Moustafa Elshafei, Husni Al-Muhtaseb and Mansour Alghamdi, “Statistical Methods for Automatic Diacritization of Arabic text”, Proceedings 18th National computer Conference, Riyadh, March 26-29, 2006

Statistical Methods for Automatic diacritization of Arabic text

Moustafa Elshafei[1], Husni Al-Muhtaseb[1], and Mansour Alghamdi[2]

1-King Fahd University of Petroleum and Minerals, Dhahran, Saudi Arabia

2- King Abdulaziz City of Science and Technology, Riyadh, Saudi Arabia.

Keywords: *Arabization, tashkeel, diacritization, vowel restoration, statistical methods.*

Abstract

In this paper, the issue of adding diacritics Tashkeel to undiacritized Arabic text using statistical methods for language modeling is addressed. The approach requires a large corpus of fully diacritized text for extracting the language monograms, bigrams, and trigrams for words and letters. Search algorithms are then used o find the best probable sequence of diacritized words of a given undiacritized word sequence. The word sequence of undiacritized Arabic text is considered an observation sequence from a hidden Markov Model, where the hidden states are the possible diacritized expressions of the words. The optimal sequence of diacritized words (or states) is then efficiently obtained using Viterbi Algorithm. We present an evaluation of the basic algorithm using the Qur’an’s text, and discuss various ramifications for improving the performance of this approach.

Introduction

The absence of the diacritics in modern Arabic text is one of the most critical problems facing computer processing of Arabic text. Readers of Arabic can restore the proper diacritics of the text, but when it comes to computer processing, the computer still needs to be provided with algorithms to mimic the human ability to identify the proper diacritics of the text. Such tool is an essential infrastructure for applications such as Text-to-Speech (Elshafei el. 2002), Automatic Translation (Trost 1991), and Arabic data mining applications (Mustafa 1998).

The problem of automatic generation of the Arabic diacritic marks, *Al Tashkeel Al-Aly*, is known in the literature under various translations, e.g., automatic vocalization, vowelization, diacritization, accent restoration, and vowel restoration. In this paper the term diacritization will be used due to the fact that the missing symbols do not represent only the vowels but represent, in addition to that, gemination *shaddah,* lack of vowels *sukoon,* and *Tanween* n-suffixes.

The formal approach to the problem of automatic restoration of the diacritic marks of Arabic text involves a complex integration of the Arabic morphological, syntactic, and semantic rules (El-Saadani and Hashis, 1988, Shatta, 1994, Khoja, 2001, and Farghali et. Al, 2003). A morphological rule matches the undiacritized word to known patterns or templates and recognizes prefixes and suffixes. Syntax applies specific syntactic rules to determine the word-final diacritic marks by applying Finite State Automata. Semantics helps to resolve ambiguous cases and to filter out hypothesis. Among the systems using the above approach is ArabDiac $^{(T)}$ of RDI of Egypt [1], Al-Alamia of Kuwait [2] and CIMOS of France [3]. The second approach is the data-centered approach, where a large corpus of text is used to extract language statistics for estimating the missing diacritical marks, or used to train artificial neural networks. Among the later approach is the work of (Sultan, 2001) who investigated application of Neural networks for vowel restoration of Arabic text.

The approach here falls under the general class of statistical methods in pattern recognition, and has been applied successfully in the speech recognition field. The system is first trained using a corpus of fully diacritized text, and preferably covering specific domains. The system generates word vocabulary list and determine the frequency of each word. It also generates word bigrams and trigrams, and performs other preprocessing and post processing steps to deal with exceptions and to overcome some known limitations.

Arabic writing system consists of 36 letter forms which represent the Arabic consonants. These are: ا, آ , أ , إ , ئ , ؤ , ء , ى , ة , ب , ت , ث , ج , ح , خ , د , ذ , ر , ز , س , ش , ص , ض , ط , ظ , ع , غ , ف , ق , ك , ل , م , ن , هـ , و and ي . Each Arabic letter represents a single consonant with some exceptions: أ , إ , ئ , ؤ and ء represent the glottal stop, but are written in different forms depending on the consonant position in the word and its adjacent phonemes. ا symbolizes the glottal stop or the long low vowel, depending on its position in the word and sentence. و and ي are long vowels when preceded by a vowel of its nature, i. e, *dhammah* and *kasrah*, respectively, and they are consonant otherwise.

In addition to the consonant symbols, there are 8 diacritics. A diacritic may be placed above or below a letter (see Table 1). The first three diacritics represent the Arabic short vowels, and the last three are the *tanween* that occur only in word-final position.

Almost all modern Arabic texts are written using the consonant symbols only, i. e, the letters without the vowel symbols or the diacritic marks. A word such as “علم” when diacritized can be: “عَلَم” flag, “عِلْم” science, “عُلِمَ” it was known, “عَلِمَ” he knew, “عَلَّمَ” he taught or “عُلِّمَ” he was taught. Arabic readers infer the appropriate diacritics based on the linguistic knowledge and the context. However in the case of a text-to-speech or automatic translation system, Arabic letters need to be diacritized, otherwise, the system will not be able to know which word to select.

There are general rules for diacritizating Arabic text. For example, *Shaddah* and *sukoon* do not follow a word-initial letter; *tanween* (last three diacritics in Table 1) comes only in word-final position. At the same time, certain letters have their own rules of diacritization: ا, آ and ى are not followed by a diacritic; ة, ء and إ are not followed by *shaddah*.

Diacritic	IPA	Definition	Sample
َ	a	*fathah* (low short vowel)	بَ
ُ	u	*dhammah* (high back rounded vowel)	بُ
ِ	i	*kasrah* (high front vowel)	
ّ	:	*shaddah* (geminate: consonant is doubled in duration)	
ْ	∅	*sukoon* (the letter is not diacritized nor geminated)	
ً	an	*tanween fathah* (low vowel + alveolar nasal)	بً
ٌ	un	*tanween dhammah* (high back rounded vowel + alveolar nasal)	بٌ
ٍ	in	*tanween kasrah* (high front vowel + alveolar nasal)	بٍ

Table 1. Arabic diacritics with their International Phonetic Alphabet representations (IPA), definitions and samples with the voiced bilabial letter ب.

In Section 2 we present the formulation of the problem, while in Section 3 we outline the basic algorithm. In Section 4 we describe the training set and its processing, and in Section 5 we present detailed evaluation of the results and the modification to eliminate certain classes of restoration errors.

Problem Formulation

We assume we have a large training set of Arabic text with full diacritical marks, $\mathbf{T_V}$, and its corresponding undiacritized text, $\mathbf{T_U}$. We then generate a word list, $\mathbf{L_V} = \{v_i\}_1^{N_v}$, of fully diacritized vocabulary words in $\mathbf{T_V}$. We also generate a table, $\mathbf{f_V}$, of the frequency of occurrence of each word in $\mathbf{L_V}$, such that $\mathbf{f_V}(k)$ is the number of occurrence of v_k in the training text $\mathbf{T_V}$. Similarly, we construct $\mathbf{L_U}$ of all undiacritized words in $\mathbf{T_U}$. Let $\Gamma(.) : \mathbf{L_V} \to \mathbf{L_U}$ be the mapping from $\mathbf{L_V}$ to $\mathbf{L_U}$; For each word $u_k \in \mathbf{L_U}$ we define a subset $V_k \subset \mathbf{L_V}$ corresponding to all the diacritized words that are mapped to u_k, i.e. $V_k = \{v \in \mathbf{L}_V ; \Gamma(v) = u_k\}$.
Now, given a word sequence (without diacritical marks)

$$W = w_1 w_2 ....w_M ; w_t \in \mathbf{L_U} ; t = 1,2,..,M \quad ; \qquad (1)$$

We wish to determine the most probable diacritized word sequence:

$$D = d_1 d_2 ....d_M \qquad (2)$$

Where $d_t = v_j = L_V(j)$ for some $j \in [1, N_v];$ and for $t = 1,2,...M$. We assume that $w_t = u_k = L_u(k)$ for some $k \in [1, N_u];$ for $t = 1,2,...M$, that is to say that all the words in (1) exist in $\mathbf{L_U}$.
The word sequence D may be chosen to maximize the posteriori probability $P(D \mid W)$, i.e. the best diacritized word sequence, $\hat{D}$, satisfies

$$\hat{D} = \underset{D}{\arg\max}\, P(D/W) \qquad (3)$$

The conditional probability $P(D \mid W)$ can be written as

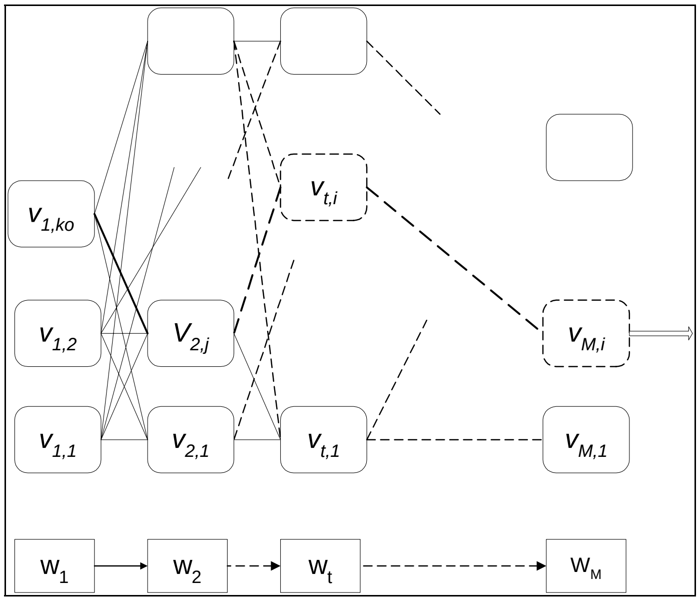
$$\begin{aligned} P(D/W) &= P(d_1 d_2 .....d_m \mid w_1 w_2 ......w_m) \\ &= P(d_1 \mid w_1 w_2 ......w_m) P(d_2 \mid d_1 ; w_1 w_2 ......w_m) \\ &\quad P(d_3 \mid d_1 d_2 ; w_1 w_2 ......w_m) ......P(d_m \mid d_1 ..d_{m-1} ; w_1 ......w_m) \end{aligned} \qquad (4)$$

In Bigram language modeling, each word is assumed to depend only on its previous word in a first order Markov chain (Huang et.al., 2001), i.e.

$$P(d_1 d_2 .....d_m \mid w_1 w_2 ......w_m) = P(d_1 \mid w_1) \prod_{t=2}^{m} P(d_t \mid d_{t-1} ; w_{t-1} w_t) \qquad (5)$$

The search for the best sequence of diacritized words which maximizes (3) can be illustrated with the help of Figure 1. The shown finite state diagram can be viewed as a Hidden Markov Model (HMM), where the observation sequence is the undiacritized word sequence *W*, while the possible diacritized words, $v_{t,j}$, of each word w_t represent the hidden states. The problem can then be formulated as finding the best state sequence given the observation *W*. The solution of this problem is usually approximated using Viterbi Algorithm VA, and can be seen as an application of dynamic programming for finding a maximum probability path in a graph with weighted arcs. The computation proceeds in a column-wise manner, at every input word index *t,* it updates the scores of the nodes in a column by means of recursion formulas, which involve the value of the probability of the best paths ending at the previous column, and the transition probabilities of the words.

Figure 1. HMM states corresponding to the word sequence W.



Let us define $\phi(t,i)$ to be the probability of the most likely partial state sequence or path until time t, and ending at the i^th state (the i^th diacritized word corresponding to w_t.). The algorithm proceeds in the following steps:

Step 1: Initialization

$$\phi(1,i) = P(v_{1,i} \mid w_1) \tag{6}$$

Step 2: Induction

Let k_t be the index of the word w_t in $\mathbf{L_U}$, i.e. $w_t = \mathbf{L_U}(k_t)$; and let $n_v(t)$ be the cardinal of the subset V_{k_t}.

$$\phi(t,i) = \max_j \{\phi(t-1,j)P(v_{t,i} \mid v_{t-1,j}; w_{t-1} w_t)\}, \quad j = 1,2,...,n_v(t);\ \text{and}\ t = 2,3,..M \tag{7}$$

$$U(t,i) = \arg\max_j \{\phi(t-1,j)P(v_{t,i} \mid v_{t-1,j}; w_{t-1} w_t)\}, \quad j = 1,2,...,n_v(t);\ \text{and}\ t = 2,3,..M \tag{8}$$

Step 3: Best Path

$$U(M, i_{best}) = \arg\max_{j} \{\phi(M, j)\} \quad j = 1,2,...,n_v(M) \tag{9}$$

Step 4: Back Tracking

$$\begin{aligned} &i_M = i_{best} \\ &d_t = v_{t,i_t}, \text{ and } i_{t-1} = U(t, i_t); \text{ for } t = M, M-1,......2 \\ &d_1 = v_{1,i_1} \\ &D = d_1 d_2 ....d_M \end{aligned} \tag{10}$$

By keeping track of the state *j* giving the maximum value in the recursion formulas (9) and (10), it is possible, at the end of the input sequence, to retrieve the states visited by the best path. The Viterbi algorithms have a time complexity $O(M * K_u^2)$, where *M* is the length of the input sentence, and K_u is the average number of the states corresponding to the words in the input sequence.

All the probabilities needed for computing the Viterbi recursion can be obtained from the statistics of training sets, and stored for on line Viterbi calculations. The probabilities are basically the unigram and bigram that are derived from the frequency of occurrence of individual words or frequency of occurrence of joint words.

The Training Database

The system should be trained based on domains of knowledge, e. g. sports, weather, local news, international news, business, economics, religion, etc. For testing purpose we started by a fully diacritized transcript of The Holy Quran (HQ). HQ's list file consists of 78,679 words and 607,849 characters with no spacing. The text is converted to a word list after removing numbers, special symbols, and the Arabic letter extension character. A word is defined here to be any sequence of letters and diacritical marks delimited by the space character or a punctuation mark. Next, a table of the vocabulary and their number of occurrence is constructed. Two words $A = a_1 a_2 ... a_{m_a}$ and $B = b_1 b_2 .... b_{m_b}$ are considered identical in the regular sense if $R(A, B) = 1$, where $R(A, B)$ is defined as follows:

$$R(A,B) = \begin{cases} 1 \text{ if } m_a = m_b \text{ and } a_i = b_i \text{ for } i = 1,2,...,m_a \\ \qquad\qquad 0 \quad \text{otherwise} \end{cases}$$

One problem with the above definition of word similarity is that the S*ukoon* diacritical mark does not consistently appear explicitly in the text. A metric is designed so that two words are still considered identical even if one of them is missing one or more S*ukoon*. Define the mapping $S(.) : \mathbf{L_V} \rightarrow \mathbf{L_V}$ which strips words from S*ukoon* diacritical marks. Let $S(A) = A^0$, and $S(B) = B^0$. Then two words *A* and *B* are said to be R^0 identical, $R^0(A,B)=1$, if $R(A^0,B^0)=1$. When generating $\mathbf{L_V}$, all words in $\mathbf{T_V}$ which are R^0 identical are represented by a single word in the vocabulary $\mathbf{L_V}$. Next, each word in the table $\mathbf{L_V}$ is mapped to its undiacritized base $\mathbf{L_U}$. A database is generated which contains the undiacritized word bases, and a list of the corresponding diacritized words and their corresponding counts. Next, we generated a table of the word bigram. In this case, two two-word sequences are considered identical if their corresponding words are R^0 identical. The data base generated is called “bigram_db”.

Another problem in creating the training databases is that there are many articles, and common short words that are not fully diacritized. The missing diacritical marks represent a serious problem. At the minimum it unnecessarily increases the size of $\mathbf{L}_\mathbf{V}$, and creates ambiguity as it would not be clear if the missing diacritics is simple S*ukoon* or not. The problem is partially alleviated by creating a lexicon of exception cases of common words and articles which usually appear partially or totally undiacritized.

Experimental Results

The vocabulary list came to 18,623 diacritized words, composed of 179,910 characters. The corresponding table of base words is made up of 15,006 words consisting of 80,864 characters. The maximum number of *tashkeel* words for any undiacritized word was found to be 12. The test set consists of 50 randomly selected sentences from the entire Quran text. The test set contains 995 words and 7657 characters. Initially, when applying the Viterbi algorithm in its basic form, it produces 230 errors in letter diacritization in 4234 undiacritized characters, that is 5.43% errors in diacritic marks of letters. The analysis of these errors is important to explore future directions of improving the basic algorithm.

The first class of errors turned out to be due to inconsistent representation of *tashkeel* in the training data bases. Examples of these cases are (الَّا،الأَ), (لا،لّا،لا). To overcome these errors, the training set is preprocessed to normalize these cases, and to insure consistent diacritization of words. Then we test the algorithm after preprocessing the training set and test set, the number of errors generated by the algorithm came down to 175 errors, or about 4.1% error rate in the diacritical marks restoration of letters.

The second class of errors is caused mainly by a few articles and short words, and accounts for 41 cases, e.g., (إنَّ , إن) ، (مَن ، مِن). The ambiguity in determining the proper form of these short words could hopefully be resolved by using higher order grams, and restricting some articles to a single diacritized form. The third class of errors occurs in determining the end cases of words. This class accounts for 94 errors. The formal approach to resolve these cases is to implement a syntax analyzer. The syntax processing can be inserted as a post processing stage after the Viterbi algorithm. It was noticed that a considerable number of these end-case errors were repeated, and occurred in a few frequently used words. Accordingly, a simple approach to reduce the number of end case errors could be devised based on higher order grams for those most frequently used words. For example, the error in resolving the end cases of the word *Allah* الله accounts for 15 errors. The use of a trigram or 4-gram for such frequent words is expected to resolve the majority of these cases. The rest of the cases are more difficult to resolve and may require higher order grams to resolve, or a post processing stage of the resulting diacritized text using knowledge-based morph-syntax word correction.

In summary, the basic HMM approach achieves 4.1% letter error rate. The letter error rate could be further reduced to about 2.5 % by using a preprocessing stage, and using trigram for a selected number of short words and the most frequent words. Elimination of the rest of the errors may require a syntax analyzer and other more involved natural language processing.

Another important issue is that the proposed algorithm assumes that all the words in the given undiacritized word sequence *W* exist in $\mathbf{L}_\mathbf{U}$. If a word is not in the list, the program may fail to give any reasonable estimation of the word sequence. Similar statistical methods to generate a word with full diacritical marks can be developed based on knowledge of the unigram, bigram, and trigram of the letter sequence. When we applied Viterbi algorithm to restore the diacritical marks of individual words based on the letter statistics, it gave success rate less than 72% .

The algorithm presented in this paper assumes as well that the input word sequence is totally undiacritized. However, in reality, the input text may contain partial diacritization. The algorithm needs to be modified to take into consideration the presence of partial diacritization to improve the efficiency of the algorithm and enhance its performance.

Conclusion

The paper proposes the use of HMM approach to solve the problem of automatic generation of the diacritical marks of the Arabic text. The basic form of the algorithm achieves an error rate of about 4.1%. The use of a preprocessing stage and trigrams for selected number of words and articles may improve the performance to about 2.5% error rate. Further improvement may require some knowledge-based tools involving morphology-syntax analysis.

Acknowledgment

The authors would like to acknowledge King Abdulaziz City for Science and Technology and King Fahd University of Petroleum and Minerals for their support.

References

El-Barhamttoushi, H. M. (2002), "Arabic speech lexical engine", International Conference on Intelligent Computing & Information Systems. 1st. Cairo,Egypt, Jun 24-26.

El-Saadani, T. A. Hashish, M. A, (1988), "Semiautomatic vowelization of Arabic verbs ", Egyptian Computer ... p88-93.

Elshafei , Moustafa, Husni Al-Muhtaseb, Mansour Al-Ghamdi, (2002)" Techniques for high quality Arabic speech synthesis", Information Sciences 140(3-4),pp. 255-267.
Farghaly, A. and J. Snellar (2003),"Intuitive Coding of the Arabic Lexicon", SYSTRAN, MT, Summit IX Workshop, Machine Translation for Semitic Languages: Issues and Approaches, New Orleans, Louisiana, U.S.A.

Huang, X., Acero, A., and Hon, H. (2001), *"Spoken Language Processing"*, Prentice Hall PTR, New Jersey, USA.

Khoja, Shereen. (2001)," APT: Arabic Part-of-speech Tagger", Proceedings of the Student Workshop at the Second Meeting of the North American Chapter of the

Association for Computational Linguistics (NAACL2001), Carnegie Mellon University, Pittsburgh, Pennsylvania. June 2001.

Mustafa, Suleiman Hussein (1998), "Arabic string searching in the context of character code standards and orthographic variations", Computer Standards & Interfaces, Volume 20, Issue 1, 16 November, pp. 31-51

Shatta, Usama, (1994), "A systemic functional syntax analyzer and case-marker generator for speech acts in Arabic", International Conference for Statistics, Computer Science, Scientific & Social Applications. 19th. Cairo, Egypt, Apr 9-14.

Sulttan, H, (2001), "Automatic Arabic diacritization using neural networks", Scientific Bulletin of Faculty of Engineering Ain-Shams University : Electrical Engineering. v36 n4 pt2 pp 501-510.

Trost, Harald (1991), "Recognition and generation of word forms for natural language understanding systems. Integrating two-level morphology and feature unification", Applied Artificial Intelligence, v 5, n 4, October, pp. 411-457

[1] http://www.rdi-eg.com/rdi/Research/Research.asp
[2] http://www.sakhr.com/
[3] http://www.cimos.com/