PREPROCESSING FOR PPM: COMPRESSING UTF-8 ENCODED NATURAL LANGUAGE TEXT

William J. Teahan[1] and Khaled M. Alhawiti[2]

[1] School of Computer Science, University of Wales, Bangor, UK.
[2]School of Computes and Information Technology, University of Tabuk, KSA.

ABSTRACT

In this paper, several new universal preprocessing techniques are described to improve Prediction by Partial Matching (PPM) compression of UTF-8 encoded natural language text. These methods essentially adjust the alphabet in some manner (for example, by expanding or reducing it) prior to the compression algorithm then being applied to the amended text. Firstly, a simple bigraphs (two-byte) substitution technique is described that leads to significant improvement in compression for many languages when they are encoded by the Unicode scheme (25% for Arabic text, 14% for Armenian, 9% for Persian, 15% for Russian, 1% for Chinese text, and over 5% for both English and Welsh text). Secondly, a new preprocessing technique that outputs separate vocabulary and symbols streams – that are subsequently encoded separately – is also investigated. This also leads to significant improvement in compression for many languages (24% for Arabic text, 30% for Armenian, 32% for Persian and 35% for Russian). Finally, novel preprocessing and postprocessing techniques for lossy and lossless text compression of Arabic text are described for dotted and non-dotted forms of the language.

KEYWORDS

Preprocessing, PPM, UTF-8, Encoding.

1. BACKGROUND

1.1. Prediction by Partial Matching (PPM)

One of the most powerful text compression techniques is Prediction by Partial Match (PPM), which was first introduced by Cleary and Witten [1]. A series of improvements have been applied to the original PPM algorithm, such as the PPMC version by Moffat [2] and PPM* by Cleary & Teahan [3]. The PPM text compression algorithm applies a statistical approach; it simply uses the set of previous symbols to predict the upcoming symbol in the stream. Variants of the PPM algorithm (such as PPMC and PPMD) are distinguished by the escape mechanism used to back-off to lower order models when new symbols are encountered in the context. PPM has also been applied successfully too many natural language processing (NLP) applications such as cryptology, language identification, and text correction [4], [5].

1.2. Universal text preprocessing for data compression

Abel and Teahan [6] presented several universal text preprocessing techniques that they applied prior to the application of various standard text compression algorithms. They found that in many cases the compression performance was significantly improved by applying the text processing techniques. In order to recover the original file during decoding, the decompression algorithm was applied first, and then postprocessing was performed that reversed the effect of the pre-processing stage.

One method first described in [4] that they found effective for English text is to substitute bigraphs with a single further unique symbol (essentially expanding the alphabet). This bigraph substitution method (described in more detail in section 2), however, was only applied to English ASCII text and its effectiveness for other languages, and other encoding schemes (such as UTF-8) has not been explored previously.

1.3. UTF-8 encoding

UTF-8 has come to be the most popular character encoding method used on the Web and in applications [7]. Figure 1 shows the percentage of websites using various character encodings and clearly shows that UFT-8 is the most popular today.

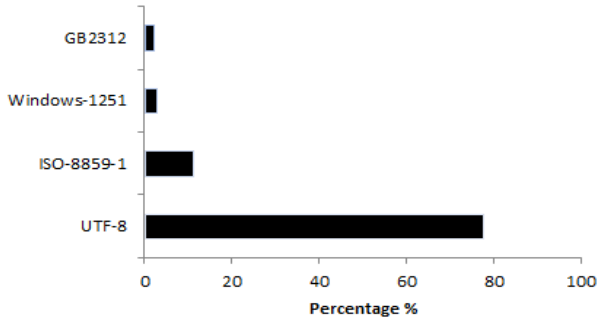

Figure. 1 Percentage of websites that use various character encodings.

UTF-8 is a multi-byte variable width encoding scheme. It uses the ASCII code (0-127) to represent a Latin character using a single byte, and then uses up to 4-bytes for other alphabets although most alphabets require only two bytes per character. The importance of Unicode derives from its compatibility with ASCII, thus it encodes English letters with single byte characters. Unicode also derives importance from its compactness and efficiency in most scripts that require more than one byte to encode, such as Arabic, Japanese and Chinese.

Surprisingly, considering UTF-8's popularity as an encoding scheme, there have been very few publications that have investigated the problem of finding the most effective compression of UTF-8 text. Fenwick and Brierly [8] concluded that for UTF-8 text "accepted 'good' compressors such as finite-context PPM do not necessarily work well".

This paper presents universal UTF-8 preprocessing algorithms to be performed prior to the use of the PPM compression scheme. The PPM algorithm itself is used unchanged (as a black- box component), and only parameters such as the escape mechanism and the order of the model have been adjusted. The impact of the text preprocessing algorithms are examined using different file sizes and text genres from the Bangor Arabic Compression Corpus (BACC) [9] of Arabic text and other corpora such as the Hamshahri corpus of Persian text [10], the HC corpus of Armenian text [11], the HC corpus of Russian text [11], the LCMC corpus of Chinese text [12], the CEG corpus of Welsh text [13], and the Brown [14] and LOB [15] corpora of American and British English text respectively.

Table 1. Bigraphs and their frequency for five different corpora.

Bigraph	Arabic CCA	Vietnamese HC	Armenian HC	English Brown	English LOB
1	ال	ng	ու	th	th
2	لم	nh	ան	he	he
3	ية	th	եր	in	in
4	لا	ch	ար	er	er
5	من	an	կա	an	an
6	في	hi	այ	re	re
7	وا	ha	հն	on	on
8	ما	ho	ան	en	en
9	ات	tr	ակ	at	nd
10	ان	là	որ	or	at
11	ها	on	րա	nd	es
12	لت	ay	ում	es	is
13	ين	én	հա	is	or
14	عل	kh	նա	ti	ar
15	لى	hư	Մա	te	ed
16	را	đi	եկ	ed	to
17	لي	em	են	ar	of
18	ار	ai	ան	st	ng
19	نا	có	րո	it	te
20	لأ	mà	ում	of	it
Total	474512	249584	844800	581117	562085
Percentage	16.61	9.79%	21.46	9.69	9.56

2. BIGRAPH SUBSTITUTION FOR PPM (BS-PPM)

2.1. Bigraphs and languages

By its nature, languages contain words that have many repeated bigraph characters, with two characters often appearing together in the same order such as “th” and “en” in English and “ ال” and “من” in Arabic text. Usually, each language has common bigraphs that represent a significant percentage of the text. For example, examining the most frequent 20 bigraphs over five different languages using 500,000 words from various corpora (CCA [16], HC-Vietnamese [11], HC-Armenian, Brown and LOB corpora) produces some interesting results as showed in Table 1.
The top 20 bigraphs take up almost 10% of the Vietnamese and English texts. For Arabic and Armenian texts, the top ranked bigraphs take up significantly more at over 16% and 21% respectively. Clearly, dealing with bigraphs for these texts is an important consideration.

Bigraphs can be employed to enhance the compression performance over standard PPM by using preprocessing and postprocessing techniques before and after the compression and decompression, as shown in Figure 2, as follows. (BS-PPM.).

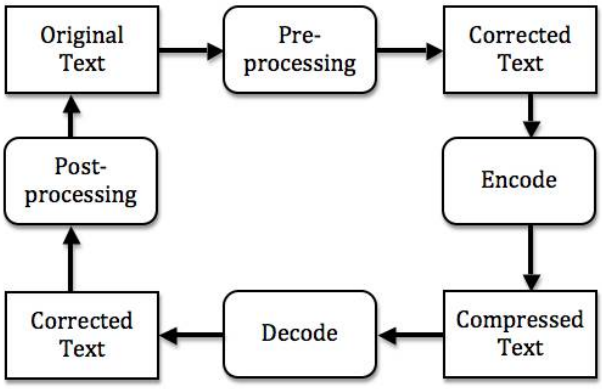

Figure.2. The use of preprocessing and postprocessing for compression.

2.2. Preprocessing and postprocessing

The preprocessing technique involves sequentially processing the text replacing the most frequent bigraphs (i.e. sequence of two bytes) in the order that they appear, with unique single characters for each. For most natural language texts, we have found that replacing the top 100 bigraphs works best with the alphabet effectively expanding by 100 further characters as a result.

The postprocessing operation simply performs the reverse mapping, by replacing the new unique characters with the original equivalent two byte bigraphs.

2.3. Experimental results for BS-PPM

Table 2 shows the results of BS-PPM (using order 4 PPMD) compared with other well-known compression schemes such as ABC2.4 [17] and bzip2 [18] using UTF-8 encoded files from the BACC for Arabic text, HC-Russian for Russian text, HC-Armenian for Armenian text, Hamshahri corpus for Persian text, LCMC for Chinese text, CEG for Welsh text, and the Brown and LOB corpora for American and British English text respectively.

Clearly, BS-PPM works very well on UTF-8 encoded texts in many languages, such as Arabic, Persian, Armenian, Russian, Welsh, Chinese, and English since it records significant improvements over other methods in terms of compression rate (in bits per character or bpc). In all cases for these languages, BS-PPM significantly outperforms the other compression methods, as it has the best results shown in bold font and also significantly outperforms standard PPM itself. For example, for Arabic text, BS-PPM shows a 25.1% improvement over standard PPM. For Armenian text, BS-PPM shows a 14.6% improvement over ABC2.4, 25% over Bzip2 and 30.8% over standard PPM.

For Persian text, BS-PPM showed an 8.7% improvement over ABC2.4, 17.7% over Bzip2 and 28% over standard PPM. For Russian text, BS-PPM showed a 14.5% improvement over ABC2.4, 26.3% over Bzip2 and 35.3% over standard PPM. Also, there was a significant improvement in compression rates for both American and British English with BS-PPM recording a 14.6% improvement over Bzip2 for the Brown corpus and 14.4% for the LOB corpus. This rep- resents an 8.3% improvement over ABC2.4 for the Brown corpus and 8.4% for the LOB corpus, 33.5% over gzip for Brown and 33.8% for LOB and 5.8% over standard PPM for Brown and 5.9% for LOB.

In order to examine the impact of bigraph coding using different order PPM models, Table 3 shows the results of both PPM and BS-PPM for orders 1 through 7. The results for order 4 from Table 2 are repeated in the table for clarity. The best result for each row is shown in bold font.
For Arabic text, the best compression rate occurred when using order 7; for Armenian, Persian and Chinese text, the best compression rate occurred when using order 5; for Russian text the best compression rate was for order 6; and for Welsh and English text, order 4 produced the best compression rate for both American and British texts and Welsh text as well.

Table 2. BS-PPM vs. other methods applied to various language texts.

Language	**Corpus Text File**	**Size** (bytes)	**Bzip2** (bpc)	**ABC2.4** (bpc)	**Gzip** (bpc)	**PPM Order4** (bpc)	**BS-PPM** Order4 (bpc)
Arabic	BACC	50411735	1.45	1.43	2.14	1.79	**1.34**
Armenian	HC	36700160	1.56	1.37	2.39	1.69	**1.17**
Chinese	LCMC	4555457	2.65	2.57	3.47	2.49	**2.46**
English	Brown	5998528	2.46	2.29	3.16	2.23	**2.10**
English	LOB	5877271	2.43	2.27	3.14	2.21	**2.08**
Persian	Hamshahri	41567603	1.53	1.38	2.22	1.75	**1.26**
Russian	HC	52428800	1.52	1.31	2.45	1.73	**1.12**
Welsh	CEG	6169422	2.55	2.34	3.19	2.30	**2.14**
Average			2.02	1.86	2.77	2.02	**1.70**

Table 3. Compression results over eight corpora and for different orders of PPM and BS-PPM.

Fil	**Language**	**PPM** order 1	**BS-PPM** order 1	**PPM** order 2	**BS-PPM** order 2	**PPM** order 3	**BS-PPM** order 3	**PPM** order 4	**BS-PPM** order 4	**PPM** order 5	**BS-PPM** order 5	**PPM** order 6	**BS-PPM** order 6	**PPM** order 7	**BS-PPM** order 7
BACC	Arabic	2.42	2.03	2.17	1.68	2.04	1.45	1.79	1.34	1.66	1.30	1.51	1.28	1.44	**1.27**
H	Armenian	2.43	2.06	2.02	1.49	1.90	1.26	1.69	1.17	1.61	**1.15**	1.42	1.16	1.36	1.18
LCMC	Chinese	4.03	3.72	3.01	2.86	2.66	2.58	2.49	2.46	2.46	**2.44**	2.47	2.45	2.49	2.46
English	Brown	3.67	3.13	3.02	2.34	2.51	2.12	2.23	**2.10**	2.16	2.13	2.17	2.17	2.22	2.21
English	LOB	3.66	3.11	2.99	2.32	2.48	2.1	2.21	**2.08**	2.14	2.11	2.15	2.15	2.19	2.18
Hamshahri	Persian	2.29	1.92	2.09	1.53	1.96	1.3	1.75	1.26	1.60	**1.17**	1.42	1.17	1.33	1.18
H	Russian	2.47	1.95	2.08	1.48	1.97	1.23	1.73	1.12	1.63	1.09	1.41	**1.08**	1.33	1.09
CE	Welsh	3.66	3.2	3.07	2.44	2.60	2.18	2.30	**2.14**	2.20	2.17	2.21	2.21	2.25	2.24

These results show that an impressive improvement in compression performance is possible for natural language texts using the bigraph substitution method. The next section describes a new method that also uses preprocessing techniques and also yields impressive improvements in compression performance.

3. CHARACTER SUBSTITUTION FOR PPM (CS-PPM)

UTF-8 is a variable-length, byte-based encoding and, therefore, a bigraph-based substitution method as described in the previous section may not be best suited for languages where two-byte encoding is not the norm. This is illustrated by the results for Chinese texts, which in UTF-8 encoding often requires 3 or 4 bytes to encode each character. This suggests a character, rather than a bigraph, substitution method might also yield impressive results.

3.1. Preprocessing and postprocessing

Unlike the bigraph substitution method just described, which replaces the top 100 bigraphs in the text during the preprocessing stage, our character substitution method (called CS-PPM) substitutes all the UTF-8 multi-byte or single-byte character sequences in the original text. Two output files are produced as a result of the preprocessing; one contains a stream of UTF-8

characters (called the 'vocabulary' stream) and the other contains a list of symbol numbers (called the 'symbols' stream).

Whenever a new character is encountered in the original text, this character is assigned a symbol number equal to the current symbols count, which is then incremented. The symbols count is initialised to 1 for the first character. When that same character is encountered later on in the text, the symbol number assigned to it is written out to the symbols output file. At the same time, the UTF-8 character-byte sequence for new characters is written out to a separate vocabulary output file.

Both the vocabulary output file and the symbols output file need to be compressed during the encoding stage. Therefore, two files also need to be decoded separately, with the vocabulary output file requiring decoding first in order to define the reverse mapping between symbols and UTF-8 characters during the post- processing stage.

We found the following method works well at encoding the two files. For encoding the vocabulary output file, standard order 1 byte- based PPM is quite effective. For the symbols output file, where the symbol numbers can get quite large for some languages, a similar technique to word-based PPM [4] works well with the alphabet size being unbounded. Another finding is that an order 4 model works best among the experimented languages.

3.2. Experimental Results for CS-PPM

Table 4 lists results for PPM and CS-PPM on Arabic text using files from the BACC corpus. In all cases, there is a significant improvement in performance for CS-PPM over PPM, as shown in column 4 of the table. The improvement is noticeable most for the largest files in the corpus with almost 25% improvement for the file bookcollection1. Column 5 provides the percentage cost of encoding the symbols output file and the last column provides the percentage cost of encoding the vocabulary output file. The results show that the symbols output file consistently takes up 75 to 80% of the overall encoding cost.

Table 5 lists results for PPM and CS-PPM on various language texts, with results from bookcollection1 from the BACC corpus repeated on the first row for comparison. The languages for which CS-PPM is most effective are Arabic (23.3% improvement), Armenian (30.4%), Persian (31.5%) and Russian (35.2%).

3.3. Character Substitution of Arabic for PPM (CSA-PPM)

In this section, we describe a third method that is tailored specifically for Arabic text. We call the method Character Substitution of Arabic for PPM (CSA-PPM).

Unlike CS-PPM which produces two output files (symbol and vocabulary stream), one out- put file is produced as a result of the pre-processing method of CSA-PPM. Since we can assume in advance that we will be encoding Arabic text, we can directly substitute the characters with the equivalent number of the UTF-8 scheme to eliminate the need for a vocabulary output file altogether, which will decrease the size of the output compressed file, as shown in Table 6.
The savings from not having to encode the characters that make up the alphabet leads to significant improvements for the small sized files. However, for large files, the improvements are minimal when compared to the overall compressed file size.

4. DOTTED (LOSSLESS) AND NON-DOTTED (LOSSY) COMPRESSION OF ARABIC TEXT

Data compression is divided into two main categories, lossless and lossy compression. On the other hand, text compression and, more specifically, the problem of natural language text compression, is usually considered to be lossless compression since changing the text in natural language will usually alter the meaning. Despite this, there have been a few papers that have considered the problem, from the satirical paper [19] to the more recent paper [20].

The Arabic language comprises 28 letters, fifteen of them that are dotted and thirteen of them non-dotted, as shown in Table 7.

Dots above and below the letter give it a different pronunciation, such as one dot below " ب " which is equivalent to a B in English and two dots above "ت" which is equivalent to the letter T in English, and so on. In old Arabic texts, all letters were not originally dotted but, despite some ambiguity as a result, this could still be easily identified by native Arabic readers familiar with the Arabic language. Figure 4 below shows a sample ancient Arabic script that uses the non-dotted form [21].

Due to the ambiguity of identifying the correct letter by non-native Arabic language learners, since 820 A.D, these letters have become dotted [22]. We exploited this historical feature of the language to improve the compression rate of Arabic in some cases by preprocessing the Arabic text prior to compression with PPM and recovering it during the postprocessing stage after decompression, as explained in more detail below.

Table 4. PPM vs. CS-PPM for the BACC.

File	**Size** (bytes)	**PPM** Order4 (bpc)	**CS-PPM** (bpc)	**Improv.** (%)	**Percentage** of encoding symbols (%)	**Percentage** of encoding vocabulary (%)
economic	15877	2.03	**1.90**	6.4	76.9	23.1
education	26892	2.06	**1.96**	4.9	76.0	24.0
sports	31609	1.95	**1.77**	9.2	78.2	21.8
culture	34683	2.03	**1.88**	7.4	76.8	23.2
artandmusic	42453	2.08	**1.93**	7.2	76.1	23.9
political	47403	1.99	**1.74**	11.7	78.4	21.6
articles	103625	1.94	**1.76**	9.3	78.2	21.8
press	547963	1.83	**1.57**	13.7	80.4	19.6
Novel1	857995	1.87	**1.63**	12.8	79.6	20.4
Novel2	908364	1.86	**1.62**	12.9	79.7	20.3
Novel3	1022265	1.86	**1.61**	13.4	80.0	20.1
bookcollection1	50411735	1.81	**1.3**	24.6	82.7	17.3
bookcollection2	199150079	1.78	**1.4**	23.3	83.2	16.8
Average		1.94	**1.7**	11.4	78.6	21.4

Table 5. Results for PPM and CS-PPM on various language texts.

Language	Corpus text file	Size (bytes)	PPM (bpc)	CS-PPM (bpc)	Improve. (%)	Percentage of encoding symbols (%)	Percentage of encoding Vocabulary (%)
Arabic	bookcollection1	50411735	1.80	**1.38**	23.3	82.7	17.3
Armenian	HC	36700160	1.69	**1.18**	30.4	85.3	14.7
Chinese	LCMC	4555457	2.49	**2.37**	4.9	70.6	29.4
English	Brown	5998528	2.23	**2.15**	3.5	73.1	26.9
English	LOB	5877271	2.21	**2.13**	3.6	73.4	26.6
Persian	Hamshahri	41567603	1.75	**1.20**	31.5	85.0	15.0
Russian	HC	52428800	1.73	**1.12**	35.2	86.0	14.0
Welsh	CEG	6169422	2.3	**2.20**	4.5	72.6	27.5
Average			2.03	**1.72**	17.3	78.6	21.4

4.1. Preprocessing Arabic text for lossy non-dotted compression

During the first stage, letters such as"ب ت ث ن ي" the dots to generate the lossy (non-dotted) are normalized by removing version of the original text before it is compressed using PPM. For example, "ي"becomes "ى" and letters such as "ج خ" are normalized to be "ج", and letters such as "ذ ز ش ض ظ غ ف ق" are normalized to be "د ر س ص ط ع ف".

4.2. Encoding and Decoding

During the encoding stage, the pre-processed (lossy) text is compressed using PPM. During the decoding stage, the compressed text is decoded using PPM to produce the lossy version of the original text.

Table 6. CS-PPM vs. CSA-PPM.

File name	Size (Bytes)	CSA-PPM Compressed output (Bytes)	CS-PPM Compressed output (Bytes)	Difference (Bytes)
economic	15877	**3629**	3681	-52
education	26892	**6427**	6516	-89
sport	31609	**6856**	6934	-78
culture	34683	**8009**	8097	-88
artandmusic	42453	**10148**	10246	-98
political	47403	**10144**	10238	-94
articles	103625	**22683**	22813	-130
press	547963	**108083**	108339	-256
novel1	857995	**176215**	176444	-229
novel2	908364	**186302**	186505	-203
novel3	1022265	**206690**	206902	-212
shortstories	1041952	**201613**	201859	-246
literature	19187425	**3312444**	3312864	-420
history	30714551	**4403343**	4403730	-387
bookcollection1	50411735	**9370563**	9371244	-681
Bookcollection2	199150079	**32249015**	32249566	-551

Table 7.Arabic letters.

Dotted	Non-Dotted
ب ت ث ج خ ذ ز ش ض ظ غ ف ق ن ي	أ ح د ر س ط ع ك ل م ه و ى

Figure. 3.Sample of Arabic script prior to 820 A.D. (UmAlqura, 2014)

4.3. Recovering the Lossless Version of the Text

In this stage, the lossy text is automatically corrected using a Viterbi-based algorithm [23] using PPM character models [5] in order to try to recover the original text. This is done by finding the most probable corrected sequence when a match occurs in the text from the list of corrections shown in Table 8. The most probable sequence is selected which is the sequence that has the best encoding according to a PPM character-based language model trained on text from the BACC corpus as it is a large and representative corpus of contemporary Arabic language.

The Viterbi-based correction method will make mistakes, so these have to be encoded separately to inform the decoder which of the dotted forms (zero, one or two) each character should take. In our implementation, we directly encode the file position of the character that has been incorrectly dotted, which requires $\lceil \log 2\ N \rceil$ bits, where N is the number of characters in the file. One further bit is needed to encode the number of dots, either one or two dots, with the default form not requiring correction being zero dots.

4.4. Experimental Results

To examine our new method, we conducted experiments using the CCA and the EASC [24]. This is done by corpora while training the PPM-based corrector on the BACC.

As anticipated, lossy compression showed significant improvement over dotted compression by over 7% for EASC and over 5% for CCA, respectively. When the lossy form of the text was corrected using the Viterbi-based correction method in order to recover the original non-dotted text, a small number of errors were discovered. These errors made up 0.66% of the EASC and 0.43% for the CCA. When the cost of encoding these errors was added in (see the last column in Table 9), the lossless scheme still resulted in a significant improvement in compression (1.78 bpc compared to 1.86 bpc for the EASC and 1.74 bpc compared to 1.83 bpc for the CCA).

In order to investigate this result further, 10-fold cross-validation was performed on the BACC with the results shown in Table 9. The corpus was split into 10 equal parts (splits S1 through S10 as in the first column of the table). Training of the Viterbi-based corrector was then done on 9/10 of the corpus and testing on the other 1/10, and this was repeated 10 times using each split for testing.

The non-dotted compression shows an improvement of 2 to 5% over the dotted texts. However, a significant number of errors were made by the correction software as shown in column six of the table and this resulted in a lossless compression result (as shown in the last column) worse than the dotted compression result, shown in column three. The worse result is probably due to the varied nature of the BACC with files covering a variety of subjects, such as economics, education, sport, and culture, for example.

Table 8. List of corrections.

List of corrections		
ب → ب	ت → ب	ث → ب
ن → ب	ي → ب	ج → ح
خ → ح	ح → ح	ذ → د
د → د	ز → ر	ر → ر
ش → س	س → س	ض → ص
ص → ص	ظ → ط	ط → ط
غ → ع	ع → ع	ف → ق
ق → ق	ة → ه	ه → ه

Table 9. Dotted PPM vs. Non-Dotted PPM.

File	Size (Bytes)	PPM of dotted text (bpc)	PPM of non-dotted text (bpc)	Improve. %	Number of error	Error %	PPM of non-dotted text with dots corrected (bpc)
EASC	630321	1.86	**1.72**	7.53	2301	0.66	1.78
CCA	6265790	1.83	**1.73**	5.46	15155	0.43	1.74

Table.10. Ten-fold cross-validation.

BACC Split	Size (bytes)	PPM of dotted text (bpc)	PPM of non-dotted text (bpc)	Improve %	Number of errors	Error %	PPM of non-dotted text with dots corrected (bpc)
S1	31188787	1.76	1.69	3.98	129662	0.74	1.8
S2	31351880	1.70	1.65	2.94	101282	0.58	1.74
S3	31394336	1.71	1.65	3.51	107259	0.61	1.74
S4	31249486	1.82	1.74	4.40	229401	1.31	1.94
S5	31369255	1.69	1.64	2.96	101334	0.58	1.73
S6	31253465	1.71	1.65	3.51	95563	0.54	1.73
S7	30931867	1.77	1.71	3.39	171525	0.98	1.86
S8	31045940	1.64	1.6	2.43	103952	0.59	1.70
S9	31394039	1.75	1.67	4.57	138527	0.79	1.79
S10	31434684	1.75	1.67	4.57	155972	0.89	1.81

5. CONCLUSIONS

The PPM compression performance on UTF-8 encoded natural language text can be significantly improved by applying preprocessing and postprocessing techniques that rely on adjusting the alphabet (for example, by expanding or reducing the number of symbols). The BS-PPM, CS-PPM, CSA-PPM and Non-Dotted PPM techniques described in this paper are all examples of these adjustments with the first three working by expanding the alphabet and the last one working by reducing the alphabet.

The results show that serious consideration should be made not only to further investigation of text pre/postprocessing techniques, but also into mechanisms for incorporating their effect directly into the compression algorithm itself (therefore obviating the need for performing the pre/postprocessing separately). Improvements in performance of over 35% using straightforward substitution techniques on UTF-8 text indicate that significant gains can still be made for the well-researched PPM algorithm.

REFERENCES

[1] J.G.Cleary and I.Witten, "Data compression using adaptive coding and partial string matching," Communications, IEEE Transactions on, vol.32, no.4, pp.396–402, 1984.

[2] A.Moffat, "Implementing the PPM data compression scheme," IEEE Transactions on Communications, vol.38, no.11, pp.1917–1921, 1990.

[3] J.G.Cleary and W.J.Teahan, "Unbounded length contexts for PPM," The Computer Journal, vol.40, no.2 and 3, pp. 67–75, 1997.

[4] W.J.Teahan, "Modelling English text," Ph.D. dissertation, University of Waikato, 1998.

[5] W.J.Teahan, S.Inglis, J. G. Cleary, and G. Holmes, "Correcting English text using ppm models," in Proceedings of the Data Compression Conference, 1998. DCC'98. IEEE, 1998, pp. 289–298.

[6] J.Abel and W. J.Teahan, "Universal text preprocessing for data compression," IEEE Transactions on Computers, vol. 54, no. 5, pp. 497–507, 2005.

[7] E.A.Jari, "Efficiency lossless data techniques for Arabic text compression," International Journal of Computer Science & Information Technology (IJCSIT), vol.6, No 5, October 2014.

[8] BuiltWith, "UTF-8 usage statistics," 2012. [Online].Available: http://trends.builtwith.com/encoding/UTF-8.

[9] P.M.Fenwick and S.Brierley, "Compression of unicode files." in Data Compression Conference, 1998, p. 547.
[10] W.J.Teahan and K.M.Alhawiti, "Designcompilation and preliminary statistics of compression corpus of written Arabic," Bangor University, Tech. Rep., 2013. [Online]. Available: http://pages.bangor.ac.uk/ eepe04/index.html.
[11] A.AleAhmad, H.Amiri, E.Darrudi, M. Rahgozar, and F. Oroumchian, "Hamshahri: A standard Persian text collection," Knowledge-Based Systems, vol.22, no.5, pp.382–387, 2009.
[12] H.Christensen, "HC Corpora," 3013. [Online]. Available: http://www.corpora.heliohost.org/download.html
[13] A.M. McEnery and Z. Xiao, "The Lancaster corpus of Mandarin Chinese: A corpus for monolingual and contrastive language study," Religion, vol. 17, pp. 3–4, 2004.
[14] N.Ellis, C. O'Dochartaigh, W. Hicks, M. Morgan, and N. Laporte, "Cronfa electroneg o gymraeg (ceg): A 1 million word lexical database and frequency count for Welsh," 2001.
[15] W.N. Francis and H. Kucera, "Brown corpus manual," Brown University Department of Linguistics, 1979.
[16] S.Johansson, "The LOB corpus of British English texts: presentation and comments," ALLC journal, vol. 1, no. 1, pp. 25–36, 1980.
[17] L.Al-Sulaiti and E. Atwell, "The design of a corpus of contemporary Arabic." International Journal of Corpus Linguistics, vol.11, no.2, 2006.
[18] M.Burrows, D. J. Wheeler, M. Burrows, and D. J. Wheeler, "A block-sorting lossless data compression algorithm," 1994.
[19] J.Seward, "bzip2," 1998. [Online]. Available: http://www.bzip.org.
[20] I.H. Witten, T. C. Bell, A. Moffat, C. G. Nevill-Manning, T.C.Smith, and H.Thimbleby, "Semantic and generative models for lossy text compression," The Computer Journal, vol. 37, no. 2, pp. 83–87, 1994.
[21] E.C¸elikel C¸ ankaya, V. Palaniappan, and S. Latifi, "Exploiting redundancy to achieve lossy text compres- sion," Pamukkale University Journal of Engineering Sciences, vol. 16, no. 3, 2011.
[22] UmAlqura, "Letter by Prophet Muhammad peace be upon him," 2014. [Online]. Available: http://uqu.edu.sa/page/ar/39589.
[23] M.K. Baik, History of Arabic Language. Dar Sa'ad Aldin, Damascus, 1992, vol. 1.
[24] A.J. Viterbi, "Error bounds for convolutional codes and an asymptotically optimum decoding algorithm," Infor- mation Theory, IEEE Transactions on, vol.13, no.2, pp. 260–269, 1967.
[25] M.El-Haj, U.Kruschwitz, and C.Fox, "Using Mechanical Turk to create a corpus of Arabic summaries," in Proceedings of the Seventh conference on International Language Resources and Evaluation, 2010.

Authors

William J. Teahan I am currently a Lecturer in the School of Computer Science at the University of Wales at Bangor. My work involves research into Artificial Intelligence and Intelligent Agents. Ongoing research has also specifically focused on applying text compression-based language models to natural language processing (NLP), text categorization and text mining. Before I came to Bangor, I was a research fellow with the Information Retrieval Group under Prof. David Harper at The Robert Gordon University in Aberdeen, Scotland from 1999-2000; an invited researcher in the Information Theory Dept. at Lund University in Sweden in 1999; and a Research Assistant in the Machine Learning and Digital Libraries Labs at the University of Waikato in New Zealand in 1998. At Waikato, I completed my Ph.D. in 1998 on applying text compression models to the problem of modelling English text.

Khaled M. Alhawiti I am currently the dean of the School of Computers and information technology at the University of Tabuk in Saudi Arabia. My work involves research into natural language processing. I have been working with Dr. Teahan since 2011 on applying text compression models to the problem of modelling Arabic text and other NLP applications for Arabic.