Automatic Alphabet Recognition*

MAAYAN GEFFET   mary@cs.huji.ac.il
YAIR WISEMAN   wiseman@cs.huji.ac.il
DROR FEITELSON   feit@cs.huji.ac.il
Hebrew University Jerusalem

Abstract.   The last step of the Information Retrieval process is to display the found documents to the user. However, some difficulties might occur at that point. English texts are usually written in the ASCII standard. Unlike the English language, many languages have different character sets, and do not have one standard. This plurality of standards causes problems, especially in a web environment, where one may download a document with an unknown standard. This paper suggests a purely automatic way of finding the standard which was used by the document writer based on the statistical letters distribution in the language. We developed a vector-space-based method that creates frequencies vectors for each letter of the language and then matches a new document's vectors to the pre-computed templates. The algorithm was applied on various types of corpora in Hebrew, Russian and English, and provides an efficient solution to the stated problem in most cases.

Keywords: natural language alphabet, characters set, letters' mapping

1.   Introduction

As the Internet has become a part of contemporary life, people all over the world access the web and look for information on a daily basis. The importance of national language support is growing together with the increasing popularity and spread of the network.

Most languages are alphabetical, so they have a constant set of letters. Thus, all the words are composed from them. However, in computer documents, the same letter may be encoded in different ways. Some of these different methods are part of the language usage, such as the distinction between upper case and lower case letters in English. Other distinctions are formatting tools used by document authors, such as the distinction between boldface and *italics*.

"Encoding" is a mapping of letters within the document to binary codes. In information technology, the letters are represented by byte-long codes. Unicode uses two bytes codes. Sometimes, different codes are assigned to the same letter of the alphabet according to the encoding type. For example, English is commonly represented by the ASCII code, which defines a 7-bit value for each lower case letter, upper case letter, and punctuation symbol. This has not always been so; early IBM mainframes used EBCDIC (IBM Character Data Representation Architecture, Reference and Registry,

---

*A preliminary version appeared in SCI2002, Orlando, FL, pp. 122–128.

SC09-2196-00, Dec. 1996), which utilizes a different encoding. The ISO-8859 standard stipulates that ASCII be used, and today practically all English documents use ASCII. When such a document is downloaded, there is no ambiguity with regard to which letters should be displayed.

As ASCII only defines 7-bit codes, the first bit in the byte-long encoding used by computers is always 0. The second set of 128 values, in which the first bit is 1, is undefined by ASCII. Thus fonts that translate the ASCII codes to graphical representations are free to use whatever representation they wish for the additional codes. A common usage is to use special characters; either Latin characters with special markings (e.g. á, ă, à, ě, o, ç, ì) or characters of foreign languages. Note that common punctuation symbols do not have to be included, as they are already defined by ASCII, and this has become a de-facto standard for other languages as well.

The ISO-8859 standard defines the mapping of the higher 128 codes for Latin alphabets and for many other languages, including Cyrillic, Arabic, Greek, and Hebrew alphabets (these use different numbers: 8859-5 is Cyrillic, 8859-6 is Arabic, etc.) (Information Technology 1998). An alternative is the Unicode standard, which uses 16-bit codes to provide unique codes to the symbols needed for all commonly used alphabets in the world. Regrettably, these standards have not come to dominate usage, as opposed to the dominance of ASCII for English. For example, the standard used by Microsoft software for Hebrew characters (known as Windows-1255) (Northrup 1999) does not conform to ISO-8859. Thus, it is relatively common to download a document in a non-English language and find that it uses an unexpected encoding of letters. The result is that the wrong glyphs are displayed and the document cannot be read (figure 1). Such cases occur quite often while exploring non-English sites on the web. The typical solution is to try different fonts supported by the browser with the hope that one of them uses the same mapping as the document.

Another typical problem for Semitic languages is bi-directional text, where some text is read right-to-left (e.g. words) and some other characters (e.g. numbers) are read left-to-right. There are two ways to represent this type of text: (i) logical, and (ii) visual. In the logically encoded text, the characters are stored in the order they are typed by the human user, using the direction marking flags ('*ltr*' for left-to-right, '*rtl*' for right-to-left). Obviously, there is a need for a special conversion algorithm to display logically stored text. The most common algorithm is the Unicode Bi-directional algorithm (The Unicode Consortium 2000). The visual representation stores the text as it should be displayed on the screen device. As this method has many disadvantages (Northrup 1999), the logical representation has become a standard and is supported by the Internet Explorer 5 (Bracewell and Karp 1998) and the Netscape Navigator 6.1 Browsers (Smith 2001). The existing standards for web pages, HTML-4 (Graham 1998) and XHTML 1.0 (Kennedy and Musciano 2000) (latest version, which is a reformulation of HTML 4 in XML 1.0), support Bi-directional text as defined in Unicode's Bi-directional algorithm (Northrup 1999, Jaeger 2002). According to this standard users supply a *language, a charset* (to identify the encoding standard) and a dir (the direction of a page—'*rtl*' or '*ltr*') parameters in their HTML pages in order to allow a browser to process a document correctly. Unfortunately, many Hebrew documents are encoded visually in various non-standard ways, and therefore cannot be displayed even by

Figure 1. Results of displaying a Hebrew document with different fonts, using a Hebrew version of Netscape (Smith 2001). The top one provides the correct decoding of the letters. The second font does not define any glyphs for the higher 128 codes, so the browser displays question marks for all the unknown symbols. Only spaces are displayed correctly. The third font uses the Unicode standard, which shows some special graphical symbols. The last one is a Cyrillic font, which displays Cyrillic letters that share the same code as the desired Hebrew ones.

standard-conforming browsers. Thus, even if the user knows the language of the document, it is not enough for correct decoding.

Our goal in this paper is to develop a methodology that can be used by a browser to automatically determine which encoding was used in a document. This will allow the browser to choose the correct font for displaying the document, without requiring a trial-and-error search by the user. We assume that the decoded document, and the language and the direction of the document are given as an input to our algorithm. It should be noted that the solutions for the language recognition and the bi-directional text problems

can be easily derived from our method as well. The outline of the algorithm is given below.

2. Template-based recognition

We aim to find an efficient statistical solution to the text encoding recognition problem, with no knowledge of morphological or syntactic rules. Therefore, this is a general method for any alphabetic language. The basic premise is to pre-compute universal templates of letter distribution in a language based on various corpora examples, and then compare them to the statistics of a given document that needs to be decoded.

2.1. Position vectors

The first problem determining which kind of statistics analysis to use. It has been shown in Wiseman (2000) that the distribution of letters in a given human language is generally similar in many texts. There are always exceptions to this rule. For example, the first chapter of the Hebrew Bible doesn't contain the letter "Samech" (S). This was likely a purposeful omission, so that it does not reflect a regular Hebrew text.

Unfortunately, a good recognition cannot be achieved by this information. There are groups of 3–4 letters (e.g. A, B, $, and N) which have very close frequencies, so it will be hard or even impossible to distinguish between them (figure 2). We are interested in getting more detailed characteristics of each letter in the language. This leads us to a vector-space-based model. Our first attempt was to construct a "*position vector*" for each letter, by counting its occurrences at every position in the word separately.
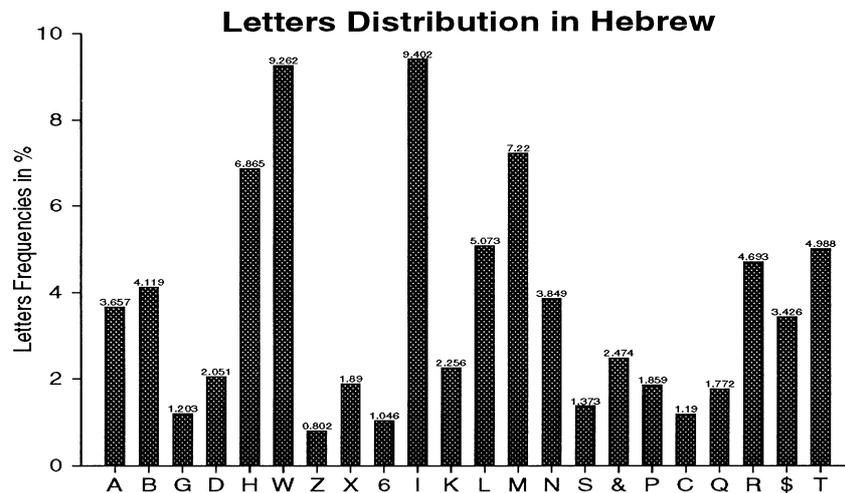

Figure 2.   Letters frequencies distribution in Hebrew Scientific Journals (HSJ in Table 1). The Hebrew letters are displayed using their Latin transliteration table from the appendix.

More formally, define $count_{l,i}$ to be the number of times letter l appears in position i in the word. The 20th position in the vector is dedicated for the letter's occurrences at the last position in the word. For example, for the word "bicycle", the last—the 20th position of the letter's "e" vector will be incremented (and not the 7th position in the vector). Thus, $count_{l,i} = count_{l,i}^{not-last} + count_{l,i}^{last}$. The position vector P_l is then defined to be $(a_1, a_2, \ldots, a_{19}, a_{last})$, where $a_i = \frac{count_{l,i}^{not-last}}{total} * 100$, and $a_{last} = \frac{\sum_i count_{l,i}^{last}}{total} * 100$. *total* denotes the total number of all the letters in the text and words are assumed to be shorter than 20 letters long. Note, that practically all encodings agree on the ASCII standard, so the white spaces, newlines and punctuation marks will be encoded identically in all of them. Based on this information, the document can be parsed into words and the position of each letter in the word can be discovered. It should be noted that the direction of the text is given as an input.

Comparing the vectors of frequencies of the letters, rather than single values, provides much more accurate results (figures 3, 4 and 7). Some letters tend to appear more often in the beginning of a word, and others in the middle or in the end. For instance, articles, prepositions and verb prefixes are single letters which appear in the beginning of words very frequently in Hebrew, while plural form suffixes will be often found in the end. In Hebrew and Arabic there are also several letters that are written differently in different positions. In Hebrew, these are special forms in the final position, while in Arabic there may be special forms for both the first and the final positions. Such forms will therefore never appear in the middle, and their vectors will include all zeros except for the correct position. The non-final form of these letters, on the other hand, will have a zero at the first and/or last place, as for M in figure 3.
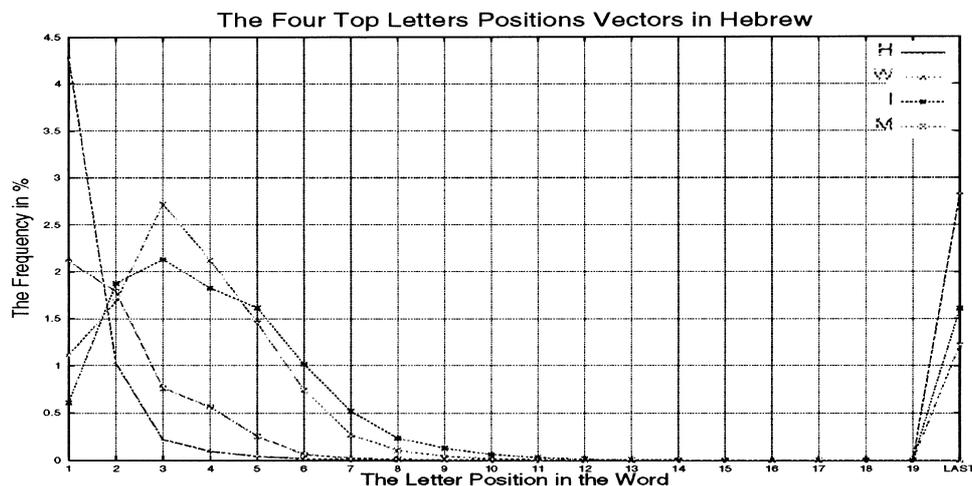

Figure 3. *Position vectors* for the four most frequent letters (I, W, H, M) in the Hebrew Scientific Journals (HSJ in Table 1). While in this and other graphs the x axis is actually discrete, plotting lines that connect the values aids the eye in making the comparison. As we can see the vectors are very different.
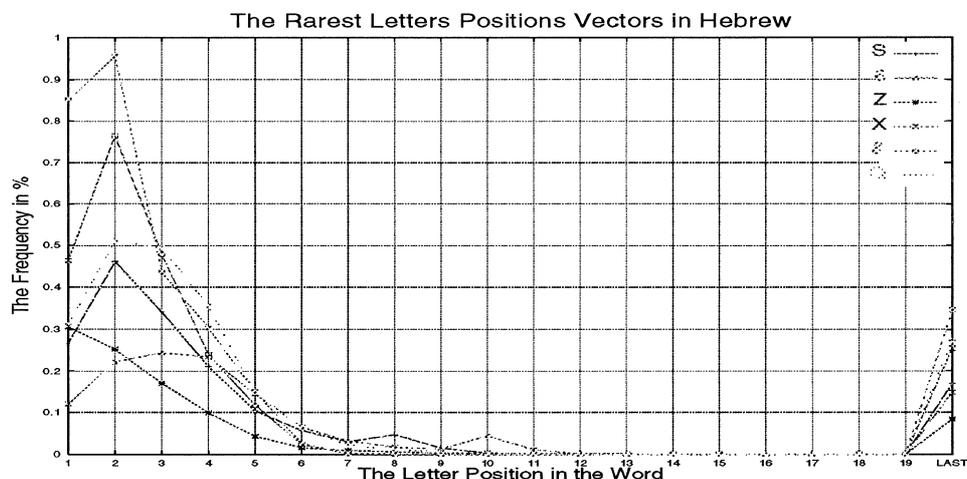
Figure 4.   Position vectors for the six rarest letters (S, 6, X, &, Q, Z) are quite characterizing as well.

2.2. *Environmental vectors*

So far, we looked at the text as a 0th order Markov Chain, as we treated each letter independently. However, it may be helpful to consider the closest neighbors of the letter in order to identify it, and for that purpose to extend our model to a higher order Markov Chain.

The basic approach of using statistical letters distribution in the language was proposed by Shannon (1948). In Benedetto (2002) the authors use statistical information on the distribution of n-grams of letters in different languages in order to construct clusters of similar languages. Damashek and Huffman employed n-grams of letter statistics to classify documents by their topic (Damashek 1995, Huffman 1995). Markov Chains are also commonly used in a human language processing to define compression rules (Bookstein and Klein 1990, Cormack and Horspool 1987) and to compute the probability of the next letter by its precedents (Horspool and Cormack 1986, Yoon et al. 1999). For example, in English "q" tends to be followed by "u", while "x" almost never occurs after "z" (Wiseman 2000, Ziv and Lempel 1978). Such rules are a very strong feature in English, but less so in Hebrew writing, which puts almost no restrictions on letters combinations, since it does not contain vowels. Nevertheless, the differences in the probabilities of different pairs are sufficient to aid in recognition.

First, we collect information about all possible pairs of letter occurrences in the corpus. This data may be viewed as a matrix M of the size: $|Alphabet| \times |Alphabet|$, where every cell M_{ij} contains the frequency, $a_{i,j}$, of the corresponding pair of letters, where $a_{i,j} = \frac{count_{i,j}}{total} * 100$. The next step is to use this matrix of pairs to find characteristics for individual letters. We notice that rows and columns of M represent the subsequent and preceding vectors of all the letters, respectively. So here again we took a vector-space-model to represent a letter's closest environment (figures 5 and 8). For each letter l in the alphabet, we define its "*environmental vector*" to be its row M_l of frequencies of different successors: $M_l = (a_{i,A},$
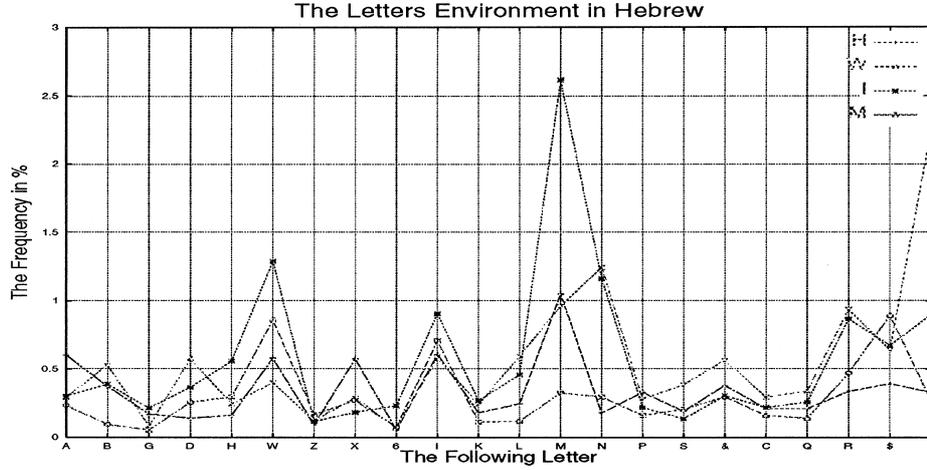
Figure 5. *Environmental vectors* of the most frequent letters in the Hebrew Scientific Journals (HSJ in Table 1). The difference between the letters is even bigger than as determined by the *position vectors* in figure 3.

$a_{i,B}, \ldots, a_{i,Z}$), where $a_{l,s} \frac{count_{l,s}}{total} * 100$, where s is the letter $l's$ successor in the text, so the complexity of the algorithm that constructs the environmental vectors is $O(|Alphabet|^s)$.

The main problem of both algorithms is ambiguity, when several distinct letters in the document are mapped to a single letter in the template.

Note that we still did not use the information contained in the columns of the matrix, the precedents "*environmental vectors*" (*PM*). This redundant information is useful to disambiguate the results, thus increasing our model to 2nd order Markov chains. Another way to eliminate ambiguity is to combine the two proposed algorithms, the "*environmental vectors*" and the "*position vectors*", in the following way: the basic recognition is done by the former technique as it usually outperforms the latter one, followed by the "*position vectors*" results to correct errors obtained from the "*environmental vectors*" algorithm. The "*combined*" method resulted in better accuracy precentages, as shown in Table 1.

2.3.   On-line matching

Once the off-line construction of templates for various languages is completed, the system can start to work on-line, getting new documents and matching their vectors to the templates.

The matching procedure receives two sets of vectors, V_1, V_2 generated from two texts, and output is a set of pairs, that are the closest to each other: $PS = \{(v, w) : v \in V_1, w \in V_2, f(v, w) = \min_{wt \in V_2} f(v, wl)\}$. V_1 usually stands for the template vectors and V_2 for the new text vectors set.

Another question to be discussed in this context is the f function, i.e. the vectors distance metries. We experimented with two versions of the norm formula: $f_1 = \|V_1 - V_2\|_1 = \sum_{1 \leq i \leq n} |x_i - y_i|$ vs. $f_2 = \|V_1 - V_2\|_2 = \sqrt{\sum_{1 \leq i \leq n} (x_i - y_i)^2}$. The latter norm decreased

Table 1. The combined method results table.

Sources			Accuracy in %		
Language	Test no.	Compared types	Positions vectors	Environmental vectors	Combined method
Hebrew	1	HN-HN**	100	100	100
Hebrew	2	HN1 - HN2***	91(2,1)*	100	100
Hebrew	3	HSJ - HSJ	100	100	100
Hebrew	4	HSJ - HN	100	100	100
Hebrew	5	HS - HS	94(2,0)	94(2,0)	100
Hebrew	6	HB - HB	100	100	100
Hebrew	7	HS - HN	76(6,2)	91(2,1)	91(2,1)
Hebrew	8	HB - HS	76(8,0)	91(3,0)	91(3,0)
Hebrew	9	HB - HSJ	76(6,2)	94(2,0)	94(2,0)
English	10	EC - EC	100	100	100
English	11	ES - ES	100	100	100
English	12	EL - EL	100	100	100
English	13	EL - ES	94(0,2)	94(2,0)	100
English	14	EL - EC	94(2,0)	94(2,0)	94(2,0)
English	15	ES - EC	76(4,4)	85(3,2)	91(1,2)
Russian	16	RN - RN	100	100	100
Russian	17	RS - RS	100	100	100
Russian	18	RP - RP	100	100	100
Russian	19	RD - RD	100	100	100
Russian	20	RP - RD	91(3,0)	91(2,1)	94(2,0)
Russian	21	RN - RP	80(5,2)	88(4,0)	88(4,0)

*The numbers of the errors of two types: (i) multiple matches (including the correct one), and (ii) unmatched letters, are shown in the parenthesis, respectively.
**We denote X - X for comparison of two distinct parts of the same source.
***We denote HN1 - HN2 for comparison of two different newspapers.

the accuracy by up to 10% as demonstrated in figure 9, so we chose f_1 as the better metric for our purposes. We also tried to use the well-known Kullback-Leibler divergence metric, but it produced very poor results (less than 10% matches). This can be explained by the fact that the KL-divergence metric works with probabilities instead of frequencies of the letters co-occurrences.

We noticed that sometimes the direction of mapping had a crucial influence on the accuracy. Therefore, mapping is executed in both directions: $V_1 \Rightarrow V_2$ and $V_2 \Rightarrow V_1$ to reduce ambiguity and to ensure that every letter gets a pair from the other set. The two results are then merged. Thus, the final number of errors is limited by the number of errors the best of the directions made. This helped increase the hit ratio by up to 6% in 25% of the cases as shown in figure 9.

2.4.   The combined method

The final version of the proposed algorithm is summarized below:

0. Compute positions (P_1) and environmental (M_1, PM_1) vectors for the templates set. This is done off-line.
1. Get a new document from the user.
2. Compute positions (P_2) and environmental (M_2, PM_2) vectors for the document.
3. Pick a template, either the default "Newspapers Style" or according to a user selection.
4. Compare the successors' environmental vectors of the template, M_1, to those of the document, M_2, using f_1. The resulting pairs of letters are stored in PS (the matched pairs set).
5. For letters that got no or several mappings do:

   - Execute the 4th step mapping in the opposite direction:
     $M_2 \Rightarrow M_1$
   - Check:

     a) If a letter $l_j^{(2)}$ in M_2 was mapped to n different letters in M_1 and if it got a unique mapping $l_i^{(1)}$ now (which is one of those n):
     – Add a pair ($l_I^{(1)}, l_j^{(2)}$) as a match in *PS*.
     b) If a letter $l_j^{(2)}$ in M_2 was not mapped at all and if it got a unique mapping $l_i^{(1)}$ now:
     – Add a pair ($l_i^{(1)}, l_j^{(2)}$) as a match in *PS*.

6. For letters that are still not resolved:

   - Compare the precedents environmental vectors of the template, PM_1, and the document, PM_2, in both directions, using f_1. Repeat step 5 with PM_1, and PM_2.
   - Compare the position vectors of the template, P_1 and the document, P_2, in both directions, using f_1. Repeat step 5 with P_1, and P_2.

7. If there is only one unidentified letter $l_k^{(2)}$ left in the document alphabet:

   - Match it to the remaining letter in the language alphabet.

2.5.   Results of the combined method

We tested our methods on various types of texts in three languages: Hebrew, Russian, and English. The source types in Hebrew experiments were on-line newspapers (HN, HN1, HN2), the Parliament protocols (HS), which represent the conversation language, several on-line scientific journals (HSJ) (Hebrew resources, http://www.snunit.k12.il/), and the Jewish Bible (HB). The English sources included Computer Science text (EC) (The Conversation English resource, http://www.athel.com), Conversation language (ES) (The Scientific English resource, http://citeseer.nj.nec.com/cs), and the complete works of William Shakespeare (EL) (The English Literature resources, http://www.chemicool.com/). The Russian corpus contained on-line newspapers (RN), scientific articles collection (RS),
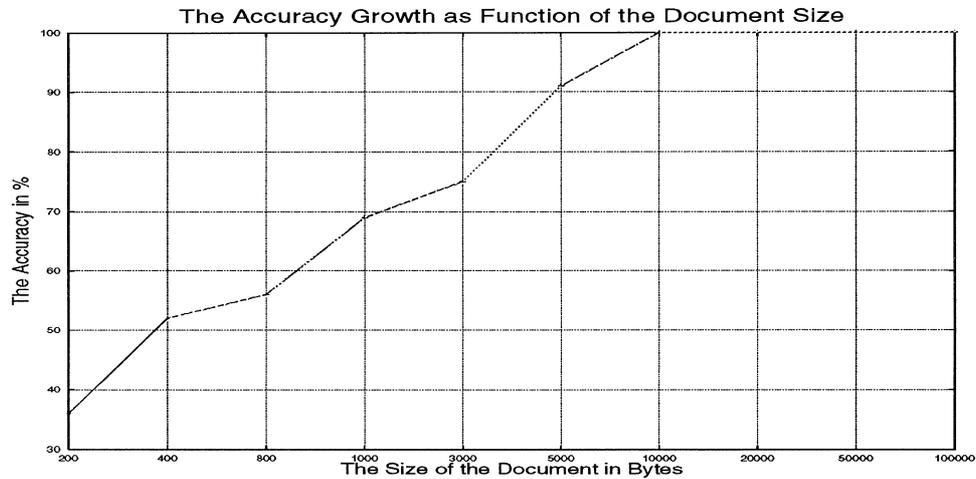
Figure 6. The document size has a crucial influence on the accuracy of the statistical encoding recognition. The larger the document, the more accurate the results. Naturally, the accuracy is quite low for extremely small documents (of 30–150 words). It grows steadily with increasing text size, and reaches a maximum accuracy in texts 700–1,500 words in length.

and the prose of A. S. Pushkin (RP), and F. M. Dostojevsky (RD) (Russian resources, http://ruslit.virtualave.net.).

We also examined the influence of the text size on the computed statistics in order to find the lower bound on the new document size. We ran the algorithms on texts of sizes varying from 200 Bytes (30–40 words) to 100 MB (~15 million words). Significant changes occur below 10 K (~1,500 words), mostly in the bottom half of this range; the difference between 5 K and 10 KB was of 2–3 letters. Starting from 10 K the matching results never changed (as shown in figure 6).

Below are illustrative graphs for different stages of the algorithm (figures 7, 8, and 9). The comparative results of the described algorithms are detailed in Table 1. Both vector methods in isolation succeeded for homogeneous corpora, but produced some mismatching for different types of text. The best case is, therefore, when we compared two similar sources, such as two newspapers, the same author's books or two halves of the same source, (e.g. the Bible), that was divided into two parts and matched one to the other. The worst case is when comparing ancient text to modern one, or written to conversational language. The lower hit ratio for conversational language samples can be explained by their high number of participants, since conversational language has almost no norms or restrictions. In order to slove this problem, we need to find some common basis for all the language styles and genres.

3. Automatically generated dictionary

A list of frequently used words can provide a common basis for a language. So we would like to construct a dictionary of the most common words constructed out of a large corpus.

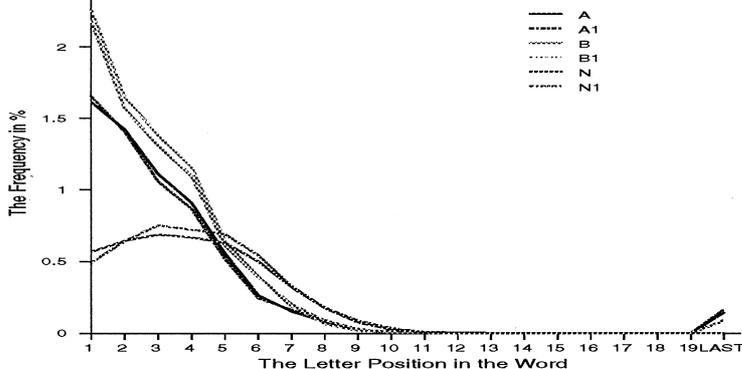## The Letters "Positions Vectors" Comparison in Hebrew

Figure 7. Example of identification of 3 letters that have similar frequencies, based on position vectors. The vectors are for A, B, N, obtained from two different texts in Hebrew; Scientific Journals (HSJ in Table 1) vs. Newspapers articles (test no. 4 in Table 1). despite the similar frequencies, there is no ambiguity in matching by position vectors: the two vectors for each letter are nearly overlapping. Note that N never occurs at the last position since it has a special character for the final form.

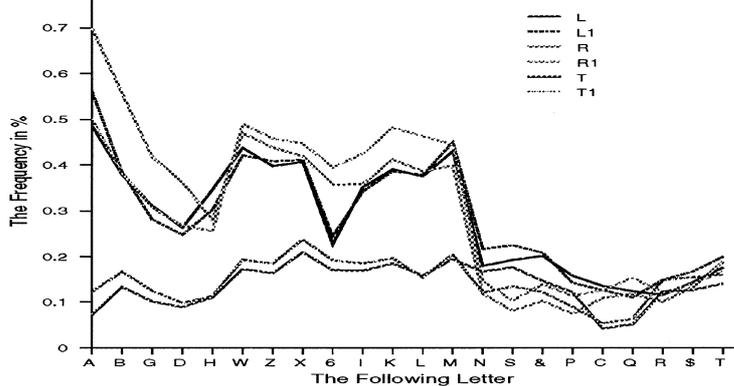## The Letters "Environmental Vectors" Comparison in Hebrew

Figure 8. Matching three medium-frequency letters from Hebrew Scientific Journals (HSJ in Table 1) vs. Newspapers articles (test no. 4 in Table 1), using environmental vectors. These vectors consist of over 20 meaningful points (dimensions) while in the position vectors usually only the first 10 and the last one were informative, therefore the similarity of the corresponding letter vectors is even more distinct in this model.

Building a dictionary is a very common method in other fields, like Data Compression (Ziv and Lempel 1978) and Speech Recognition (Sloboda 1995). For this purpose the algorithm takes a large text. We can use the text that was employed for template generation in Section 2. The text is split into words. We consider a "word" any sequence of letters surrounded by non-letter characters. The algorithm counts the words and sorts them according to their number of appearances. Then, we can easily take the *N* most frequent words.
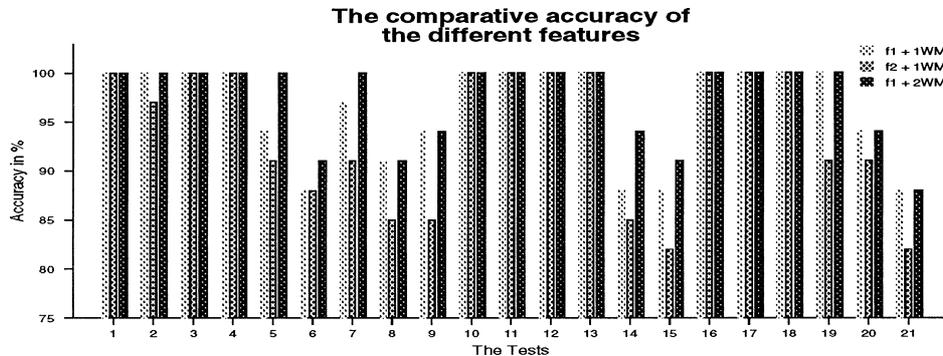
Figure 9. Comparison of the accuracy achieved by 3 on-line matching schemes, based on different combinations of the vector distance norm f_1 vs. f_2 and the use of one-way or two-way mapping (denoted '$1WM$' or '$2WM$'). The test numbers refer to Table 1. As we can see, the $f1 + 2WM$ combination is always better than or equal to the other two.

From this point, the dictionary integrates into our algorithm. We have some letters that were mapped to several differnt letters in the template by the combined algorithm from Section 2. For each mapping of such a letter, the modified algorithm fetches from the dictionary the N most frequent words. Then these words are searched in the original text. The mapping of a letter with the highest number of found words is assessed to be the correct one.

Afterward, the "disqualified" mappings have to be assigned to alternative letters. For this we use the combined algorithm described in the previous section. We look for the best matching letter, excluding the previously disqualified ones. This procedure is repeated until there are no ambiguous mappings or when there is no change in the letters' mapping table. A sketch of the algorithm is given below:

1. Automatic Dictionary Generation out of large text (off-line):

   (a) Split the text into words.
   (b) Build a dictionary containing these words.
   (c) Sort the dictionary according to the frequency of the words occurrences.

2. Given a new document to be decoded:

   (a) For each letter l with an ambiguous mapping:

      i. For each possible mapping m of l:

         A. Fetch from the dictionary a list L of the N most frequent words containing m.
         B. Substitute all the letters of all the words in L by their current mappings.
         C. Search for the resulting words in the given document.
         D. Set the mapping with the highest number of words occurrences to be the correct one.

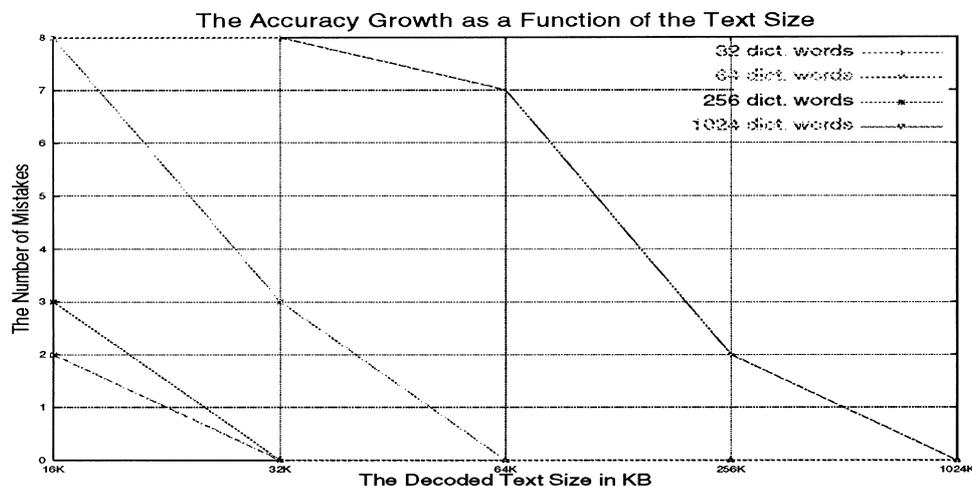   (b) For the disqualified mappings apply the combined method from Section 2 to assign them to alternative letters.

Figure 10. The number of mismatchings in the worst case as a function of text size for various numbers of words fetched from the dictionary.

We have run the new algorithm on the test files from Section 2, and the accuracy has been increased. The number of the mapping mistakes in the worst case (comparing different texts types) as a function of N most frequent words and the size of the text can be seen in figure 10. The conclusion from the graph is that 100% mapping accuracy can be achieved by either working with a large enough document or increasing the number of words fetched from the dictionary.

According to our method, there is no need for a special computerized linguistic resource, rather we generate our dictionary automatically from the text. We use any text of the language. Such a text can usually be found on the web. Moreover, the algorithm looks just for *N* words in the dictionary. Since N is a constant, the algorithm will perform $|Alphabet|$ iterations of letters fixing in the worst case. Actually, this means that our algorithm's complexity is $|Alphabet|^4$, since theoretically each letter can be mapped to every other letter in the alphabet until it receives the correct interpretation.

4.   Conclusions and future work

We developed and presented a purely automatic method for the document encoding recognition, based on a vector-space model. It produced excellent results for languages from different families: Semitic, Slavic, and Indo-European. The time complexity of the vectors generation process is $\Theta(n)$, where *n* is a number of characters in the new document. The matching procedure performance is bound by $\Theta(|Alphabet|^4)$.

We would like to suggest some possible extensions and further applications of our research. The method may be naturally used to easily identify the language of the document by comparing its statistics to pre-computed templates of given languages. Only one of them will have a relatively small number of unmatched or multiple-matched letters. It can also

be applied to determine document direction ('*ltr*', '*rtl*') and representation type (*visual*, *logical*) by simply running the "*position vectors*" algorithm in both ways and comparing the results to the templates. Obviously, only one of the two obtained vectors sets will match the templates, which reveals the correct direction and representation of the text. These are common difficulties since people tend to omit this information despite the fact that it is mandated by the HTML 4 standard. In summary, the only item our algorithm needs in order to recognize the encoding of a given piece of text is the list of candidate languages to choose from. Given this, it executes the three following stages, first it identifies the language, then the representation type and the direction, and finally the character set.

It is interesting to note that we received consistently different frequencies for different text types. Furthermore, it is a known fact in Linguistics (Yalin 1942) that every person uses certain very common stop words and prepositions with a constant individual frequency. There are also extreme examples, such as the first Prime Minister of Israel, David Ben-Gurion, who never used the case preposition "AT". This feature can be very effective in identifying the authorship of written documents. We suggest to apply the presented method to classify texts according to their date, author, and style. First, we prepare sample vectors (templates) for various types of text and then compare them to a new document vectors and index it to the closest vectors category, respectively.

The templates vectors as described so far were once calculated and then remain static. Another possible extension is to generate and update the templates as new documents arrive and are decoded, and thus make error correction dynamically.

Appendix

In this paper we used the Latin transliteration of the Hebrew and Russian Letters. The Russian alphabet consists of 33 letters: 21 consonants, 10 vowels and two letters without sound—soft sign and hard sign (Russian transliteration, www.geocities.com/Colosseum/Track/7635/). Undotted Hebrew alphabet consists of 22 letters, all of them are consonants, and 5 of them have a special "final" form (Segal and Itai, Hebrew transliteration, http://www.cs.technion. ac.il/~serelgl/bxi/hmntx/teud.html) for a total of 27 symbols.

The Russian-Latin and Hebrew-Latin Transliteration Table:

Russian	Latin	Hebrew	Latin
ah	A	alef	A
beh	B	bet	B
veh	V	gimel	G
geh	G	dalet	D
deh	D	hei	H
yeh	Ye	waw	W
yo	Yo	zain	Z
zheh	Zh	chet	X

(*Continued on next page.*)

(*Continued*).

Russian	Latin	Hebrew	Latin
zeh	Z	tet	6
ee	I	yod	I
short ee	J	kaf final	k
kah	K	kaf	K
ehl	L	lamed	L
ehm	M	mem final	m
ehn	N	mem	M
oh	O	nun final	n
peh	P	nun	N
ehr	R	samech	S
ehs	S	ain	&
teh	T	pei final	p
oo	U	pei	P
ehf	F	tzadi final	c
khah	H	tzadi	C
tseh	Ts	quf	Q
cheh	Ch	resh	R
shah	Sh	shin/sin	$
schyah	Sch	tav	T
hard sign	'		
i	Y		
soft sign	,		
eh	E		
yoo	Yu		
yah	Ya		

References

Benedetto D, Caglioti E and Loreto V (2002) Language trees and zipping. Physical Review Letter., 88(4).

Bookstein A and Klein ST (1990) Compression, information theory and grammars: A unified approach. ACM Trans. on Information Systems, 8:27–49.

Bracewell M and Karp DA (1998) O'Reilly Utilities—Quick Solutions for Windows 98 Annoyances, O'Reilly & Associates, Inc.

Cormack GV and Horspool RN (1987) Data compression using dynamic Markov modelling. Computer Journal, 30(6):541–550.

Damashek M (1995) Gauging similarity via N-Grams: Language-independent categorization of text. Science, 246:843–848.

Graham IS (1998) HTML 4.0 Sourcebook. Wiley Computer Publishing, New York, pp. 450–451.

Hebrew resources, http://www.snunit.k12.il/.

Horspool RN and Cormack GV (1986) Dynamic Markov modelling—A prediction algorithm. In: Proc. 19th Hawaii International Conference on System, Sciences, vol. II, pp. 700–707.

Huffman S (1995) Acquaintance: Language-independent document categorization by N-grams. The Fourth Text REtrieval Conference (TREC-4), Nov. Gaithersburg, Maryland, USA.

IBM Character Data Representation Architecture, Reference and Registry; (Dec. 1996) SC09-2196-00.

Information Technology (1998) ISONET Manual, ISO/IEC 8859, Jersey City. NJ.

Jaeger G (2002) Some notes on the formal froperties of bidirectional optimality theory, Journal of Logic, Language, and Information, 11(4):427–451.

Kennedy B and Musciano C (2000) HTML & XHTML: The Definitive Guide, O'Reilly & Associates, Inc., 4th edition, Section 15.1.

Northrup A (1999) Introducing Microsoft Windows2000 Server, Microsoft Press, Washington, pp. 15–16.

Russian transliteration, www.geocities.com/Colosseum/Track/7635/

Russian resources, http://ruslit.virtualave.net.

Segal E and Itai A, Hebrew transliteration, http://www.cs.technion.ac.il/~erelsgl/bxi/hmntx/teud.html

Shannon CE (1948) A mathematical theory of communication. Bell System Tech. Journal, 27:398–403.

Sloboda T (1995) Dictionary learning: Performance through consistency. In: Proc. of ICASSP '95, Detroit, MI, pp. 453–456.

Smith B (2001) SUN Microsystems Unveils Netscape 6 for Solaris, Sun's Press Releases, Brookline, MA.

The Conversation English resource, http://www.athel.com

The Scientific English resource, http://citeseer.nj.nec.com/cs

The English Literature resources, http://www.chemicool.com/

The Unicode Consortium (2000) The Unicode Standard, Version 3.0, Addison-Wesley Developers Press, Reading, MA.

Wiseman Y (2000) Parallel Compression, Ph.D. Thesis, Computer Science Dept., Bar-Ilan University, Ramat-Gan, Israel, pp. 76–79.

Yalin D (1942) Grammar of the Hebrew Language, R. Mass Press, Jerusalem, (In Hebrew).

Yoon HS, Soh J, Min B and Yang HS (1999) Recognition of Alphabetical Hand Gestures Using Hidden Markov Model, IEICE Transactions Fundamentals, E82-A(7):1358–1366.

Ziv J and Lempel A (1978) Compression of individual sequences via variable-rate coding. IEEE Trans. on Information Theory IT-24, pp. 530–536.