develop further improvements for the new methods for different text languages (see chapter 5).

• Evaluate these methods by comparing them with well-known compression methods (see chapters 4 and 5).

• Develop improved word-based compression models for PPM by parsing the text to construct a word-based CFG which is then compressed using PPM (see chapter 6).

• Apply one of these new methods to the problem of the classification of text (see chapter 7).

1.3 Research Questions

The specific research questions for this study that relate to the aim and objectives are as follows:

1. What is the best grammar-based compression model for compressing various natural language texts (see chapters 4, 5 and 6)?

2. Do grammar-based methods perform better than other common compression methods for the Arabic language specifically, as it is not related to English and has thus far been under-researched compared to research into English text compression (see chapters 4, 5 and 6)?

3. Does better compression lead to better classification when we apply improved compression models (see chapter 7)?

4. Do the classification results match between languages for a parallel corpus [1] to the problem of text classification (see chapter 7)?

[1] A parallel corpus contains a collecting of texts of two or more languages.

1.4 Contributions

The specific contributions of this study are as follows:

• New methods based on both grammar-based modelling and PPM compression for different language texts have been developed and evaluated.

• Further improvements to these new methods have also been developed and evaluated.

• A new grammar-based word compression scheme has also been proposed and evaluated.

• An effective and novel method for text classification based on the new compression method has also been investigated.

The main contribution of this thesis is to suggest effective, novel schemes using grammars for compressing different language texts and applying this method to the problem of classification. The methods achieve significantly better compression for different natural language texts compared to other well-known compression methods. We describe further improvements using a two-pass scheme in which grammar-based pre-processing is applied again in a second pass through the text. In addition, a new method has been designed especially for Arabic text. Significantly, GRB-PPM achieves better results in classification schemes for both Arabic and English texts in a parallel corpus.

The objectives listed above have been achieved by the development of the GRB-PPM, GRT-PPM, GRBB-PPM and GRTT-PPM models. Grammar based pre-processing for PPM (GR-PPM) has shown positive results when applied to languages as varied as Arabic, Persian, English, Chinese and Welsh. Specifically, GRW-PPM has shown very good results, especially for Arabic text, when compared with other word-based compression methods.

Additionally, using GRB-PPM to classify text in a parallel corpus confirmed that better compression leads to better classification. We have also shown that the results are similar across languages in the parallel corpus.

1.5 Publications

Two journal papers based on this study have already been published, with a third paper to be submitted in the future. Table 1.1 lists the specific journal papers that relate to this study. The first, entitled "Grammar based pre-processing for PPM (GR-PPM)", describes the new methods based on both grammar-based compression and PPM compression for different language texts. The paper investigates several GR-PPM methods in both first pass (GRB-PPM and GRT-PPM) and second pass (GRBB-PPM and GRTT-PPM). The experimental results discussed in this paper show that GR-PPM achieves significant improvements in compression rates over existing methods. The paper was published in the *International Journal of Computer Science & Information Technology (IJCSIT)* in 2017. The insights obtained from this paper have been an important basis for this thesis, as discussed in chapters 4 and 5.

The second paper, entitled "Word-Based Grammars for PPM", upon which chapter 6 is based, discusses a new word-based method that achieves the best compression rates not only for Arabic text but also for other languages, such as Persian, that use Arabic script in their writing systems. The word-based method constructs a CFG for the text; this grammar is then encoded using PPM to achieve excellent compression rates. In addition, word-based compression schemes generally adapt directly to the text being compressed in an on-line manner (as PPM does), rather than using dictionaries created from general sources. The paper was published in *the International Journal of Advanced Computer Science and Applications (IJACSA)* in 2017. The main contribution of this paper is the improved word-based compression method for PPM that is achieved by parsing the text to construct a word-based CFG, which is then compressed using PPM.

The third journal paper, entitled "GRB-PPM Classification", is discussed in Chapter 7. The paper addresses the use of the new method discussed in chapter 4 to classify the genre of a parallel corpus for both English and Arabic. The experimental results show that GRB-PPM produces good results for Arabic text using both static and dynamic approaches; for English, GRB-PPM produces good results using a dynamic approach. We also confirm that better compression leads to better classification and that the results are similar across languages in the parallel corpus. This paper will be submitted to *Computer Speech and Language (CSL)* in the very near future.

TABLE 1.1: Publications that relate to this study

1	**Title**	Grammar-Based Pre-Processing for PPM
	Authors	William J. Teahan and Nojood O. Aljehane
	Submitted to	*International Journal of Computer Science and Information Technology (IJCSIT)*
	Year	2017
	Status	Published
2	**Title**	Word-Based Grammars for PPM
	Authors	Nojood O. Aljehane and William J. Teahan
	Submitted to	*International Journal of Advanced Computer Science and Applications (IJACSA)*
	Year	2017
	Status	Published
3	**Title**	GRB-PPM Genre Classification
	Authors	Nojood O. Aljehane and William J. Teahan
	Will be submitted to	*Computer Speech and Language (CSL)*
	Year	2018
	Status	Pending

1.6 Organisation of this Dissertation

This thesis is organised into eight chapters:

• Chapter 1 is the introduction. In this chapter the background and motivation have been introduced. Also, the aim and objectives in this thesis have been discussed. The contributions and publication also have been reviewed in more detail.

• Chapter 2 reviews Arabic language and its features that are relevant for this study. Also, Arabic encoding methods have been reviewed. Some corpora that have been used in thesis have also been discussed.

• Chapter 3 reviews the literature related to this research being conducted. We review statical language modelling for text compression. Text encodings for English and Arabic have also been discussed. Text Compression has been discussed. We then review grammar-based codes.

• In chapter 4, we introduce a new approach based on CFGs for compressing text files. GR-PPM uses both CFGs and PPM as a general-purpose adaptive compression method

for text files that produces significantly improved results in compression for various natural languages when compared to standard PPM and other well-known methods.

• In chapter 5, we describe further improvements using a recursive grammar pre-process -ing scheme where grammar-based pre-processing is applied again in a second pass through the text. We then apply the algorithms to the files in the Calgary Corpus. This method achieves significantly improved results when compared with other compression algorithms, including a grammar-based approach, the Sequitur algorithm.

• In chapter 6, we present a new word-based grammar method for compressing natural language text using our CFG scheme and PPM. The results show significant improvements, especially for Arabic and languages, such as Persian, that use Arabic in their writing systems.

• In chapter 7, we investigate a compression-based classification method that uses our new GRB-PPM method to confirm whether better compression leads to better classification; the results showed that this is indeed the case.

• A summary of the dissertation is presented in chapter 8, along with suggestions for future work.

CHAPTER 2

ARABIC LANGUAGES OVERVIEW

Contents

2.1 Introduction

A language has been defined as a system of communication using words, grammar and sounds or communication used by people in a certain country (Clackson, 2007).

This chapter provides contextual information about Arabic and is organized as follows as this relates specifically to research question 2. Firstly, it reviews Arabic's geographical spread in section 2.2. Secondly, section 2.3 discuses Arabic characters and its written system, after which section 2.4 reviews Arabic morphology. Section 2.5 outlines digital Arabic content, and section 2.6 reviews different encoding methods created for Arabic script. Section 2.7 describes certain Arabic language corpora that have been used for Arabic natural language. A summary and discussion is provided in Section 2.8.

2.2 Arabic: One of the Most Widely Spoken Languages in the World

Arabic (العربية, al-arabiyyah) is the native language of between 290 to 420 million people, while more than one billion use Arabic as a second language (Simons et al., 2017). It is considered a divine language for the whole Muslim community, as the Holy Qur'an and most Islamic books were written in Arabic. It is a Semitic language, a group that includes other languages like Phoenician, Hebrew, Aramaic and Tigrinya, all of which use a similar structure. It is regarded as one of the richest languages on Earth due to its influential history. The Arabic language has influenced other languages in western Asia, such as Urdu, Persian and Kurdish, both directly and indirectly.

Table 2.1 shows the most common languages based on their number of speakers. The Chinese language is the most widely spoken language, with English coming next(Simons et al., 2017). Arabic is the fifth most-spoken language, based on the Ethnologue website, in 2017.

The majority of Arabic speakers are found in 22 countries, mostly in the Middle East and North Africa, *"from Iraq in the north to Somalia in the south, Bahrain in the east to the western shores of Mauritania"*(Rasheed, 2008). It is divided into numerous regional

TABLE 2.1: List of 12 languages with the largest total numbers of speakers (Simons et al., 2017)

Rank	Language	Speakers (million)
1	Mandarin Chinese	1090
2	English	983
3	Hindustani	544
4	Spanish	527
5	Arabic	422
6	Malay	281
7	Russian	267
8	Bengali	261
9	Portuguese	229
10	French	229
11	Hausa	150
12	Punjabi	148

dialects or variants (Rasheed, 2008). In addition, there are differences from one Arabic-speaking nation to another and even differences between social groups within the same nation (Holes, 1995).

2.3 The Arabic Character and Writing System

As noted above, the Arabic language is in the Semitic language group, which developed a writing system over time from the Aramaic and Nabataean scripts (Elbeheri et al., 2006). Arabic has twenty-eight basic letters, which are: "ا ب ت ث ج ح خ د ذ ر ز س ش ص ض ط ظ ع غ ف ق ك ل م ن ه و ي", and eight diacritical marks (call -ed Tashkeel تشكيل). Tashkeel are optional and are generally added above or below letters to help readers recognize Arabic words correctly. Arabic has three vowels, which are "ا و ي"; the rest of the letters are consonants. Arabic has two genders (masculine and feminine) and singular, dual and plural forms. The latter are exemplified by "هو" for masculine singular and "هي" for feminine singular, "هما" for masculine and feminine dual and "هم" and "هن" for masculine and feminine plural forms, respectively.

Arabic letters can be fully vowelised, partially vowelised or unvowelised. Table 2.2 shows the various vowelised states of Arabic characters. Arabic words in their natural

TABLE 2.2: Vowelised state examples for Arabic text

Vowelisation States	Examples
Fully vowelised	العَلَمُ
Partially vowelised	العَلم
Unvowelised	العلم

form are separated by spaces. The Arabic writing system is written and read from right to left; unlike English, Arabic does not have uppercase or lowercase forms.

The Arabic script is the world's second most widely used writing system, based on the number of nations that use it in their writing systems, and the third most widely used by number of people, after the Chinese and Latin writing systems (Encyclopædia Britannica, 2015). It is cursive, which means that most of its characters are joined together by means of ligatures; the shapes of many letters within a word depend on their position (Elbeheri et al., 2006). Unlike English script, for example, in which all letters are connected within a word, Arabic script has 22 letters that can be joined with preceding and following letters by short horizontal lines; the other six Arabic letters, (و ا ذ د ز ر), can be joined only to a preceding letter.

Arabic has various script styles, such as naskh, ruq'ah and Kufic. Kufic is the oldest script in the Arabic writing system. It is a modification of the Nabataean script; the term Kufic comes from the region of Kufa, Iraq as shown in Figure 2.1 which is an image of the oldest Qur'an in the Kufic script (Trezise, 2002).

FIGURE 2.1: The Qur'an in Kufic script (Trezise, 2002)

The formal written system for Arabic text can be divided into two types: Modern Standard Arabic and Classical Arabic (Ryding, 2005; Najeeb et al., 2014). Modern Standard Arabic (MSA), which is used in today's written language, is derived from Classical Arabic. MSA is one of the six official languages of the United Nations (UN) (United Nations, 2017). The language used in the Holy Quran and much Arabic literature is

written in Classical Arabic (CA) as opposed to MSA; there are many differences in vocabulary and style. Some vocabulary used in Classical Arabic is not used today, and native speakers for Arabic may need to rely on a translation of these words to understand CA text. On the other hand, MSA is the form used by native Arabic speakers in their everyday interactions and in written material, lectures, TV shows, and so on. (See Figure 2.2 for an example of MSA and Figure 2.3 for CA).

وعلمت "الوطن" أن نادي الاهلي بصدد الاستفادة من مراكز عدد من اللاعبين
الذين لا يرغب في استمرارهم مع الفريق الموسم المقبل، وذلك عبر تسويقهم
لفرق منافسة بمقايضتهم بنجوم آخرين.

FIGURE 2.2: A sample of Modern Standard Arabic from the BACC corpus

كما يشرف الولد بشرف الوالد قد يشرف الوالد بشرف الولد ولله در ابن الرومي في قوله:
وكم أب قد علا بابن ذرى حسب كما علت برسول الله عدنان

FIGURE 2.3: A sample of Classical Arabic from the BACC corpus

2.4 Arabic morphology

Arabic shares many features with other Semitic languages, one of which is that words can be constructed from a basic root (usually three letters) into which different vowel patterns are then inserted. In other words, a word can be derivative or non-derivative. A derivative word is constructed from a basic root depending on a template known as a morphological balance or a predefined palette. Figure 2.4 shows an example of words derived from the root علم, which represents the concept of 'learning'. Fundamentally, non-derivative words are functional or nouns borrowed from other languages (Elbeheri et al., 2006; Versteegh, 2001).

By applying different patterns to the single root علم (a-l-m), numerous words may be formed:

- عَلِمَ (alima) 'to know'

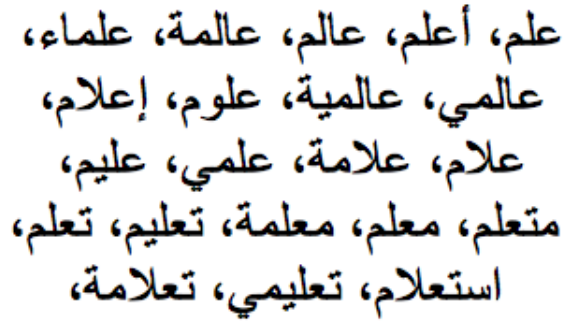


FIGURE 2.4: Several words derived from the same root

- يَعْلَمُ (yalamu) ‘to be informed’
- عَلَّمَ, allama ‘to teach (something)’
- يُعَلِّمُ, yuallimu ‘to mark’
- عِلْمٌ, ilm ‘knowledge’
- عُلُومٌ, ulum ‘science’
- عَلَمٌ, alam ‘sign’
- أَعْلَامٌ, aelam ‘flags’

Like other Semitic languages, Arabic has a rich and complex morphology, and the same word can be formed in many different ways (Elbeheri et al., 2006; Versteegh, 2001). A word can be changed into different forms by adding a suffix, a prefix or both, as occurs with “ قلم, which means “pen”. When the prefix “ال” is added to the word root, a second form, “ القلم ”, is created; it means “the pen”. A third form is created when the suffix “ان ” is added to the word, givng us “قلمان”, which means “two pens”. When both a prefix “ال” and a suffix “ان ” are added to the word root, the new form “القلمان” is created, which means “the two pens”.

2.5 Digital Arabic Content

Nowadays, information technology and digital content play an enormously important role in the world. Digital data includes text, audio, video and images and has radically changed the way that people do business, learn and socialize.

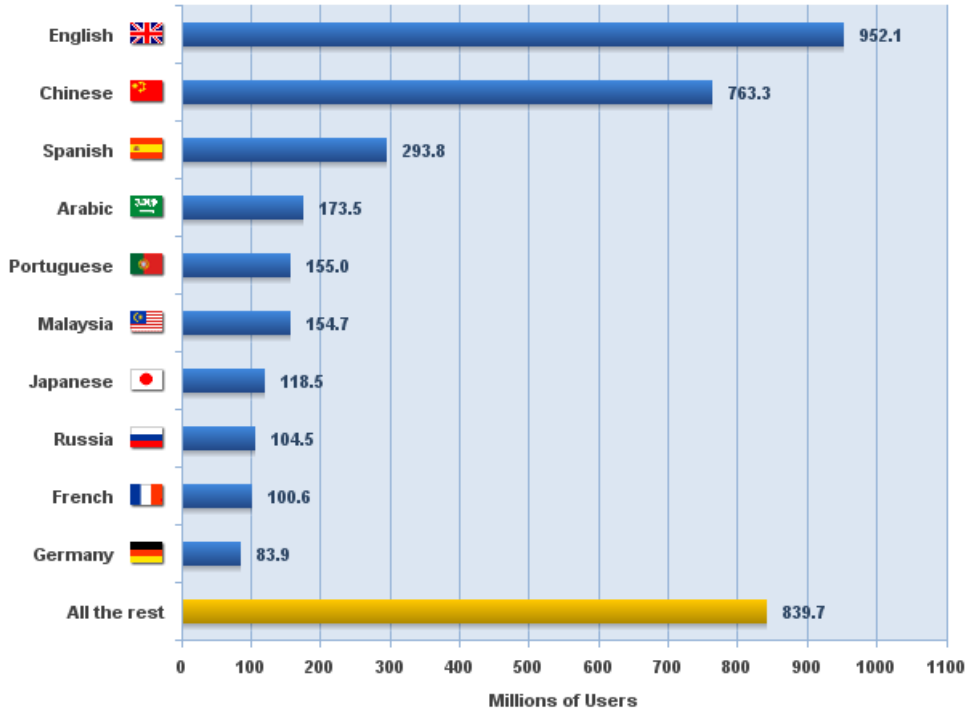


FIGURE 2.5: Top ten languages among internet users (Marketing, 2017)

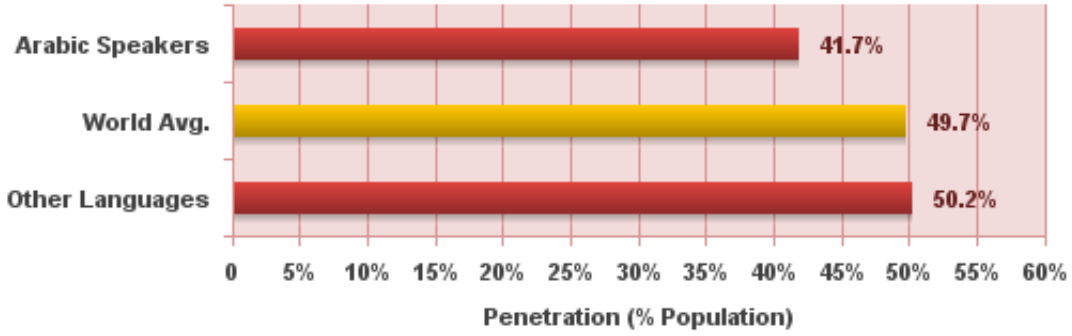


FIGURE 2.6: Global internet penetration of Arabic users in 2017 (Marketing, 2017)

According to the Marketing website, English is the most popular language on the internet, with more than 950 million (25.5% of the world's internet users), after which Chinese speakers come next, as shown in Figure 2.5. Arabic speakers are the fourth largest

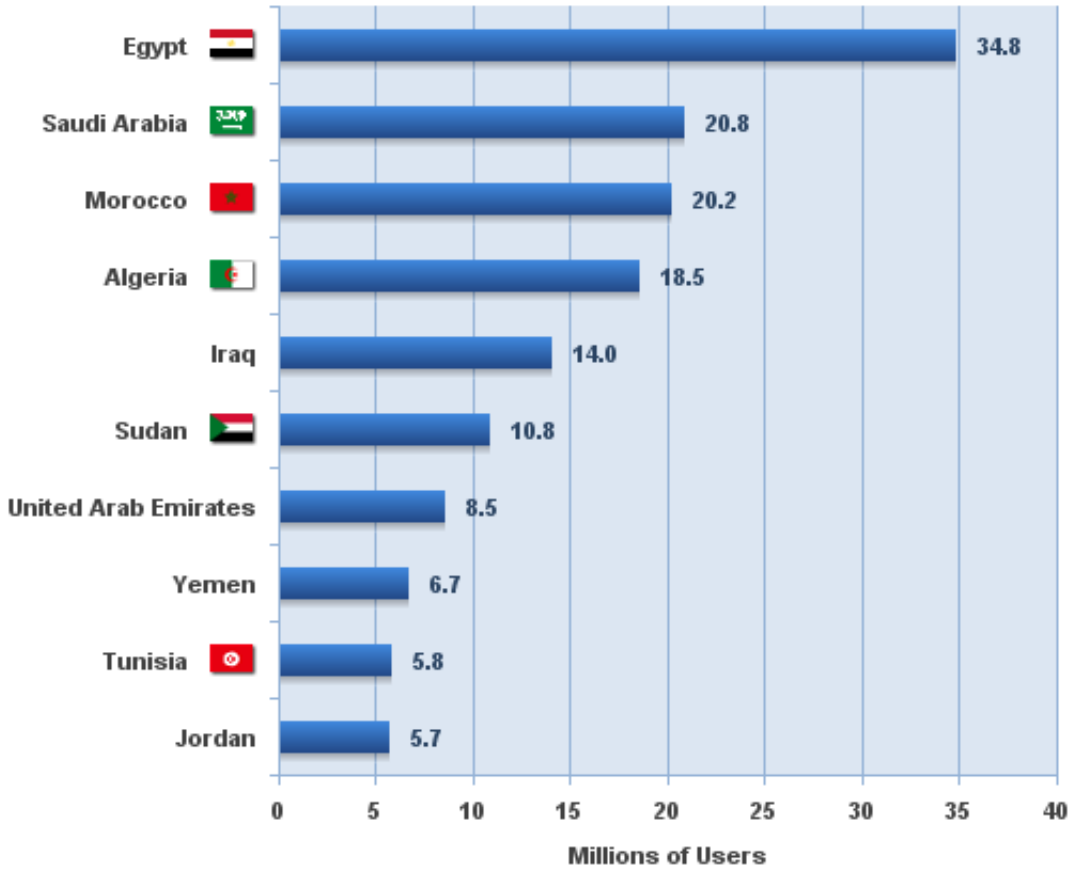


FIGURE 2.7: Estimated Arabic-speaking internet users in March 2017 (Marketing, 2017)

internet users, with more than 173 million Arabic-speaking internet users. Arabic users come from 23 countries in which Arabic is the primary language. As Figure 2.6 illustrates, Egypt has the most Arabic internet users (19.4%), then Saudi Arabia 11.8% and Morocco 11.5% (Marketing, 2017). Figure 2.7 shows that Arabic speakers are among the largest groups of internet users, since they constitute 5.6% of all internet users in the world (Marketing, 2017).

2.6 Arabic Encoding methods

Unlike English letters, Arabic letters do not use separate uppercase and lowercase forms. However, some letters in Arabic have different forms depending on their position in a word. For example, the letter b "ب " has three constituent shapes: at the front of a word it is "بـ"; in the middle it is "ـبـ"; and at the end it is "ـب". Moreover,

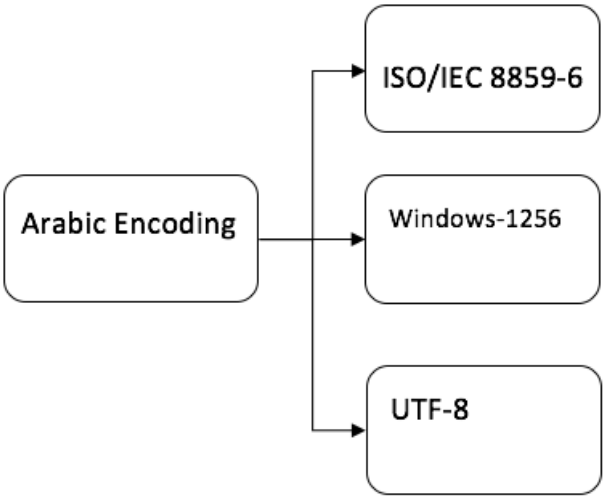
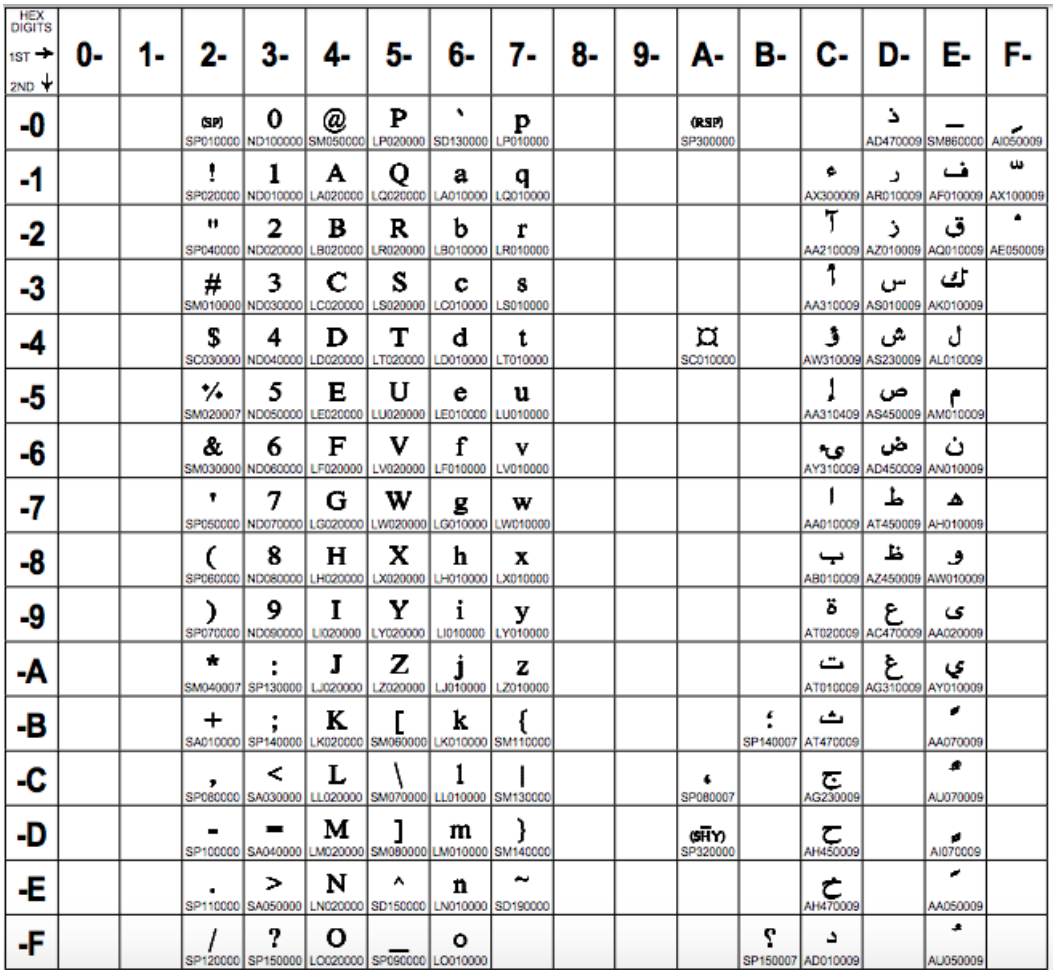


FIGURE 2.8: Different Arabic encoding methods

some Arabic letters have the same constituent shape but are distinct from each other by dots placed below or above the letters. For instance, the letter b "ب " has a similar shape to the letter t "ت ", but the difference is the dots; the letter "ب " has one dot below, while the letter "ت " has two dots above. In addition, some letters in Arabic do not have to be connected with the next letter in the word, such as "ا د ذ و ز ر ". Because of the variety of shapes and forms found in Arabic letters, many different schemes have been developed for encoding Arabic letters; they are discussed in the next section. Figure 2.8 shows the different Arabic encoding types to be discussed below .

2.6.1 ISO (8859-6) Arabic Encoding

The International Organization for Standardization (ISO 8859-6) is one of the common character encodings that is designed for Arabic languages. This encoding uses an 8-bit character method and was issued by the by the European Computer Manufacturers Association. It is designed to encode only standard Arabic letters and is not used for other languages that use Arabic script in their written forms, such as Persian, Kurdish and Urdu (see Figure 2.9).



HEX DIGITS 1ST → 2ND ↓	0-	1-	2-	3-	4-	5-	6-	7-	8-	9-	A-	B-	C-	D-	E-	F-
-0			(SP) SP010000	0 ND100000	@ SM050000	P LP020000	` SD130000	p LP010000			(RSP) SP300000			ذ AD470009	ـ SM860000	ً AJ050009
-1			! SP020000	1 ND010000	A LA020000	Q LQ020000	a LA010000	q LQ010000					ء AX300009	ر AR010009	ف AF010009	ّ AX100009
-2			" SP040000	2 ND020000	B LB020000	R LR020000	b LB010000	r LR010000					آ AA210009	ز AZ010009	ق AQ010009	ٌ AE050009
-3			# SM010000	3 ND030000	C LC020000	S LS020000	c LC010000	s LS010000					أ AA310009	س AS010009	ك AK010009	
-4			$ SC030000	4 ND040000	D LD020000	T LT020000	d LD010000	t LT010000			¤ SC010000		ؤ AW310009	ش AS230009	ل AL010009	
-5			٪ SM020007	5 ND050000	E LE020000	U LU020000	e LE010000	u LU010000					إ AA310409	ص AS450009	م AM010009	
-6			& SM030000	6 ND060000	F LF020000	V LV020000	f LF010000	v LV010000					ئ AY310009	ض AD450009	ن AN010009	
-7			' SP050000	7 ND070000	G LG020000	W LW020000	g LG010000	w LW010000					ا AA010009	ط AT450009	ه AH010009	
-8			(SP060000	8 ND080000	H LH020000	X LX020000	h LH010000	x LX010000					ب AB010009	ظ AZ450009	و AW010009	
-9			) SP070000	9 ND090000	I LI020000	Y LY020000	i LI010000	y LY010000					ة AT020009	ع AC470009	ى AA020009	
-A			* SM040007	: SP130000	J LJ020000	Z LZ020000	j LJ010000	z LZ010000					ت AT010009	غ AG310009	ي AY010009	
-B			+ SA010000	; SP140000	K LK020000	[SM060000	k LK010000	{ SM110000				؛ SP140007	ث AT470009		ً AA070009	
-C			, SP080000	< SA030000	L LL020000	\ SM070000	l LL010000	| SM130000			، SP080007		ج AG230009		ٌ AJ070009	
-D			- SP100000	= SA040000	M LM020000	] SM080000	m LM010000	} SM140000			(SHY) SP320000		ح AH450009		ٍ AI070009	
-E			. SP110000	> SA050000	N LN020000	^ SD150000	n LN010000	~ SD190000					خ AH470009		ٍ AA050009	
-F			/ SP120000	? SP150000	O LO020000	_ SP090000	o LO010000					؟ SP150007	د AD010009		ْ AU050009	

FIGURE 2.9: ISO 8859-6 Arabic encoding (IBM Corp, 1996)

2.6.2 Windows-1256

Windows-1256 was designed by Microsoft and also uses an 8-bit character encoding method. Windows-1256 encodes not only characters in Arabic but also in languages that use Arabic characters in their written systems in Microsoft Windows, such as Persian, Kurdish and Urdu. Windows-1256 encodes not only standard forms but also isolated forms (single forms) (Unicode Consortium, 1991). However, it is not compatible with other Arabic encodings, such as the ISO 8859-6 scheme. Each character in figure 2.10 is presented with its Unicode equivalent.

	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
00	NUL 0000	STX 0001	SOT 0002	ETX 0003	EOT 0004	ENQ 0005	ACK 0006	BEL 0007	BS 0008	HT 0009	LF 000A	VT 000B	FF 000C	CR 000D	SO 000E	SI 000F
10	DLE 0010	DC1 0011	DC2 0012	DC3 0013	DC4 0014	NAK 0015	SYN 0016	ETB 0017	CAN 0018	EM 0019	SUB 001A	ESC 001B	FS 001C	GS 001D	RS 001E	US 001F
20	SP 0020	! 0021	" 0022	# 0023	$ 0024	% 0025	& 0026	' 0027	(0028	) 0029	* 002A	+ 002B	, 002C	- 002D	. 002E	/ 002F
30	0 0030	1 0031	2 0032	3 0033	4 0034	5 0035	6 0036	7 0037	8 0038	9 0039	: 003A	; 003B	< 003C	= 003D	> 003E	? 003F
40	@ 0040	A 0041	B 0042	C 0043	D 0044	E 0045	F 0046	G 0047	H 0048	I 0049	J 004A	K 004B	L 004C	M 004D	N 004E	O 004F
50	P 0050	Q 0051	R 0052	S 0053	T 0054	U 0055	V 0056	W 0057	X 0058	Y 0059	Z 005A	[005B	\ 005C	] 005D	^ 005E	_ 005F
60	` 0060	a 0061	b 0062	c 0063	d 0064	e 0065	f 0066	g 0067	h 0068	i 0069	j 006A	k 006B	l 006C	m 006D	n 006E	o 006F
70	p 0070	q 0071	r 0072	s 0073	t 0074	u 0075	v 0076	w 0077	x 0078	y 0079	z 007A	{ 007B	| 007C	} 007D	~ 007E	DEL 007F
80	€ 20AC	پ 067E	‚ 201A	ƒ 0192	„ 201E	… 2026	† 2020	‡ 2021	ˆ 02C6	‰ 2030	ٹ 0679	‹ 2039	Œ 0152	چ 0686	ژ 0698	ڈ 0688
90	گ 06AF	‘ 2018	’ 2019	“ 201C	” 201D	• 2022	– 2013	— 2014	ک 06A9	™ 2122	ڑ 0691	› 203A	œ 0153	ZWNJ 200C	ZWJ 200D	ں 06BA
A0	NBSP 00A0	، 060C	¢ 00A2	£ 00A3	¤ 00A4	¥ 00A5	¦ 00A6	§ 00A7	¨ 00A8	© 00A9	ھ 06BE	« 00AB	¬ 00AC	- 00AD	® 00AE	¯ 00AF
B0	° 00B0	± 00B1	² 00B2	³ 00B3	´ 00B4	µ 00B5	¶ 00B6	· 00B7	¸ 00B8	¹ 00B9	؛ 061B	» 00BB	¼ 00BC	½ 00BD	¾ 00BE	؟ 061F
C0	ہ 06C1	ء 0621	آ 0622	أ 0623	ؤ 0624	إ 0625	ئ 0626	ا 0627	ب 0628	ة 0629	ت 062A	ث 062B	ج 062C	ح 062D	خ 062E	د 062F
D0	ذ 0630	ر 0631	ز 0632	س 0633	ش 0634	ص 0635	ض 0636	× 00D7	ط 0637	ظ 0638	ع 0639	غ 063A	ـ 0640	ف 0641	ق 0642	ك 0643
E0	à 00E0	ل 0644	â 00E2	م 0645	ن 0646	ه 0647	و 0648	ç 00E7	è 00E8	é 00E9	ê 00EA	ë 00EB	ى 0649	ي 064A	î 00EE	ï 00EF
F0	ً 064B	ٌ 064C	ٍ 064D	َ 064E	ô 00F4	ُ 064F	ِ 0650	÷ 00F7	ّ 0651	ù 00F9	ْ 0652	û 00FB	ü 00FC	LTR 200E	RTL 200F	ے 06D2

FIGURE 2.10: Windows-1256 Arabic encoding (Unicode Consortium, 1991)

2.6.3 UTF-8 Encoding

UTF-8 is the most popular encoding method for different languages. It also combines different languages and is capable of encoding all potential characters. UTF-8 encoding is commonly used in websites and in applications that use multiple languages, such as YouTube, Google and Facebook (Benoit, 2013). Figure 2.11 illustrates that UTF-8 had exceeded all other language encodings on the web, as recorded by Google. It is clear from Figure 2.11 that other encoding methods, such as ASCII, declined precipitously from 2001 to 2010 (Mark, 2010).

UTF-8 uses ASCII character encodings to represent English letters in only one byte and uses one to four bytes to represent other languages that need more than one byte per letter, such as Chinese, Japanese and Arabic. For example, in Arabic, UTF-8 encodes each letter plus the supplementary symbols by using two bytes. Because other encoding

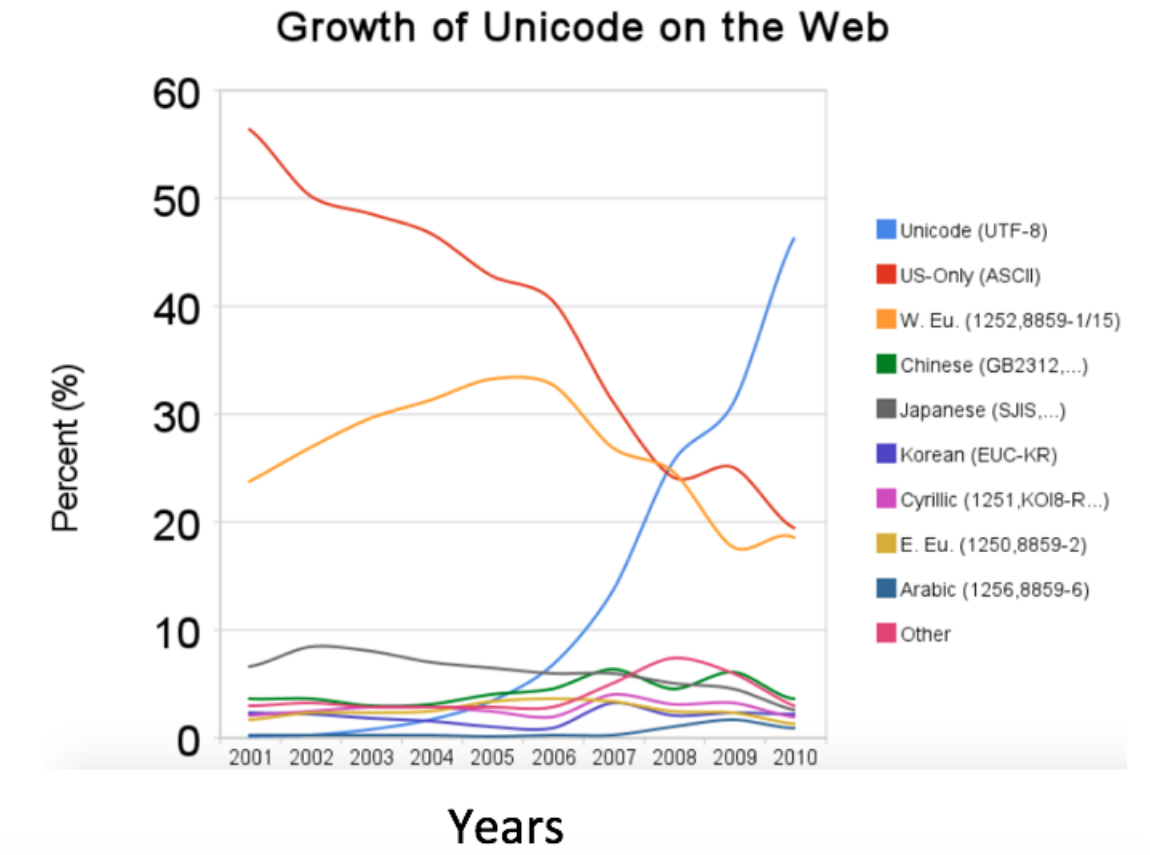


FIGURE 2.11: The popularity of the UTF-8 scheme compared with other encoding methods (Mark, 2010)

methods use only the Arabic letter itself, UTF-8 is used widely in many applications and operating systems, such as Linux and Apple Mac (Mark, 2010).

2.7 Arabic Language Corpora

A corpus is a set of large structured text, usually organised electronically, that processes syntax of natural language and analyses grammar (McEnery and Wilson, 2004). It may consist of text in one language or text in multiple languages as occurs in a parallel corpus. Moreover, a parallel corpus can be used in text compression, text translations and contrastive linguistics. Corpora are useful in several streams of linguistic research, such as speech recognition, machine translation and language teaching (Yoon and Hirvela, 2004). In the present study, corpora are used for comparing the effectiveness of different compression and classification algorithms.

The purpose of including this information here is to provide more detail on the specific Arabic corpora used in later chapters for the experimental evaluation. There are many Arabic corpora some of them include diacritic characters (tashkeel), some of them do not. In this section, we review several Arabic language corpora that are used to standardize results of experiments in compressing Arabic text.

2.7.1 The Corpus of Contemporary Arabic (CCA)

The Corpus of Contemporary Arabic (CCA) was introduced by Latifa Alsuliti and Eric Atwell in 2006 at the University of Leeds (Alsulaiti and Atwell., 2006). The users targeted by this corpus are language engineers, language teachers and learners of Arabic as an additional language. The CCA text files were derived from websites, and the audio files were acquired from Radio Qatar. It has a group of 415 text files, nearly 842,680 words and 15 different categories, including stories, economics, education, health and medicine, interviews, politics and religion. The CCA also has a small set of spoken files that are divided into the three categories of sports, education and entertainment. The text files in CCA were encoded using the UTF-8 method. We use this corpus to analysis the top 20 most common words in the Arabic language.

2.7.2 Bangor Arabic Compression Corpus (BACC)

BACC is a 31 million-word corpus created by Teahan and Alhawiti in 2013 Teahan and Alhawiti (2013). It was constructed primarily to preform compression experiments on Arabic text. It contains 16 text files in different categories such as education, economic, sports, culture, political, articles, press and history. These files were constructed from books, magazines and websites; they are divided in terms of size into four categories (small, medium, large and very large). The text files in the BACC corpus were encoded using UTF-8 encoding. The text files in the BACC corpus are taken from two Arabic language categories, CA and MSA (Teahan and Alhawiti, 2014). This corpus does not include diacritic characters which means the less bytes will be used in text compression.

2.7.3 Corpus A

Corpus A consists of 57 million words. It is a bilingual English/Arabic parallel corpus that was created by Alkahtani in 2015 (Alkahtani, 2015). The corpus was collected from the *Al Hayat* newspaper website and from the open source online corpus (OPUS). Corpus A is classified into several genres such as cinema, issues, stories and books. Corpus A has been used to clarify that the genera classification are slimier cross the languages.

2.8 Summary and Discussion

In this chapter, we have firstly reviewed the basic fundamentals of Arabic as this is relevant for research question 2, including its characters, writing system and morphology. We have then discussed a number of important text encoding approaches with particular attention to those involving the Arabic language and script. We found that UTF-8 is not only used widely for languages that need more than one byte to be encoded but also can combine different languages in the same file, which is useful for parallel corpora.

Finally, we have reviewed the Arabic corpora that are the main tools for all the experiments in this study. A number of Arabic corpora created in recent years have been described. The purpose of this chapter is to show the differences between Arabic's basic characteristics and those in other languages, such as English. There are important differences between Arabic and English, such as writing system and morphology, and similarities such as the size of their alphabets.

CHAPTER 3

LITERATURE REVIEW

Contents

3.1 Introduction

Natural language processing (NLP) is a field of study in computer science that is used in different applications and systems that are able to understand human natural language. There are several applications that depend on NLP, such as speech recognition, machine translation, text processing and artificial intelligence (Chowdhury, 2003).

Statistical NLP uses probabilistic, statistical and stochastic methods to solve problems in statistical grammar like statistical parsing, stochastic context-free grammar and hidden Markov models. Difficulties became apparent when processing long sentences that were highly ambiguous and required realistic grammars that could offer many possible analyses. On the other hand, Markov models and corpora methods were investigated for disambiguation (Manning and Schutze, 1999).

In this chapter, we review some basic ideas in statistical NLP as we adopt a statistical NLP approach for Arabic and English in our work. First, we introduce the concept of statistical language modelling in section 3.2, and then discuss the idea of n-gram language models in section 3.3 and the idea of n-graph language models in section 3.4. In section 3.5, we discuss the concepts of entropy and cross-entropy. Section 3.6 discusses the sparse data problem in statistical language modelling. In section 3.7, we illustrate various methods of text encoding. We discuss the fundamentals of text compression in section 3.8. The concept of grammar-based codes is discussed in section 3.9. Finally, a summary and discussion are provided in section 3.10.

3.2 Statistical Language Modelling

A statistical language model assigns a probability distribution $p(x_1, ...x_n)$ to sequences of characters or words (Ponte and Croft, 1998). A language model is useful for different applications such as speech recognition (Jelinek et al., 1975), automatic spelling correction (Kernighan et al., 1990) and machine translation (Brown et al., 1990). Text compression or more generally lossless compression is another application for distinct sequence prediction.

Witten and Bell stated that the language model for a natural language and how language models are both important for different applications in NLP. A statistical language

model for text is a set of information which approximates the structure and statistics of the text that actually exists. The model assigns a probability for each symbol (e.g., letter or character or word). Authorship, text compression and spell checking are some areas that are important for modelling these applications (Witten and Bell, 1990).

3.3 n-gram language models

An n-gram language model can be words, but more generally can also be sequences of other units such as letters, syllables or phonemes. The n-gram estimates the probability of a given word appearing in a sequence based on the previous words in the text.

Let $p(S)$ be the probability of a sequence S of m words $w_1, w_2, w_3, ..., w_m$ given by the formula:

$$P(S) = p(w_1)p(w_2|w_1)p(w_3|w_1,w_2)...p(w_m|w_1,...,w_{m-1}) \tag{3.1}$$

$$= \prod_{i=1}^{m}(w_i|w_1,w_2,...w_{i-1}). \tag{3.2}$$

Here, w_i is the word being predicted and $w_1, w_2, w_3, ..., w_{i-1}$ is the history or conditional context. Clearly, using a full history to build the language model would be computationally expensive. We can solve this problem by using the Markov assumption, where the conditioning context is considered equivalent to the preceding $w-1$ words (Jelinek, 1990). For instance, in the bigram model, only the previous word is used to determine probability:

$$P(S) \approx \prod_{i=1}^{m} p(w_i|w_{i-1}). \tag{3.3}$$

In the same way, the probability for a trigram model is determined by using the previous two words:

$$P(S) \approx \prod_{i=1}^{m} p(w_i|w_{i-2},w_{i-1}). \tag{3.4}$$

Models such as the bigram and trigram models that use Markov assumptions to predict the next word are referred to as Markov models. Generally, an n-gram model is known as an order $n-1$ Markov model (Markov, 1913) .

3.4 n-graph language model

An n-graph is an instance or specific case of a n-gram, where the unit is a character. In an n-graph language model, Markov models are also used to predict the next character in the text, based on previous characters. Let $p(S)$ be the probability of a sequence S of m characters $c_1, c_2, c_3, ..., c_m$ is given by the formula:

$$P(S) = \prod_{i=1}^{m} p(c_i | c_1, c_2 ..., c_{i-1}). \tag{3.5}$$

A 5-graph model uses the previous four characters to predict upcoming characters and is given by the formula:

$$P(S) \approx \prod_{i=1}^{m} p(c_i | c_{i-4}, c_{i-3}, c_{i-2}, c_{i-1}). \tag{3.6}$$

Character-based $n-$graph models have been widely applied in different applications in NLP and other applications, such as text compression, cryptography, OCR and spell checking (Teahan, 1998).

3.5 Entropy and Cross-Entropy

In 1948, Claude E. Shannon (Shannon, 1948) introduced the idea of the entropy of language, after which entropy became a fundamental idea in information theory.

Entropy means the measure of inability to predict the content of a message. The entropy for a language is low when the upcoming character is able to be predicted fairly easily. On the other hand, entropy is large when the upcoming character is very difficult to predict. In addition, the length after the message is encoded and the message's entropy should be equal (Bell et al., 1990).

Let X be a discrete random sequence with possible values $x_1, x_2, x_3, ..., x_n$, D be an alphabet of this sequence and $p(x)$ be the probability of X. The entropy E of discrete sequence X is:

$$E(X) = -\sum_{x_i \in D} p(x_i) \log p(x_i). \tag{3.7}$$

For instance, suppose there are sequences of text comprising letters a, b, c with probabilities $\frac{1}{2}$, $\frac{1}{2}$, $\frac{1}{10}$. We can find the entropy for these alphabets, or the average number of bits that each symbol needs to be encoded, as follows:

$$\begin{aligned} E &= -p(a)\log_2 p(a) - p(b)\log p(b) - p(c)\log_2 p(c) \\ &= -\frac{1}{2}\log\frac{1}{2} - \frac{1}{2}\log\frac{1}{2} - \frac{1}{10}\log\frac{1}{10} \\ &= 5.2 \quad bits. \end{aligned} \tag{3.8}$$

Teahan states that entropy is "a measure of how much uncertainty is involved in the selection of a symbol — the greater the entropy, the greater the uncertainty. It can also be considered a measure of the information content of the message — more probable messages convey less information than less probable ones" (Teahan, 1998).

Let $X = x_1, x_2, x_3, ..., x_n$ be a series of symbols in language modelling; equation 3.7 would thus be reformulated as:

$$E(X) = -\sum_{x_i \in D} p(x_1, x_2, x_3, ..., x_n) \log p(x_1, x_2, x_3, ..., x_n), \tag{3.9}$$

and the equation for per-word entropy is:

$$\frac{1}{n}E(X) = -\frac{1}{n}\sum_{x_i \in D} p(x_1, x_2, x_3, ..., x_n) \log p(x_1, x_2, x_3, ..., x_n). \tag{3.10}$$

In general, the entropy of a language with probability distribution L when the length of the message becomes very large is defined as:

$$E(X) = \lim_{x \to \infty} -\frac{1}{n}\sum_{x_i \in D} p(x_1, x_2, x_3, ..., x_n) \log p(x_1, x_2, x_3, ..., x_n). \tag{3.11}$$

The probability distribution of the language L is normally not known. Nevertheless, an upper bound to $H(L)$ can still be obtained by applying a model M for language L as an approximation:

$$H(L, M) = -\sum P_M(x_1, x_2, x_3, ..., x_n) \log P_M(x_1, x_2, x_3, ..., x_n). \tag{3.12}$$

The model $P_M(x_1, x_2, x_3, ..., x_n)$ is applied to estimate the probabilities. The entropy $H(L)$ is always less than or equal to the *cross-entropy* $H(L, M)$, as this is dependent on

the best possible source for the language model (Teahan, 2000).

$$H(L) \leqslant H(L,M). \tag{3.13}$$

In 1998, Teahan achieved a good compression rate to estimate the entropy of English by using PPM 1.48 bpc (to measure the compression rate, bit per character (bpc) is used) (Teahan, 1998). To measure the compression rate, bpc is used (the lower bpc compression rate is better) where:

$$bpc = \frac{Encoded\ file\ size(Bytes) \times 8}{Original\ file\ size(Bytes)}$$

To estimate an upper bound for entropy, text compression can be applied directly (Brown et al., 1992). Encoding the number of bits $b_M(x_1,x_2,x_3,...,x_n)$ for a sequence of symbols in the text $(x_1,x_2,x_3,...,x_n)$ by applying model M is defined as:

$$H(L,M) = \lim_{x\to\infty} \frac{1}{n} b_M(x_1,x_2,x_3,...,x_n), \tag{3.14}$$

where the number of bits demanded per character is $H(L,M)$ to encode a long series of text messages extracted from L.

Cross-entropy presents a measure of how successful the language model is on the test text: the closer the cross-entropy $H(L,M)$ is to $H(L)$, the less inexact the language model is. Thus, to find the best language model, one measures the cross-entropy; the better the performance of the model, the lower the cross-entropy (Brown et al., 1992).

3.6 The Sparse Data Problem

In section 3.2, we noted that an n-gram model language had to perform the prediction tasks $P(x_n|x_1,x_2,...,x_{n-1})$ and estimate the probabilities for a trigram model as follows:

$$p(w_i|w_{i-2},w_{i-1}) = \frac{C(w_{i-2},w_{i-1},w_i)}{C(w_{i-2},w_{i-1})}. \tag{3.15}$$

where C counts the number of times that token occurs in the text. In English text, many trigrams are rare. The problem of how to handle unseen words or rare events is known as *the sparse data problem*. (Teahan, 1998)

There are three approaches to processing n-gram language models: *blending, estimating* and *smoothing* (Chen, 1996).

The blending strategy is a process that combines predictions from different models into an overall probability. One way to combine various predictions is to assign for each model a weight and estimate the weighted sum of the probabilities (Bell et al., 1990) .

Estimating involves the problem of determining a good probability of estimating the next word. One approach is the *maximum likelihood (ML)* estimate:

$$P_{ML}(w_1, w_2, ..., w_n) = \frac{C(w_1, w_2, ..., w_n)}{T} \tag{3.16}$$

and

$$P_{ML}(w_n|w_1, w_2, ..., w_{n-1}) = \frac{C(w_1, w_2, ..., w_n)}{C(w_1, w_2, ..., w_{n-1})}, \tag{3.17}$$

where $C(w_1, w_2, ..., w_n)$ denotes the number of times the sequence of symbols $w_1, w_2, ..., w_n$ occur in the given text and T denotes the total number of sequence symbols in the training data or text. However, an immediate problem is encountered for the ML estimator. When the sequence of symbols $C(w_1, w_2, ..., w_n)$ has never occurred in the preceding text or is extremely rare, then the probability estimates for the language model are zero, and it is therefore unable to predict these novel symbols. (Chen, 1996).

A simple approach to overcoming the sparse data problem is to use more training text to train the language model. This is a direct approach that usually results in an improvement in the model's predictive performance, but the problem with this approach is the size of the training text or corpus required. To train large amounts of data, greater resources are required (Teahan, 1998).

Another simple approach to avoid the problem of zero probability and build a more compact language models is called *smoothing*, a process used to correct the probability estimate. One smoothing approach employs Laplace's Law (Laplace, 1814) by adding one to the various counts:

$$P_{LAP}(x_n|x_1,x_2,...,x_{n-1}) = \frac{C(x_1,x_2,...,x_n)+1}{C(x_1,x_2,...,x_{n-1})}. \tag{3.18}$$

In 1953, Good suggested the *Good-Turing* estimate (Good, 1953), which was initially used for estimating the size of species in population biology and then used as a basis for smoothing techniques. This method is based on the supposition that distribution frequency is binomial (Church and Gale, 1991).

If $r > 0$ and $r = C(x_1,x_2,...,x_n)$ is the frequency and N_r denotes the frequencies of frequency r:

$$P_G(x_1,x_2,...,x_n) = \frac{r^*}{N}, \tag{3.19}$$

where

$$r^* = (r+1)\frac{N_{r+1}}{N_r}. \tag{3.20}$$

If $r = 0$, the frequency r^* is thus reduced to:

$$r^* = \frac{N_1}{N_0} \tag{3.21}$$

and

$$P_G(x_1,X_2,...,X_n) = \frac{N_1}{N_0 N}. \tag{3.22}$$

From equations (3.21) and (3.22), we can see that the number of unseen occurrences of words N_0 is provided, but a crude estimate can be defined from V, where V is the size of the vocabulary, as follows:

$$N_0 = V^n - \sum_{r>0} N_r. \tag{3.23}$$

From the previous equation $\sum_{r>0} N_r \ll V^n$, then $N_0 \approx V^n$. For instance, the Essex Arabic Summaries Corpus (EASC) corpus has 58,973 characters (Elhaj et al., 2010), so N_0 for the bigram model is estimated as V^2 = 3,477,814,729. By comparing equation (3.17) with (3.23), we can see that the Good-Turing estimate makes the same adjustment for count r^* of unseen words in the n-gram.

TABLE 3.1: Common character encodings for several languages (Benoit, 2013)

Character Encoding	Byte(s) per Character	Language
ASCII	1	English
ISO-8859-1	1	West European
ISO-8859-2	1	Central and East European
ISO-8859-3	1	South European
ISO-8859-4	1	North European
ISO-8859-5	1	Slavic languages
ISO-8859-6	1	Arabic language
ISO-8859-7	1	Modern Greek
ISO-8859-8	1	Hebrew
ISO-8859-9	1	Turkish
ISO-8859-10	1	Nordic
ISO-8859-11	1	Thai language
ISO-8859-13	1	Baltic Rim
ISO-8859-14	1	Celtic
ISO-8859-15	1	East-Asian languages
ISO-8859-16	1	South-Eastern Europe
Shift JIS	2	Japanese
EUC-JP	up to 3	Japanese
Big5	2	Traditional Chinese
GB18030	up to 4	Simplified and Traditional Chinese

3.7 Text Encodings

A text encoding is used to present a repertoire of characters in a text language (Unicode Consortium, 1991). A number of text encodings have been designed for different natural languages.

Table 3.1 illustrates some general text encodings for different languages, such as ASCII for English and ISO-8859-6 for Arabic. It can be seen that most languages are encoded by using only one byte. There are two common encodings for Chinese language, Big5 and GB18030. Japanese also has two common encodings, Shift JIS and EUC-JP. Because of large alphabet sizes in Chinese and Japanese, their encodings require at least two bytes per character. In the following sections, English and Arabic encodings are discussed.

TABLE 3.2: The basic ASCII code (Coded Character Set, 1986)

Dec	Char	Dec	Char	Dec	Char	Dec	Char	Dec	Char
32	Space	52	4	72	H	92	\	112	p
33	!	53	5	73	I	93	]	113	q
34	”	54	6	74	J	94	^	114	r
35	#	55	7	75	K	95	_	115	s
36	$	56	8	76	L	96	‘	116	t
37	%	57	9	77	M	97	a	117	u
38	&	58	:	78	N	98	b	118	v
39	’	59	;	79	O	99	c	119	w
40	(	60	<	80	P	100	d	120	x
41	)	61	=	81	Q	101	e	121	y
42	*	62	>	82	R	102	f	122	z
43	+	63	?	83	S	103	g	123	{
44	‘	64	@	84	T	104	h	124	—
45	-	65	A	85	U	105	i	125	}
46	.	66	B	86	V	106	g	126	∼
47	/	67	C	87	W	107	k	127	DEL
48	0	68	D	88	X	108	l		
49	1	69	E	89	Y	109	m		
50	2	70	F	90	Z	110	n		
51	3	71	G	91	[	111	o		

3.7.1 English Encodings

ASCII is the first encoding system designed for information exchange (Benoit, 2013). ASCII codes are based on 128 characters containing English letters, numbers, popular symbols and basic punctuation, as well as 33 non-printing characters. Table 3.2 shows the basic ASCII code characters, with “Dec” referring to the decimal value of each character (Coded Character Set, 1986).

There is an extended ASCII encoding that uses 128 additional characters; it is widely used in European nations to encode more characters and is known as ISO-8859-1. Table 3.1 presents the different encoding languages based on ISO-8859, such as ISO-8859-1, ISO-8859-2, ISO-8859-3, ISO-8859-4, etc. (Benoit, 2013).

3.7.2 Arabic Encodings

Arabic text has been encoded using several methods that store and present Arabic characters electronically. Arabic letters and digits are encoded by using Arabic encoding,

but contextual forms in Arabic are not encoded (Kherallah et al., 2009). Three different methods are used to encode Arabic: ISO (8859-6) Arabic Encoding, Windows-1256 and UTF-8 Encoding (which was discussed in chapter 2).

3.8 Text Compression

Text compression is used to reduce the space used to store data by using different encoding techniques. Text compression is used for encryption, error correction, error detection and many others applications (Bell et al., 1990).

The data being compressed can be text, images, video, sound and so on. When dealing with compressed data, it is crucial to ensure that the decompressed file is the same size and has the same content as the original file.

3.8.1 Lossless and Lossy Compression

Data compression has become an important factor in advancing technology for several storage media and digital data file types, such as text, images, video and audio . There are two main compression categories: lossless and lossy.

Lossless compression techniques are used when the original text is important and it is not acceptable to lose any information from that text; examples include programming files, text files and medical images. On the other hand, when data is less important and some information from the original file can be lost without a noticeable impact, a lossy compression technique can be used.

Each approach has advantages and disadvantages. Lossy compression is usually limited to audio and video such as ALAC, ATRAC, MLP and ZRLE . For images, the lossy approach provides better compression than the lossless approach such as PGF, JPEG, BPG and PNG (Senthil and Robert, 2011) .

Generally, a text file is the input stream for the encoder, which means the encoder codes the input text that contains different kinds of symbols and compresses the symbol stream to generate a compressed output file. The goal is to have the compressed file be smaller than the original; in this case, compression is effective. The compressed file is defined using the parameters that are used by the algorithm used for compression. The idea of a

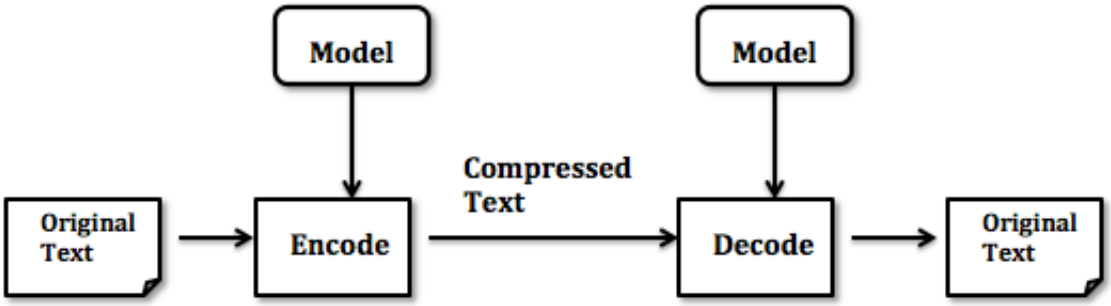


FIGURE 3.1: Using a model for compression (Bell et al., 1990)

model is helpful in recognising how a given encoder works. The model determines the parameters that need to be applied by the compression algorithm. In addition, in order to recover the original file, a decoder is needed; the decoder has knowledge of how the compressed file was encoded (Jacob et al., 2012) — see Figure 3.1.

There are four main types of text compression: statistical; dictionary; Burrows-Wheeler Compression; and grammar-based codes. Statistical compression is based on the probabilities for each symbol (a symbol may be a character or a word) such as PPM. More frequent symbols, which are more probable, require fewer bits to encode. On the other hand, dictionary-based compression uses a dictionary to replace repeated sets of characters in the text and depends on replacing a group of symbols by a matching dictionary code such as Lempel-Ziv (LZ77, LZ78 and LZW algorithms), Gzip, ZIP, xz and WinRAR.

Statistical compression and dictionary compression each have advantages and disadvantages. In terms of speed, dictionary compression is faster than statistical compression. However, statistical compression is better than dictionary compression in terms of compression rate (Bell et al., 1990). An alternative approach was devised by Michael Burrows and David Wheeler in 1994: Burrows-Wheeler Compression. The Burrows-Wheeler Transform (BWT) such as Bzip2, is a popular type of lossless compression and offers a good compression ratio with high speed (Blelloch, 2001).

In 1981, Rissanen and Langdon specified two steps to perform statical text data compression, modelling and encoding. Modelling works by estimating the probability for each character, and encoding works by using a coding technique such as arithmetic encoding to generate the compressed data(Rissanen and Langdon, 2012).

Figure 3.1 (Bell et al., 1990) shows how to use a model in order to compress data for statistical-based compression. Both the encoder and decoder use the same model. The model estimates probabilities that are used by the encoder to encode the original text into compressed data that will be sent to a decoder. The decoder uses the probabilities estimated by the model to decode the compressed data back into the original text.

In statistical compression, both the encoder and decoder use the same arithmetic coding to compress and decompress data based on similar codebooks (Bell et al., 1990).

3.8.2 Prediction by Partial Matching

In 1984, Cleary and Witten introduced PPM, which is one of the most effective statistical compression methods. Various versions of the original PPMA and PPMB (Cleary and Witten, 1984) have led to improved performance; examples include PPMC (Moffat, 1990), PPMD (Howard, 1993) and PPM* (Cleary and Teahan, 1997).

When PPM is applied to NLP, it achieves excellent results in many applications such as error detection and correction, cryptology, optical character recognition (OCR) and part-of-speech tagging (Cleary and Teahan, 1997; Teahan, 1998).

PPM depends on a bounded number of previous characters or symbols in order to make its prediction. In order to predict upcoming characters or symbols, PPM uses different orders of models, starting from the highest orders and moving down to the lowest. The escape probability estimates if a new symbol appears in the context (Cleary and Witten, 1984; Bell et al., 1990), and the model backs off to lower-order models when this occurs. Although more costly in terms of memory and speed of execution, PPM offers the best compression rate compared with other well-known compression methods such as dictionary-based methods and the BWT.

In PPM, each symbol or character is encoded via arithmetic coding using probabilities estimated by different models (Witten et al., 1987). To clarify the operation of PPM, Table 3.3 shows the state of different orders for PPM, where *k=2, 1, 0 and -1* after input string *"abcdbc"* has been handled. In this example, the maximum order is two to estimate the probability for symbols in the context. The PPM algorithm starts from the highest order *k=2*. In order to estimate the probability of the upcoming symbol or character, if the context predicts the next symbol or character successfully, the associated probability for this symbol or character will be used to encode it. Otherwise, the

TABLE 3.3: PPMC model after processing the string *"abcdbc"*

Order k=2			Order k=1			Order k=0			Order k=-1		
Prediction	c	p	Prediction	c	p	Prediction	c	p	Prediction	c	p
$ab \rightarrow c$	1	$\frac{1}{2}$	$a \rightarrow b$	1	$\frac{1}{2}$	$\rightarrow a$	1	$\frac{1}{10}$	$\rightarrow A$	1	$\frac{1}{\lvert A \rvert}$
$\rightarrow Esc$	1	$\frac{1}{2}$	$\rightarrow Esc$	1	$\frac{1}{2}$						
$bc \rightarrow d$	1	$\frac{1}{2}$	$b \rightarrow c$	2	$\frac{2}{3}$	$\rightarrow b$	2	$\frac{2}{10}$			
$\rightarrow Esc$	1	$\frac{1}{2}$	$\rightarrow Esc$	1	$\frac{1}{3}$						
$cd \rightarrow b$	1	$\frac{1}{2}$	$c \rightarrow d$	1	$\frac{1}{2}$	$\rightarrow c$	2	$\frac{2}{10}$			
$\rightarrow Esc$	1	$\frac{1}{2}$	$\rightarrow Esc$	1	$\frac{1}{2}$						
$db \rightarrow c$	1	$\frac{1}{2}$	$d \rightarrow b$	1	$\frac{1}{2}$	$\rightarrow d$	1	$\frac{1}{10}$			
$\rightarrow Esc$	1	$\frac{1}{2}$	$\rightarrow Esc$	1	$\frac{1}{2}$	$\rightarrow Esc$	4	$\frac{4}{10}$			

probability of escape will be estimated to encode an escape to move down to the next highest order, which in this case is k=1. This is repeated as needed until the encoder reaches the lowest order, which is *k=-1*. Then, the probability for all symbols or characters will be estimated and encoded by $\frac{1}{|A|}$, which is the size of the alphabet in the context. Experiments show the order that gives a good compression rate for English is order 5 (Cleary and Witten, 1984; Witten et al., 1987; Teahan, 1998). For Arabic text, experiments show that PPM algorithm gives a good compression rate at order 7 (Teahan and Alhawiti, 2015).

For example, if *"a"* follows the string *"abcdbc"*, the probability for the order 2 model would be $\frac{1}{2}$, because a successful prediction ("$ab \rightarrow c$") has been found. Suppose the string *"a"* follows the input string *"abcdbc"*; the probability of escape of $\frac{1}{2}$ for an order 2 model would be encoded arithmetically, and the process of encoding downgrades from order 2 model to the next order, which is order 1. In order 1, the encoder does not predict string *"a"*, so another escape probability $\frac{1}{2}$ is encoded, and the process of encoding downgrades to order 0. In order 0, the probability is $\frac{1}{10}$ for string *"a"*. Therefore, to encode string *"a"*, the total probability is $\frac{1}{2} \times \frac{1}{2} \times \frac{1}{10} = \frac{1}{40}$, which requires 5.2 bits (that is, $-\log_2 \frac{1}{40} = 5.2$).

If a new string *"n"* follows the input string *"abcdbc"*, this string has not appeared before, so the process encodes down to the lowest order, which is *k=-1*. Here *k=-1*, all strings are encoded with the same probabilities, which are $\frac{1}{|A|}$ where $|A|$ is the size of the alphabet. Supposing that the size of the alphabet is 256 (for ASCII), the probability

TABLE 3.4: Estimating probabilities for PPM variants

Model	Escape Probability	Symbol Probability	
PPMA	$e = \frac{1}{n+1}$	$p(s) = \frac{c(s)}{n+1}$	(Cleary and Witten, 1984)
PPMB	$e = \frac{t}{n}$	$p(s) = \frac{c(s)-1}{n}$	(Cleary and Witten, 1984)
PPMC	$e = \frac{t}{n+t}$	$p(s) = \frac{c(s)}{n+t}$	(Moffat, 1990)
PPMD	$e = \frac{t}{2n}$	$p(s) = \frac{2c(s)-1}{2n}$	(Howard, 1993)

for the new string *"n"* is $\frac{1}{256}$ for the order -1 context. Therefore, the character *"n"* is encoded by using the probability $\frac{1}{2} \times \frac{1}{2} \times \frac{4}{10} \times \frac{1}{256}$, which requires 11.4 bits.

3.8.2.1 PPM's Probability Estimation

PPM models need to compute the escape and novel symbol probability for the context model. PPM methods use two different mechanisms, *full exclusion* and *update exclusion* (these two schemes are discussed in detail in section 3.8.2.4).

Table 3.4 shows how the different PPM models estimate the probabilities. According to the equations in Table 3.4, e denotes the escape probability and $p(s)$ represents the probability for symbol s. $C(s)$ is the number of times that symbol s occurs in the context and t is the total number of unique characters that exist in the context. Let n denote the number of tokens or the total number of times that all symbols have been counted. For example, if three symbols (*a, b* and *c*) have occurred seven times previously (t=3), with symbol *a* following two times, with symbol b following three times and with symbol *c* following three times, for the models PPMA, PPMB, PPMC and PPMD, the escape probability e would be $\frac{1}{8}$, $\frac{3}{7}$, $\frac{3}{10}$ and $\frac{3}{14}$, respectively.

3.8.2.2 PPM Variants

There are many variants of the basic algorithm, such as PPMA and PPMB (Cleary and Witten, 1984), PPMC (Moffat, 1990), PPMD (Howard, 1993), PPM* (Cleary and Teahan, 1997), PPMZ (Bloom, 1998) and PPMii (Shkarin, 2002), each of which is named for the method of escape used.

PPMA and PPMB were established by Cleary and Witten (1984), PPMC was presented by Moffat Moffat (1990) and PPMD was proposed by Howard (1993). Many experimental results show that PPMD and PPMC are better than PPMA and PPMB in terms of compression rate and that PPMD typically outperforms these other methods.

PPMP also was presented another escape method, Witten and Bell (1991). PPMP depends on the assumption that tokens occur depending upon a Poisson model. Method P assesses the escape probability as:

$$e = \frac{t_1}{n_1} - \frac{t_2}{n_2} - \frac{t_3}{n_3} - \ldots \quad . \tag{3.24}$$

where t_i is the number of symbols appearing i times in the context and n_i is the number of times that all symbols have been counted.

Because n is normally very large in method P, Method X computes only the first term of the previous equation, reducing the escape probability to:

$$e \approx \frac{t_1}{n}. \tag{3.25}$$

When t_1 is equal to 0 or equal to n, methods P and X would break down because the novel symbol probability is equal to 0 or 1 in these cases. To address this problem, the probability for novel symbols is set to 1. PPMXC was introduced by Witten and Bell (1991). Method XC combines methods X and C:

$$e = \begin{cases} \frac{t_1}{n}, & \text{when } 0 < t_1 < n \\ \frac{t_1}{n+t_1}, & \text{otherwise.} \end{cases}$$

Moffat, Neal and Witten (Moffat et al., 1998) presented a new method, named PPMX1, to address the problem in method X by adding one to the count:

$$e = \frac{t_1 + 1}{n + t_1 + 1} \qquad \text{and} \qquad p(s) = \frac{c(s)}{n + t_1 + 1}. \tag{3.26}$$

Another type of PPM model is known as PPM* and depends on unbounded length contexts. PPM* was introduced by Cleary and Teahan (1997). In order to predict the next symbols, PPM* uses all substrings for the context rather than selecting only some

substrings. Although PPM* offers good compression results that are often better than PPM, it requires more memory space and execution time (Cleary and Teahan, 1997).

In 1998, Bloom developed a new PPM model called PPMZ (Bloom, 1998). PPMZ combines the features of PPM* and PPM and addresses the problem of finite context order and unbounded length context, based on two approaches known as local order estimation (LOE) and secondary escape probability estimation (SEE). Experimental results shows that PPMZ achieves good results on the Calgary corpus, a standard corpus used for evaluating different compression algorithms.

Shkarin developed PPMii in 2002; it uses a technique called information inheritance. PPMii solves the issue of the lack of information for long contexts by allowing the child contexts or known short contexts to inherit or pass on information from their parents. PPMii achieves better results in terms of both compression speed and compression rate (Shkarin, 2002).

Wu and Teahan developed a new PPM model for Chinese called PPM-ch. The main idea in this algorithm is that PPM-ch does not use exclusions to process the Chinese language text faster because of the large Chinese alphabet. Wu and Teahan then presented PPMO in 2008. It depends on encoding two separate streams: order streams and symbols streams. Two steps involving these separate streams are required to encode Chinese texts. The order is encoded first, after which symbols are encoded (Wu and Teahan, 2008).

More recently, Alhawiti and Teahan introduced other PPM models called BS-PPM and CS-PPM (Teahan and Alhawiti, 2015). The idea behind BS-PPM is to use the most frequent bigraphs by using pre-processing techniques before the encoding stage and postprocessing techniques after the decoding stage. BS-PPM achieves good results for different languages including English, Arabic, Welsh and Armenian. CS-PPM splits the encoding process into two separate vocabulary and symbol streams. CS-PPM obtains good compression rates for English, Arabic, Welsh, Armenian, Kurdish, Persian and Russian.

Figure 3.2 shows that the compression ratios for different PPM methods using the Calgary corpus has been decreasing over time. For example, in 1990, the compression rate was up to 2.45 bpc for the PPMC model. The ratio decreased to between 2.32 bpc and 2.35 bpc by 1993 and 1997 for both PPMD and PPM*. During the period between 1998 and 2002, PPMZ and PPMii were devised to give better results than all previous

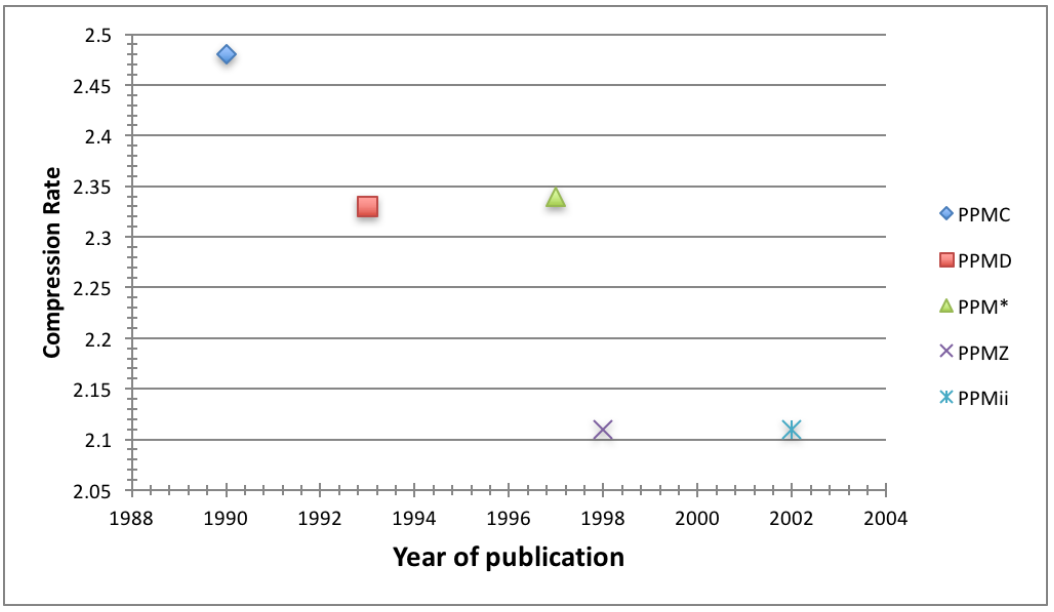


FIGURE 3.2: Compression ratios of various PPM methods using the Calgary Corpus mapped against year of publication

PPM methods, with a compression ratio of 2.11 bpc (the lower bpc compression rate is better).

Table 3.5 summarises the different PPM variants; it displays the name of the variant in column one, the date it was first published in column two and the description of the different PPM variants in column three.

3.8.2.3 Comparing PPM variations

Many experiments demonstrate that PPM can be used not only for character-based methods but also for word-based methods. In the character-based method, PPMD is better than other methods such as PPMA, PPMB and PPMC. On the other hand, PPMP and PPMX are poorer than PPMXC in performance. In the word-based method, Moffat employed PPMC to encode words (Moffat, 1989). PPMX1 works very well in word-based methods and speech schemes (Cleary and Teahan, 1997). Experiments show that PPMZ outperforms other methods in performance (Bloom, 1998) because it addresses the problem in previous PPM variations, which depended on a finite context order. In addition, many experiments show that PPM works very well in different languages. PPMO accomplishes good results for Chinese, and BS-PPM and CS-PPM achieve the best compression rates for languages such as Arabic, Persian, Kurdish, English, Welsh, Chinese and Russian. Table 3.6 shows the experimental results for PPM variants using

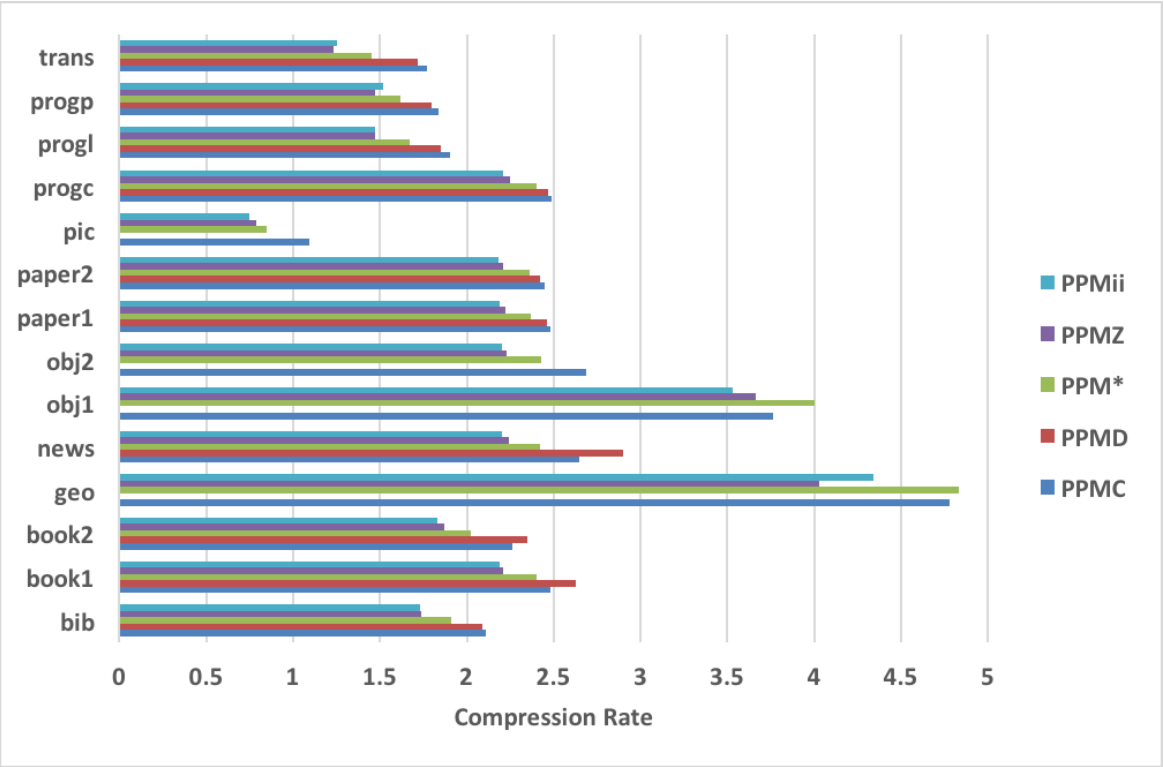


FIGURE 3.3: Compression ratios of various PPM methods using the Calgary Corpus

the Calgary Corpus; the difference in results is clear. PPMD (Howard, 1993) does not include *geo, pic, obj1* and *obj2* and showed a 6% improvement on average over PPMC (Moffat, 1990). PPM* (Cleary and Teahan, 1997) achieved a 5.64% average enhancement over PPMC. PPMZ (Bloom, 1998) and PPMii (Shkarin, 2002) achieved similar overall results of 2.11 bpc with a 14.9% average improvement over PPMC and 9.82% over PPM*.

A bar chart of the results in Table 3.6 is presented in Figure 3.3. It clearly shows that there is a high variation among PPM models. PPMC and PPMD showed high compression rates using the Calgary corpus. PPMZ and PPMii had similar results with most types of files.

3.8.2.4 PPM's Blending Mechanism

PPM models employ a blending strategy called *exclusion*. These models combine the prediction of all symbols of length less than or equal to the maximum order of the model m taken from the previous text. The exclusion technique uses the escape to exclude lower order predictions, including the order 0 model and the order -1 model

TABLE 3.5: PPM Variants

Variant	Publications	Description
PPM	Cleary and Witten (1984)	The original concept of Prediction by Partial Matching that uses Markov models to analyze probability and predicts the next characters or words based on text preceding. This method uses escape (novel) probabilities to show where matches do not appear in the models. Originally, there were two methods proposed: A and B.
PPMC	Moffat (1990)	PPMC offered a new method of calculation of novel (escape) probabilities; it is a hybrid of the original PPM methods A and B and is used to estimate the probability by calculating the number of times that an unseen symbol character has existed before in the context.
PPMP	Witten and Bell (1991)	PPMP calculates escape probability using a Poisson process model.
PPMX	Witten and Bell (1991)	In this method, the escape probability can be estimated based on the number of symbols or characters that were previously seen in the character stream only once, instead of using the randomness of all symbols or characters in the PPMP model.
PPMD	Howard (1993)	This method deals with a new symbol by a factor of $\frac{1}{2}$ rather than 1, as in PPMC.
PPM*	Cleary, Teahan and Witten (1995)	This method allows the use of unbounded length context, which employs a higher order in the model with lower use of the escape.
PPMZ	Bloom (1999)	In this method, Bloom improved three elements: LOE, unbounded deterministic context and SEE.
PPMT	Teahan and Harper (2001)	PPMT depends on the Viterbi algorithm, which uses text mining to categorize the text before the compressed process.
PPMii	Shkarin (2002)	PPMii is the first model that uses a form of information inheritance to initialize a new context at a high order level from a lower order context.
PPM-ch PPMO	Wu and Teahan (2008)	Created for Chinese text, PPM-ch uses pre-processing and does not use exclusion escapes to encode Chinese text.
BS-PPM CS-PPM	Teahan and Alhawiti (2014)	In the BS-PPM method, Teahan and Alhawiti used UTF-8 encoding and pre-processing techniques by replacing 100 bigraphs in order to improve the compression ratio for different languages. CS-PPM also uses UTF-8 encoding to substitute all multi-byte or single-byte characters; it then produces two files, a vocabulary stream and a symbol stream, each of which is encoded separately.

predictions, from the final probability assessment. Generally, the order 0 model is used to predict a character based on its unconditioned probabilities, while the order -1 model is used to make sure that all possible characters are assigned a finite probability (Teahan, 1998).

The blended probability of symbol s, as stated by Bell, Cleary and Witten, is given by (Bell et al., 1990):

$$p(s) = \sum_{i=-1}^{m} q_i p_i(s), \tag{3.27}$$

where p_i and q_i are the probabilities and the weight assigned to the order i model, respectively. The non-zero weights are estimated by the predictions from lower orders in order to avoid zero probabilities.

TABLE 3.6: Compression ratios of various PPM methods applied to the Calgary corpus

Files	PPMC	PPMD	PPM*	PPMZ	PPMii
bib	2.11	2.09	1.91	1.74	1.73
book1	2.48	2.63	2.40	2.21	2.19
book2	2.26	2.35	2.02	1.87	1.83
geo	4.78	-	4.83	4.03	4.34
news	2.65	2.90	2.42	2.24	2.20
obj1	3.76	-	4.00	3.66	3.53
obj2	2.69	-	2.43	2.23	2.20
paper1	2.48	2.46	2.37	2.22	2.19
paper2	2.45	2.42	2.36	2.21	2.18
pic	1.09	-	0.85	0.79	0.75
progc	2.49	2.47	2.40	2.25	2.21
progl	1.90	1.85	1.67	1.47	1.47
progp	1.84	1.80	1.62	1.47	1.52
trans	1.77	1.72	1.45	1.23	1.25
Avg.	2.48	2.33	2.34	2.11	2.11

The equivalent weight w_o of the escape probabilities in the PPM model can be calculated by:

$$w_o = (1 - e_o) \prod_{i=o+1}^{l} e_i \qquad -1 \leqslant o < l$$

and

$$w_l = 1 - e_l.$$

Let e_o be the probability of the escape in order o and the highest order that makes a non-zero prediction be l, where $-1 \leqslant o \leq 1$.

The weighted contribution estimation of the PPM model to the blended probability of the symbol s is thus:

$$w_o p_o(s) = (1 - e_o) p_o(s) \prod_{i=o+1}^{l} e_i. \tag{3.28}$$

Further improvement was made to PPM's blending algorithm by Bell, Cleary and Witten with what they called *full exclusion* (Bell et al., 1990). Using this mechanism, the characters predicted by a higher order will not be predicted in a lower order context since these characters would already have been encoded. Using full exclusion requires extra computation, since each symbol has to be checked (Teahan, 1998).

Another mechanism called *update exclusion* was introduced by Moffat (1990). With this mechanism, the count is updated in the context only if it is not predicted by a higher order context. Update exclusion improves the program execution time, since it removes the need to update the counts in the lower order contexts (Bell et al., 1990).

3.9 Grammar-based codes

Grammar-based compression algorithms depend on using a CFG to help compress the input text. The grammar and text are compressed by arithmetic coding or by different statistical encoders. Grammar-based compression schemes often outperform other compression schemes (Sayood, 2017). Three examples of grammar-based compression schemes are the Sequitur algorithm (Nevill-Manning and Witten, 1997), the Re-Pair algorithm (Larsson and Moffat, 2000) and irreducible grammars (Kieffer and Yang, 2000).

3.9.1 Sequitur algorithm

Sequitur is an on-line algorithm that was developed by Nevill-Manning and Witten (1997). It uses a hierarchical structure as specified by a recursive grammar that is generated iteratively in order to compress the text. Sequitur depends on repeatedly adding rules to the grammar for the most frequent digram sequences (which may consist of terminal symbols that appear in the text or non-terminal symbols that have previously been added to the grammar). The rule for the start symbol S shows the current state of the corrected text sequence as it is being processed. To see how the Sequitur algorithm works more clearly, the following example is provided.

The idea is to generate a grammar for the sequence "ababbbabab" and encode the grammar in order to help encode the sequence. The sequence will be processed from left to right. Two rules should be satisfied: digram uniqueness and the utility rule (see section 3.9.3 for more details).

Note that we have a duplicate digram, "ab", so a new rule $A \rightarrow ab$ is generated and we get:

$S \rightarrow AAbbAA$

$A \rightarrow ab$.

Next, a new rule is defined $B \rightarrow AA$ which results in:

$S \rightarrow BbbB$

$A \rightarrow ab$

$B \rightarrow AA$.

In the Sequitur algorithm, the grammars are encoded by using an implicit coding method (Sayood, 2017).

3.9.2 Re-Pair algorithm

The Re-Pair algorithm proposed by Larsson and Moffat in 2000 is off-line. Like Sequitur, Re-Pair replaces the most frequent pair of symbols with a new symbol in the source message, essentially extending the alphabet. The frequencies of symbol pairs are then re-evaluated, and the process repeats until there are no longer any pairs of symbols that occur twice. This off-line process can be considered generating a dictionary, after which an explicit representation of the dictionary is encoded as part of the compressed message (Larsson and Moffat, 2000). To see how Re-Pair works, the following phrase is used "ybaab•ybaab•ybaab•ybaab•ydd•ydd•ydd".

Note that the pair **•y** occurs most often. By replacing this pair by a new rule, we get

$S \rightarrow$ *ybaabAbaabAbaabAbaabAddAddAdd*

$A \rightarrow$ •*y*.

As we can see, there are several pairs that exist with the same frequency, such as *ba, aa* and *ab*. While they are equally frequent pairs, Larsson and Moffat suggest that picking one or the other will not affect the final result. To solve this issue, they replace the pairs in the sequence in order from left to right. A new rule is thus generated, and the string *ba* is replaced with rule *B* in the sequence.

$S \rightarrow yBabABabABabABabAddAddAdd$

$B \rightarrow ba$.

The next rule is

$C \rightarrow ab$

making the sequence

$S \rightarrow yBCABCABCABCAddAddAdd$.

From the previous sequence, there are two pairs with the same frequency, with both occuring four times: *BC* and *CA*. In this case, rule *A* occurs first, so the new rule is

$D \rightarrow CA$

and the sequence becomes

$S \rightarrow yBDBDBDBDddAddAdd$.

It is clear that the pair *BD* occurs most frequently (four times), so the new rule *E* is created:

$E \rightarrow BD$

and the sequence becomes

$S \rightarrow yEEEEddAddAdd$.

The next most frequent pair is *dd*:

$F \rightarrow dd$,

which gives the sequence

$S \rightarrow yEEEEFAFAF$.

Now, there are three pairs that occur twice: *EE, FA* and *AF*. The pair *AF* is replaced by the new rule *G*:

$G \rightarrow AF$,

resulting in the sequence

$S \rightarrow yEEEEFGG$.

Lastly, there is only one pair that occurs twice:

$H \rightarrow EE$,

which gives the final sequence

$S \rightarrow yHHFGG$.

The final set of rules is:

$A \rightarrow \bullet y$

$B \rightarrow ba$

$C \rightarrow ab$

$D \rightarrow CA$

$E \rightarrow BD$

$F \rightarrow dd$

$G \rightarrow AF$

$H \rightarrow EE$.

In the Re-Pair algorithm, the final sequence is encoded by applying a zero order entropy code (Larsson and Moffat, 2000).

3.9.3 Irreducible Grammars

Kieffer and Yang introduced irreducible grammars, a lossless compression scheme that transforms an input text into a context-free grammar; then, the grammar can be encoded by using arithmetic coding. Irreducible grammars use four rules to generate the grammar (Kieffer and Yang, 2000). The rules are as follows:

Rule 1: To ensure all the rules on the right side are used only once (in the Sequitur algorithm, this is called rule utility).

Rule 2: To ensure that no string of a length greater than one occurs more than once in the grammar (in the Sequitur algorithm, this is called digram uniqueness).

Rule 3: If there are two rules that each contain the same string, then a new rule is created and replaced with a new variable symbol.

Rule 4: If there is a string that is part of a string on the right hand side and there is a rule that generates this string, then this string is replaced with the variable symbol.

To see how these rules are used, the following example is provided.

To generate a grammar for the the sequence “bababababbbbaabbbbababab” using Rule 2 for repeated substrings *babab*, we get

$S \rightarrow bababababbbbaabbbbababab$

$A \rightarrow babab$.

Now, Rule 2 is applied again for repeated substrings *abbba*:

$S \rightarrow AabbBBA$

$A \rightarrow babab$

$B \rightarrow abbba$.

As we can see that the substring ”ab” occurs in both rule A and rule B, by using Rule 3 we get:

$S \rightarrow AabbBBA$

$A \rightarrow bCC$

$B \rightarrow Cbba$

$C \rightarrow ab$.

Finally, applying Rule 4, the grammar becomes:

$S \rightarrow ACbBBA$

$A \rightarrow bCC$

$B \rightarrow Cbba$

$C \rightarrow ba$.

3.10 Summary and Discussion

In this chapter, we have first reviewed several fundamental theories in NLP, along with important concepts of statistical language modelling. We have then reviewed the basic use of NLP and why it is important for other applications.

Text compression is the basic approach of this study and is discussed in this chapter. Some other language models have been examined that are essential for characters and words. We have also reviewed a number of important text encoding schemes and the types that are used, especially for English and Arabic. This chapter has also examined the difference between the types of data compression, lossless and lossy, and the uses of both techniques. We have reviewed various PPM models with an eye to which ones offer the most effective statistical compression methods.

Finally, grammar-based codes have been discussed; they use a CFG to help compress text. The three techniques of grammar-based compression schemes have been reviewed: the Sequitur algorithm, the Re-Pair algorithm and irreducible grammars.

The next chapter presents a new scheme that uses grammar-based codes and the PPM model to achieve the best compression rate for different natural language texts and the Calgary corpus.

CHAPTER 4

GRAMMAR-BASED PRE-PROCESSING FOR PPM

Contents

4.1 Introduction

One of the primary motivations for our research is the application of grammar-based techniques with PPM to NLP (as per question 1). Therefore, we introduce in this chapter the results of the use of a new grammar-based approach for PPM on natural language texts. PPM has achieved excellent results in various NLP applications such as language identification and segmentation, text categorisation, cryptology and OCR (Teahan, 1998).

The new approach uses a grammar-based compression algorithm that depends on using a CFG to help compress the input text. Two examples of grammar- based compression schemes are the Sequitur algorithm (Nevill-Manning and Witten, 1997) and the Re-Pair algorithm (Larsson and Moffat, 2000), as discussed in the previous chapter.

The new approach uses preprocessing prior to PPM. Abel and Teahan (2005) discuss text pre-processing techniques that have been found useful at improving the overall compression for the Gzip, BWT and PPM schemes. The techniques work in such a way that they can easily be reversed while decoding in a post-processing stage that follows the decompression stage. The methods discussed include a long list of prior work in this area and several new techniques. We present experimental results that show significant improvement in overall compression for the Calgary Corpus using our new grammar-based preprocessing approach.

In this chapter we use both grammar-based compression and PPM model for different text compression. Our new approach is discussed in the section 4.2. Then in section 4.3, we discuss experimental results on natural language texts by comparing how well the new scheme performs compared to other well-known methods. The summary and conclusions are presented in the final section.

Purpose of Chapter and state as contributed to a journal published (W. Teahan and N. Aljehane, "GRAMMAR-BASED PRE-PROCESSING FOR PPM," International Journal of Computer Science & Information Technology (IJCSIT) Vol 9, No 1, February 2017).

4.2 Grammar based pre-processing for PPM (GR-PPM)

In this section, a new off-line approach based on CFGs is presented for compressing text files. This algorithm, which we call *GR-PPM* (short for *Grammar based pre-processing for PPM*), uses both CFGs and PPM to serve as a general-purpose adaptive compression method for text files.

In our approach, we essentially use the N most frequent n-graphs (e.g., bigraphs when n=2) in the source files to first generate a grammar with one rule for each n-graph. Every time one of these n-graphs occurs in the text, we then substitute the single unique non-terminal symbol as specified by its rule in the grammar in a single pass through the file. This is done during the pre-processing phase prior to the compression phase in a manner that allows the text to be easily regenerated during the post-processing stage. For bigraphs, for example, we call the variant of our scheme *GRB-PPM (Single-Pass Grammar Bigraphs for PPM)*. Our new method shows significantly positive results when replacing the N most frequent symbols for bigraphs (for example, when N=100) but also using trigraphs (when n=3) in a variant which we have called *GRT-PPM (Single-Pass Grammar Trigraphs for PPM)*.

Each natural language text contains a high percentage of common n-graphs that comprise a significant proportion of the text (Teahan, 1998). Substitution of these n-graphs using our CFG scheme and standard PPM can significantly improve overall compression, as shown below. For example, natural languages contain common sequences of two characters (bigraphs) that often repeat in the same order in many different words, such as the English *"th"*, *"ed"* and *"in"* and the Arabic "لا", "ال" and "في" and so on.

Table 4.1 shows the 20 most frequent bigraphs for different languages, including Arabic, Persian, English and Welsh, from several corpora. For English, the Brown Corpus for American English (Francis and Kucera, 1979) and the LOB Corpus for British English (Johansson, 1986) were used. For Arabic, we use the BACC as reference corpus (Teahan and Alhawiti, 2014). For Persian, the Hamshahri corpus is used (AleAhmad et al., 2009). The LCMC corpus is used for Chinese (McEnery and Xiao, 2004), and the CEG corpus for Welsh (Ellis et al., 2001). The frequencies of these characters in reference corpora can be used to define the n-graphs that will be substituted (without the need to encode the grammar separately, making it possible to have the algorithm work in an on-line manner rather than off-line). However, although this method can be

TABLE 4.1: Bigraphs and their frequency in four corpora

Bigraph	Arabic	Persian	English	Welsh
1	ال	ان	th	dd
2	لم	ای	he	yn
3	له	ار	in	ww
4	ما	را	er	yd
5	فا	ند	an	ed
6	عل	ست	re	ae
7	بي	در	on	ol
8	في	اس	en	an
9	ان	ها	at	ar
10	نا	دا	or	di
11	عن	از	nd	th
12	لا	ای	es	wy
13	من	با	is	od
14	حم	رد	ti	ym
15	اب	ما	te	we
16	نا	ام	ed	go
17	لى	به	ar	da
18	ول	ود	st	ia
19	بن	ال	it	er
20	لل	ور	of	ei

quite effective, what we have found to be most effective for our scheme is to determine the list of *n*-graphs that define the grammar in a single or double pass through the text that will be compressed prior to the compression phase, and then encoding the grammar separately, along with the corrected text, which is encoded using PPM.

Our method replaces common sequences similar to both Sequitur and Re-Pair, but unlike them, this is not done iteratively on the current most common digram sequence or phrase, but is done by replacing the most common sequences, as the text is processed from beginning to end in a single pass (although further passes may occur later). In addition, the PPM algorithm is used as the encoder once the common sequences

have been replaced, whereas Re-Pair uses a dictionary-based approach for the encoding stage. Like Re-Pair, our method is only off-line during the phase that generates the grammar.

Our approach adapts the bigraph replacement text pre-processing approach of Teahan (1998) by using an off-line technique to generate the list of bigraphs first from the source file being compressed. This approach is considered within a grammar-based context, and the approach is further extended by considering multiple passes and recursive grammars.

Figure 4.1 shows the whole GR-PPM process. First, the CFG will be generated from the original source files by taking the N most frequent n-graphs and replacing them with non-terminal symbols, as defined by their rules in the grammar. After the rules are produced, the sender will do grammar-based pre-processing to correct the text. Then, the corrected text is encoded using PPMD, and the resulting compressed text is sent to the receiver. The receiver then decodes the text by using PPMD to decompress the compressed file. Grammar-based post-processing then facilitates the reverse mapping by replacing the encoded non-terminal symbols with the original n characters or n-graphs.

One final step is the need to encode the grammar so that it is also known by the receiver since, as stated, we have found that we can achieve better compression by separately encoding a grammar generated from the source file itself instead of using a general grammar (known to both sender and receiver) generated from a reference corpus. We choose to use a very simple method for encoding the grammar—simply transmitting the lists of bigraphs or trigraphs directly. Thus, encoding a grammar containing $N = 100$ (we use arbitrarily value) rules for the GRB-PPM scheme where $n = 2$ and character size is 1 byte (for ASCII text files, say) will incur an overhead of $N \times n$ bytes or 200 bytes for each file being encoded.

Table 4.2 illustrates the process of GRB-PPM and GRT-PPM using a line taken from a song by Manfred Mann (as a good example of repeating characters). The sequence is "*singing she looked good she looked fine*". For GRB-PPM, for example, there are four bigraphs which are repeated more than once in the text: *in*, *sh*, *oo* and *ed* . Thus, these bigraphs will be substituted with new non-terminal symbols , say A, B, C and D, respectively, and included in the grammar (for example, if we choose $N = 4$). For GRT-PPM, there are four trigraphs that appear more than once in the text: *ing, she, loo*

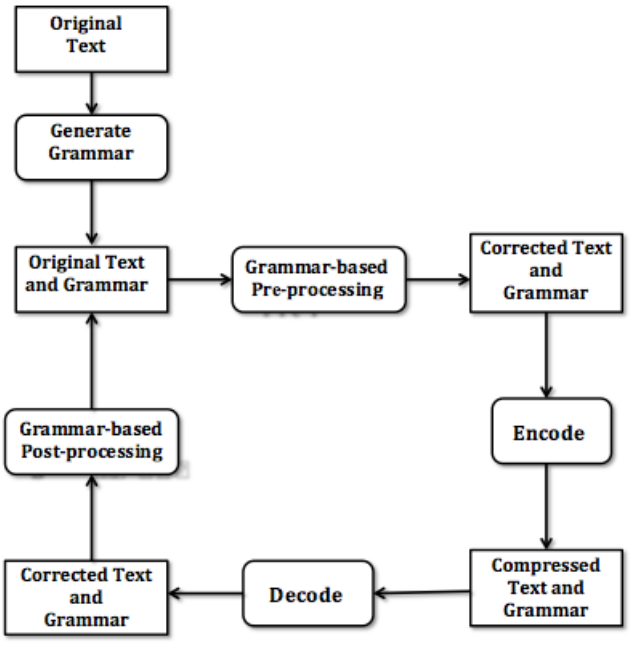


FIGURE 4.1: The complete process of Grammar-Based Pre-Processing for Prediction by Partial Matching (GR-PPM)

TABLE 4.2: An example of how GR-PPM works with a sample text

GR-PPM	Grammar	String and Corrected Strings
		singing.she.looked.good.she.looked.fine
GRB-PPM	$A \rightarrow in,\ B \rightarrow sh,\ C \rightarrow oo,\ D \rightarrow ed$	sAgAg.Be.lCkD.gCd.Be.lCkD.fAe
GRT-PPM	$E \rightarrow ing,\ F \rightarrow she,\ G \rightarrow loo,\ H \rightarrow ked$	sEE.F.GH.good.F.GH.fine

and *ked*. These trigraphs will be replaced with new non-terminal symbols, say *E*, *F*, *G* and *H*, respectively. The substitution is executed as the text is processed from left to right for both GRB-PPM and GRT-PPM. In practice, non-terminal are symbol number for English starting from 257 until 300 for N=4 (for ASCII). For Arabic, non-terminal also symbol number range from 432 until 436 for N=4 (for UTF8). For example, in the table 4.2 non-terminal symbols are (A=257, B=258, C=259 and D=300).

The grammar in both GRB-PPM and GRT-PPM shares the same characteristic, which is that no pair of characters appears in the grammar more than once. This property ensures that every bigraph in the grammar is unique, a property called *non-terminal uniqueness* in the terminology proposed by Nevill-Manning and Witten (1997). To make sure that

each rule in the grammar is useful, the second property, referred to as *rule utility*, is that every rule in the grammar is used more than once in the corrected text sequence. These two features are in the grammar that GR-PPM generates and are now discussed in more detail.

4.2.1 Non-terminal uniqueness

For the more general case (i.e. considering n-graphs, not just bigraphs), each n-graph has to appear only once in the grammar and is also replaced once by a non-terminal symbol. To prevent the same n-graph from occurring elsewhere in the corrected text sequence, each n-graph is replaced based on the n-graph that is generated by the algorithm. In the example in Table 4.2, the list of most frequent bigraphs that form the grammar is added at the same time as the text is processed in each pass. When *in* appears in the text more than once, the new non-terminal symbol A is substituted. By contrast, in the Sequitur algorithm (Nevill-Manning and Witten, 1997), only the most frequent digram (or bigraph using our terminology) is added incrementally to the grammar.

4.2.2 Rule Utility

Every rule in the grammar should be used more than once to ensure that the rule utility constraint is applied. For example, when *oo* appears in the text more than once in table 4.2, the new non-terminal symbol C is substituted. In our approach, rule utility does not require creating and deleting rules, which makes the rules more stable. This method avoids the requirement of exterior data structures. However, in the Sequitur algorithm, when a new letter in the input appears in the first rule S, the grammar creates a new rule and deletes the old digram (Nevill-Manning and Witten, 1997). The process of deleting and creating rules in the grammar each time a new symbol appears is unstable and inefficient. In the Sequitur approach, the grammar is dynamically created. To avoid separate data structures, we apply a multi-pass approach for GR-PPM and add multiple symbols to the grammar at the same time, which leads to greater stability and efficiency.

Figure 4.2 summarises the algorithm using pseudo-code. Line 1 is for a loop to define how many passes the algorithm performs (from 1 up to a maximum of P passes). Lines

```
1  For each pass up to P
2     Find the N most frequent bigraphs in the current sequence
       that do not contain a space or a punctuation character
3     For each of the N most frequent n-graphs do
4         Substitute the new n-graph with its non-terminal symbol
5     End for
6  End for
7  Use PPMD to encode the corrected sequence
```

FIGURE 4.2: Pseudo-code for the GR-PPM algorithm

2 to 4 are for a loop to find the N most frequent bigraphs and replace them with non-terminal symbols. Lines 3 through 5 implement the bigraphs' utility constraints. Line 7 compresses the final text file by using PPMD after N bigraphs have been replaced.

4.3 Experimental Results

This section discusses experimental results for variants of the GR-PPM method described above for compression of various text files and compares our results with other well-known schemes.

We have found that variants of the GR-PPM algorithm achieve the best compression ratios for texts in different languages, such as English, Arabic, Chinese, Welsh and Persian. We have also compared the results with those of different compression methods that are known to obtain good results, including Gzip and Bzip2. BS-PPM uses order 4 PPMD to compress UTF-8 text files and is a recently published variant of PPM that achieves excellent results for natural language texts (Teahan and Alhawiti, 2015). For both GRB-PPM and GRT-PPM, we use the 100 most frequent bigraphs or trigraphs and order 4 PPMD for the encoding stage.

The Lempel-Ziv technique (LZ) depends on dictionary-based techniques. Gzip is one type of Lempel-Ziv coding; it uses LZ77 (Ziv and Lempel, 1977). Gzip is available to download at http://www.gzip.org. Bzip2 compression uses the Burrows-Wheeler compression algorithm and Huffman coding. Bzip2 is available for download at http://www.bzip.org/downloads.html.

TABLE 4.3: GRB-PPM and GRT-PPM compared with other compression methods for different natural language texts

File	Language	Size	Gzip (bpc)	Bzip2 (bpc)	PPMD4 (bpc)	BS-PPM (bpc)	GRT-PPM (bpc)	GRB-PPM
Brown	American English	5968707	3.05	2.33	2.14	2.11	2.03	**1.99**
LOB	British English	6085270	2.95	2.22	2.03	2.10	1.92	**1.88**
LCMC	Chinese	5379203	2.58	1.64	1.61	2.49	1.61	**1.59**
BACC	Arabic	69497469	2.14	1.41	1.68	1.32	1.29	**1.21**
Hamshahri	Persian	53471934	2.58	1.64	1.68	1.25	1.31	**1.22**
CEG	Welsh	6753317	2.91	1.95	1.69	2.30	1.57	**1.51**
Average			2.14	1.86	1.80	1.92	1.62	**1.56**

Table 4.3 compares the results of using GRB-PPM and GRT-PPM (using $N = 100$ (we use arbitrarily value) and order 4 PPMD) with different versions of well-known compression methods, such as Gzip, Bzip2 and BS-PPM order 4. (PPMD has become the de facto standard for comparing variants of the PPM algorithm and so is included with the results listed here.) We do these experiments with different text files in different languages, including American English, British English, Arabic, Chinese, Welsh and Persian.

It is clear that GRB-PPM achieves the best compression rate (shown in bold) in bpc for all cases in different languages. In addition, GRB-PPM is significantly better than several other compression methods. With Arabic text, for example, GRB-PPM shows a nearly 45% improvement over Gzip and an approximately 15% improvement over Bzip2. For Chinese, GRB-PPM shows a 36% improvement over BS-PPM and a 38% improvement over Gzip. For the Brown corpus, GRB-PPM shows a nearly 35% improvement over Gzip and an approximately 15% improvement over Bzip2. For the LOB corpus, GRB-PPM shows a 36% improvement over Gzip and a 15% improvement over Bzip2. For Welsh, GRB-PPM shows a 31% improvement over BS-PPM and a 48% improvement over Gzip. Finally, for Persian, GRB-PPM shows a nearly 50% improvement over Gzip and an approximately 22% improvement over Bzip2.

Figure 4.3 shows the compression rates for all the compression schemes in different languages, including English, Arabic, Chinese, Welsh and Persian. GRB-PPM clearly

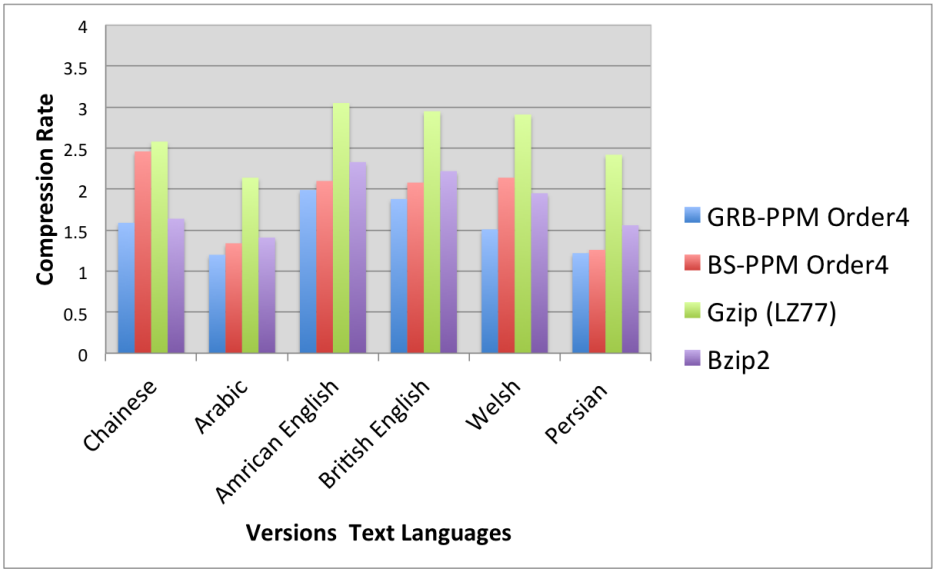


FIGURE 4.3: Compression rates for texts in different languages

achieves significant outcomes in multiple languages when compared with different compression schemes. As we can see in Figure 4.3, for American English, Gzip has the worst compression rate, with Chinese, British and Welsh text between 3 bpc and 2.5 bpc. For Bzip2, American English text and British English have better compression rates of between 2.3 bpc and 2.2 bpc. For Chinese and Welsh text, order 4 BS-PPM has poor compression rates.

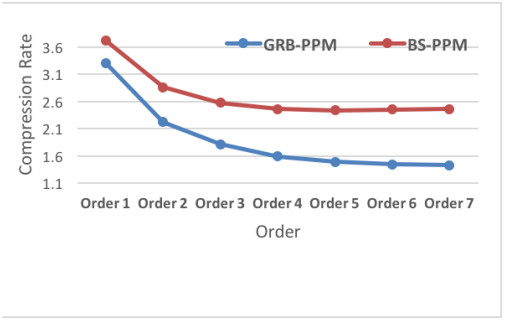
We use different PPM orders from 1 through 7 to inspect and compare GRB-PPM and BS-PPM. Table 4.4 shows the results for both GRB-PPM and BS-PPM; the best results are shown in bold for each language.

For Chinese and Arabic, GRB-PPM order 7 offers the best compression rate, better not only than other orders but also when compared with BS-PPM order 7. For American English and British English, GRB-PPM order 4 offers the best compression rate compared with other orders and with BS-PPM order 4. For Welsh, GRB-PPM order 5 offers the best compression rate compared with other orders and with BS-PPM order 5. For Persian, GRB-PPM order 6 offers the best compression rate compared with other orders and with BS-PPM order 6 (Figure 4.4). The experimental results show that in English language order 5 is best compared to other orders because there is less data available in the longer contexts.

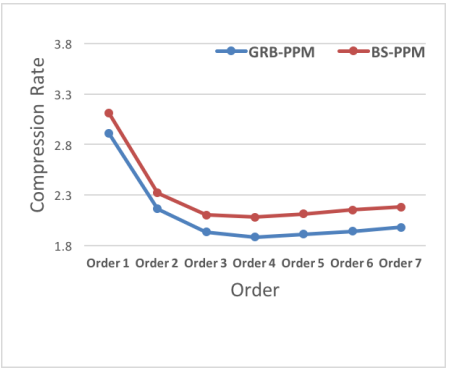
Table 4.5 shows the compression rates in bytes of both PPM and GRB-PPM for different Arabic file sizes from the BACC corpus. It is clear that the large files in the

(A) Brown Corpus

(B) LOB Corpus



(C) LCMC Corpus



(D) BACC Corpus

(E) CEG Corpus

(F) Hamshahri Corpus

FIGURE 4.4: Comparing GRB-PPM and BS-PPM for texts indifferent languages in different orders

TABLE 4.4: Compression results over six corpora and for various orders of BS-PPM and grammar bigraphs for prediction by partial matching (GRB-PPM)

File	BS-PPM1 (bpc)	GRB-PPM1 (bpc)	BS-PPM2 (bpc)	GRB-PPM2 (bpc)	BS-PPM3 (bpc)	GRB-PPM3 (bpc)	BS-PPM4 (bpc)	GRB-PPM4 (bpc)
LCMC	3.72	3.30	2.86	2.22	2.58	1.81	2.49	1.59
BACC	2.03	1.83	1.68	1.45	1.45	1.27	1.32	1.21
Brown	3.13	3.03	2.34	2.25	2.12	2.02	2.11	**1.99**
LOB	3.11	2.91	3.32	2.16	2.10	1.93	2.10	**1.88**
CEG	3.20	2.85	2.44	1.90	2.18	1.59	2.20	1.51
Hamsh	1.92	1.75	1.53	1.49	1.30	1.27	1.25	1.22

File	BS-PPM5 (bpc)	GRB-PPM5 (bpc)	BS-PPM6 (bpc)	GRB-PPM6 (bpc)	BS-PPM7 (bpc)	GRB-PPM7 (bpc)
LCMC	2.44	1.49	2.45	1.44	2.46	**1.43**
BACC	1.30	1.18	1.28	1.18	1.27	**1.17**
Brown	2.13	2.05	2.17	2.05	2.21	2.08
LOB	2.11	1.94	2.15	1.94	2.18	1.98
CEG	2.17	**1.50**	2.21	1.51	2.24	1.53
Hamsh	1.17	1.16	1.17	**1.14**	1.18	1.15

BACC, bookcollection, history and literature, achieve better compression rates in bpc terms using GRB-PPM as opposed to PPM. With small and medium file sizes in the BACC corpus, the compression rate with GRB-PPM is not as good it was with the large files. For GRT-PPM, small files showed the biggest improvements in compression rates, with the best results shown in bold in Table 4.6. However, in the large files, GRB-PPM achieved the best results in compression rates in terms of overall compressed files size. For medium file sizes, such as articles, press and novel1, the improvements were minimal compared to other files.

The character-based GRB-PPM model generates the best compression ratio amongst all methods tested for large text files, showing a significant improvement of 5 to 30% over PPM (Figure 4.5).

Table 4.7 shows the execution time for PPMD, GRB-PPM and GT-PPM under an order 4 model. We select the Brown, LOB, BACC and CEG text as test files. The configuration of our test machine is 4 GB GHz Intel Core i5, with 4GB internal memory.

The execution time of PPMD is nearly two times faster than GRB-PPM and GRT-PPM for the Brown corpus. For the LOB corpus, GRT-PPM is better than GRB-PPM, but

TABLE 4.5: PPM vs. GRB-PPM for the BACC corpus

File name	Size (bytes)	PPM	GRB-PPM (bpc)	Improvements (%)
economic	15924	2.03	**1.92**	5.4
sport	31706	1.95	**1.79**	8.2
culture	34760	2.03	**1.91**	5.9
artandmusic	42648	2.08	**1.96**	5.7
political	47556	1.97	**1.73**	12.1
articles	103839	1.94	**1.76**	9.2
press	549063	1.83	**1.57**	14.2
novel1	860680	1.87	**1.63**	12.8
novel2	912604	1.86	**1.63**	12.3
novel3	1023987	1.86	**1.60**	13.9
shortstories	1041952	1.82	**1.51**	17.0
literature	19187425	1.76	**1.34**	23.8
history	30714551	1.59	**1.10**	30.8
bookcollection	201693734	1.74	**1.24**	28.7

TABLE 4.6: GRB-PPM vs. GRT-PPM for the BACC corpus

File name	Size (bytes)	GRB-PPM	GRT-PPM	Improvements (%)
economic	15924	1.92	**1.91**	-0.5
sport	31706	1.79	**1.77**	-1.1
culture	34760	1.91	**1.89**	-1.0
artandmusic	42648	1.96	**1.95**	-0.5
political	47556	**1.73**	1.74	0.5
articles	103839	1.76	1.76	0
press	549063	1.57	1.57	0
novel1	860680	1.63	1.63	0
novel2	912604	1.63	1.63	0
novel3	1023987	1.60	1.60	0
shortstories	1041952	**1.51**	1.52	0.6
literature	19187425	**1.34**	1.35	0.7
history	30714551	**1.10**	1.13	2.7
bookcollection	201693734	**1.24**	1.28	3.2

PPMD is faster than both GRB-PPM and GRT-PPM. For the CEG corpus, PPMD is faster than GRT-PPM, and GRT-PPM is better than GRB-PPM. For Arabic, PPMD is slightly faster than GRB-PPM and GRT-PPM. In the four languages overall, it is clear that compression time is faster than decompression time.

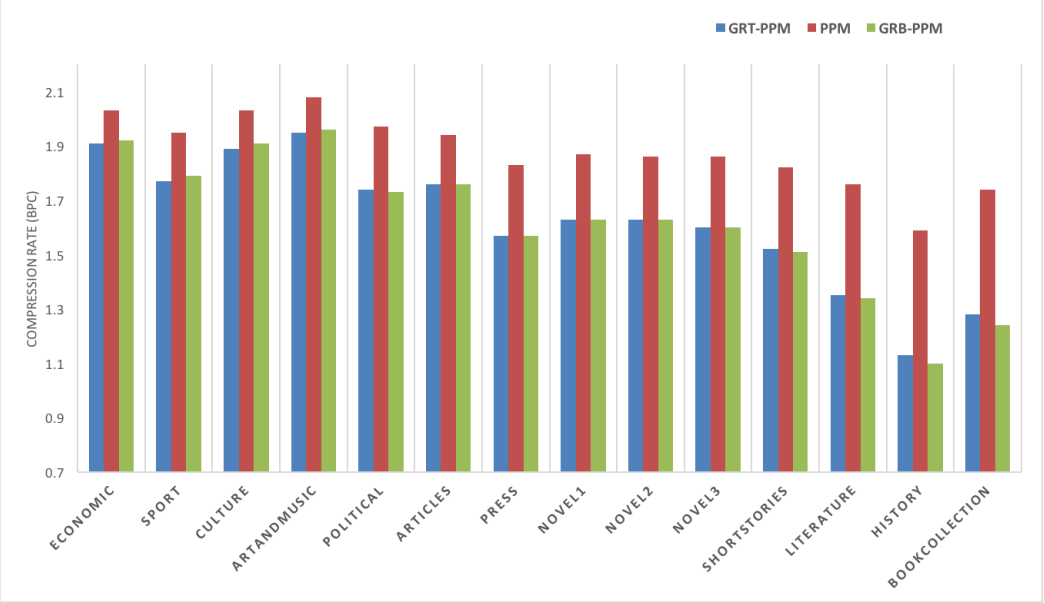


FIGURE 4.5: Comparing between various methods of PPM over the BACC Corpus

TABLE 4.7: Execution times for PPMD Order 4, GRB-PPM Order 4 and GRT-PPM Order 4

File	Size	PPMD Order 4		GRB-PPM Order 4		GRT-PPM Order 4	
		Encode Time	Decode Time	Encode Time	Decode Time	Encode Time	Decode Time
Brown	5968707	10.57	11.24	17.65	18.21	14.01	15.07
LOB	6085270	10.73	12.02	16.89	17.67	15.45	16.55
BACC	69497469	9.98	11.61	11.87	12.23	10.05	11.34
CEG	6753317	11.14	11.97	16.39	16.56	16.23	16.78

Table 4.8 illustrates the execution times when using GRB-PPM with orders 1 through 7. Compression times are comparable no matter which order is chosen.

From Table 4.8 and Table 4.9, it is clear that GRT-PPM is faster than GRB-PPM on average for various languages and different orders.

Figure 4.6 shows the executions time in GRB-PPM for different corpora using orders 1 through 7. Meanwhile, Figure 4.7 illustrates the executions time in GRT-PPM for different corpora. For LOB corpus and BACC, GRT-PPM is faster using order 1. These results are a reflection of the way the PPM escape probability estimates are calculated for order 1 contexts.

TABLE 4.8: Encoding execution times for GRB-PPM, Orders 1 through 7

Order	Brown	LOB	BACC	CEG
	Time (seconds)	Time (seconds)	Time (seconds)	Time (seconds)
Order 1	16.99	15.03	13.90	18.45
Order 2	16.56	15.12	12.35	17.97
Order 3	16.17	16.56	11.57	17.03
Order 4	17.65	16.89	11.87	16.39
Order 5	17.90	17.23	12.75	15.78
Order 6	18.32	17.56	12.90	15.71
Order 7	18.70	17.87	13.03	15.32

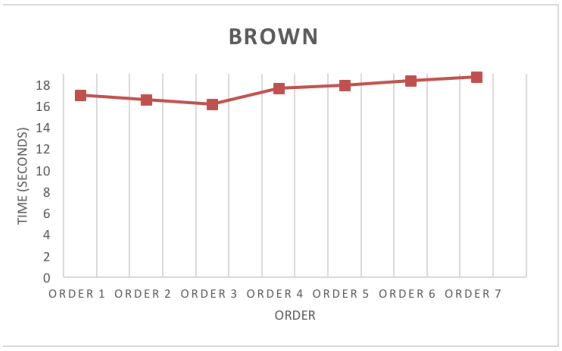
TABLE 4.9: Encoding execution times for GRT-PPM, Orders 1 through 7

Order	Brown	LOB	BACC	CEG
	Time (seconds)	Time (seconds)	Time (seconds)	Time (seconds)
Order 1	13.50	14.02	9.76	15.98
Order 2	13.78	14.78	9.85	15.40
Order 3	14.56	15.14	10.39	14.67
Order 4	14.01	15.45	10.05	14.45
Order 5	15.78	16.66	11.65	16.23
Order 6	15.98	16.12	11.90	17.70
Order 7	16.54	17.78	12.21	17.51

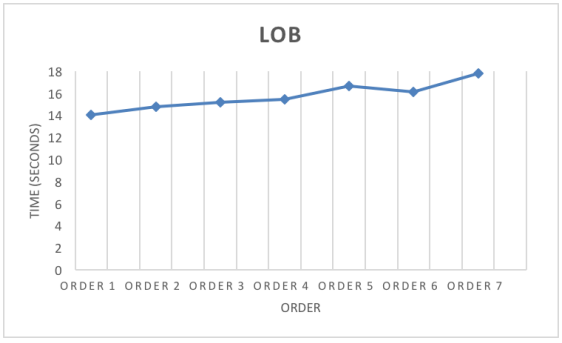
4.4 Summary and Discussion

A new approach has been introduced for improving compression of different natural language texts. The new approach, GR-PPM, is based on two fundamental approaches (CFGs and PPM) and is a general-purpose adaptive compression method for text files. The idea for GBR-PPM and GRT-PPM is to operate by substituting a repeated symbol (bigraph or trigraph) with a non-terminal symbol from a grammatical rule in a CFG before using PPM to compress the text files. This algorithm is maintained by two constraints. The first constraint is non-terminal uniqueness; each pair (sequence of terminals) can only appear once in the grammar. The second constraint is rule utility: each pair should be used more than once. Experimental results show that GRB-PPM achieves the best compression ratios for texts in different languages, such as English, Arabic, Chinese, Welsh and Persian. This technique also leads to significantly outperforms other compression schemes, such as Gzip, Bzip2 and BS-PPM order 4. Figure 4.4 shows that for Chinese and Arabic, GRB-PPM order 7 obtains the best compression

rate when compared with BS-PPM order 7. For American English and British English, GRB-PPM order 4 offers the best compression rate compared with other orders the compression rates become higher from order 5 to order 7. For Welsh, GRB-PPM order 5 offers the best compression rate compared with other orders which compression rates offer higher in orders 1 until 4 then orders 6 and 7. For Persian, GRB-PPM order 6 offers the best compression rate compared with other orders as the compression ratios offer higher orders. From the insights of the experimental results from the execution time of GRB-PPM in order 3 is faster than other orders for the Brown and BACC corpus. For the LOB corpus, GRB-PPM is faster in order 1. The experimental results also show that CEG is better in order 7. For GRT-PPM the execution time shows better results in order 1, for Brown and LOB corpus. For BACC and CEG corpus order 2 is better than other orders. In the four languages overall, it is clear that GRT-PPM time is faster than GRB-PPM. The experimental results also show that GRB-PPM is better than GRT-PPM in terms of compression rate, while GRT-PPM is better in execution time than GRB-PPM. These results are a reflection of the way the GRB-PPM and GRT-PPM escape probability estimates are calculated for the lower order contexts compared with the higher order contexts. Also, GRB-PPM and GRT-PPM require extra computation, since these models require three passes through the text determining the list of n-graphs that define the grammar in the first pass prior to the compression phase; correcting the text using the grammar in the second pass and then encoding it using PPM in the third pass.



(A) Brown Corpus

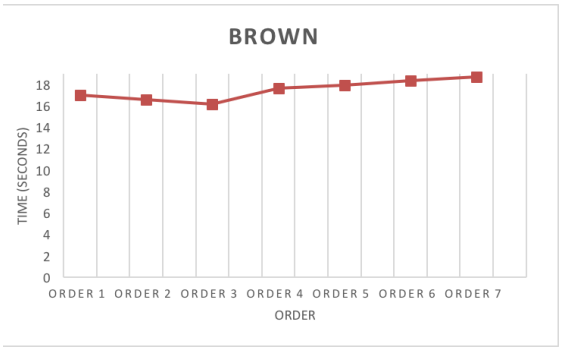


(B) LOB Corpus

(C) BACC Corpus

(D) CEG Corpus

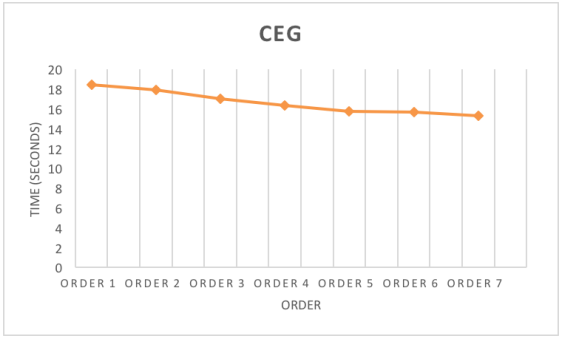
FIGURE 4.6: Compression times with GRB-PPM for texts in different languages



(A) Brown Corpus

(B) LOB Corpus

(C) BACC Corpus



(D) CEG Corpus

FIGURE 4.7: Compression Time in GRT-PPM for different language texts

CHAPTER 5

RECURSIVE GRAMMAR PRE-PROCESSING FOR PPM

Contents

5.1 Introduction

In the previous chapter, we discussed improvements to PPM text compression achieved by using various techniques. In this chapter, we investigate further improvements using a two-pass scheme in which grammar-based pre-processing is applied again in a second pass through the text. The two passes yield a significant enhancement in compression of natural language texts over known compression methods, as well as on the Calgary Corpus, a standard corpus used to compare text compression algorithms. The rest of the chapter is organised as follows. Our further improvements are discussed in section 5.2. We then discuss experimental results on natural language texts and the Calgary Corpus by comparing how well the new scheme performs compared to other well-known methods in section 5.3. A summary and discussion is presented in the final section.

Part of the work in this chapter has been published in a journal paper (W. Teahan and N. Aljehane, "GRAMMAR-BASED PRE-PROCESSING FOR PPM," International Journal of Computer Science & Information Technology (IJCSIT) Vol 9, No 1, February 2017).

5.2 Recursive grammar pre-processing for PPM

In this section, a new approach is applied in a second pass through the text based on CFG. This algorithm, which we call the second pass of *GR-PPM*, uses recursive CFGs and PPM.

In our approach, we basically use the N most frequent n-graphs in the second pass from previous files (e.g., the files generated by first pass of GR-PPM) to first generate a second grammar with one rule for each n-graph. We use these multiple passes to repeatedly substitute commonly occurring sequences of n-graphs and non-terminal symbols as specified by their rules in the grammar in a second pass through the file. This is done during the second pre-processing phase prior to the compression phase in a stage that allows the text to be regenerated during the post-processing stage. For bigraphs in the second pass, for instance, we call the variant of our method *GRBB-PPM (Two-Pass Grammar Bigraphs for PPM)*. Our new scheme shows good results not only when replacing the N most frequent symbols for bigraphs to bigraphs (for instance,

TABLE 5.1: An example of how GRBB-PPM works with a sample text

Pass	Grammar	String & Corrected Strings
		singing.do.wah.diddy.diddy.dum.diddy.do
1st	$A \rightarrow in,\ B \rightarrow do,\ C \rightarrow di,\ D \rightarrow dd$	sAgAg.B.wah.CDy.CDy.dum.CDy.B
2nd	$E \rightarrow Ag,\ F \rightarrow CD$	sEE.B.wah.Fy.Fy.dum.Fy.B

when N=100), but also using trigraphs to trigraphs (when n=3) in a variant which we have called *GRTT-PPM (Two-Pass Grammar Trigraphs for PPM)*.

As described in the previous chapter, the frequencies of common sequences of characters in reference corpora such as the Brown Corpus, the LOB Corpus and the BACC corpus can be used to define the n-graphs that will be substituted. However, although this method can be quite effective, what we have found to be most effective for our scheme is to determine the list of n-graphs that define the grammar in a double pass through the text being compressed prior to the compression phase, and then encoding the grammar separately, along with the corrected text, which is encoded using PPM.

Table 5.1 illustrates the GRBB-PPM process using a line taken from the song by Manfred Mann. The sequence is "*do wah diddy diddy dum diddy do*" (one published paper uses this song as a good example of repeating characters (Larsson and Moffat, 2000)). For GRB-PPM, there are four bigraphs (i.e. N=4) in the first pass through the text: *in, do, di and dd* . These bigraphs are replaced with new non-terminal symbols, say A, B, C and D, respectively. In *GRBB-PPM*, new rules are formed in a second pass through the corrected text that resulted from the first pass for the further bigraphs, *Ag, CD*. These will then be represented with non-terminals *E and F*, which are added to the expanded grammar. After all bigraph substitutions have been made, the message is reduced to the new sequence *sEE.B.wah.Fy.Fy.dum.Fy.B*.

Note that we ignore spaces and any punctuation characters because, based on our experiments, including these symbols decreases the compression rate. Moreover, the grammar will be transmitted to the receiver with the original text after all bigraphs are replaced in the original text with their non-terminal symbols. In the above example, the number of symbols is reduced from 32 in the original text to 22 in the first pass (GRB-PPM) and to 17 in the next pass (GRBB-PPM).

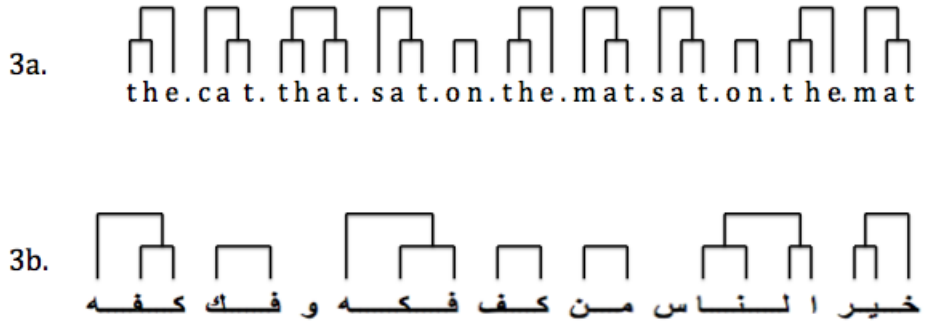


FIGURE 5.1: Hierarchical structure in the grammars generated by our algorithm for sample sequences in two languages: (a) English (b) Arabic

The grammars in both GRBB-PPM and GRTT-PPM share the same characteristics, which is that no pair of characters appears in the grammar more than once, a property called *non-terminal uniqueness*, using the terminology proposed by Nevill-Manning and Witten (1997). The second feature *rule utility*, is that every rule in the grammar is used more than once in the corrected text sequence. These two features are in the grammar that the first and second passes that GR-PPM generates.

In order to illustrate our method further, Figures 5.1a and 5.1b show the hierarchical grammatical structures that are generated by our approach for two different languages, English and Arabic. The hierarchical grammatical structures are formed based on the most frequent two characters or bigraphs in each text. For example, in Figure 5.1a, the word *the* is split into *th* and *e* (*th* is the bigraph in GRB-PPM, and the non-terminal that represents it and the letter *e* forms the second bigraph for the next pass for GRBB-PPM), and so on for other words in the texts. The same algorithm generates the Arabic version in Figure 5.1b, where the word كـفـه is split into كـف and هـ in a similar way (كـف is the bigraph and هـ is the second bigraph). We use fullstops for spaces to make them more visible, but spaces will not be considered in our algorithms and structures. On the other hand, in the Sequitur algorithm, spaces are part of the algorithm, which means they are also part of the grammatical structures generated by the algorithm.

5.3 Experimental Results

This section discusses experimental results for our method variants, GRBB-PPM and GRTT-PPM (described above), in compression of various text files when compared with

other well-known schemes.

We found that the GRBB-PPM and GRTT-PPM algorithms achieve the best compression ratios for texts in different languages, such as English, Arabic, Chinese, Welsh and Persian. We have also compared the results with those of different compression methods that are known to obtain good results, including PPMC and Sequitur. For both GRBB-PPM and GRTT-PPM, we use the 100 most frequent bigraphs or trigraphs for both passes and order 4 PPMD for the encoding stage.

TABLE 5.2: GRBB-PPM and GRTT-PPM compared with other compression methods using different natural language texts

File	Language	Size	Sequitur (bpc)	PPMD4 (bpc)	GRTT-PPM (bpc)	GRBB-PPM (bpc)
Brown	American English	5968707	2.55	2.11	2.00	**1.97**
LOB	British English	6085270	2.34	2.03	1.92	**1.88**
LCMC	Chinese	5379203	**1.45**	1.61	1.61	1.59
BACC	Arabic	69497469	1.47	1.68	1.29	**1.20**
Hamshahri	Persian	53471934	1.42	1.68	1.31	**1.22**
CEG	Welsh	6753317	2.04	1.69	1.54	**1.49**
Average			1.87	1.80	1.60	**1.55**

Table 5.2 shows the results of the different second pass variants of GRBB-PPM and GRTT-PPM. The results for the second pass variants GRBB-PPM and GRTT-PPM have been compared with PPMD order 4 and the results of the grammar-based compression algorithm Sequitur (since GRBB-PPM order 4 was found to be the best- see Table 5.5). It is clear that GRBB-PPM achieves the best compression rate for almost all cases in the different language texts, with only the single result on the Chinese text, the LCMC corpus, being better for the Sequitur algorithm. For Arabic, GRBB-PPM shows a 29% improvement over PPMD and a nearly 19% improvement over the Sequitur algorithm. For American English, Brown GRBB-PPM shows an 8% improvement over PPMD and a nearly 23% improvement over Sequitur. For British English, LOB GRBB-PPM shows a 20% improvement over the Sequitur algorithm. For Welsh, GRBB-PPM shows a 12% improvement over PPMD and a 27% improvement over the Sequitur algorithm.

TABLE 5.3: Performance of various compression schemes on the Calgary Corpus

File	Size (bytes)	PPMC (bbc)	Gzip (bpc)	Sequitur (bpc)	GRB-PPM (bpc)	GRBB-PPM (bpc)
bib	111261	2.12	2.53	2.48	1.87	**1.85**
book1	768771	2.52	3.26	2.82	**2.25**	**2.25**
book2	610856	2.28	2.70	2.46	**1.91**	**1.91**
news	377109	2.77	3.07	2.85	**2.32**	**2.32**
paper1	53161	2.48	2.79	2.89	2.34	**2.32**
paper2	82199	2.46	2.89	2.87	2.29	**2.26**
pic	513216	0.98	0.82	0.90	**0.81**	**0.81**
progc	39611	2.49	2.69	2.83	2.36	**2.33**
progl	71646	1.87	1.81	1.95	1.66	**1.61**
progp	49379	1.82	1.82	1.87	1.70	**1.64**
trans	93695	1.75	1.62	1.69	1.48	**1.45**
Average		2.14	2.29	2.32	1.91	**1.88**

For Persian, GRBB-PPM shows a 27% improvement over PPMD and a nearly 14% improvement over Sequitur.

Table 5.3 shows the compression rates for PPMC, Sequitur (Nevill-Manning and Witten, 1997), Gzip, GRB-PPM and GRBB-PPM on the Calgary corpus. Overall, the GRBB-PPM algorithm outperforms all the well-known compression methods. For the Sequitur algorithm, GRBB-PPM shows, on average, a nearly 19% improvement and an average 17%, 12% and 1% improvement over Gzip, PPMC and GRB-PPM, respectively. Although GRBB-PPM achieves similar results to GRB-PPM on the book1, book2, news and pic files, GRBB-PPM is better than GRB-PPM for the other files.

In order to investigate the effect of this semi-recursive grammar-based scheme further, other orders besides order 4 were also investigated. Table 5.4 shows that GRBB-PPM outperforms GRB-PPM and PPMD on the Calgary corpus in orders 1 and 2; the results show enormous differences between them. GRBB-PPM order 1 demonstrates, on average, a nearly 22% improvement over PPMD1. GRBB-PPM in order 1 achieves a good result, with an average 6.2% improvement rate over GRB-PPM. GRBB-PPM in order 2 achieves, on average, an approximately 20% improvement over PPMD2 and a 2.2% improvement over GRB-PPM order 2.

We use different GRB-PPM orders from 1 through 7 to compare and inspect GRBB-PPM. Table 5.5 shows the results for both GRB-PPM and GRBB-PPM; the best results

TABLE 5.4: Performance of GRBB-PPM, GRB-PPM and PPMD and in ordesr1 and 2 (bpc)

File	GRBB-PPM1 (bpc)	GRB-PPM1 (bpc)	PPMD1 (bpc)	GRBB-PPM2 (bpc)	GRB-PPM2 (bpc)	PPMD2 (bpc)
bib	**2.78**	2.94	3.45	**2.10**	2.16	2.63
book1	**2.81**	3.00	3.60	**2.26**	2.34	2.89
book2	**2.87**	3.08	3.77	**2.07**	2.15	2.88
news	**3.41**	3.65	4.10	**2.57**	2.65	3.24
paper1	**2.98**	3.18	3.81	**2.42**	2.45	2.90
paper2	**2.74**	2.94	3.61	**2.30**	2.35	2.85
progc	**3.13**	3.31	3.84	**2.52**	2.55	2.87
progl	**2.49**	2.67	3.31	**1.86**	1.92	2.33
progp	**2.38**	2.58	3.35	**1.83**	1.86	2.26
trans	**2.82**	3.00	3.48	**1.87**	1.94	2.35
avg.	**2.84**	3.03	3.63	**2.18**	2.23	2.72

TABLE 5.5: Compression results over six corpora and for various orders of GRB-PPM and GRBB-PPM

File	GRB PPM1 (bpc)	GRBB-PPM1 (bpc)	GRB-PPM2 (bpc)	GRBB-PPM2 (bpc)	GRB-PPM3 (bpc)	GRBB-PPM3 (bpc)	GRB-PPM4 (bpc)	GRBB-PPM4 (bpc)
LCMC	3.30	3.30	2.22	2.22	2.81	1.81	1.59	1.59
BACC	1.83	1.76	1.45	1.40	1.27	1.24	1.21	1.20
Brown	3.03	2.85	2.25	2.15	2.02	1.98	1.99	**1.97**
LOB	2.91	2.74	2.16	2.06	1.93	1.89	1.88	**1.88**
CEG	2.85	2.65	1.90	1.80	1.59	1.55	1.51	1.51
Hamsh	1.75	1.70	1.49	1.41	1.27	1.25	1.22	1.22

File	GRB-PPM5 (bpc)	GRBB-PPM5 (bpc)	GRB-PPM6 (bpc)	GRBB-PPM6 (bpc)	GRB-PPM7 (bpc)	GRBB-PPM7 (bpc)
LCMC	1.49	1.49	1.44	1.44	1.43	**1.43**
BACC	1.18	1.17	1.18	1.17	1.17	**1.16**
Brown	2.05	2.01	2.05	2.05	2.06	2.08
LOB	1.91	1.91	1.94	1.94	1.97	1.98
CEG	1.50	**1.49**	1.51	1.50	1.53	1.52
Hamsh	1.16	1.15	1.14	**1.13**	1.15	1.15

are shown in bold for each language.

An interesting comparison can be made between the GRB-PPM and GRBB-PPM models in the lowest order. GRBB-PPM from orders 1 through 4 significantly outperforms GRB-PPM in the same orders for American English and British English. For example, in the Brown corpus and the LOB corpus, GRBB-PPM order 1 shows a nearly 6% improvement over GRB-PPM. For Welsh, GRBB-PPM order 1 offers a better compression rate than GRB-PPM order 1, with a nearly 9% improvement over GRB-PPM. For Arabic, GRBB-PPM shows a nearly 4% improvement over GRB-PPM order 1. For Chinese text, it is clear that the recursive grammar does not change the results for different orders of GRBB-PPM and GRB-PPM.

TABLE 5.6: Compression results over six corpora and for various orders of GRT-PPM and GRTT-PPM).

File	GRTT PPM1 (bpc)	GRT-PPM1 (bpc)	GRTT-PPM2 (bpc)	GRT-PPM2 (bpc)	GRTT-PPM3 (bpc)	GRT-PPM3 (bpc)	GRTT-PPM4 (bpc)	GRT-PPM4 (bpc)
LCMC	3.30	3.30	2.24	2.24	2.83	1.83	1.61	1.61
BACC	1.83	1.90	1.52	1.32	1.30	1.34	1.29	1.29
Brown	3.09	3.25	2.41	2.57	2.09	2.17	**2.00**	2.03
LOB	2.97	3.12	2.31	2.46	1.99	2.07	**1.90**	1.92
CEG	2.85	3.03	2.05	2.20	1.67	1.75	1.54	1.57
Hamsh	1.81	1.92	1.42	1.35	1.32	1.33	1.31	1.31

File	GRTT-PPM5 (bpc)	GRT-PPM5 (bpc)	GRTT-PPM6 (bpc)	GRT-PPM6 (bpc)	GRTT-PPM7 (bpc)	GRT-PPM7 (bpc)
LCMC	1.51	1.51	1.45	1.45	**1.43**	**1.43**
BACC	1.24	1.25	1.23	1.23	**1.22**	1.23
Brown	2.02	2.02	2.05	2.05	2.08	2.09
LOB	1.91	1.91	1.94	1.95	1.98	1.99
CEG	**1.50**	1.52	1.51	1.52	1.53	1.54
Hamsh	1.26	1.27	**1.23**	1.24	1.25	1.25

Table 5.6 presents the results for both GRT-PPM and GRTT-PPM orders from 1 through 7. The best results are shown in bold for each language. Table 5.5 and Table 5.6 show that GRBB-PPM is better than GRTT-PPM in the lowest order for different languages .

Table 5.7 presents the compression rates for Sequitur and GRBB-PPM on the BACC corpus. It is clear that GRBB-PPM achieves the best compression rate in almost all cases in text files of different sizes, with more than a 21% imrpovement for all BACC corpus files. Table 5.8 shows that GRTT-PPM achieves better results for small files than

TABLE 5.7: Sequitur vs. GRBB-PPM for the BACC corpus

File name	Size (bytes)	Sequitur	GRBB-PPM	Improve-ments (%)
economic	15924	2.51	**2.00**	25.5
sport	31706	2.31	**1.83**	26.2
culture	34760	2.43	**1.95**	24.6
artandmusic	42648	2.52	**2.00**	26.0
political	47556	2.20	**1.76**	25.0
articles	103839	2.26	**1.79**	26.2
press	549063	2.04	**1.58**	29.1
novel1	860680	2.02	**1.64**	23.1
novel2	912604	2.10	**1.65**	27.2
novel3	1023987	2.02	**1.61**	25.4
shortstories	1041952	1.96	**1.52**	28.9
literature	19187425	1.66	**1.34**	23.8
bookcollection	201693734	1.50	**1.23**	21.9

TABLE 5.8: GRTT-PPM vs. GRBB-PPM for the BACC corpus

File name	Size (bytes)	GRTT-PPM	GRBB-PPM	Improve-ments (%)
economic	15924	**1.97**	2.00	- 1.5
sport	31706	**1.81**	1.83	- 1.0
culture	34760	**1.92**	1.95	- 1.5
artandmusic	42648	**1.97**	2.00	- 1.5
political	47556	1.76	1.76	0
articles	103839	**1.78**	1.79	- 0.5
press	549063	**1.57**	1.58	- 0.6
novel1	860680	1.64	1.64	0
novel2	912604	**1.63**	1.65	-1.2
novel3	1023987	1.61	1.61	0
shortstories	1041952	**1.51**	1.52	-0.6
literature	19187425	1.34	1.34	0
history	30714551	1.11	**1.09**	1.8
bookcollection	201693734	1.26	**1.23**	2.4

large files. For example, with small files such as economic, sport, culture and articles, GRTT-PPM achieves more than a 1% improvement over GRBB-PPM. However, for large files, GRBB-PPM is between, with between 1.70% and 2.50% improvements in compression rates. For medium file sizes, the improvements are minimal compared to the overall compressed file size.

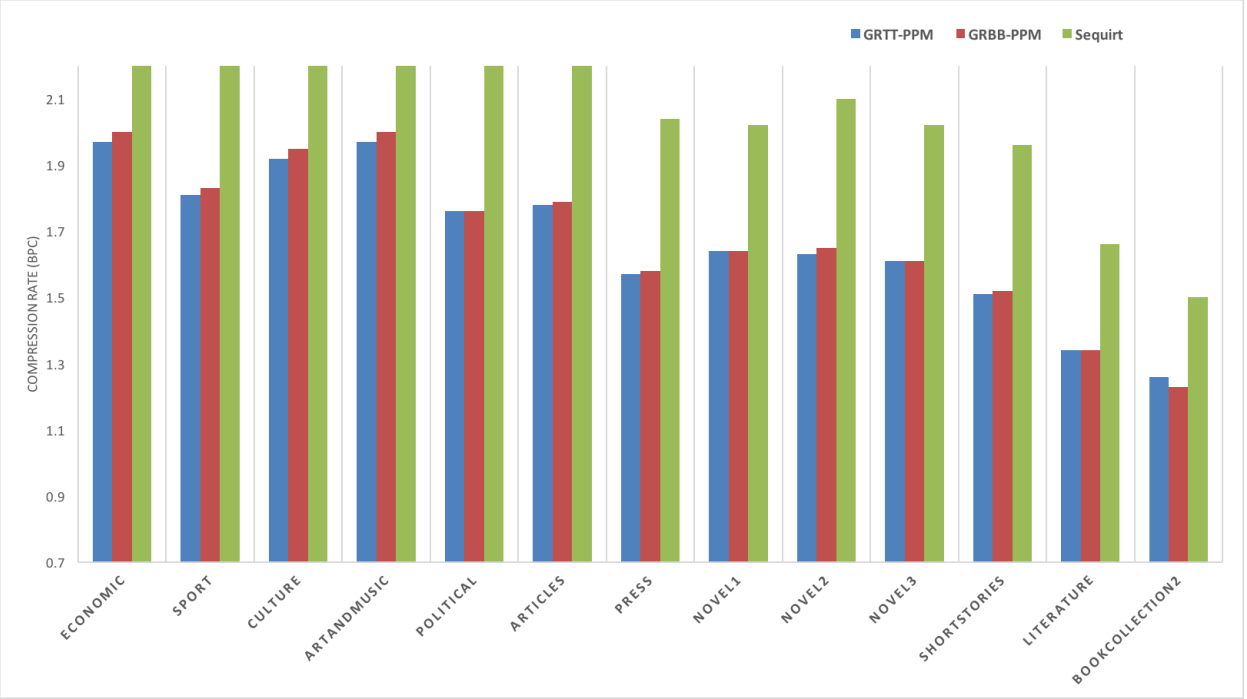


FIGURE 5.2: Comparing various well-known compression schemes using the BACC Corpus

It is clear from figure 5.2 that the Sequitur method has the worst compression rates for different sizes using the BACC corpus.

TABLE 5.9: Encoding execution times for GRBB-PPM using orders 1 through 7

Order	Brown	LOB	BACC	CEG
	Time (seconds)	Time (seconds)	Time (seconds)	Time (seconds)
Order 1	18.32	19.11	12.48	21.48
Order 2	18.67	19.44	12.66	21.73
Order 3	19.05	19.76	14.05	21.82
Order 4	19.12	20.02	14.63	22.71
Order 5	19.45	20.55	14.76	23.47
Order 6	20.03	21.12	15.09	24.37
Order 7	20.87	21.08	15.34	24.45

Table 5.9 and Table 5.10 illustrate the execution times when using GRBB-PPM and GRTT-PPM at orders 1 through 7. The configuration of our test machine was 4 GB GHz intel Core i5, with 4GB internal memory. Execution times are comparable between the various files, no matter which order is chosen. It is also clear that GRTT-PPM is faster than GRBB-PPM for multiple languages and different orders.

TABLE 5.10: Encoding execution times for GRTT-PPM using orders 1 through 7

Order	Brown	LOB	BACC	CEG
	Time (seconds)	Time (seconds)	Time (seconds)	Time (seconds)
Order 1	17.11	18.2 1	10.42	17.68
Order 2	17.23	18.43	11.12	18.15
Order 3	18.08	19.10	11.20	18.64
Order 4	18.46	19.12	11.54	19.34
Order 5	18.94	19.76	12.97	21.13
Order 6	19.27	20.05	13.34	21.56
Order 7	19.24	20.68	13.39	22.25

As shown in Figures 5.3 and 5.4, the results show on average that GRBB-PPM and GRTT-PPM at order 1 in the Brown, LOB, BACC and CEG corpora are faster than with other orders (for the same reasons that were stated above concerning how PPM order 1 calculations are made), with order 7 the slowest for these corpora. For order 2 and higher, the time increases with the increasing number of calculations as the contexts get longer.

5.4 Summary and Discussion

We have presented further improvements of the GR-PPM compression algorithm designed for different natural language texts. This method uses a two-pass scheme instead of a one-pass scheme. To our knowledge, this approach is the first recursive grammar pre-processing scheme for PPM. No other study has successfully used a two-pass scheme to process text. This approach uses two constraints: non-terminal uniqueness and rule utility, as discussed in the previous chapter. Experimental results show that the GRBB-PPM and GRTT-PPM schemes yield a significant enhancement in compression of natural language texts over known compression methods, just as they do on the Calgary Corpus, a standard corpus used to compare text compression algorithms.

From the insights of the experimental results from the execution time of GRBB-PPM in order 1 is faster than other orders for the Brown and LOB corpus. For the BACC and CEG corpus, GRBB-PPM is faster in order 2. The experimental results also show that languages overall, GRTT-PPM are better in order 1. In the four languages overall, it is clear that GRTT-PPM time is faster than GRBB-PPM as a few of numbers of symbols have to be encoded compared with the numbers of symbols in GRBB-PPM. Moreover,

the experimental results show that GRBB-PPM is better than GRTT-PPM in terms of compression rate.

These results are a reflection of the way the GRBB-PPM and GRTT-PPM (in the recursive grammar) escape probability estimates are calculated for the lower order contexts. For the higher order contexts, the time increases with the increasing number of calculations as the contexts get longer. Moreover, GRBB-PPM and GRTT-PPM require extra computation, since these models use multiple passes to repeatedly substitute commonly occurring sequences of n-graphs and non-terminal symbols as specified by their rules in the grammar in a second pass through the file. This is done during the second pre-processing phase prior to the compression phase

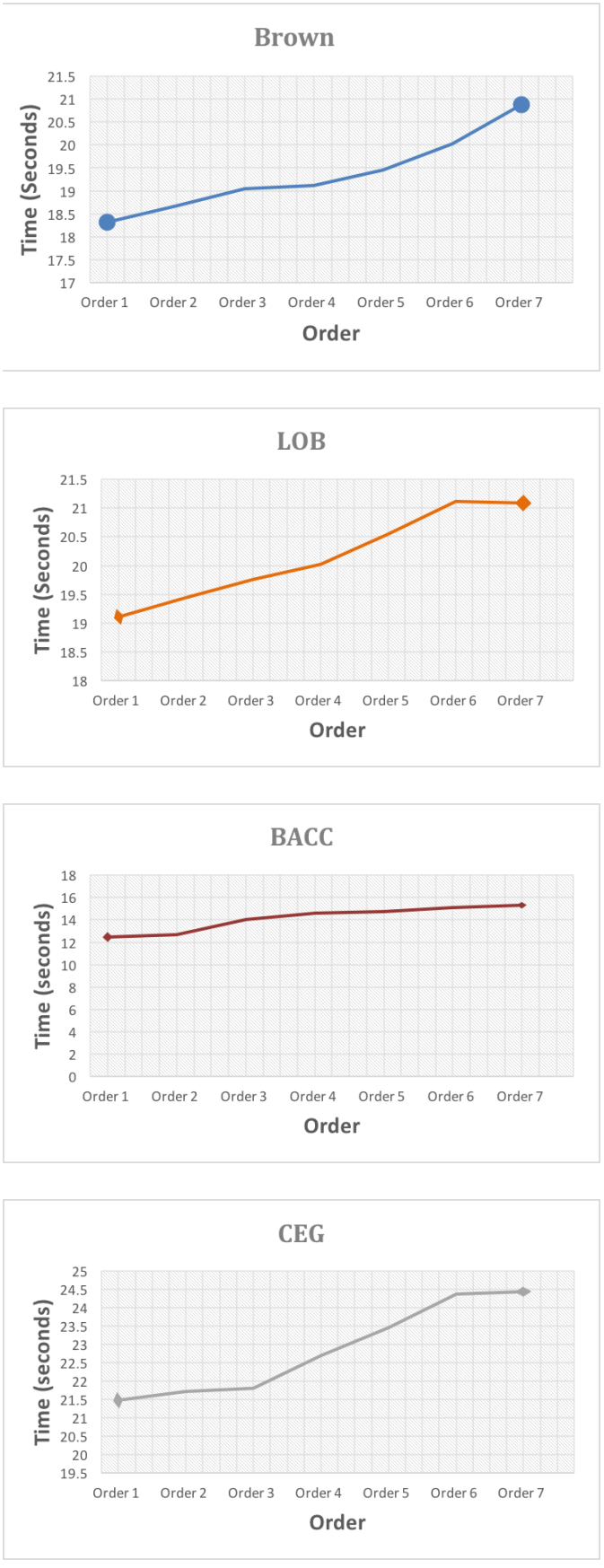


FIGURE 5.3: Comparing execution times for GRBB-PPM from orders 1 through 7 for different corpora

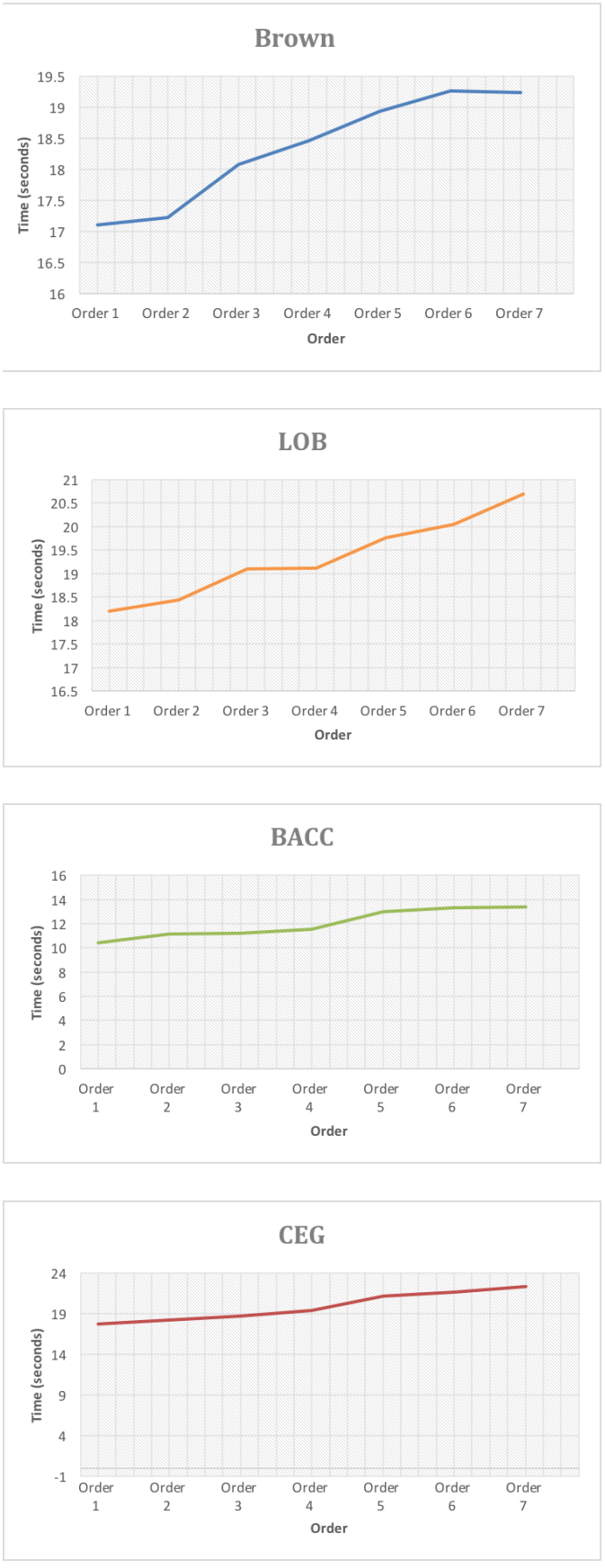


FIGURE 5.4: Comparing execution times for GRTT-PPM using orders 1 through 7 with different corpora

CHAPTER 6

GRAMMAR WORD-BASED PPM

Contents

6.1 Introduction

In the previous chapter, we discussed new algorithms to provide better compression and address the problem of encoding contexts for different language including Arabic, English, Welsh, Persian and Chinese. These algorithms, which depend on encoding a CFG, use a character-based method as the main unit of encoding.

Prediction in PPM depends on a the bounded number of previous characters or symbols, effectively using a Markov-based approach. In PPM, to predict the next character or symbol, different orders of models are used, starting from the highest order down to the lowest orders. An escape probability estimates if a new symbol appears in the context (Cleary and Witten, 1984; Moffat, 1990) and if an escape is encoded, the algorithm will back off to a lower order model. Despite the cost in terms of memory and the speed of execution, PPM usually attains better compression rates than other well-known compression methods. The "full exclusions" mechanism (Cleary and Witten, 1984) is used to significantly improve compression by excluding the prediction of higher order symbols when an escape has occurred, since these characters were not encoded (Bell et al., 1990). Experimental results show that not using full exclusions speeds up the execution time of programs, although compression ratios are reduced.

However, when a PPM approach is applied to words rather than characters, it is not clear what method is most effective for encoding text because there are issues of how to encode the spaces and punctuation along with the text, how to deal with capitalised words, whether to treat digit sequences differently, how to deal with the much larger alphabet when using full exclusions and so on. This is compounded when considering certain languages, such as Arabic, which have a rich morphological structure that may present further types of difficulties for word-based compression not found in languages such as English.

As an illustration, the lists below in Table 6.1 show the most common words in each of the examined texts. They are based on an analysis of the Brown Corpus for American English, the LOB Corpus for British English, the BACC and CCA Corpora for Arabic text, the Hamshahri corpus for Persian text and the CEG corpus for Welsh text.

Substitution of these words using our CFG scheme and standard PPM can significantly improve overall compression, as shown below. For example, natural languages contain common sequences of words that often recur in the same order, such as in English "the"

TABLE 6.1: The 20 most common words in the Brown, LOB, BACC, CCA and Hamshahri text corpora

Rank	Brown Corpus	LOB Corpus	BACC Corpus	CCA Corpus	Hamshahri
1	the	the	في	في	برورش
2	of	of	من	من	دستور
3	and	and	على	على	سمتی
4	to	a	و	ان	اين
5	a	in	ما	إلى	حميد
6	in	that	ان	التي	در
7	that	is	إلى	عن	اعلام
8	is	was	عن	ما	بنيادی
9	was	for	لا	لا	اظهارات
10	for	it	الذي	هذا	صف
11	with	to	على	هذه	كند
12	The	be	إلى	الذي	افزود
13	as	his	أو	أو	برای
14	he	as	بين	و	بر
15	it	on	مع	كان	آول
16	his	The	كان	مع	كردند
17	on	his	له	لم	كجا
18	be	at	ثم	كل	كه
19	from	as	لك	ذلك	كذشته
20	had	had	هذا	بين	اداره

,“of” and “and”, and for the Arabic language in the BACC corpus, such as “من”, “في” and so on. Table 6.1 shows that the most common word for both American and British English is ”the”. However, for these corpora if one treats capitalised words as being distinct (that is, “the” is treated as distinct from “The”), we find that the word “The” also appears in the top 20 ranked words, but at different ranks (12 for the Brown Corpus versus 16 for the LOB Corpus). In contrast, the word “had” appears with the same rank for both corpora. Certain words such as “from” and “at” appear in the list for one corpus but not for the other.

For Arabic text, the most common word for both the BACC and ACC Corpora is “في” (in). Nevertheless, we find that the word “ان” (that) also appears in the top 20 words,

but at different ranks (4 for CCA versus 6 for BACC). In contrast, the word "من" (from) appears with the same rank in both corpora. Certain words such as "التي" (which) and "له" (for him) appear in the list for one corpus but not for the other. For Persian text in the Hamshahri Corpus, even though it uses Arabic script, the top 20 ranked words are noticeably different due to the differences between the two languages.

From these lists, it is clear even from examining only the 20 most common words that there are important differences, so word-based compression schemes have to adapt directly to the text being compressed in an on-line manner (as PPM does) rather than using dictionaries created from general sources. Another factor is that since the most frequent words represent a significant proportion of the text, adaptive word-based schemes can often lead to improved compression for many languages. An added advantage of such schemes is that many fewer symbols need to be encoded (for example, in English, there are on average approximately five times fewer word symbols than there are character symbols). However, finding the most effective word-based compression remains an open problem with word-based schemes being under-researched compared to character-based schemes. The comparison between the effectiveness of word-based schemes with character-based and parts-of-speech (tag) schemes also provides an interesting tool for performing further linguistic analysis (Teahan, 1998). The main contribution of the work described in this chapter is an improved word-based compression method for PPM, which is achieved by parsing the text to construct a word-based CFG that is then compressed using PPM.

The rest of the chapter is organized as follows. Previous work is discussed first, and our new approach is discussed in the next section. We discuss experimental results for natural language texts in order to evaluate how well the new scheme performs compared to other well-known methods. A summary and discussion is presented in the final section.

Purpose of Chapter and state as contributed to a journal published (N. Aljehane and W. Teahan , "Word-Based Grammars for PPM," the International Journal of Advanced Computer Science and Applications (IJACSA), Vol 8, No 10, Oct 2017).

6.2 Previous Work

As stated in earlier chapters, standard PPM word-based models predict the forthcoming symbol, starting from the highest order context; however, when the upcoming symbol

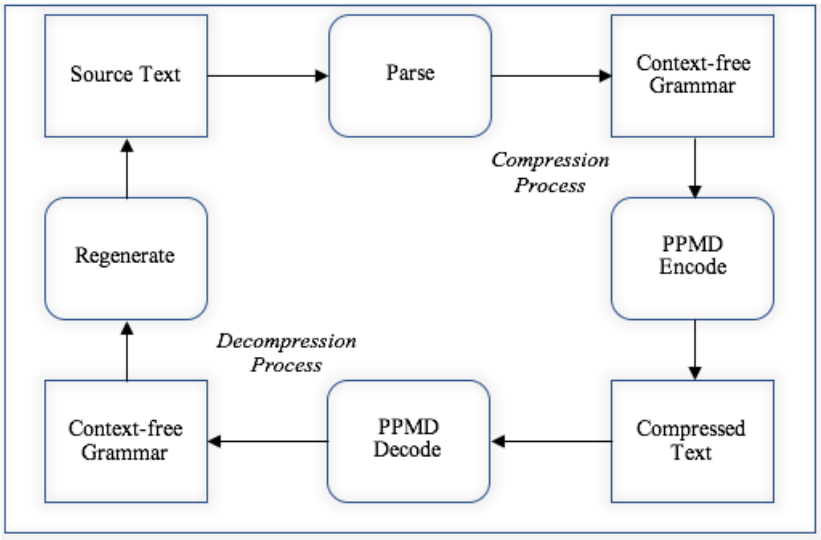


FIGURE 6.1: The complete compression and decompression process of GRW-PPM

has not appeared in this context, then a lower context is used and an escape symbol is encoded. A number of methods have been used to estimate the probability for these escape symbols (Witten and Bell, 1991; Teahan, 1998).

Experiments indicate that the $X1$ method has the best performance for English text in most cases (Teahan, 1998). This method is given by the formula:

$$e = \frac{t_1 + 1}{T_d + t_1 + 1}. \tag{6.1}$$

Here, t_1 denotes the number of symbols seen only once previously in the context, and T_d is the frequency with which the symbol occurs in the context. This method estimates the escape symbol probability proportionate to the number of words that have appeared only once in the text.

Experiments using English show that the word-based models in Table 6.2 provide better performance than other models (Teahan, 1998).

Model $C|C^5$, is a PPM character model of order five that predicts the probability of character symbols and is used as a compression baseline. In this model, the formula for the probability of text string S of m characters is given by:

$$P(S) = \prod_{i=1}^{m} p(c_i|c_{i-1}c_{i-2}c_{i-3}c_{i-4}c_{i-5}), \tag{6.2}$$

TABLE 6.2: Some models for predicting characters and words (Teahan, 1998)

$C\|C^5$ Model	$W\|W$ Model
$p(c_i\|c_{i-1}c_{i-2}c_{i-3}c_{i-4}c_{i-5})$	$p(w_i\|w_{i-1})$
$\hookrightarrow p(c_i\|c_{i-1}c_{i-2}c_{i-3}c_{i-4})$	$\hookrightarrow p(w_i)$
$\hookrightarrow p(c_i\|c_{i-1}c_{i-2}c_{i-3})$	$\hookrightarrow$ *Character model*
$\hookrightarrow p(c_i\|c_{i-1}c_{i-2})$	
$\hookrightarrow p(c_i\|c_{i-1})$	
$\hookrightarrow p(c_i)$	
$\hookrightarrow p_e q(c_i)$	

where the preceding five characters in the text are used to estimate the probability of the forthcoming symbol. This estimate of the probability for the previous formula depends on the escape method (in Table 6.2; the symbol $\hookrightarrow$ denotes an escape). In character-based models, if the highest order fails to predict forthcoming symbol, the probability of escape is encoded to move down to the next highest order.

The second model $W|W$ is a PPM order 1 word-based (i.e., bigram) model that predicts the probability of word symbols. In this model, the estimation of probability depends on the previous word in the text and predicts the probability for the forthcoming word, as represented by the following formula for the probability of text string S of m words:

$$P(S) = \prod_{i=1}^{m} p(w_i|w_{i-1}), \tag{6.3}$$

where p denotes the probability of the symbols in the sequence of the text S based on words. If the word is not predicted by this model, then an escape is encoded down to the order 0 model. If the word still has not been seen in this context, then a further escape is encoded, followed by each character in the word being encoded separately using the standard PPM character-based model.

6.3 Word-based grammars for PPM (GRW-PPM)

A new approach based on using word-based CFGs for compressing text files is presented here. This algorithm, which we call GRW-PPM (short for Grammar word-based pre-processing for PPM), uses both a CFG and PPM as the basis of a universal general-purpose compression method for text files.

In our approach, we essentially parse words, digits, spaces and punctuation in the source file to first generate a grammar with rules and terminal and non-terminal symbols representing each of these text elements. We then substitute every time one of these text elements occurs in the source text with the single unique non-terminal symbol specified by its rule in the grammar. This is done during the pre-processing phase prior to the PPM compression phase, which is applied to the sequences of non-terminal symbols for words, digits and spaces and punctuation separately.

Our method replaces sequences of words (n-grams) in the text as they are processed from beginning to end in a single pre-processing pass. The PPM algorithm is used as the encoder once the sequences have been replaced. Unlike PPM, our method is off-line during the phase which generates the grammar.

Our approach adapts the $W|W$ word-based method and the character n-graph replacement pre-processing approach of Teahan (1998) by using an off-line technique to generate the list of word n-grams first from the source file being compressed. However, our approach is considered within a grammar-based context instead. The main difference from prior word-based schemes (such as $W|W$) is the use of PPM to encode the sequence of word symbols directly without the need to escape to a separate character-level encoding and the treatment of digits as word symbols (see below).

The grammar in GRW-PPM shares a key characteristic as Sequitur by Nevill-Manning and Witten (1997) and GR-PPM (discussed in chapter 4) in that no pair of symbols appears in the grammar more than once. This property ensures that every n-gram in the grammar is unique, a property which as stated is called non-terminal uniqueness in the terminology proposed by Neville-Manning and Witten. To make sure that each rule in the grammar is useful, the second property, referred to as rule utility, is that every rule in the grammar is used more than once in the corrected text sequence.

Figure 6.1 shows the whole process of GRW-PPM. First, the original text will be parsed and word, digit and space/punctuation tokens will be extracted; then, the CFG will be generated by replacing them in the text wherever they occur with the non-terminal symbols defined by their rules in the grammar. After the rules have been produced, the grammar is encoded using PPMD, and the resulting compressed text is then sent to the receiver. The receiver decodes the grammar by using PPMD to decompress the compressed file that was sent. The reverse mapping is then facilitated by using the decoded grammar to regenerate the original source text.

TABLE 6.3: An example of the grammar generated by GRW-PPM for a sample English text

Sequence:

The.song.“Do.Wah.Diddy.Diddy.Dum.Diddy.Do”.was.recorded.on.11.June.1964.and .released.on.10.July.

Grammar:

$S \rightarrow S_1\ S_2\ S_3\ S_4\ S_5\ S_5\ S_6\ S_5\ S_3\ S_7\ S_8\ S_9\ S_D\ S_{10}\ S_D\ S_{11}\ S_{12}\ S_9\ S_D\ S_{13}$

$V \rightarrow S_1\ S_2\ S_3\ S_4\ S_5\ S_6\ S_7\ S_8\ S_9\ S_{10}\ S_{11}\ S_{12}\ S_{13}$

$D \rightarrow D_1\ D_2\ D_3$

$P \rightarrow P_1\ P_2\ P_1\ P_1\ P_1\ P_1\ P_1\ P_1\ P_3\ P_1\ P_1\ P_1\ P_1\ P_1\ P_1\ P_1\ P_1\ P_1\ P_1\ P_4$

$S_1 \rightarrow$ “The”

$S_2 \rightarrow$ “song”

$S_3 \rightarrow$“Do”

$S_4 \rightarrow$ “Wah”

$S_5 \rightarrow$ “Diddy”

$S_6 \rightarrow$“Dum”

$S_7 \rightarrow$ “was”

$S_8 \rightarrow$ “recorded”

$S_9 \rightarrow$ “on”

$S_{10} \rightarrow$ “June”

$S_{11} \rightarrow$ “and”

$S_{12} \rightarrow$ “released”

$S_{13} \rightarrow$ “July”

$D_1 \rightarrow$ “11”

$D_2 \rightarrow$ “1964”

$D_3 \rightarrow$ “10”

$P_1 \rightarrow$ “.”

$P_2 \rightarrow$ “ .“ ”

$P_3 \rightarrow$ “ .” ”

$P_4 \rightarrow$ “.”

Table 6.3 illustrates the process of GRW-PPM using the following phrase from the song by Manfred Mann: *‘The.song.“Do.Wah.Diddy.Diddy.Dum.Diddy.Do”.was.recorded.on. 11.June.1964.and.released.on.10.July.’*. First, the original text will be parsed from left to right and new non-terminal word and digit symbols ($S_1\ S_2\ S_3\ S_4\ S_5\ S_5\ S_6\ S_5$ .... $S_{12}\ S_9\ S_D\ S_{13}$) will be substituted for each unique word (defined as being separated by the intervening space and punctuation symbols). For this example (and for the experiments described below), we use single words (unigrams), although the method works in a similar way for word bigrams and trigrams. Referring to Table 6.3, we replace the unigram “The” with non-terminal symbol S_1, unigram “song” with non-terminal symbol S_2, unigram “Do” with non-terminal symbol S_3 and so on. We use bullet points for spaces in this chapter to make them more visible. Spaces (white space) and punctuation define the word boundaries (i.e., each word is made up of sequences of anything that is not white space or punctuation).

Table 6.4 shows the same process for a sample Arabic text (figure 6.2) which translates into English as follows: “The number of shares traded in the “Saudi” market were more than 277 thousand shares, and the number of transactions were more than 132 thousand transactions.” However, in this case the unigrams are generated from right to left. Each

TABLE 6.4: An example of how GRW-PPM works for a sample Arabic text (using the same example as for Table 6.3)

Sequence:

و بلغ.عدد.الأسهم.في.السوق.”السعودي“.أكثر.من.277.ألف.,سهم.و بلغ عدد.الصفقات .أكثر.من.132.ألف.صفقة.

Grammar:

$S \rightarrow S_1\ S_2\ S_3\ S_4\ S_5\ S_6\ S_7\ S_3\ S_7\ S_8\ S_D\ S_9\ S_{10}\ S_1\ S_2\ S_{11}\ S_7\ S_8\ S_D\ S_9\ S_{12}$

$V \rightarrow S_1\ S_2\ S_3\ S_4\ S_5\ S_6\ S_7\ S_8\ S_9\ S_{10}\ S_{11}\ S_{12}$

$D \rightarrow D_1\ D_2$

$P \rightarrow P_1\ P_1\ P_1\ P_1\ P_2\ P_3\ P_1\ P_1\ P_1\ P_1\ P_1\ P_1\ P_1\ P_1\ P_1\ P_1\ P_1\ P_4$

$S_1 \rightarrow$ “و بلغ”

$S_2 \rightarrow$ “عدد”

$S_3 \rightarrow$ “الأسهم”

$S_4 \rightarrow$ “في”

$S_5 \rightarrow$ “السوق”

$S_6 \rightarrow$ “السعودي”

$S_7 \rightarrow$ “أكثر”

$S_8 \rightarrow$ “من”

$S_9 \rightarrow$ “ألف”

$S_{10} \rightarrow$ “سهم”

$S_{11} \rightarrow$ “الصفقات”

$S_{12} \rightarrow$ “صفقة”

$D_1 \rightarrow$ “277”

$D_2 \rightarrow$ “132”

$P_1 \rightarrow$ “.”

$P_2 \rightarrow$ “ .“ ”

$P_3 \rightarrow$ “ .” ”

$P_4 \rightarrow$ “.”

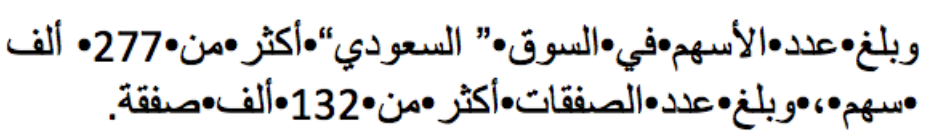


FIGURE 6.2: Example of Arabic Text

unique Arabic unigram has a non-terminal symbol associated with it. For instance, the words “ و بلغ ” ,“ عدد ” and “ الأسهم ” are replaced by non-terminal symbols S_1 to S_3, respectively.

In the grammar examples, the S rule is used to represent the word and digit symbols sequence. Separate rules (S_1, S_2, S_3 ...) are used, one for each word, to specify each symbol's content directly using a non-terminal (denoted by characters surrounded by “'s”). The V rule enumerates each of these words in order; it is used to represent the vocabulary (the sequence of unique words as they occur in the text). Each digit sequence

is encoded within the S sequence by using a special symbol to indicate the positions of the digits in the sequence (as represented by S_D in the above examples). The actual content of each digit symbol is specified by the D rule and encoded separately to the word and digit symbols. We also process spaces and any punctuation characters in order to be able to fully decode the original text. These are represented by the P rules for the grammars in the above examples and are similarly encoded separately to the word and digit unigram symbols. Moreover, the grammar will be transmitted to the receiver once it has been constructed after all unigrams are substituted in the original text with their non-terminal symbols.

The grammar represents a complete description of the text, so it is possible to devise a lossless text compression scheme by directly encoding it in some manner, as long as it is possible for the decoder to be able to regenerate the complete source text losslessly once the grammar has been decoded.

As stated previously, we have found one effective means for encoding the grammar is to use PPM. Specifically, the grammar is encoded by using PPMD to separately encode the four main elements (words, vocabulary, digits and spaces/punctuation as represented by the S, V, D and P rules). For Rule S, we can encode the sequence of symbol numbers or letters that appear in the rule. For example, in Table 6.3, the sequence of symbol numbers/letters for Rule S is as follows: 1 2 3 4 5 5 6 5 3 7 8 9 D 10 D 11 12 9 D 13. This represents the sequence of id numbers assigned to each unique word with id numbers starting from 1 and incrementing by one whenever a new word is encountered. The letter D indicates that a digit sequence has occurred. Clearly, the sequence for rule S will be highly repetitive for long sequences of natural language text because of the presence of repeated words and frequent function words (such as “the” and “and” for English and “من ”and “في ” for Arabic, as shown in Table 6.1). More specifically, we have found PPMD to be very effective at encoding this sequence. However, unlike $W|W$ (which uses similar PPM-like methods to encode word symbols in this manner), our method simply uses PPMD with a fixed maximum alphabet size (since this is known when the grammar has been fully constructed for the whole text). In addition, our method does not need to encode an escape down to the individual character-level, as $W|W$ does, in order to encode novel words when they occur. Instead, it uses the standard PPMD encoding mechanism (where a novel symbol will be encoded using a default order -1 model with all symbols equiprobable).

For practical purposes, rule V and rules S_1, S_2, S_3, ... can simply be represented as a string of text that contains all the unique words as they appear in the source text, one after another, with a separator (such as a space character) used to indicate the end of the previous word and the beginning of the next one. Similarly, we can use the same encoding technique for the digit sequences for rule D and rules D_1, D_2, D_3,... and for the spaces and punctuation for rule P and rules P_1,P_2,P_3,... That is, both the digits and punctuation can be encoded effectively by using PPMD to encode one text string that contains all the unique digit sequences and another text string that contains the unique space and punctuation sequences, respectively. A space character can be used as a separator for the digits, but for the punctuation, a different separator is needed. We use the letter "W" as the separator to mark where words are located.

As an illustration, Table 6.5 presents the symbols or text that are being encoded for the four elements (symbols, vocabulary, digits, spaces and punctuation) for the beginning of the Brown corpus. All are encoded directly by PPMD as text except for the symbols element, which is treated as a sequence of numbers instead.

TABLE 6.5: What the different text elements look like for the beginning of the Brown Corpus

Brown Corpus (text at the start of the corpus):
The Fulton County Grand Jury said Friday an investigation
of Atlanta's recent primary election produced "no evidence" that
any irregularities took place. The jury further said in term-end
presentments that the City Executive Committee, which had over-all
...

Symbols	Vocabulary	Digits	Spaces & Punctuation
2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 2 25 26 72 7 28 29 30 20 31 32 33 34 35 36 37 38 39 11 31 16 40 3141 42 43 11 31 32 11 12 44 31 45 27 35 31 16 46 47 2 48 49 28 25 36 50 51 52 3 53 54 55 56 57 58 59 60 11 61 ...	The Fulton County Grand Jury said Friday an investigation of Atlanta s recent primary election produced no evidence that any irregularities took place jury further in term end presentments the City Executive Committee which had over all charge ...	1 1 2 2 1913 71 74 637 1937 1923 1 13 1962 8 1961 100 30 3 4 1958 50 10 87 31 29 5 13 1 119 402 18 17 63 31 300 000 6 13 451 500 157 460 88 000 182 17 000 1 000 12 3 81 65 4 22 1 4 250 114 4 5 000 000 15 000 000 24 12 30 24 4 150 13 1961 62 10 ...	W W W W W W W W W W W'W W W W W "W W" W W W W W. W W W W W W-W W W W W W W, W W W-W W W W W, "W W W W W W W W W W" W W W W W W W W W. W W-W W W W W W W W W W W W W W W W W W "W" W W W-W W W W W W W-W W W W&. "W W W W W W W W W", W W W, "W W W W W W W, W W W W W W W W W W". W W W W W W W ...

The decompression process first uses PPMD to decode the four separate elements and then reconstructs the full grammar from them. During the subsequent regeneration phase, the grammar is then used to regenerate the original source text exactly, character for character (i.e., the method is completely lossless). Whenever a previously unseen symbol is encountered as the sequence specified by the S rule is being processed, the current word is read from the sequence specified by the V rule; then, the position is moved along to the next word. The P rule is used to insert the punctuation between the word and digit symbols as they are encountered in the S rule. Whenever a digit is signified by the S_D symbol for this rule, the current digit symbol is read from the sequence specified by the D rule and then inserted into the decoded output sequence; the position then moves along to the next digit symbol.

Algorithm 1 summarizes the algorithm using pseudo-code. Lines 1 through 15 are for the n-gram tokenizer. Line 3 starts the **for** loop to read the n-grams in the input file. Lines 4 through 9 check whether the n-gram is a word; if it is, it prints the n-gram to the vocabulary file, assigns each id number with ids for unique n-grams increasing with each new n-gram that is found and also prints a W to the spaces & punctuation file. Lines 10 through 13 check whether this n-gram is a digit; if it is, it adds this digit to the digit file and prints W to the spaces & punctuation file. Lines 14 and 15 check whether this n-gram is punctuation or a space; if so, these are added to the spaces & punctuation file. Line 16 compresses the final text for the four files by using PPMD.

An additional improvement to our approach, in terms of both compression and execution speed, can be gained by further processing the files in the following manner. The main disadvantage of the symbols file is that it consists of many singletons that occur only once in the text and doubletons that occurs only twice (ISO, 1996). Singletons and doubletons are detrimental to encoding efficiency because they do not give any useful reference information (Ye and Cosman, 2003). In addition, singletons incur an unnecessary extra cost in our scheme, because their symbol numbers are unique and therefore cause the alphabet size to be incremented by one each time they occur, which is frequently due to the Zipf's Law-like nature of natural language text. As a result, the alphabet size can be substantially higher when these are present. A large alphabet for PPM is undesirable when using the full exclusions mechanism (Cleary and Witten, 1984) that PPM uses for its encoding, as it substantially slows execution speeds due to the need to exclude symbols already seen in the higher orders from lower order predictions.

Algorithm 1: Pseudo-code for the GRW-PPM.

Input: The source text file
Output: Compressed text

```
1  Open Vocabulary, SYMBOLS, DIGITS and SPACES & PUNC. files
2  id ← 2
3  for each n-gram in input file do
4      if ngram is a new word then
5          print n-gram to Vocabulary file
6          Assign id to each n-gram
7          print id to SYMBOLS file
8          print W to SPACES & PUNC. file
9          id ← id + 1
10     else if n-gram is a digit then
11         print n-gram to DIGITS file
12         print number 1 to SYMBOLS file
13         print W to SPACES & PUNC. file
14     else
15         print n-gram to SPACES & PUNC. file
16 Use PPMD to encode the four files.
```

In order to overcome these problems and therefore improve our new method, we process the symbols file to replace all singletons in that file with the same special symbol wherever they occur. For example, the symbols stream "1 6 7 6 7 7 4 5" has three singletons —- 1, 4 and 5. These singletons are replaced by a special symbol (Φ, say) and the symbols sequence being encoded becomes "Φ 6 7 6 7 7 Φ Φ". Each singleton can be readily decoded once the special symbol is encountered in the symbols stream, which signals the decoder to read the characters for the word from the next set of characters in the vocabulary stream until the next word separator character is encountered. For our example, let's say that the characters in the vocabulary stream are "*one six seven four five*". When replacing just singletons in the symbols stream, there is no need to change this vocabulary stream, since the decoder will have all the necessary information to decode each word because singletons only occur once. The only effect is that the symbols stream becomes slightly more compressible with a much smaller alphabet, which significantly speeds up compression when performing full exclusions, as shown below.

We also have an option to replace doubletons and tripletons (and so on) wherever they

occur in the symbols file if we wish. However, when replacing non-singletons in this case, there is no way to decode the characters when the word is being replaced the second time or subsequent times (for tripletons etc.), so a simple expedient is to repeat the word character for character in the vocabulary stream whenever it reoccurs. Using the previous example again, if we were to replace singletons and doubletons but not tripletons, then the symbols sequence would now be encoded as "Φ Φ 7 Φ 7 7 Φ Φ", since the symbol 6 appears twice but the symbol 7 appears three times. In the vocabulary stream in this case, the characters for symbol 6 would appear twice, i.e. it would now become *"one six seven six four five"* since the word *"six"* is a doubleton and therefore appears again in this sequence. Clearly, the size of the vocabulary stream will now grow because of the presence of the repeated words; this can affect the overall compression, but that disadvantage is offset by the significantly faster processing since the alphabet size in the symbols stream is much smaller.

In the experimental results below, we use the following labels for the variants of our algorithm: GRW-PPM for our standard algorithm; GRW1-PPM for when singletons are replaced by the special symbol; GRW2-PPM for when both singletons and doubletons are replaced; GRW3-PPM for when all the singletons, doubletons and tripletons are replaced and GRW4-PPM for when all the singletons, doubletons, tripletons and quadrupletons are replaced.

6.4 Experimental Results

This section discusses experimental results using GRW- PPM and the variants described above for compression of various text files. We compare our new method with other compression schemes. We also discuss the encoding execution times for GRW-PPM with and without using the full exclusions mechanism that PPM uses for its encoding.

In this experiment, GRW-PPM encoding is divided into four parts: vocabulary, symbols, digits and spaces and punctuation. Order 5 PPMD is used for the vocabulary, order 1 PPMD for the symbols, order 4 PPMD for the digits and for spaces and Punctuation. Experiments showed that these particular orders were the most effective at compressing the different text elements.

Table 6.6 illustrates the compression ratio for the four parts. The compression ratio is calculated by multiplying the compressed output size in bytes times 8 and dividing by

TABLE 6.6: Compression ratios for GRW-PPM of the four parts using different text files

File	Language	size	Encoding Vocab. (bpc)	Encoding Symbols (bpc)	Encoding Digits (bpc)	Encoding Spaces & Punc.	Overall (bpc)
BROWN	American English	5968707	0.226	1.698	0.014	0.278	2.21
LOB	British English	6085270	0.217	1.628	0.016	0.191	2.05
BACC	Arabic	31018167	0.143	1.078	0.006	0.173	1.40
CEG	Welsh	6753317	0.147	1.284	0.089	0.214	1.73
Hamshahri	Persian	1120834	0.311	0.982	0.042	0.101	1.43

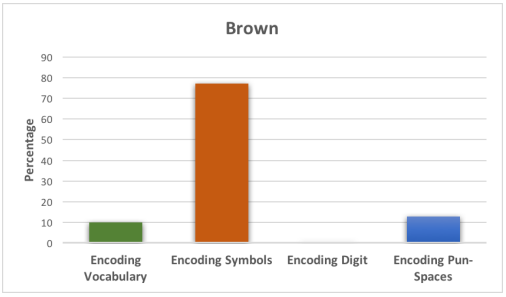
the original input file size in order to determine the contribution each part has to the overall encoding cost. As shown in the table, the digits part has the smallest compression rate for the different languages. The compression rates for vocabulary and spaces and punctuation are also small compared to the symbols part for the Brown, LOB, CEG, Hamshahri and BACC corpora.

Figure 6.3 shows the percentage of compression ratio for the four parts in the Brown, LOB, CEG, Hamshahri and BACC corpora. For Brown corpus, the result indicates that 77 percent (1.698 out of 2.21) of the compression rate overall about the symbol encoding. In contrast, 10 percent (0.226 out of 2.21) of the compression rate overall about the vocabulary encoding.

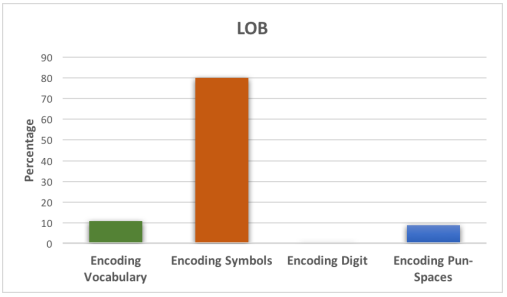
As shown in Table 6.7, GRW3-PPM at order 1 significantly outperforms GRW-PPM at order 1, as it has the best compression ratio for the corpora being compressed. The improvement of GRW3-PPM over GRW-PPM occurs for all texts; it ranges from over 2% to 4.2% for the BACC corpus of Arabic text.

In the experiments shown in Tables Table 6.7 and 6.11 for different text files, we found that the full exclusions mechanism improves compression rates. However, this increases the execution time slightly because all symbols are removed for full exclusions for predictions in a lower order level if they have already been seen in a higher order. (There may be many symbols requiring exclusion, depending on the context.) The configuration of our test machine is 4 GB GHz Intel Core i5, with 4GB internal memory.

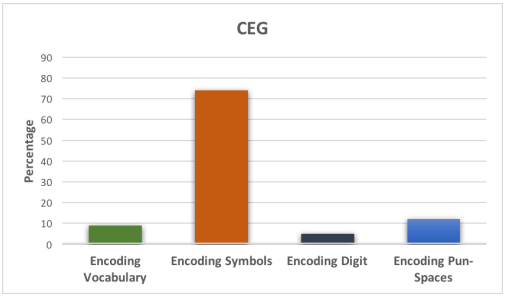
It is clear from Table 6.8 and Table 6.10 that not using full exclusions results in a poorer compression rate. The improvement of GRW1-PPM and GRW2-PPM with full exclusions over GRW1-PPM and GRW2-PPM without full exclusions ranges on average



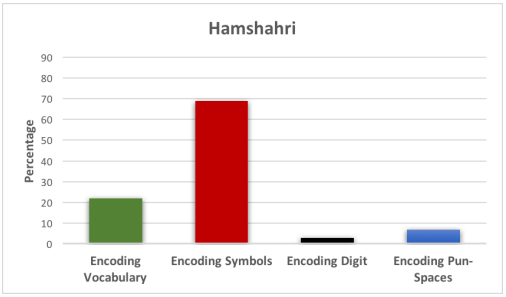
(A) Brown Corpus



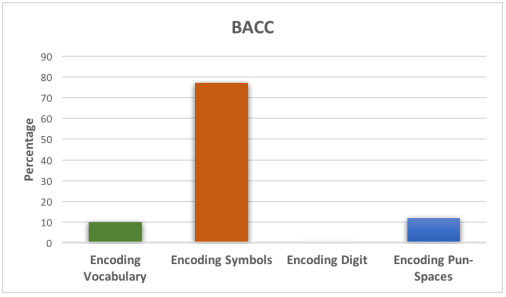
(B) LOB Corpus



(C) CEG Corpus



(D) Hamsh. Corpus



(E) BACC Corpus

FIGURE 6.3: The compression ratio for GRW-PPM in the four parts for Brown, LOB, BACC, CEG and Hamshahri corpora

TABLE 6.7: Compression Ratios for GRW-PPM with full exclusions compared with the performance of GRW1-PPM, GRW2-PPM and GRW3-PPM for different natural languages

	Using full exclusions				
File	GRW-PPM (bpc)	GRW1-PPM (bpc)	GRW2-PPM (bpc)	GRW3-PPM (bpc)	GRW4-PPM (bpc)
Brown	2.21	2.16	2.15	**2.14**	**2.14**
LOB	2.03	1.99	**1.98**	**1.98**	**1.98**
BACC	1.40	1.35	**1.34**	**1.34**	**1.34**
CEG	1.73	1.70	**1.69**	**1.69**	**1.69**
Hamsh.	1.43	1.41	1.40	**1.39**	**1.39**
Average	1.76	1.72	**1.71**	**1.71**	**1.71**

TABLE 6.8: GRW-PPM without full exclusions compared with performance of GRW1-PPM, GRW2-PPM and GRW3-PPM for different natural languages

File	Without full exclusions				
	GRW-PPM (bpc)	GRW1-PPM (bpc)	GRW2-PPM (bpc)	GRW3-PPM (bpc)	GRW4-PPM (bpc)
Brown	2.35	2.33	2.23	2.23	2.23
LOB	2.14	2.11	2.07	2.06	2.06
BACC	1.49	1.45	1.43	1.43	1.43
CEG	1.80	1.76	1.76	1.75	1.75
Hamsh.	1.52	1.48	1.46	1.46	1.46
Average	1.86	1.82	1.79	1.79	1.79

from just over 4% to 5.4% for all texts. However, the advantage in not performing full exclusions is that speed improves by an average of 3% to 20% for different texts.

Table 6.9 shows an interesting result when comparing GRW-PPM and GRW3-PPM with PPMD and $W|W$. It is clear that GRW3-PPM on average significantly outperforms $W|W$. GRW3-PPM shows an average 7.1% improvement over $W|W$. It also illustrates that there are significant differences between each of the compression methods for different languages.

For instance, for American English, $W|W$ achieves the best compression rate of all models, with a 3.6% improvement over GRW-PPM and a 0.45% improvement over GRW3-PPM. For British English, $W|W$ achieves a 4.3% improvement over GRW-PPM and a 1.0% improvement over GRW3-PPM. For Welsh, GRW3-PPM and PPMD attain a 2.3% improvement over GRW-PPM and an approximately 1.0% improvement over $W|W$. For Arabic, GRW3-PPM outperforms the other models, attaining a 14.6% improvement over PPMD and a 15.7% significant improvement over $W|W$. For Persian, GRW3-PPM exceeds all other models, with a 22.3% improvement over $W|W$ (Figure 6.4). The execution times for both full exclusions and without full exclusions are less for GRW1-PPM compared with GRW4-PPM due to there are fewer symbols needing to be encoded in GRW4-PPM.

TABLE 6.9: Comparing the PPMD, PPM word-based, GRW-PPM and GRW3-PPM models

File	size	PPMD Order 4 (bpc)	$W\|W$ Order 1 (bpc)	GRW-PPM Order 1 (bpc)	GRW3-PPM Order 1 (bpc)
BROWN	5968707	2.22	**2.13**	2.21	2.14
LOB	6085270	2.03	**1.96**	2.05	1.98
BACC	31018167	1.57	1.59	1.40	**1.34**
CEG	6753317	**1.69**	1.70	1.73	**1.69**
Hamsh.	1120834	1.75	1.79	1.43	**1.39**
Average		1.85	1.83	1.76	**1.70**

TABLE 6.10: Execution times for GRW1-PPM, GRW2-PPM and GRW3-PPM without using full exclusions

	With not using full exclusions			
File	GRW1-PPM (seconds)	GRW2-PPM (seconds)	GRW3-PPM (seconds)	GRW4-PPM (seconds)
Brown	722.25	481.15	389.04	320.10
LOB	596.83	583.66	353.13	296.92
BACC	5655.20	4156.35	3229.16	3179.45
CEG	275.82	198.56	193.31	138.03
Hamsh.	2544.21	1375.30	965.34	843.35

6.5 Compression experiments comparing GR-PPM and GRW-PPM

In this section, we report on some interesting experimental results when comparing the compression ratios between GR-PPM (as discussed in chapter 4) and GRW-PPM. Table 6.12 lists the results for different PPM variants: order 4 GRT-PPM, order 4 GRB-PPM from the chapter 4 and GRW-PPM from this chapter.

The PPM character-based grammar models PPM, GRT-PPM and GRB-PPM outperform the grammar word-based models GRW-PPM in different languages. For the Brown corpus of American English, GRB-PPM achieves a 7.4% improvement in bpc over that obtained with GRW-PPM. For the LOB corpus of British English, GRB-PPM attains an 8% improvement in compression rate over the output by GRW-PPM. For Arabic,

TABLE 6.11: Execution times for GRW1-PPM, GRW2-PPM and GRW3-PPM using full exclusions

File	Using full exclusions			
	GRW1-PPM (seconds)	GRW2-PPM (seconds)	GRW3-PPM (seconds)	GRW4-PPM4 (seconds)
Brown	760.83	600.05	471.92	342.59
LOB	670.20	436.98	328.12	329.57
BACC	6149.99	5292.56	3693.99	3320.88
CEG	302.71	264.58	239.65	173.57
Hamsh.	3260.91	2062.56	1268.33	916.22

TABLE 6.12: Comparing GRB-PPM, GRT-PPM and GRW-PPM model performance

File	size	GRT-PPM Order 4 (bpc)	GRB-PPM Order 4 (bpc)	GRW-PPM Order 1 (bpc)
BROWN	5968707	2.03	**1.99**	2.21
LOB	6085270	1.92	**1.88**	2.05
BACC	31018167	1.29	**1.21**	1.40
CEG	6753317	1.57	**1.51**	1.73
Hamshahri	1120834	1.31	**1.22**	1.43
Averge		1.62	**1.56**	1.76

GRB-PPM outperforms the grammar word-based GRW-PPM, achieving a nearly 15% improvement in compression rate. To some extent, these results are a reflection of the rich morphological nature of Arabic text, in which a word can take many different forms. For instance, " كتب" which means 'write', has been formed in many ways such as: يكتب اكتب كاتَب استكتب مكتوب كتبت كتبا كتبوا . For Persian text, GRB-PPM achieves a 14.6% improvement in bpc over the results generated with GRW-PPM. For Welsh text, GRB-PPM achieves a 12% improvement in compression rate over GRW-PPM (Figure 6.5). Although the improved word-based models now produce poorer results than the character-based models, this is interesting because these results are different from the previous results, which found word-based models to perform better (Moffat, 1989; Teahan, 1998; Teahan and Alhawiti, 2015).

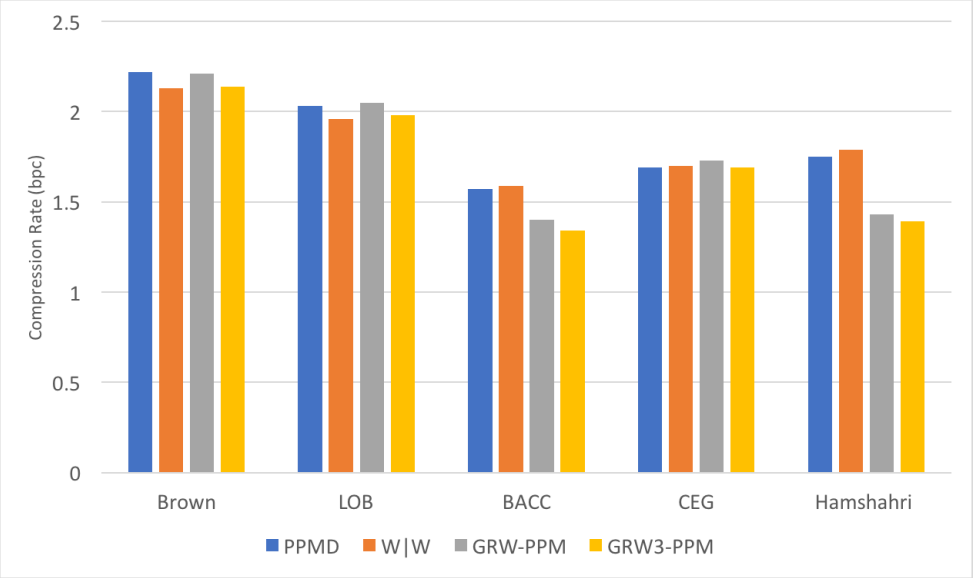


FIGURE 6.4: Comparing compression performance of the various methods for different languages

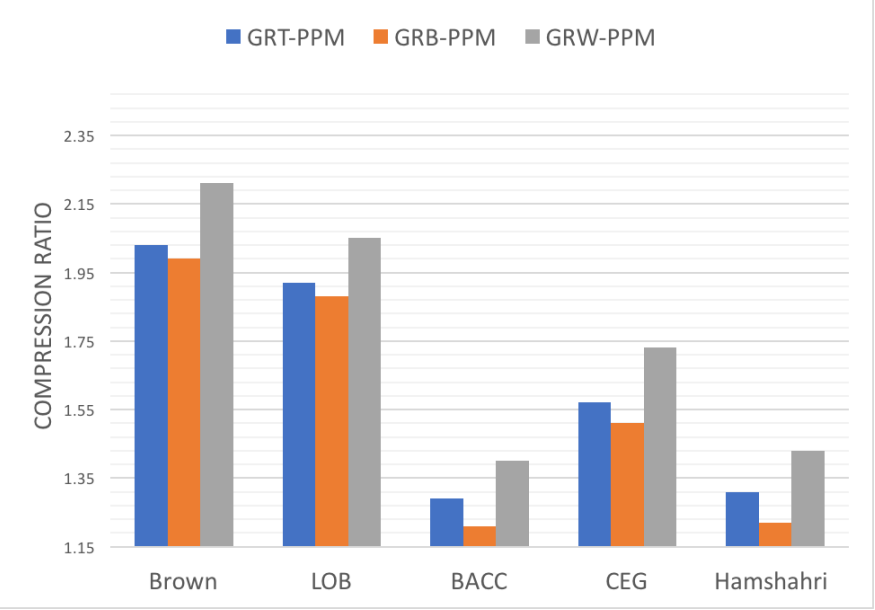


FIGURE 6.5: Comparing various versions of grammar-based PPM with different languages

6.6 Summary and Discussion

In this chapter, a new word-based grammar scheme (GRW-PPM) has been described for compressing natural language text. Our method creates a CFG by replacing words and repeated sequences of digits, spaces and punctuation with non-terminal symbols in the text as it is processed from beginning to end in a single pre-processing pass. The PPM text compression algorithm is then used as the encoder of the sequences of non-terminal sequences once they have been constructed for the whole text. Unlike PPM, which is an on-line method, our method is off-line during the phase that generates the grammar.

In our experimental evaluation, GRW-PPM has been compared with other well-known schemes using various language corpora for English, Welsh, Arabic and Persian. The best-performing scheme for languages using Arabic script (Arabic and Persian) is GRW-PPM, followed by the previous best performing the standard character-based PPMD scheme, then word-based PPM models ($W|W$). For English, our experiments show that the word-based PPM models ($W|W$) are the best when compared with standard PPM and GRW-PPM. For Welsh, the best results are achieved using the standard character-based PPMD scheme. These results are a reflection of the rich morphological nature of Arabic text, in which a word can take many different forms. GRW-PPM offers good results not only for Arabic language and language that uses Arabic in their writing system such as Persian but also for the Welsh language as a word can take different forms. Also, GRW3-PPM outperforms the other models for languages that take different forms such as Arabic, Persian, and Welsh. In the four languages overall, it is clear that the execution times without full exclusions is faster than the full exclusions. Using full exclusion requires extra computation, since each symbol has to be checked.

CHAPTER 7

GRB-PPM GENRE CLASSIFICATION

Contents

7.1 Introduction

The amount of electronic text for analysis has vastly increased, and the need has arisen to categorize it. An important step is to be able to classify the text or information among some known set of relevant categories or classes. Classifying text or documents has become significant because much of the research area in text mining and information retrieval involves classification in some way (Mohri et al., 2012). Good classification is also essential for many natural language processing applications such as text summarization (Yousefi-Azar and Hamey, 2016), information extraction (Aaditya and Mandowara, 2016), language identification (Alghamdi et al., 2016), dialect identification (Zaidan and Callison-Burch, 2014) and sentiment classification (Pang et al., 2002).

The purpose of this chapter is to apply the new method developed in Chapter 4 to a specific natural language processing application: genre classification. From this we can measure whether this method also has possibilities for NLP beyond the main application that has been investigated so far: text compression.

This chapter is organized as follows: Section 7.2 illustrates previous work used for classification. Then in section 7.3, we present evaluation techniques for text classification. In section 7.4, GRB-PPM text classification is presented. We discuss experimental results using our GRB-PPM classification method in section 7.5. A summary and discussion of our study is presented in section 7.6.

7.2 Text Classification

Text classification can also be called text categorization. Fragoudis et al.(Fragoudis et al., 2005) define text classification as *"the task of assigning one or more predefined categories to natural language text documents, based on their contents"*. Classification is a supervised learning form of machine learning, as it requires training data which is already categorized in order to build models that can be used to categorize unknown data, known as test data (Figure 7.1) (Mohri et al., 2012).

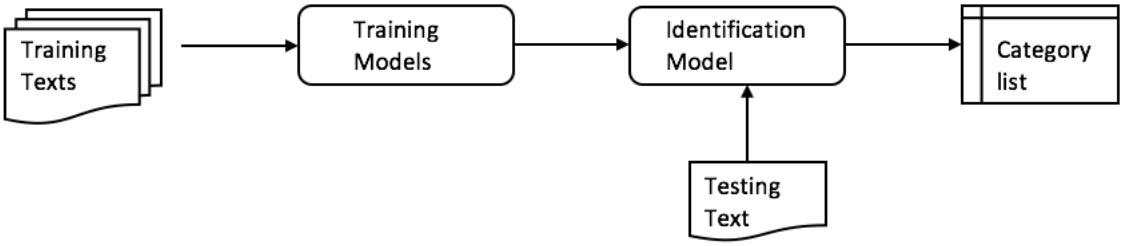


FIGURE 7.1: Text Classification Process (Mohri et al., 2012)

Several pre-processing steps are applied by traditional machine learning algorithms for text categorization, such as stemming, feature selection and extraction, stop word removal and tokenization (Dumais et al., 1998). In addition, feature-based approaches normally process features based on words. There are a number of problems when using feature-based approaches to classify text, such as choosing features before pre-processing, the need to define the morphological variants of words and to decide whether to remove non-alphabetic symbols and digits (Frank et al., 2000). In recent years, many studies have shown that using a compression-based approach based on the PPM scheme for text classification can outperform traditional text classification methods for different languages (Thomas, 2001; Alkhazi and Teahan, 2017). A preliminary investigation into the efficiency of applying the PPM model to categorize by genre was also presented by McCallum and Nigam in 1998, employing the Newsgroups data set (McCallum and Nigam, 1998).

7.3 Evaluation Techniques

In order to evaluate how well a classification model performs, measure the success of the classification technique and compare the results against other techniques, a number of evaluation criteria may be used. The most common evaluation criteria are (i) *accuracy*, (ii) *precision*, (iii) *recall* and (iv) *F-measure*. Each of these is discussed in more detail in the next subsection, along with the concept of a confusion matrix.

7.3.1 K-fold Cross validation

Data in this method is randomly split into K equal-sized folds or subsets. Each single fold or subset is used for testing the model, with the remaining subsets used for training

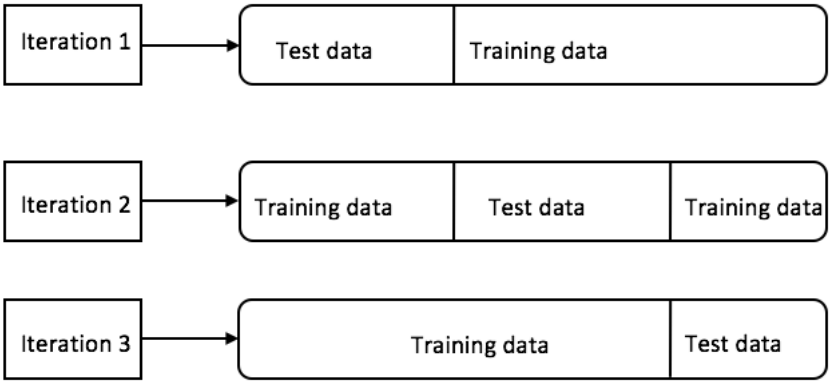


FIGURE 7.2: K-fold-Cross validation (k=3)

TABLE 7.1: Confusion matrix of two classes

Predicted Class	**Actual Class**	
	Positive	Negative
Positive	TP (True Positive)	FP (False Positive)
Negative	FN (False Negative)	TN (True Negative)

data. The cross-validation is then repeated K times, as shown in figure 7.2 when $K = 3$ (Mitchell, 1997).

7.3.2 Confusion Matrix

A confusion matrix is a simple way to record the performance of classification models. The actual classes are displayed in the columns, and the predicted classes are listed in the rows, as shown in Table 7.1 (Witten and Frank, 2005).

The meaning of the variables listed in the Table are as follows:

- *TP* (True Positive): This is number of cases that the prediction correctly classified.

- *TN* (True Negative): This is number of cases that the prediction correctly categorized as negative.

- *FP* (False Positive): This is number of cases that the prediction incorrectly categorized as positive.

- *FN* (False negative): This is number of cases that the prediction incorrectly categorized as negative.

All the evaluation measures described below can be calculated from the confusion matrix.

7.3.2.1 Accuracy

The accuracy of a classifier indicates the quality of a classifier. The accuracy can be calculated by:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{7.1}$$

7.3.2.2 Precision

Precision is an indicator of the documents that were correctly predicted as positive and is calculated by:

$$Precision = \frac{TP}{TP + FP} \tag{7.2}$$

7.3.2.3 Recall

Recall is the proportion of actual classes correctly categorised as positive and can be defined as:

$$Recall = \frac{TP}{TP + FN}. \tag{7.3}$$

7.3.2.4 F-measure

F-measure is a calculation based on both precision and recall.

$$F - measure = \frac{2 \times Precision \times Recall}{Precision + Recall}. \tag{7.4}$$

TABLE 7.2: Protocols for text categorization (Thomas, 2001)

	Dynamic Model	Static Model
Concatenation of training	AMDL (Protocol II)	SMDL (Protocol I)
Non-Concatenation of training	BCN (Protocol IV)	Protocol III

7.3.3 Protocols

In 2005, Marton *et al.* described three different compression-based approaches for text classification in the literature: SMDL (the standard minimum description length), AMDL (approximate MDL) and BCN (the best compression neighbour). Many compression-based methods have been used to classify different corpora under these three procedures (Marton et al., 2005). In 2011, Thomas introduced what he called "protocols", which have two attributes: the first involves whether dynamic or static models are used (the former are updated continuously whereas the latter remain fixed once training is completed); the second involves whether training documents are concatenated together into a single model or whether separate models are used for each document, as shown in Table 7.2.

Protocol I, SMDL, and protocol III both use static models. However, AMDL (protocol II) and BCN (protocol IV) use dynamic models. Moreover, AMDL and SMDL concatenate the training documents in the same class, which decreases the number of calculations needed when compared to non-concatenation of the training documents in the same class. That approach requires BCN to provide calculations for each training separately. Thomas notes that concatenated models not only achieve the best results for both static and dynamic approaches but are also faster than non-concatenated models (Thomas, 2001).

7.4 GRB-PPM Genre Classification

The GRB-PPM genre classification covers only classification of genre, not dialect, sentiment or other text classification tasks.

The minimum cross-entropy method calculated using PPM text compression has been found effective in text categorization and shows a good improvement over other methods (Teahan, 2000; Teahan and Harper, 2001).

Cross-entropy provides an important measurement, as it presents how well the estimated model is doing on the test document. Cross-entropy presents a useful measure for analysing the accuracy of competing models, with the lowest cross-entropy for the model concluded to be the "best". In our case, the correct genre of the text T is predicted as follows:

$$\hat{\theta} = argmin_i H(T|S_i) \tag{7.5}$$

where $H(T|S)$ is an approximation of the entropy of T with respect to S. For a given text T of length n and a model P, the cross-entropy is calculated by the following formula:

$$\begin{aligned} H(T|S) &= -\log_2 P(T) \\ &= -\sum_{i=1}^{n} \log_2 P(x_i|x_1 ... x_{n-1}), \end{aligned} \tag{7.6}$$

where $P(x_i|x_1,...,x_{n-1})$ indicates the probability of characters x_i for each context being encoded (Teahan, 2000). Each test text is compressed using the class models, and the class is chosen from the model used for training that has the minimum value or obtains the best compression (Teahan, 1998). In practice, it is calculated using a PPM Markov-based scheme that assumes a maximum fixed order of five and can be estimated using the following formula:

$$H(T|S) = -\sum_{i=1}^{n} \log_2 P(x_i|x_{i-5} ... x_{i-1}), \tag{7.7}$$

where $P()$ denotes the probabilities estimated by the order 5 PPM model. PPM normally is an online scheme with its model being adaptive and dynamically updated as the document or text is processed consecutively. An alternate static variant primes the model by using typical some training text, then compresses the test text without updating the model (Teahan, 2000). In the next section, we discuss experimental results using our new GRB-PPM approach to classify text by genre on the parallel Arabic-English corpus A (Alkahtani, 2015) which is based on OPUS. We use this corpus because it is

a parallel Arabic-English corpus that enables us to examine English and Arabic classification together. The genres covered by the corpus include such topics as *cinema, crimes, decisions, geographies, issues, stories* and *books*.

7.5 Experimental Results

In order to see how efficient our GRB-PPM method is for classification and compare our experimental results with other well-known compression-based classification methods at categorising English and Arabic text, four experiments were conducted: (1) English compression experiments; (2) Arabic compression experiments; (3) English classification experiments and (4) Arabic classification experiments. The first experiment examined the compression rates for English text using GRB-PPM and well-known compression methods. The second experiment examined the compression rates of Arabic text by applying GRB-PPM and other compression schemes. The third experiment produced the results of classifying English text by applying both static and dynamic models. The final experiment produced the results of classifying Arabic text using the previous models.

We applied our experimental results to corpus A (Alkahtani, 2015). As stated, this data set is a parallel corpus and has many categorises such as cinema, crimes, decisions, geographies, issues, stories and books. Table 7.3 shows the categories for corpus A and their size for both English and Arabic texts.

Both PPMD and GRB-PPM compression were performed using the TMT Toolkit to obtain a compression codelength estimate (Teahan, , in press). This toolkit allows both static and dynamic models to be produced from training text for both PPMD and GRB-PPM. The test files were first spilt into ten folds to perform 10-fold cross-validation, with nine training texts and one test text per class in each fold.

7.5.1 English compression experiments

This section discusses experimental results for GRB-PPM compared with other well-known schemes. Table 7.4 compares the results of using GRB-PPM using N=100 and order 5 with other compression methods such as Compress, Gzip, Bzip2 and PPMD order 5. It is clear that GRB-PPM is significantly better than the other compression

TABLE 7.3: The size of each category of corpus A in English text

Category	English Size (Mbytes)	Arabic Size (Mbytes)
cinema	26.0	51.3
crimes	6.2	9.6
decisions	10.8	15.8
geographies	10.6	15.6
issues	6.8	10.1
stories	19.9	29.3
books	7.2	10.5

methods. In addition, GRB-PPM achieves the best compression rate (shown in bold) in bpc for all cases in different genres. For instance, on average, GRB-PPM shows a 53% improvement over Compress, a 41% improvement over Gzip, a 17% improvement over Bzip2 and a nearly 10% improvement over PPMD. The last column indicates that GRB-PPM shows an improvement of 6.4 to 17 % over PPMD; GRB-PPM clearly outperforms PPMD over all genres. The experimental results from Table 7.4 have also been graphed in Figure 7.3.

TABLE 7.4: Compression rates for different compression-based methods for English text from corpus A

Genre	Size	Compress	Gzip	Bzip2	PPMD (bpc)	GRB-PPM (bpc)	Improvement (%)
cinema	26018997	3.06	2.60	1.9	1.72	**1.61**	6.4
crimes	6181161	3.08	2.54	1.86	1.73	**1.63**	5.8
decisions	10760957	2.56	1.83	1.19	1.12	**0.92**	9.0
geographies	10635962	2.88	2.23	1.61	1.44	**1.31**	9.0
issues	6827714	2.94	2.27	1.66	1.52	**1.39**	9.0
stories	19861483	3.07	2.58	1.89	1.69	**1.57**	7.1
books	7213471	2.55	1.82	1.18	1.13	**0.94**	17.0
Average		2.87	2.26	1.61	1.47	**1.33**	10.0

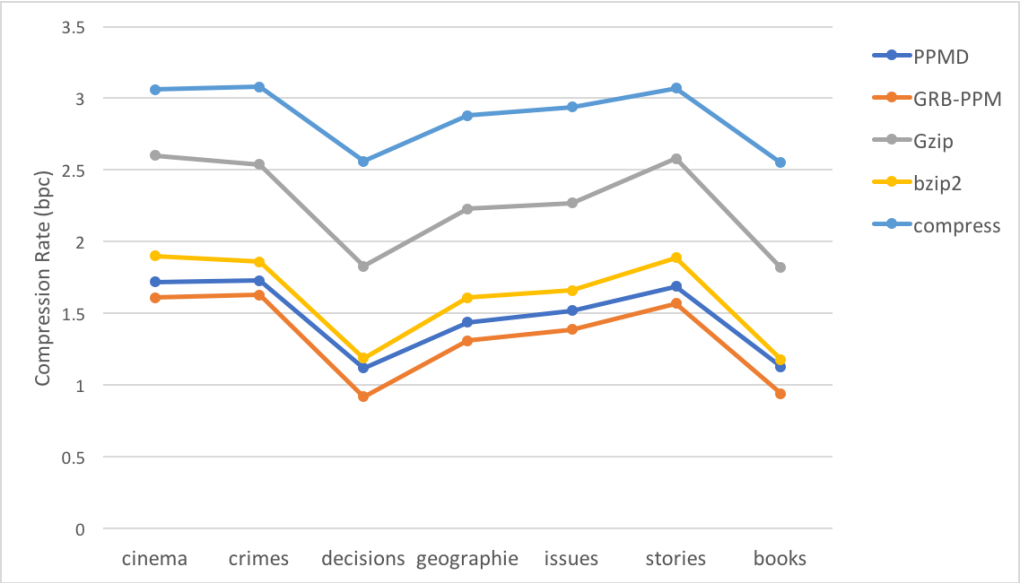


FIGURE 7.3: Comparing compression rates for English text of different compression-based methods in corpus A

7.5.2 Arabic compression experiments

For Arabic text, the experimental results for GRB-PPM are compared with well-known compression methods and show that GRB-PPM is significantly better than other compression schemes in different genres for corpus A. For example, as shown in table 7.5 and graphed in Figure 7.4, GRB-PPM shows, on average, a 55% improvement over Compress, a 46% improvement over Gzip, a nearly 20% improvement over Bzip2 and a nearly 30% improvement over PPMD. The last column indicates that GRB-PPM shows an improvement of 21.8 to 48.0 % over PPMD, so GRB-PPM clearly outperforms PPMD in all genres.

7.5.3 English classification experiments

Our experimental results for classification used both dynamic models, where the models are updated continuously, and static PPM models, where models are not updated once the training is completed (as per protocols from Table 7.2). In the dynamic approach, we use an off-the-shelf compression-based algorithms to estimate the relative entropy (Khmelev and Teahan, 2003). Here, for some compressor COMP (such as Compress,

TABLE 7.5: Compression rates for different compression-based methods for Arabic text in corpus A

Genre	Size	Compress	Gzip	Bzip2	PPMD (bpc)	GRB-PPM (bpc)	Improvement (%)
cinema	51257604	2.56	2.34	1.58	1.70	**1.33**	21.8
crimes	9579211	2.34	2.04	1.35	1.59	**1.18**	25.0
decisions	15770413	2.06	1.55	1.10	1.25	**0.65**	48.0
geographies	15622256	2.25	1.84	1.21	1.44	**1.11**	23.0
issues	10130055	2.29	1.86	1.23	1.46	**1.01**	30.8
stories	29261180	2.39	2.11	1.44	1.61	**1.20**	25.4
books	10526752	2.06	1.56	1.02	1.25	**0.68**	45.6
Average		2.27	1.90	1.27	1.47	**1.02**	30.6

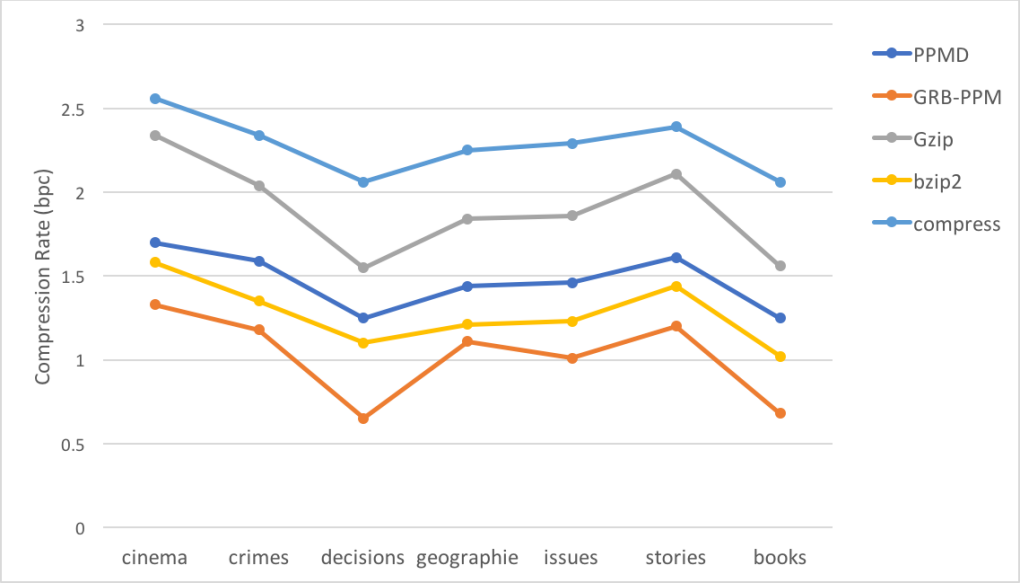


FIGURE 7.4: Comparing compression rates for Arabic text of different compression-based methods in corpus A

Gzip, Bzip2, PPMD and GRB-PPM), let $L_{Comp}(T)$ be the compressed length of text T. The relative entropy is then calculated by the following formula:

$$H(T|S) = L_{Comp}(T \oplus S_i) - L_{Comp}(T). \tag{7.8}$$

The relative entropy is estimated by subtracting the difference between the compressed text of concatenating the test text S_i onto the end of the training text T and the compressed size of training text T. In PPMD and GRB-PPM, the correct class of the text T

is predicted using formula (7.5).

The static models are created by the toolkit from training data. These models do not change when used to compress the testing text. The codelengths are compared and the file with the smallest codelength is chosen for each class label for both dynamic and static models, as in formula (7.5).

Classifying the genres such as cinema, crimes, decisions, geographies, issues, stories and books for English text using static models for the PPMD compression algorithm achieved an accuracy of 0.95, a precision of 0.85, a recall of 0.83 and an F-measure of 0.82. The results in Table 7.6 show the confusion matrix for PPMD for English text. For the GRB-PPM method, we found that the accuracy was about 0.94, precision 0.80, recall 0.79 and F-measure 0.79; the confusion matrix for GRB-PPM is shown in Table 7.7.

TABLE 7.6: PPMD confusion matrix for English text using static models

	cinema	crimes	decisions	geographies	issue	stories	books
cinema	100	0	0	0	0	0	0
crimes	0	62	0	35	0	3	0
decisions	0	0	98	0	0	0	2
geographies	0	0	0	100	0	0	0
issue	0	7	0	6	85	2	0
stories	0	0	6	0	15	74	5
books	1	0	37	0	0	0	62

TABLE 7.7: GRB-PPM confusion matrix for English text using static models.

	cinema	crimes	decisions	geographies	issue	stories	books
cinema	100	0	0	0	0	0	0
crimes	0	83	0	16	1	0	0
decisions	0	0	63	0	4	0	33
geographies	0	26	0	74	0	0	0
issue	0	14	2	5	77	2	2
stories	0	15	3	0	10	67	5
books	0	0	17	0	5	0	78

The steps of the dynamic model experiment are as follows:

• The dynamic models created for each genre and the compressed files for each genre are generated by using GRB-PPM and PPMD order 5 to compress the texts.

• Then, the sizes of the compressed files are compared and the smaller compressed file is chosen for each class label.

Table 7.8 displays the results of the dynamic model for GRB-PPM, Table 7.9 for the PPMD, Table 7.10 for Compress, Table 7.11 for Gzip and Table 7.12 for Bzip2. The tables show the experimental results, on average, for accuracy, precision, recall and F-measure for each fold of GRB-PPM, PPMD, Compress, Gzip and Bzip2.

TABLE 7.8: Classification results of GRB-PPM with English text for dynamic models

Fold	Accuracy	Precision	Recall	F-Measure
1	0.96	0.92	0.88	0.86
2	0.95	0.82	0.81	0.81
3	0.95	0.89	0.78	0.78
4	0.95	0.83	0.81	0.80
5	0.96	0.92	0.88	0.86
6	0.96	0.92	0.87	0.87
7	0.94	0.86	0.80	0.76
8	0.95	0.86	0.84	0.84
9	0.94	0.83	0.84	0.82
10	0.95	0.89	0.82	0.81
Average	0.95	0.87	0.83	0.82

Table 7.13 compares the genre categorisation of English text using different compression methods, with the best results obtained in terms of accuracy, recall, precision and F-measure highlighted in bold and graphed in Figure 7.5. It is clear that GRB-PPM order 5 outperforms the other methods (Compress, Gzip, Bzip2 and PPMD) for English text.

In order to compare the classification results for English, a summary of results is provided in Table 7.14 and graphed in figure 7.6. They show that GRB-PPM performs better than PPMD in the dynamic case, which is the opposite for the static case in terms of accuracy, recall, precision and F-measure.

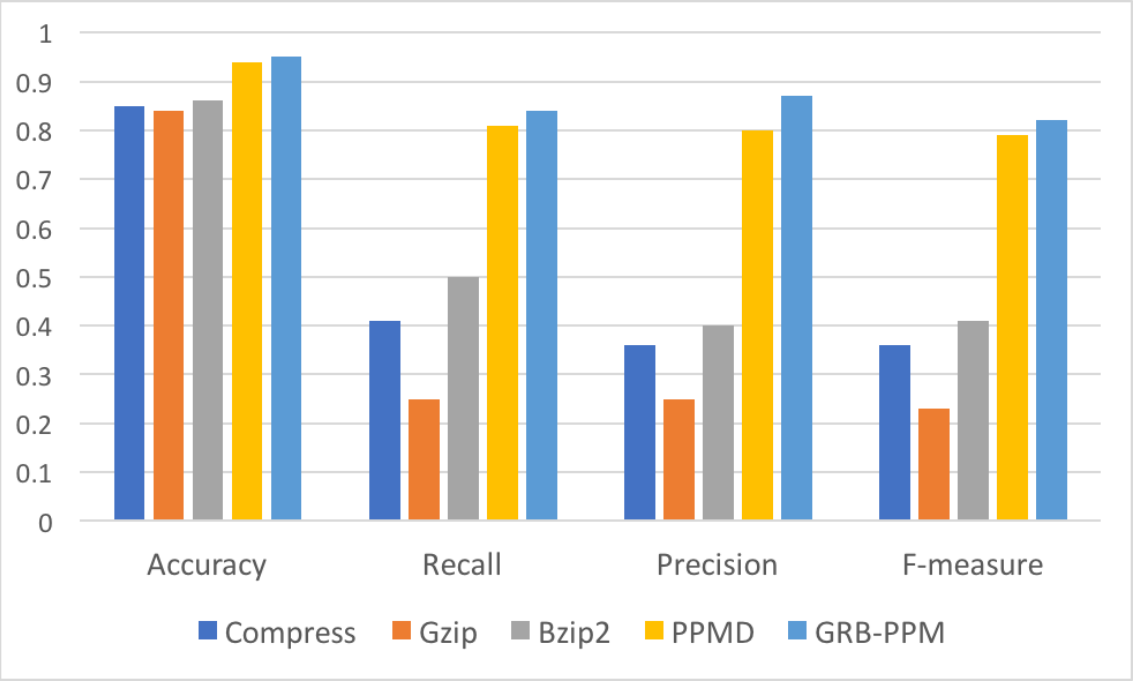


FIGURE 7.5: Comparing classification results for English text of different compression-based methods using dynamic models

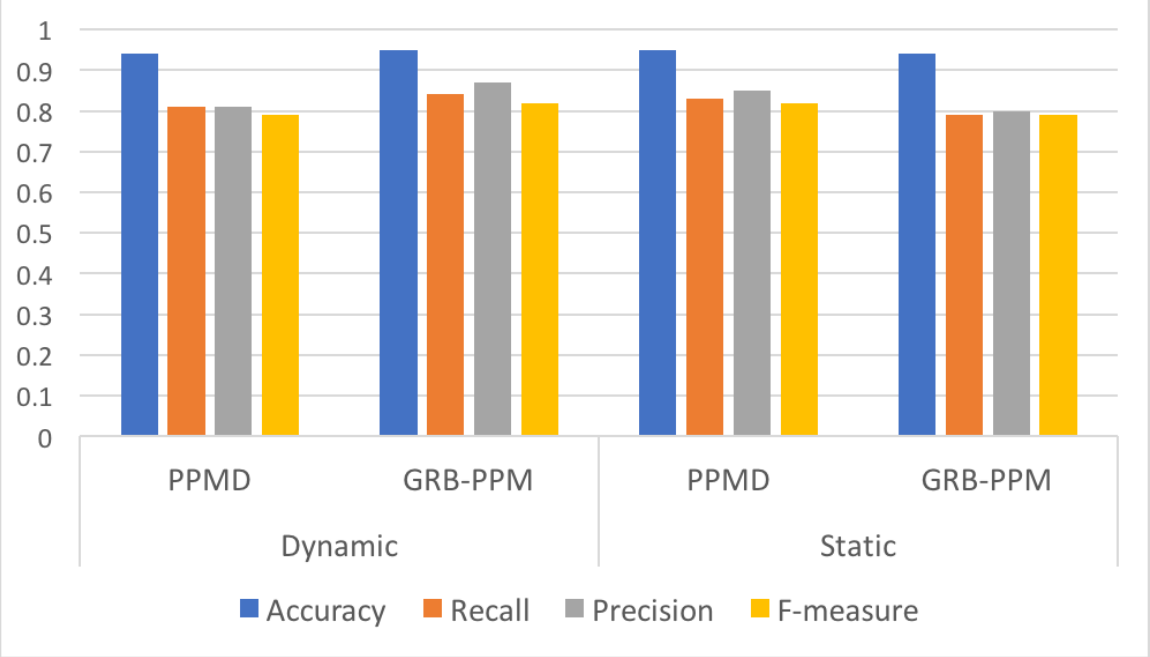


FIGURE 7.6: Comparing static and dynamic classification results in English text with concatenated training

TABLE 7.9: Classification results of PPMD with English text for dynamic models

Fold	Accuracy	Precision	Recall	F-Measure
1	0.99	0.98	1	0.99
2	0.97	0.94	0.91	0.91
3	0.93	0.71	0.75	0.69
4	0.93	0.76	0.79	0.74
5	0.94	0.73	0.78	0.88
6	0.96	0.91	0.88	0.85
7	0.96	0.86	0.85	0.69
8	0.92	0.78	0.70	0.73
9	0.94	0.70	0.78	0.68
10	0.92	0.72	0.69	0.68
Average	0.94	0.80	0.81	0.79

TABLE 7.10: Classification results of Compress of English text for dynamic models

Fold	Accuracy	Precision	Recall	F-Measure
1	0.85	0.33	0.33	0.32
2	0.81	0.17	0.81	0.17
3	0.84	0.39	0.40	0.38
4	0.82	0.23	0.26	0.24
5	0.85	0.30	0.38	0.33
6	0.87	0.62	0.51	0.50
7	0.84	0.31	0.40	0.33
8	0.89	0.46	0.60	0.49
9	0.88	0.36	0.48	0.39
10	0.89	0.48	0.60	0.50
Average	0.85	0.36	0.41	0.36

7.5.4 Arabic classification experiments

This section reports the experiment results applied to Arabic text as part of the evaluation of GRB-PPM and other compression methods. We use the same static and dynamic models that were discussed in the previous section.

The confusion matrix for the static case for Arabic text for GRB-PPM order 5 is shown in Table 7.15 and for PPMD order 5 in Table 7.16. Classification using GRB-PPM obtained an accuracy of 0.95, a precision of 0.86, a recall of 0.84 and an F-measure

TABLE 7.11: Classification results of Gzip for English text for dynamic models

Fold	Accuracy	Precision	Recall	F-Measure
1	0.83	0.23	0.21	0.21
2	0.82	0.10	0.16	0.12
3	0.85	0.15	0.20	0.17
4	0.84	0.14	0.20	0.15
5	0.84	0.28	0.21	0.19
6	0.85	0.42	0.25	0.24
7	0.83	0.13	0.21	0.16
8	0.85	0.23	0.23	0.20
9	0.86	0.23	0.28	0.25
10	0.91	0.64	0.63	0.61
Average	0.84	0.25	0.25	0.23

TABLE 7.12: Classification results of Bzip2 with English text for dynamic models

Fold	Accuracy	Precision	Recall	F-Measure
1	0.88	0.37	0.50	0.42
2	0.84	0.30	0.35	0.24
3	0.86	0.36	0.50	0.40
4	0.85	0.22	0.38	0.28
5	0.86	0.32	0.50	0.38
6	0.88	0.52	0.60	0.51
7	0.87	0.40	0.53	0.40
8	0.89	0.51	0.60	0.52
9	0.88	0.54	0.58	0.49
10	0.88	0.49	0.53	0.47
Average	0.86	0.40	0.50	0.41

of 0.84. Classification using PPMD showed an accuracy of 0.94, a precision of 0.81, a recall of 0.79 and an F-measure of 0.79. Clearly, GRB-PPM order 5 outperforms PPMD order 5 in static models using Arabic text.

For the dynamic case, we applied the same technique explained in the previous section. It is clear that GRB-PPM achieves better results than other compression schemes. Table 7.17 displays the classification results of the dynamic model for GRB-PPM, Table 7.18 for PPMD, Table 7.19 for Compress, Table 7.20 for Gzip and Table 7.21 for

TABLE 7.13: Genre categorisation of English text using different compression methods with corpus A for dynamic models.

	Compress	Gzip	Bzip2	PPMD	GRB-PPM
Accuracy	0.85	0.84	0.86	0.94	**0.95**
Recall	0.41	0.25	0.50	0.81	**0.83**
Precision	0.36	0.25	0.40	0.80	**0.87**
F-measure	0.36	0.23	0.41	0.79	**0.82**

TABLE 7.14: Accuracies achieved by using dynamic and static models on English text with concatenated training

	Dynamic		Static	
	PPMD	GRB-PPM	PPMD	GRB-PPM
Accuracy	0.94	**0.95**	**0.95**	0.94
Recall	0.81	**0.83**	**0.83**	0.79
Precision	0.80	**0.87**	0.85	0.80
F-measure	0.79	**0.82**	**0.82**	0.79

Bzip2. The tables show the results for each fold and an overall average for accuracy, precision, recall and F-measure for all five schemes.

TABLE 7.15: GRB-PPM confusion matrix with Arabic text for static models

	cinema	crimes	decisions	geographies	issue	stories	books
cinema	100	0	0	0	0	0	0
crimes	0	97	0	3	1	0	0
decisions	0	0	69	0	4	0	30
geographies	0	25	0	75	0	0	0
issue	0	15	0	5	85	0	0
stories	0	4	3	1	11	75	6
books	0	1	10	0	0	0	89

Table 7.22 and Figure 7.7 present the outcomes of the different compression methods for Arabic text. Clearly, GRB-PPM significantly outperforms the other methods, with an average accuracy of 0.95, an average precision of 0.86, an average recall of 0.81 and an average F-measure of 0.80.

Table 7.23 compares the outcomes of static and dynamic classification results for GRB-PPM and PPMD using Arabic text; they are also graphed in Figure 7.8. It is clear that

TABLE 7.16: PPMD confusion matrix with Arabic text for static models

	cinema	crimes	decisions	geographies	issue	stories	books
cinema	100	0	0	0	0	0	0
crimes	0	66	0	32	0	2	0
decisions	0	0	86	0	0	0	14
geographies	0	2	0	98	0	0	0
issue	0	7	0	1	92	0	0
stories	0	0	5	0	15	73	7
books	0	0	59	0	0	0	41

TABLE 7.17: Classification results of GRB-PPM with Arabic text for dynamic models

Fold	Accuracy	Precision	Recall	F-Measure
1	0.97	0.90	0.88	0.89
2	0.95	0.88	0.84	0.84
3	0.94	0.87	0.77	0.72
4	0.95	0.84	0.82	0.82
5	0.95	0.84	0.80	0.79
6	0.96	0.92	0.87	0.84
7	0.96	0.89	0.88	0.88
8	0.94	0.87	0.78	0.78
9	0.94	0.87	0.77	0.72
10	0.94	0.77	0.78	0.76
Average	0.95	0.86	0.81	0.80

the GRB-PPM achieves better results than PPMD for Arabic text in terms of accuracy, recall, precision and F-measure.

Table 7.24 and Table 7.25 show the amount of time in seconds required to perform the compression experiments for static and dynamic models. The dynamic model in both Arabic and English took longer to produce results than static model, due to the fact that it had to compress the training text twice because of the "off-the-shelf" method of calculating the relative entropy for equation (7.8).

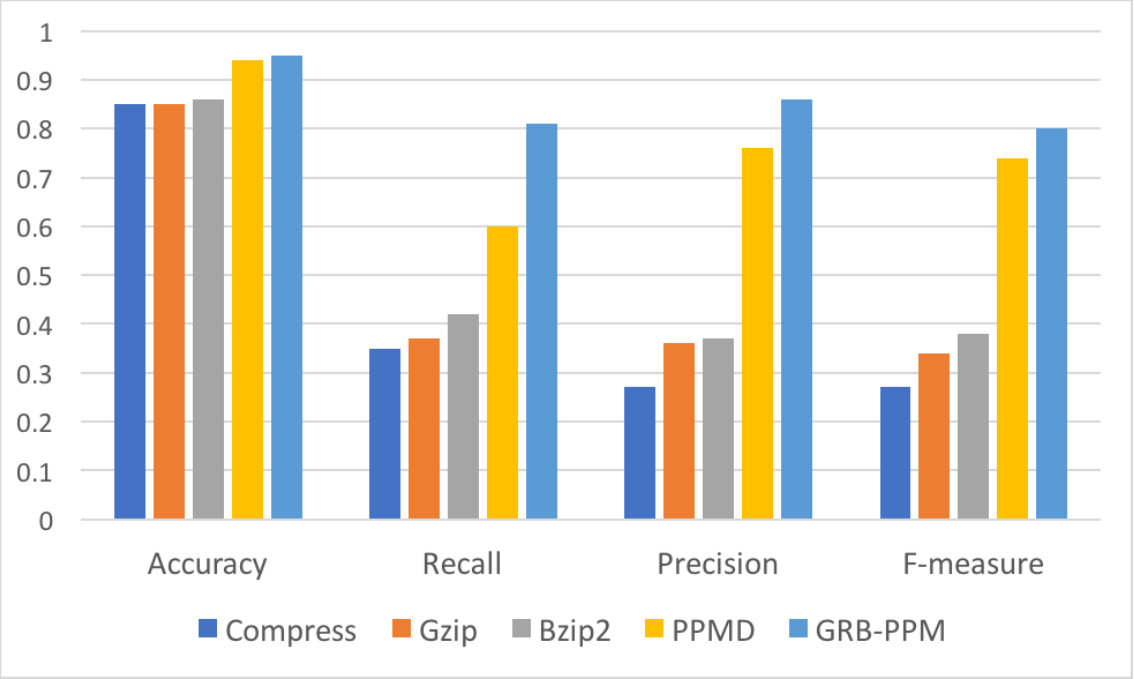


FIGURE 7.7: Comparing classification results for Arabic text of different compression-based methods using dynamic models

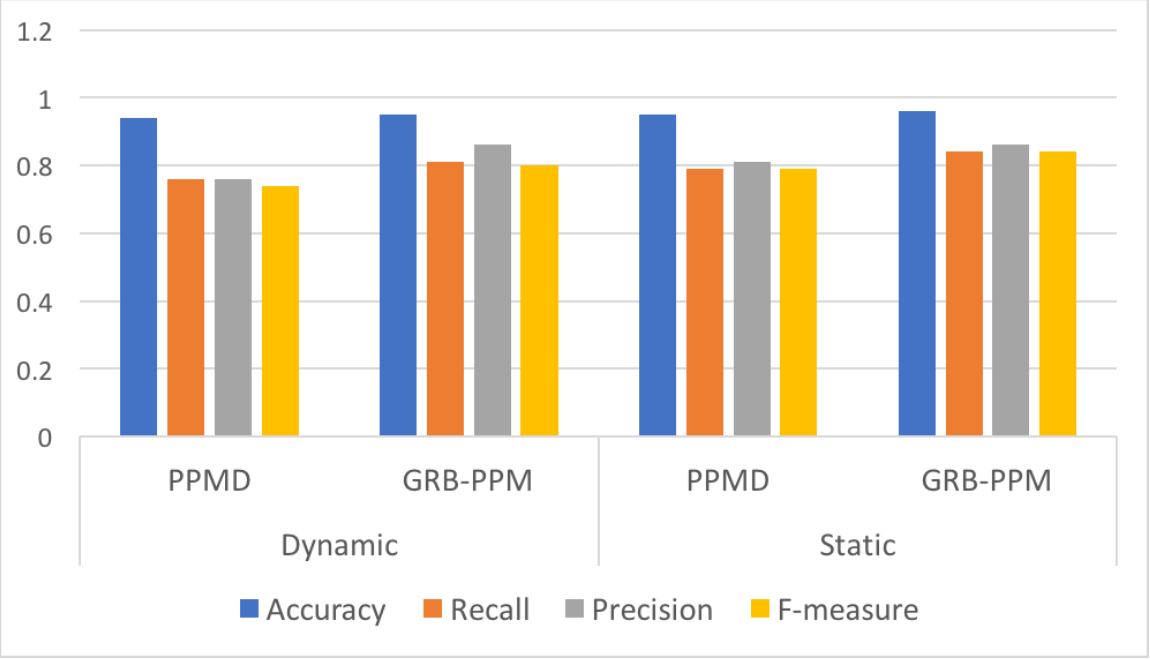


FIGURE 7.8: Comparing static and dynamic classification results on Arabic text with concatenated training

TABLE 7.18: Classification results of PPMD with Arabic text for dynamic models

Fold	Accuracy	Precision	Recall	F-Measure
1	0.98	0.94	0.94	0.94
2	0.97	0.92	0.87	0.88
3	0.92	0.75	0.70	0.64
4	0.94	0.74	0.78	0.75
5	0.93	0.66	0.74	0.69
6	0.96	0.88	0.85	0.85
7	0.95	0.83	0.82	0.82
8	0.91	0.73	0.64	0.64
9	0.91	0.59	0.67	0.61
10	0.91	060	0.67	0.62
Average	0.94	0.76	0.76	0.74

TABLE 7.19: Classification results of Compress for Arabic text for dynamic models

Fold	Accuracy	Precision	Recall	F-Measure
1	0.84	0.13	0.33	0.18
2	0.88	0.49	0.53	0.47
3	0.83	0.12	0.21	0.15
4	0.85	0.26	0.34	0.27
5	0.84	0.17	0.27	0.18
6	0.87	0.28	0.40	0.32
7	0.86	0.38	0.42	0.35
8	0.84	0.18	0.28	0.21
9	0.86	0.43	0.38	0.31
10	0.85	0.31	0.37	0.30
Average	0.85	0.27	0.35	0.27

7.6 Summary and Discussion

In this chapter, we performed genre classification experiments using corpus A for English and Arabic texts using GRB-PPM and PPMD order 5 and other well-known compression methods. Both static and dynamic models were applied to different texts. Four experiments were conducted in this chapter to compress and classify the genres of corpus A. First, a compression of English text was performed, with the outcomes showing

TABLE 7.20: Classification results of Gzip with Arabic text for dynamic models

Fold	Accuracy	Precision	Recall	F-Measure
1	0.84	0.27	0.30	0.27
2	0.84	0.42	0.28	0.28
3	0.85	0.32	0.31	0.30
4	0.84	0.34	0.31	0.30
5	0.84	0.39	0.32	0.28
6	0.87	0.38	0.41	0.37
7	0.87	0.31	0.41	0.35
8	0.85	0.28	0.35	0.29
9	0.89	0.43	0.50	0.44
10	0.90	0.52	0.57	0.54
Average	0.85	0.36	0.37	0.34

TABLE 7.21: Classification results of Bzip2 with Arabic text for dynamic models

Fold	Accuracy	Precision	Recall	F-Measure
1	0.88	0.35	0.40	0.44
2	0.84	0.30	0.30	0.29
3	0.85	0.40	0.38	0.32
4	0.84	0.27	0.28	0.27
5	0.84	0.20	0.55	0.22
6	0.90	0.54	0.35	0.49
7	0.85	0.20	0.51	0.29
8	0.89	0.50	0.58	0.49
9	0.89	0.55	0.58	0.50
10	0.89	0.48	0.58	0.50
Average	0.86	0.37	0.42	0.38

that GRB-PPM order 5 and $N = 100$ (e.g. the N most frequent) achieved better results for corpus A. Second, a compression of Arabic text was examined, with the results showing that GRB-PPM outperformed the other compression results. Then, a classification of English text was accomplished using different compression methods; the results indicated that the best performance in the static model for English text was PPMD, followed by GRB-PPM. For Arabic, GRB-PPM was better, followed by PPMD. Finally, a classification of Arabic text was shown, with the results showing that the best outcomes for dynamic model for English and Arabic texts are found with GRB-PPM, with PPMD.

TABLE 7.22: Genre categorisation of Arabic text using different compression methods in corpus A for dynamic models

	Compress	Gzip	Bzip2	PPMD	GRB-PPM
Accuracy	0.85	0.85	0.86	0.94	**0.95**
Recall	0.35	0.37	0.42	0.76	**0.81**
Precision	0.27	0.36	0.47	0.76	**0.86**
F-measure	0.27	0.34	0.38	0.74	**0.80**

TABLE 7.23: Accuracies achieved by using dynamic and static models on Arabic text with concatenated training

	Dynamic		Static	
	PPMD	GRB-PPM	PPMD	GRB-PPM
Accuracy	0.94	**0.95**	0.94	**0.95**
Recall	0.76	0.81	0.79	**0.84**
Precision	0.76	**0.86**	0.81	**0.86**
F-measure	0.74	0.80	0.78	**0.84**

TABLE 7.24: Average time in seconds to calculate using GRB-PPM and PPMD on English text with concatenated training for each fold

	Dynamic		Static	
	PPMD	GRB-PPM	PPMD	GRB-PPM
Time(secs)	16498.5	600427.7	330.8	3193.7

TABLE 7.25: Average time in seconds to calculate using GRB-PPM and PPMD on Arabic text with concatenated training for each fold

	Dynamic		Static	
	PPMD	GRB-PPM	PPMD	GRB-PPM
Time(secs)	18269.3	880705.6	261.4	2797.1

In this chapter, The results confirmed that better compression leads to better classification (as per the research question 4) and that the results are similar across languages in a parallel corpus. For example, GRB-PPM outperforms PPMD in classification for both dynamic and static models. The experimental results present the outcomes of the different compression methods for English and Arabic text. Clearly, GRB-PPM significantly outperforms the other methods for Arabic, with an average accuracy of 0.95. However, PPMD outperforms the other methods for English text, with an average accuracy of 0.95. These results are a reflection of the calculation of GRB-PPM, in which a model

in GRB-PPM uses cross-entropy to present a useful measure for analysing the accuracy of competing models, with the lowest cross-entropy for the model concluded to be the "best". GRB-PPM for both static and dynamic models consumes more time than PPMD due to requires three passes through the text determining the list of n-graphs that define the grammar in the first pass prior to the compression phase; correcting the text using the grammar in the second pass and then encoding it using PPM in the third pass.

CHAPTER 8

SUMMARY AND FUTURE WORK

Contents

8.1 Summary

Firstly, this chapter reviews the achievements of this research. Secondly, it returns to the initial aim and objectives to discus whether they have been achieved. It also reviews the research questions. The most important results are highlighted in this chapter, which also reviews the contributions of the research and the experimental results. Finally, a number of suggestions to encourage future work are offered.

We apply grammar-based pre-processing prior to using the PPM compression algorithm. This achieves significantly better compression for different natural language texts than other well-known compression methods. Our method first generates a grammar based on the most common two-character sequences (bigraphs) or three-character sequences (trigraphs) in the text being compressed; it then replaces these sequences using the respective non-terminal symbols defined by the grammar in a pre-processing phase undertaken prior to compression. This leads to significantly improved results in compression for several natural languages.

We describe further improvements using a two-pass scheme where grammar-based pre-processing is applied again in a second pass through the text. We then apply the algorithms to the files in the Calgary Corpus and again achieve significantly improved results in compression when compared with other compression algorithms, including a grammar-based approach, the Sequitur algorithm.

Despite the advances of the PPM method in predicting upcoming symbols or words in English, more research is required to devise better compression methods for other languages, such as Arabic due in part to the rich morphological nature of Arabic text, where a word can take many different forms. In this dissertation, we proposed a new method that achieves the best compression rates not only for Arabic text but also for other languages that use Arabic script in their writing systems, such as Persian. Our word-based method (GRW-PPM) constructs a CFG for the text, which is then encoded using PPM to achieve excellent compression rates.

Finally, we investigated the classification of genre in English and Arabic text by using our new character-based text compression scheme (GRB-PPM). Experimental results on a parallel Arabic and English corpus show that our new method is very effective when compared with traditional compression-based classification methods. We have also confirmed that good compression leads to good classification.

The following sections discuss these results in more detail.

8.1.1 Grammar based pre-processing for PPM (GR-PPM)

Grammar based pre-processing for PPM (GR-PPM) is based on combining two basic approaches, CFG and PPM, to create a novel method compression method for text files. In this technique, we first generate a grammar based on the most common two-character sequences (bigraphs) or three-character sequences (trigraphs) in the source files to generate a grammar and substitute a repeated symbol with non-terminal symbols before using PPM to compress the corrected text sequence.

GRB-PPM (Single-Pass Grammar Bigraphs for PPM) and GRT-PPM (Single-Pass Grammar Trigraphs for PPM) are maintained by two constraints: firstly, non-terminal uniqueness, which means that each pair can only appear once in the grammar; secondly, rule utility, which requires that each rule should be used more than once. Experimental results show that the GRB-PPM achieves the best compression ratio for texts in different languages such as English, Arabic, Chinese, Welsh and Persian compared with several well-known compression methods, such as Gzip, Bzip2 and BS-PPM order 4.

8.1.2 Recursive Grammar preprocessing for PPM

A further new technique was proposed with improvements in the effectiveness of the GR-PPM compression algorithm. In this method, we use a two-pass scheme instead of a one-pass scheme and replaced the N most frequent n-grams from the files generated by GR-PPM to generate a second grammar with one rule for each n-graph. This is done during the second pre-processing phase prior to the compression phase in a stage that allows the text to be regenerated during the post-processing stage.

Our new method shows an improvement in compression results when replacing the N most frequent symbols for two-pass bigraphs. We call the variant of our method GRBB-PPM (Two-Pass Grammar Bigraphs for PPM). We also use two-pass trigraphs in a variant we have called GRTT-PPM (Two-Pass Grammar Trigraphs for PPM).

Experimental results show that the GRBB-PPM and GRTT-PPM schemes yield a significant enhancement in compression of natural language texts compared to well-known compression methods, including PPMC and Sequitur. The schemes also perform well

on the Calgary Corpus. In addition, the experimental results show that in terms of the execution times for orders 1 through 7 in different languages, GRTT-PPM is better than GRBB-PPM.

8.1.3 Grammar word-based pre-processing for PPM

A new grammar-based method (GRW-PPM) using words was also described (in chapter 6). Our new word-based method constructs a CFG for the text; this grammar is then encoded using PPM to achieve excellent compression rates. The experiments show some interesting results, especially for Arabic and the languages that use Arabic in their writing systems, such as Persian. GRW-PPM achieves better compression rates than PPM word-based compression ($W|W$). Compared with the character-based variants, it was found that the best compression method is character-based followed by word-based compression for different languages (see section 6.5) which contrasted with previous results.

8.1.4 Classification using GRB-PPM

Classification is an important natural language process application. We have also shown that our GRB-PPM scheme performs well when applied to the classification of Arabic and English.

We also found that GRB-PPM performs better than well-known compression methods in genre classification. The results confirm that better compression leads to better classification; the results are similar across languages for a parallel corpus.

8.2 Review of Aim and Objectives

The aim and objectives of this research outlined in section 1.2 have been successfully achieved in the research. Novel grammar-based methods for compression based on characters using PPM for different language texts have been designed and developed. These new methods have been compared with well-known compression methods. Further improvements to our new method have also been developed. A new word-based

grammar-based has been developed. This method has also successfully been applied to the problem of text classification.

Therefore, the particular objectives detailed in section 1.2 were achieved:

• *Design and implement novel grammar-based methods for compression based on characters using PPM.*

Several novel methods have been designed using grammars for compression and PPM for different language texts. This objective was achieved in chapter 4.

• *Develop further improvements for the new methods for different text languages.*

A new technique was developed with further improvements using a two-pass scheme. This objective was achieved in chapter 5.

• *Evaluate these methods by comparing them with well-known compression methods.*

The evaluation of these methods (GRB-PPM, GRT-PPM, GRBB-PPM, GRTT-PPM) showed that they significantly outperform well-known compression schemes such as PPMD, BS-PPM, Gzip, Bzip2 and the Sequitur algorithm. This objective was achieved in chapters 4 and 5.

• *Develop improved word-based compression models for PPM by parsing the text to construct a word-based CFG which is then compressed using PPM.*

A new grammar-based method has been developed using words; it first constructs a CFG for the text. This grammar is then encoded using PPM to produce an excellent compression rate. This objective was achieved in chapter 6.

• *Apply one of these new methods to the problem of the classification of text.*

Finally, our GRB-PPM method preforms well when applied to genre classification. The results confirm that better compression leads to better classification. This objective was achieved in chapter 7.

8.3 Review of Research Questions

The research questions of this dissertation that were laid out in section 1.3 are reviewed in this section.

The specific research questions from section 1.3 were as follows (along with a discussion of the experimental findings from this research that relate to these questions):

1. *What is the best grammar-based compression model for compressing various natural language texts?*

As shown in the experiments on the investigated models, the best compression models for different natural languages are GRBB-PPM and GRB-PPM character-based compression, followed by the GRTT-PPM and GRT-PPM compression methods. These methods significantly outperformed well-known compression schemes such as PPMD, BS-PPM, Gzip, Bzip2 and the Sequitur algorithm on different natural language texts and on the Calgary Corpus.

GRW-PPM outperformed other traditional word-based text compression methods especially for Arabic text and for other languages that use Arabic script in their writing systems, such as Persian. This research question was investigated in chapters 4, 5 and 6.

2. *Do grammar-based methods perform better than other common compression methods specifically for the Arabic language, as this a language that is not related to English and has been under-researched in the past in comparison to research into English text compression?*

Our new methods GRB-PPM, GRT-PPM, GRBB-PPM, GRTT-PPM and GRW-PPM have been compared with standard PPM, Gizp, Bzip2 and the Sequitur algorithm and the experimental results showed that our new methods performed better than the other. This research question was investigated in chapters 4, 5 and 6.

3. *Does better compression lead to better classification when we apply improved compression models?*

Our GRB-PPM achieved good results, especially for Arabic text, as investigated in chapter 7.

4. *Do the classification results match between languages for a parallel corpus to the problem of text classification?*

When we applied our methods to a parallel corpus for English and Arabic, we found that the results were similar across languages, as shown in chapter 7.

8.4 Future Work

The research presented in this dissertation has raised are a number of questions for further investigation as follows:

• GR-PPM requires three passes through the text determining the list of n-graphs that define the grammar in first pass prior to the compression phase; correcting the text using the grammar in the second pass and then encoding it using PPM in the third pass. However, we do not need to encode the grammar separately, making it possible to have the algorithm work in an online manner rather than offline. In this way the speed performance of our grammar-based compression methods can be improved.

• In the recursive grammar, we use multiple passes to repeatedly substitute commonly occurring sequences of n-graphs and non-terminal symbols as specified by their rules in the grammar in a second pass through the file. This is done during the second pre-processing phase prior to the compression phase in a stage that allows the text to be regenerated during the post-processing stage. Further improvements in compression may be possible using further passes during grammar-based preprocessing. However, improvements in processing speed also need to be investigated in order to make these schemes feasible.

• GR-PPM can also be applied to syllables in order to determine whether they are better than character-based or word-based methods. However, this will require investigating different methods for performing syllable segmentation, and it is not at this stage clear which methods would be best.

• The GRW-PPM uses both a CFG and PPM as the basis of a universal general- purpose compression method for text files such as English, Arabic, Persian and Welsh. Therefore, the GRW-PPM method can also be applied to other languages, such as Chinese.

• Compression of image and video files is an important research area. Our experiments with Calgary Corpus showed GR-PPM algorithms work well with pic file (image file), so we could try investigations applying our extended PPM to image and video files.

The idea of the recursive grammar originates from the insights of the experimental results from GR-PPM. The improvements for these models can be focused on the process of speed. The optimization of GR-PPM can also be applied to syllables to determine whether they are better than character-based or word-based methods. GRW-PPM can also be applied to other languages. Our experiments also show that the GR-PPM works well with the image file, and future work is required to investigate these features.

BIBLIOGRAPHY

Aaditya, J and Mandowara, J (2016). Text classification by combining text classifiers. *International Journal of Computer Application*, 6(2).

Abel, J. and Teahan, W. (2005). Universal text pre-processing for data compression. *IEEE Transactions on Computers*, 54(5): 497–507.

AleAhmad, A., Amiri, H., Darrudi, E., Rahgozar, M. (2009). Hamshahri: A standard Persian text collection. *Knowledge-Based System*, 22(5): 382–387.

Alghamdi, M.A., Alkhazi, I.S. and Teahan, W.J (2016). Arabic OCR evaluation tool. In *Proceedings of 7th International Conference on Computer Science and Information Technology (CSIT), IEEE, Amman*, 1–6.

Alkahtani, S. (2015). *Building and verifying parallel corpora between Arabic and English.* Ph.D. thesis, Bangor University.

Alkhazi, I. and Teahan, W. (2017). Classifying and segmenting classical and modern standard Arabic using minimum cross-entropy. *International Journal of Advanced Computer Science and Applications*, 8(4): 421–430.

Alsulaiti, L. and Atwell, E. (2006). The design of a corpus of contemporary Arabic. *International Journal of Corpus Linguistics*, 11(2):135–171.

Bell, T. C., Cleary, J. G., and Witten, I. H. (1990). Text compression. *Upper Saddle River, NJ: Prentice Hall*.

Benoit, G. (2013). Character encoding. Simmons College: 1–27.

Blelloch, E. (2001). Introduction to data compression. Pittsburgh, PA: Computer Science Department, Carnegie Mellon University.

Bloom, C. (1998). Solving the problems of context modeling. Informally published report: http://www.cbloom.com/papers.

Brown, P. F., Cocke, J., Pietra, S. D., Pietra, V. J. D., Jelinek, F., Lafferty, J. D., Mercer, R. L., and Roossin, P. S. (1990). A statistical approach to machine translation. computational linguistics. *Computational Linguistics*, 16(2): 79–85.

Brown, P. F., Pietra, V. J. D., Mercer, R. L., Pietra, S. A. D., and C.Lai, J. (1992). An estimate of an upper bound for the entropy of English. *Computational Linguistics*, 18(1): 31–40.

Chen, S. F. (1996). *Building probabilistic models for natural language*. Ph.D. thesis, Harvard University.

Chowdhury, G. (2003). Natural language processing. *Annual Review of Information Science and Technology*, 37: 51–89.

Church, K. W. and Gale, W. A. (1991). A comparison of the enhanced Good-Turing and deleted estimation methods for estimating probabilities of English bigrams. *Computer Speech and Language*, 5(5): 19–54.

Clackson, James (2007). Indo-European linguistics: An introduction. Cambridge: Cambridge University Press.

Cleary, J. G. and Teahan, W. J. (1997). Unbounded length contexts for PPM. *The Computer Journal*, 40(2/3): 1917–1921.

Cleary, J. G. and Witten, I. (1984). Data compression using adaptive coding and partial string matching. *Communications, IEEE Transactions*, 32(4): 396–402.

Coded Character Set (1986). 7-bit American standard code for information interchange. *ANSI X3.4*.

Davis, M. (2010). Unicode nearing 50% of the web. `URL:https://googleblog.blogspot.co.uk/2010/01/unicode-nearing-50-of-web.html`. [accessed 6 Aug 2017].

Dumais, S. Platt, J. Sahami, M. ,Heckerman, D. (1998). Inductive learning algorithms and representations for text categorization. In *Proceedings International Conference on Information and knowledge Management*, 148–155.

Elbeheri, G., Everatt, J., Reid, G., Al-Mannai, H. (2006). Dyslexia assessment in Arabic. *Journal of Research in Special Educational Needs*, 10: 143–152.

Elhaj, M. ,Kruschwitz, U. and Fox, C. (2010). Using mechanical turk to create a corpus of arabic summaries. In *Proceedings of the Seventh conference on International Language Resources and Evaluation*.

Ellis, N., Hicks, W., Morgan, M.,Laporte, N. (2001). Cronfa Electroneg o Gymraeg (CEG): A 1 million word lexical database and frequency count for Welsh: https://www.bangor.ac.uk/canolfanbedwyr/ceg.php.en, and [accessed 19 Feb 2016].

Encyclopædia Britannica. (2015). Arabic alphabet. `URL:https://www.britannica.com/topic/Arabic-alphabet`. [accessed 23 Aug 2017].

Fragoudis, D., Meretakis, D. and Likothanassis, S. (1990). Best terms: An efficient feature-selection algorithm for text categorization. *Knowledge and Information Systems*, 8(1): 16–33.

Francis, W. and Kucera, H. (1979). Brown corpus manual. Providence, RI: Brown University.

Frank, E., Chui, C., and Witten, I. H. (2000). Text categorization using compression models. In *Proceedings of the Conference on Data Compression*.

Good, I. J. (1953). The population frequencies of species and the estimation of population parameters. *Biometrika*, 40:237–264.

Hilbert, M. and Lopez, P. (2011). The world's technological capacity to store, communicate, and compute information. *Science*, 332(6025): 60–65.

Holes, C. (1995). *Modern Arabic: Structures, functions, and varieties*. London: Longman.

Howard, P. (1993). *The design and analysis of efficeient lossless data compression systems*. Ph.D thesis, Brown University.

Hunnisett, D and Teahan, W. (2004). Context-based methods for text categorisation. In *Proceedings of the 27th annual international ACM SIGIR conference on Research and development in information retrieval*.

Hutchins, J. (2005). The history of MT in a nutshell. URL:http://www.hutchinsweb.me.uk/Nutshell-2005.pdf. [accessed 23 May 2016].

IBM Corp (1996). Graphic character sets and code pages. URL:ftp://ftp.software.ibm.com/software/globalization/gcoc/attachments/CP01089.pdf. [accessed 2 Aug 2017].

Internet Live Stats (2016). Internet Users. URL:http://www.internetlivestats.com/internet-users/. [accessed 10 Feb 2018].

ISO/IEC JTC1/SC29/WG1 N339. (1996). Xerox Proposal for JBIG2 Coding.

Jacob, N., Somvanshi, P., and Tornekar, R. (2012). Comparative analysis of lossless text compression techniques. *International Journal of Computer Applications*, 56(3): 17–21.

Jelinek, F. (1990). Self-organized language modeling for speech recognition. In Readings in speech recognition, edited by A.Waibel and K.Lee. Burlington, MA: Morgan Kaufmann Publishers, pp. 450–456.

Jelinek, F., Bahl, L. R., and Mercer, R. L. (1975). Design of a linguistic statistical decoder for the recognition of continuous speech. *IEEE Transactions On Information Theory*, IT-21(3): 250–256.

Johansson, S. (1986). The tagged LOB corpus: User's manual. URL:http://www.hit.uib.no/icame/lobman/lob-cont.html. [accessed 2 Oct 2015].

Kernighan, M., Church, K., and Gale, W. (1990). A spelling correction program based on a noisy channel model. In *Proceedings of the Thirteenth International Conference on Computational Linguistics*, pp. 205–210.

Kherallah, M., Bouri, F., and Alimi, A. M. (2009). Toward an on-line Arabic handwriting recognition system based on visual encoding and genetic algorithm. In *Proceedings of Engineering Applications of Artificial Intelligence*, 22: 153–170.

Khmelev, D. and Teahan, W. (2003). A repetition based measure for verification of text collections and for text categorization. In *Proceedings of the 26th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 104–110.

Kieffer, J. and Yang, E. (2000). Grammar-based codes: A new class of universal lossless source codes. *IEEE Transactions on Information Theory*, 46(3): 737–754.

Laplace, P. (1814). *Essai philosophique sur les probabilites*. Paris: Mme. Ve. Courcier.

Larsson, N. J and A Moffat, A. (2000). Off-line dictionary-based compression. *Proceedings of the IEEE*, 88(11): 1722–1732.

Maamouri, M., Bies, A., Buckwalter, T., and Hubert, J. (2005). Arabic Treebank: Part 1 v. 3.0. (POS with full vocalization + syntactic analysis), Linguistic Data Consortium. https://catalog.ldc.upenn.edu/LDC2005T02. [accessed 19 Feb 2017].

Manning, C. and Schutze, H. (1999). *Foundations of Statistical Natural Language Processing*, Cambridge, MA: The MIT Press.

Markov, A. (1913). An example of statistical investigation in the text of 'Eugene Onegin' illustrating coupling of 'tests' in chains. *Academy of Sciences, St. Petersburg*, 7(6): 153–162.

Marton, Y., Wu, N. and Hellerstein, L. (2005). On compression-based text classification. In Advances in Information Retrieval: *27th European Conference on IR Research, ECIR 2005, Santiago de Compostela, Spain, March 21-23, 2005, Proceedings*, pp. 300–314.

McCallum, A. and Nigam, K. (1998). A comparison of event models for naive Bayes text classification. In *AAAI-98 Workshop on Learning for Text Categorization,* pp. 41–48.

McEnery, A. and Xiao, Z. (2004). The Lancaster Corpus of Mandarin Chinese: A corpus for monolingual and contrastive language study. *Religion*, 17: 3–4

McEnery, T. and Wilson, A. (2004). Corpus Linguistics. 2nd ed. Edinburgh: University of Edinburgh Press.

Miniwatts Marketing (2017). Internet world stats. `URL:http://www.internetworldstats.com/stats19.html`. [accessed 22 July 2017].

Mitchell, T. (1997). *Machine learning, 1st Ed*. McGraw Hill.

Moffat, A. (1989). Word-based text compression. *Software Practice and Experience*, 19(2): 185–198.

Moffat, A. (1990). Implementing the PPM data compression scheme. *IEEE Transactions on Communications*, 38(11): 1917–1921.

Moffat, A., Neal, R., and Witten, I. (1998). Arithmetic coding revisited. *ACM Transactions on Information Systems*, 16(3): 256–294.

Mohri, M., Rostamizadeh, A., and Talwalkar, A. (2012). Foundations of Machine Learning. Cambridge, MA: The MIT Press.

Najeeb M. M., Abdelkader A. A. , and Al-Zghoul, M.B. (2014). Arabic natural language processing laboratory serving Islamic sciences. *International Journal of Advanced Computer Science Applications,* 5(3): 114–117.

Nevill-Manning, C. and Witten, I. (1997). Identifying hierarchical structure in sequences: A linear-time algorithm. *Journal of Artificial Intelligence Research*, 7: 67–82.

Nevill-Manning, C. and Witten, I. (1997). Compression and explanation using hierarchical grammars. *The Computer Journal*, 40(2/3): 103–116.

Pang, B., Lee, L., and Vaithyanathan, S. (2002). Thumbs up?: Sentiment classification using machine learning techniques. *Proceedings of the ACL-02 Conference on Empirical Methods in Natural Language Processing.* Volume 10, pp. 79–86.

Ponte, J. M. and Croft, W. B. (1998). A language modeling approach to information retrieval. *Proceedings of the 21st Annual International ACM SIGIR Conference*, pp. 275–281.

Rasheed, Z. T (2008). Arabic is the tie that binds. Al-Jazeera. `URL:http://www.aljazeera.com/focus/arabunity/2008/01/200852518532541888 2.html`. [accessed 20 Aug 2017].

Rissanen, J. and Langdon, J. (2012). Universal modeling and coding. *IEEE Transactions on Information Theory*, 27(1): 12–23.

Ryding, K.C. (2005). A Reference Grammar of Modern Standard Arabic. Cambridge: Cambridge University Press.

Sayood, K. (2017). *Introduction to Data Compression. 5th Ed.* Morgan Kaufmann: San Francisco, CA.

Senthil, S. and Robert, L. (2011). Text compression algorithms: A comparative study. *ICTACT Journal of Communication Technology*, 2(2): 444–451.

Shannon, C. E. (1948). A mathematical theory of communication. *Bell System Technical Journal*, 27: 379–423 and 623–656.

Shkarin, D. (2002). PPM: One step to practicality. In *Proceedings of the Data Compression Conference, DCC*, pp. 202–211.

Simons, Gary F. and Charles D. Fennig (2017). Ethnologue: Languages of the World, 20th ed. Dallas, Texas. `URL:http://www.ethnologue.com`. [accessed 19 Aug 2017].

Teahan, W. J. (1998). *Modelling English Text.* Waikato, New Zealand: University of Waikato.

Teahan, W. J. (2000). Text classification and segmentation using minimum cross-entropy. In *Content-Based Multimedia Information Access.* Volume 2, pp. 943–961.

Teahan, W. J. (in press). *Natural Language Engineering*.

Teahan, W. and Alhawiti, K. (2013). Design compilation and preliminary statistics of compression corpus of written Arabic. Bangor University Technical Report.

Teahan, W. and Alhawiti, K. (2014). A compression-based method for ranking n-gram differences between texts and corpora evaluation. In *7th Saudi students scientific conference 2014*.

Teahan, W. and Alhawiti, K. (2015). Preprocessing for PPM: Compressing UTF-8 encoded natural language text. *International Journal of Computer Science & Information Technology*, 7(2): 41–51.

Teahan, W. and Harper, D. (2001). Using compression-based language models for text categorization. *Language Modeling for Information Retrieval. Springer*, pp. 141–165.

Thomas, D (2001). An empirical study of stream-based techniques for text categorization. Bangor University.

Trezise, W (2002). Collection of Arabic Calligraphy. URL:http://library.princeton.edu/libraries/firestone/rbsc/aids/islamic/trezise.html. [accessed 26 July 2017].

Turing, A. (1950). Computing machinery and intelligence. *Mind*, LIX(236): 433–461.

Unicode Consortium (1991). The Unicode standard: Worldwide character encoding. 1st edition. Boston, MA: Addison-Wesley Longman Publishing Co., Inc.

United Nations (2017). *Official Languages*. URL:https://www.un.org/en/sections/about-un/official-languages. [accessed 26 July 2017].

Versteegh, K. (2001). *The Arabic Language*. Edinburgh: Edinburgh University Press.

Witten, I. H. and Bell, T. C. (1990). Source models for natural language text. *International Journal of Man-Machine Studies*, 32(5): 545–579.

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

Witten, I. H., Neal, R. M., and Cleary, J. G. (1987). Arithmetic coding for data compression. *Communications of the ACM*, 30(6): 520–540.

Witten, I. H. and Frank, E. (2005). *Data Mining: Practical machine learning tools and techniques, 2nd Ed*. Morgan Kaufmann, San Francisco, CA.

Wu, P. and Teahan, W. J. (2008). A new PPM variant for Chinese text compression. *Natural Language Engineering*, 14(3): 417–430.

Ye, Y. and Cosman, P. (2003). Fast and memory efficient text image compression with JBIG2. *IEEE Transactions on Image Processing,* 12(8): 944–956.

Yoon, H. and Hirvela, A. (2004). ESL student attitudes toward corpus use in L2 writing. *Journal Of Second Language Writing*, 13(4): 257–283.

Yousefi-Azar, M. and Hamey, L. (2016). Text summarization using unsupervised deep learning. *Expert Systems with Applications*, 68: 93–105.

Zaidan, F. and Callison-Burch, C. (2014). Arabic dialect identification. *Computational Linguistics*, 40(1):171–202.

Ziv, J. and Lempel, A. (1977). A universal algorithm for sequential data compression. *IEEE Transactions on information theory*, 23(3):337–343.