Arabic Diacritization: Stats, Rules, and Hacks

Kareem Darwish **Hamdy Mubarak** **Ahmed Abdelali**
Qatar Computing Research Institute,
HBKU, Doha, Qatar
{kdarwish,hmubarak,aabdelali}@hbku.edu.qa

Abstract

In this paper, we present a new and fast state-of-the-art Arabic diacritizer that guesses the diacritics of words and then their case endings. We employ a Viterbi decoder at word-level with back-off to stem, morphological patterns, and transliteration and sequence labeling based diacritization of named entities. For case endings, we use Support Vector Machine (SVM) based ranking coupled with morphological patterns and linguistic rules to properly guess case endings. We achieve a low word level diacritization error of 3.29% and 12.77% without and with case endings respectively on a new multi-genre free of copyright test set. We are making the diacritizer available for free for research purposes.

1 Introduction

Modern Standard Arabic (MSA) text is typically written without diacritics (short vowels). During reading, Arabic readers need to reintroduce diacritics to disambiguate words and to sound them correctly given their context. Diacritics play two main roles depending on their position in the word. The final vowel in a word – case ending – which typically appears on the last letter of the stem indicates its syntactic role; while diacritics on the rest of the letters indicate the lexical choice. The role of diacritics is crucial for some applications such as text-to-speech (TTS) and can be beneficial in other Arabic processing steps such as part-of-speech (POS) tagging and word sense disambiguation. A single word can have multiple meaning given the different diacritized forms. The word "Elm"[1] could be viewed as a homograph for different lexical items. The word can be diacritized in the following ways: "Eilom" (knowledge/science), "Ealam" (flag), "Ealima" (he knew), "Eulima" (it was known), "Eal∼ama" (he taught), etc. Beginners learning Arabic get accustomed to diacritics and gradually learn to disambiguate the text without them. For final diacritics, they can help disambiguate the syntactic roles of words in sentences. Consider the syntactically ambiguous sentence: "r>Y AlmElm Altlmy∗". It most likely means: "The teacher saw the student" resulting in the diacritized version "ra>Y AalomuEal∼imu Aalt∼ilomy∗a", because Arabic prefers placing the subject ahead of the object. However, Arabic allows subjects and objects to switch positions, and hence the meaning might be "The student saw the teacher" with diacritized form "r>Y AalomuEal∼ima Aalt∼ilomy∗u".

In this paper we present a new state-of-the-art Arabic diacritizer. The diacritizer works in two cascaded steps: First, it guesses the diacritics for the core of words – disambiguating lexical choice; and then it guesses case endings – disambiguating syntactic roles. For the first step, we employ a Viterbi decoder at word-level with back-off to stem, morphological patterns, and transliteration and sequence labeling based diacritization of named entities. For the second, we employ SVM-based ranking coupled with morphological patterns and linguistic rules to properly guess case endings.

The contributions of this work is as follows:

- We introduce a back-off scheme where OOV words are diacritized using their morphological patterns (a.k.a stem templates)
- We use transliteration mining coupled with sequence labeling to guess diacritics on Arabic words based on how the words are transliterated in English.

[1]Buckwalter encoding is used exclusively in the paper.

- We employ SVM-based ranking to guess the appropriate cased endings.
- We use morphological patterns and linguistic rules to better guess case endings.
- We offer a new copyright-free multi-genre test set to measure diacritization accuracy.
- We offer a state-of-the-art Arabic diacritizer that is coded entirely in Java and can be used freely for research purposes.

2 Background

2.1 Arabic Diacritization

Significant research has addressed diacritic restoration/recovery or diacritization for Arabic and some other Semitic languages which are typically written without short vowels. Diacritization is essential for a variety of applications such as TTS as well as educational tools for language learners.

In earlier attempts for automatic diacritization, Gal (2002) used a Hidden Markov Model for diacritics restoration, tested on the Quran text, and achieved 14% word error rate (WER). Vergyri and Kirchhoff (2004) used acoustic features in conjunction with morphological and contextual constrains to train a diacritizer. They evaluated their automatic diacritization system on two corpora, namely FBIS and LDC CallHome ECA, and reported a 9% diacritics error rate (DER) without case ending, and 28% DER with case endings. The difference between WER and DER is that the latter measures the percentage of letters with incorrect diacritics. Consequently, DER values are typically lower than WER values. Nelken and Shieber (2005) used a cascade of probabilistic finite state transducers trained on the LDCs Arabic treebank news stories (Part 2). The corpus consists of 501 news stories collected from Al-Hayat newspaper with a total of 144,199 words. The cascade included a word-based language model, a letter-based language model, and a morphological model. This combination of probabilistic models achieved an accuracy of 7.33% and 23.61% WER without and with case ending respectively. Zitouni et al. (2006) trained a maximum entropy model for sequence classification to restore diacritics for each character in a word. For training, they used the LDCs Arabic Tree-bank (Part 3, version 1.0) diacritized corpus, which includes complete vocalization (full diacritics including case ending) for each word. The corpus is composed of 600 documents from the An-Nahar Newspaper with a total of 340,281 words. The maxEnt system achieved 5.5% DER and 18% WER on words without case ending. Habash and Rambow (2007) presented "MADA-D" a system that combines a tagger and a lexeme language model. The system showed that the morphological tagger along with a 3-gram language model were able to achieve the best performance of 5.5% and 14.9% WER respectively for diacritized words without and with case ending. We compare to their system in our paper. Rashwan et al. (2009) introduced a two layers stochastic system to automatically diacritize Arabic text. The system combines both morphological knowledge and the word full form features. These information is exploited through a maximum marginal probability on the full form supplemented with linguistic factorization model based on morphological analysis and POS tagging. While the first is fast, the second one used as a fall back is more accurate as it exploits more inner knowledge from the word parts and the context. Later work by Rashwan et al. (2015) used deep learning to improve diacritization accuracy and they reported a WER of 3.0% without case ending and 9.9% WER for guessing case ending. We compare to their system in our paper. Recent work by Abandah et al. (2015) uses recurrent neural networks to improve diacritization, and they report results that are better than those of MADA. Bebah et al. (2014) developed a hybrid approach that utilizes the output of the open source morphological Analyzer AlKhalil Morpho System (Mohamed Ould Abdallahi Ould et al., 2011) which outputs all possible diacritization for each word analyzed out of context. These outputs were then fed to an HMM to guess the correct diacritized form. The system was trained on a large body of text consisting of 2,463,351 vowelized words divided between NEMLAR corpus (460,000 words), Tashkeela corpus (780,000 words), and RDI corpus (1,223,351 words). The training was carried out with 90% of a corpus and the remaining 10% composed of 199,197 words was used for testing. The system achieved 9.93% WER.

For more indepth survey on relevant work on Arabic diacritization, Azmi and Almajed (2015) provide a comprehensive survey. To the exception of Belinkov and Glass (2015) which performed lower than the state-of-the-art when using a recurrent neural network for diacritization, traditionally, works on diacritic restoration rely on linguistic features and tools. MADA (Morphological Analysis and Disambiguation for Arabic) (Habash et al., 2009) and its successor MADAMIRA (Pasha et al., 2014) are among the few available tools for diacritization. The core of MADA – MADAMIRA as well – utilizes morpho syntactic features to select a proper analysis from a list of potential analyses provided by the Buckwalter Arabic Morphological Analyzer (BAMA) (Buckwalter, 2004). Using a set of SVMs trained on the Arabic Penn Treebank, MADAMIRA selects the most probable features. The analysis MADAMIRA selects consists of the diacritized form along with other features such as its lexeme, its morphological features, and English glossary.

2.2 ATB

The availability of diacritized text for training is crucial. Much of the previous work on diacritization relied on using the Arabic Penn Treebank (ATB). Though ATB is invaluable for many tasks such as POS tagging and parsing, it is sub-optimal for diacritization for the following reasons:
1. ATB is limited in terms of size with less than 570k tokens and in terms of diversity with 87,160 unique surface forms (excluding numerals). In comparison, the AFP news corpus has approximately 765,890 unique tokens (Cole et al., 2001). These limitation would lead to poor coverage.
2. ATB often uses inconsistent diacritizations. For example the word "<nh" appears 27 as "<in∼ahu" and 37 as "<inhu" where the first is the correct one. Also, the diacritic sukon "o" is sometimes omitted in ATB (ex. "kwbnhAgn" is diacritized as: "k**u**wb**i**n**o**hAg**i**n"); and default diacritics preceding long vowels are optionally used.
We thus used a large diacritized corpus that we describe in Section 3.1. When comparing with systems that were trained on the ATB, some preprocessing is required, as we show later, to make sure that we are not unfairly penalizing them.

3 Our Diacritizer

The diacritizer has two main components. The first component recovers the diacritics for the core word (i.e. word without case ending), and the second only recovers the case ending. In this section we describe: the training and test corpora we used and how we processed them; the training of our system that diacritizes core-words and guesses case ending; and our results compared to those of other systems that are described in the literature, and some relaxations we applied during evaluation to insure fair comparison.

3.1 Training and Test Corpora

For this work, we acquired a diacritized corpus from a commercial vendor containing more than 9.7 million tokens with approximately 194k unique surface forms (excluding numbers and punctuation marks). The corpus is composed mostly of modern standard Arabic (approximately 7 million words) and covers many genres including politics, economics, sports, science, etc., and the remaining 2.7 million words are mostly religious text in classical Arabic. Thus the corpus is well balanced and is considerably larger than the ATB. We manually checked random samples from the corpus and we estimate diacritization errors to be less than 1%, and diacritization is thorough with no omissions of sukon or optional diacritics. We used the corpus to build several resources as we describe in Section 3.2.

For testing, we used a new test set composed of 70 WikiNews articles (majority are from 2013 and 2014) that cover a variety of themes, namely: politics, economics, health, science and technology, sports, arts, and culture. The articles are evenly distributed among the different themes (10 per theme). The articles contain 18,300 words. We compare our results to three different systems that are described in the literature, where the authors were kind enough to either diacritize our test set or provide a working system. The systems were those of Rashwan et al. (2015), Belinkov and Glass (2015), and MADAMIRA (Habash et al., 2009; Pasha et al., 2014). The first two are recent systems and MADAMIRA is a popular tool for processing Arabic.

3.2 Data Preparation

Given a word in the diacritized corpus, we produce multiple representations of it. To illustrate the representations, we use the word "wakitAbihimo" (and their book) as our running example. 1. diacritized surface form ("wakitAbihimo"). 2. diacritized surface form without case ending. To remove case endings, we segment each word in the corpus to its underlying clitics using the Farasa segmenter (Darwish and Mubarak, 2016). For example, given the diacritized word "wakitAbihimo" (and their book), it would be segmented to the prefix "wa", stem "kitAbi", and suffix "himo". The

case ending which is attached in this case to the stem is removed, leading to "kitAb" and the full surface form "wakitAbhimo". A few things are noteworthy here, namely that diacritized suffixes are often affected by the case-ending (ex. if the case ending in the this example was "u", the suffix would have been "humo"), and the case ending may be on the attached noun suffix (ex. "p" or "At") and not on the last letter in the stem.
3. diacritized stem with and without case ending, which would be 'kitAbi" and 'kitAb" respectively for our example. If the surface form had a noun suffix, then it was considered as part of the stem. For example, the word "kitAbAt" (writings) is segmented as "kitAb+At", where "At" is the noun suffix indicating feminine plural marker. In this example, the noun suffix is treated as if it is a part of the stem.
4. diacritized template of surface form. This step involves obtaining the root from which the word is derived and the stem template. Arabic words are typically derived from a set of a few thousand roots by fitting the roots into stem templates. For example, the word "kitAb" is derived by fitting the root "ktb" into the template "**fiEAl**". In our example, the surface form template is "wa**fiEAl**ihimo" and "wa**fiEAl**himo" with and without case ending respectively. We again used the Farasa to obtain the roots and stem templates.
5. diacritized stem template with and without case ending. As shown before for the example, the stem template is "**fiEAl**" and "**fiEAli**" with and without case ending respectively.
Based on different representations, we created the following dictionaries and language models:
1. surface form dictionary, which contained surface forms without diacritics and seen diacritized forms without case ending. For example, the undiacritized word "wktAbhm" has two seen diacritized forms, namely "wakitAbhimo" and "wakitAbhumo". In this example, the diacritized form of the suffix actually depends on the case ending. We have a module that corrects for this when placing the case ending.
2. stem dictionary, which contains the stem without diacritics and seen diacritized forms without case ending. From our example, the stem "ktAb" has the diacritized forms "kitAb" and "kut~Ab".
3. surface form and stem templates without diacritics along with the most common (based on statistics on the training corpus) diacritized templates. In our example, the template "w**fEAl**hm" would be mapped to "wa**fiEAl**ihimo" and the tempalte "**fEAl**" to "**fiEAl**".
4. a bigram surface form langauge model and a unigram stem language model. Both are without case ending.

3.3 Core word diacritization

For core-word diacritic recovery, our basic system uses a bigram word model with back-off to a stem unigram model, stem template-based diacritization, and a sequence labeling based stem diacritization model. We experimented with a trigram language model instead of a bigram model, and the bigger model did not yield any gain.

3.3.1 Baseline System

In our baseline system, we used the bigram language model using the word surface forms without case endings. When given a sentence to diacritize, we would build a lattice with all previously seen diacritized forms for a word by consulting the previously constructed dictionary. If a word is not seen before, it is left as is without any diacritics. Then we use our bigram model and the Viterbi algorithm to find the most likely path in the lattice.

In the following setups, different back-offs are used when a word is not found in the dictionary.

3.3.2 Back-off to Stem

For a surface form that is not found in the surface form dictionary, the surface form is segmented and the stem is extracted. Then we search for the stem in the stem dictionary, and we use the most probable stem diacritization using the stem unigram language model. If found, then the prefixes are diacritized and attached to the stem. If not, the input surface form is used. To attach the prefixes and suffixes to the stems, some affixes have one form (namely "w" (and), "f" (then), "s" (will), "b" (with), and "l" (to)) and others change form depending on the stem and the case ending. Those that depend on the case ending (some attached pronouns) are diacritized properly after we determine the case ending. One prefix that changes form and affects the diacritic of the first letter in the word is "Al" (the). Arabic has so-called *lam qamariyyah* (literally meaning "moon *l*") and *lam shamsiyyah* ("sun *l*"), where the former is pronounced normally and the later is not pronounced with the first letter after it being stressed with a "$\sim$" (*shaddah*). An example of both are the words "Aloqamar" (the moon) pronounced as *alqamar* and "Al\$$\sim$amos" (the sun) pronounced as *ashams*.

In the case of lam qamariyyah, it receives a "o" diacritic, and the lam shamsiyyah case it is left bare.

3.3.3 Back-off based on Template

If the surface form and stem do not appear in our dictionaries, we attempt to diacritize using word and stem templates. To do so, we generate the equivalent undiacritized template for the surface form. For example, given the word "wsEAlhm" (and their coughing), we find that the underlying root is "sEl" with "wfEAlhm" and "fEAl" being the undiacritized surface form and stem templates respectively. If the surface form template is found in the previously constructed surface form template dictionary, the most likely diacritized template is used. In this example, this would be "wafiEAlhimo". If not, then we search the stem template dictionary and use the most likely diacritized template ("fiEAl" for our example).

3.3.4 Automatically Diacritized Transliterated Words

One of the important sources of words for which no diacritization candidates exist in our dictionaries and for which we can not obtain valid templates are foreign names. We devised a scheme to automatically diacritize transliterated words using transliteration mining and sequence labeling. The intuition for automatic diacritization draws from the fact that while short vowels are generally omitted in Arabic text, English vowels are often explicit. For example, the name "klntn" is written in English as "Clinton". The vowels on the English side imply that the proper Arabic diacritization is "kolinotun".

The automatic diacritization process has multiple steps. First, we need Arabic and English transliterations, which we could obtain from Wikipedia cross-lingual links. To do so, we used the Arabic Wikipedia snapshot from September 28, 2012 that has 348,873 titles including redirects, which are alternative names to articles. Of these articles, 254,145 have cross-lingual links to English Wikipedia. To find which words in English and Arabic titles would be transliterations, we used a transliteration miner akin to that of El-Kahky et al. (2011) that was trained using 3,452 parallel Arabic-English transliteration pairs. We aligned the word-pairs at character level using GIZA++ and the phrase extractor and scorer from the Moses machine translation package (Koehn et al., 2007). The alignment produced mappings between English letter sequences and Arabic letter sequences with associated mapping probabilities. Given a word in the Arabic title, we produced all its possible segmentations along with their associated mappings into English letters. We retained valid target sequences that produced a word in the corresponding English title. We further filtered out pairs where the transliteration probability was less than 0.1, obtaining 125k transliterated pairs.

Second, we trained a sequence labeler that would automatically assign the proper diacritic for each Arabic letter in a name given its English rendition. We constructed a training set of 500 Arabic-English transliteration pairs, where the Arabic is fully diacritized. For each pair, we used our transliteration miner to automatically map each Arabic letter to one or more English letters. Then given these mappings, we trained a conditional random fields (CRF) model using the CRF++ implementation (Kudo, 2005) using the following features: Arabic Letter, English mapping, is first letter in word, is last letter in word, and English is all vowels. The label was the observed diacritic.

Third, given all the transliterations we found from Wikipedia titles, we used the trained CRF model to automatically diacritize the Arabic words. In doing so, we were able to automatically diacritize more than 68k Arabic transliterations. We were not able to diacritize all of them because the transliteration miner was not able to fulfill the requirement that each Arabic letter was to be mapped to one or more English letters. An example diacritized name is "rAwlobinody" (Rawalpindi), which does not exist in the diacritization dictionary. We took a sample of 200 words that were automatically diacritized in this fashion and the accuracy of diacritization was 79%. Perhaps in the future we can utilize more training data to further improve the accuracy.

3.3.5 Word Look-up

Another method that we employed entails building a dictionary of words that we reckoned had only one possible diacritization or one that dominated all others. An example of this is the word "En", which has two diacritized forms namely "Eano" (about) and "Ean~a" (appeared). The second diacrized form, though possible, is archaic. We constructed a dictionary of such words using two methods. First, we manually constructed a set of about 393 Arabic prepositions, particles, and pronouns with and without prefixes and suffixes. Example entries in the dictionary include "fy" (in), "wfy" (and in), "fyhm" (in them), and "wfyh" (and in it). In the second, we collected statistics on the training corpus, and any word appearing at least 10 times with a diacritized form being used more than 90% of the time, we added the word and the dominating diacritized form to our dictionary. Picking the dominant form is mostly safe particularly in case such as words that start with prepositions. In other cases such as "gzp" (Gaza), the dominant form is "gaz∼apa", while we know that it is possible for it to be diacritized as "gaz∼apu" (with a different case ending). However, what we observed in our corpus is that it overwhelmingly appears as part of the collocation "qTAE gzp" (Gaza Strip), and the collocations forces "gzp" to have a specific diacritized form.

3.3.6 Results

Table 1 shows the results for the baseline system, the baseline with different back-off schemes, and the baseline with back-off and word look-up. We used the two common evaluation measures, namely error rate at word level (WER) and error rate at character/diacritic level (DER). As the results show, backing-off to using stems and word or stem templates, using automatically diacritized words, and using word look-up all had a positive effect on diacritizing core words. The most improvement was achieved when backing-up to stems and using word look-up. Automatically diacritized words had a smaller effect than we expected, because foreign named entities frequently use long vowels that are inserted during transliteration instead of short vowels. For example, "John" is typically transliterated as "jwno" (with the long vowel "w") rather than "juno". Combining the different enhancements led to even greater drops of 50% and 56% in WER and DER respectively. Compared to systems in the literature, our core word diacritizer is far ahead of MADAMIRA and that of (Belinkov and Glass, 2015). However, we lag behind the system of Rashwan et al. (2015). When score the output of different systems, we performed some relaxations to account for differences in diacritization conventions. The relaxations involved: removing default diacritics on letters followed by long vowels as in "jwno" where putting the diacritic "u" after "j" would be redundant; assuming that a letter, that is not a long vowel, with no diacritic to be followed by "o"; and removing diacritics from the determiner "Al".

System	% WER	% DER
B	6.64	2.40
S	4.69	1.44
T	5.96	1.90
TSL	6.56	2.39
WL	4.54	1.75
S+T+TSL	4.51	1.35
S+T+TSL+WL	**3.29**	**1.06**
MADAMIRA	6.73	1.91
Rashwan et al. (2015)	***3.04***	***0.95***
Belinkov and Glass (2015)	14.87	3.89

Table 1: Core word diacritization results for Baseline (B) and with back-off to Stem (S), Template based diacritization (T), and Transliteration and Sequence Labeling based diacritization (TSL), and Word look-up (WL).

3.4 Recovering Case Endings

Though the case ending of some words or word types might be fixed, such as prepositions and past tense verbs, case ending often depends on the role a word plays in the sentence. Consider the following examples: "*ahaba muHam∼adN" (Muhammad went) and "*ahaba >lY muHam∼adK" (he went to Muhammad). The name "muHam∼ad" is assigned the case "N" (nominative) when it is a subject and "K" (genitive) when it is preceded by a preposition ">lY". Ascertaining case endings for simple prepositional phrases may be easy, however determining if a word is a subject or an object may be much hard. Though parsing can help determine the role of words in a sentence, it is typically very slow and hence impractical for some real life applications such as TTS.

To determine case endings, we use a linear SVM ranking (SVM^{rank}) model that is trained on a variety of lexical, morphological, and syntactic features. The advantage of an SVM^{rank} model is that it is very fast, and it has been shown to be accurate for a variety of ranking problems (Joachims, 2006). For training, we use the implementation of Joachims (2006) with default parameter values. We also employ heuristics that include/exclude possible/impossible case ending to be considered by the SVM^{rank} model. In the following, we describe our ranking model and how we determine which case endings to consider.

3.4.1 SVM^{rank} Model

Here we describe the features we used for SVM^{rank} to determine the case endings of words.

We use the Farasa segmenter and POS tagger (Darwish and Mubarak, 2016) to determine some of the morphological, lexical, and syntactic features. To help explain the features, we use here the running example "jrY Abn AljyrAn fy AlmlEb" (the neighbors' son ran in the playground), where we will focus on assigning the case ending *C* of the word "AljyrAn" (the neighbors), which is segmented as "Al+jyrAn" and POS tagged as "DET+NOUN$_{masculine,plural}$" by Farasa. All feature values are computed on our training corpus. Position "0" is the current position, with "1" and "-1" being the next and previous positions. The features are as follows:

- $P(C|word_0)$ and $P(C|stem_0)$: this captures the likelihood that case ending can appear with the current word or stem respectively (ex. $P(C|AljyrAn)$ and $P(C|jyrAn)$).
- $P(C|word_{-1})$, $P(C|stem_{-1})$, $P(C|word_1)$, and $P(C|stem_1)$: this captures the context of the word (ex. $P(C|Abn)$, $P(C|Abn)$, $P(C|fy)$, and $P(C|fy)$).
- $P(C|POS_{word_0})$ and $P(C|POS_{stem_0})$: some case endings are likely for some POS tags and nearly impossible for others. For example, "DET" precludes tanween {"N", "K", "F"} (ex. $P(C|POS_{DET+NOUN})$ and $P(C|POS_{NOUN})$)
- $P(C|POS_{word_{-1}})$, $P(C|POS_{stem_{-1}})$, $P(C|POS_{word_1})$, and $P(C|POS_{stem_1})$: this captures local contextual information that may help in detecting some syntactic constructs (ex. $P(C|POS_{DET+NOUN})$ and $P(C|POS_{NOUN})$)
- $p(C|gen/num_{word_0})$, $p(C|gen/num_{word_{-1}})$, and $p(C|gen/num_{word_1})$: gender and number agreement may indicate dependency within a sentence (ex. $p(C|masc\_pl)$, $p(C|masc\_sing)$, and $p(C|null)$[2]).
- $p(C|prefix_0)$, $p(C|POS_{prefix_0})$, $p(C|suffix_0)$, and $p(C|POS_{suffix_0})$: prefixes and suffixes may force or preclude certain case endings (ex. $p(C|Al)$, $p(C|DET)$, $p(C|null)$, $p(C|null)$).
- $p(C|stem\_template_0)$: Some stem templates favor particular case endings over others. For example, the template "m**f**A**E**y**l**" (as in "msAHyq" (powders)) does not allow tanween (ex. $p(C|fElAn)$).
- $p(C|last\_letter_{word_0})$: the last letter in the word may force or preclude certain case endings. For example, "Y" does not accept any diacritics. For our running example, we compute $p(C|n)$.
- $p(C|POS_0, POS_{word_{-1}}, diacritic_{word_{-1}})$: case endings are affected by the diacritic of the previous word in the case of NOUN-ADJ constructs (ex. $p(C|DET+NOUN, NOUN, u)$).
- $p(C|word, word_{-1})$: this helps capture collocations or local constructs such as short prepositional phrases (ex. $p(C|AbnAljyrAn)$).
- $p(C|POS, POS_{word_{-1}}, POS_{word_1})$ and $p(C|POS, POS_{word_{-1}}, POS_{word_{-2}})$: this can help detect local constructs such as NOUN-DET+NOUN which is often an idafa construct (ex. $p(C|DET + NOUN, NOUN, PREP)$ and $p(C|DET + NOUN, NOUN, V)$).

3.4.2 Filtering Heuristics

Limiting the possible case endings that the model has to rank can improve accuracy by disallowing bad choices. We applied simple heuristics to limit possible case endings to be scored by SVM*Rank*. Some of these are based on Arabic grammar, and others are purely statistical.

- If a word or stem appears more than 1,000 times in the training corpus, then restrict possible case endings to those seen in the training corpus. Though the threshold would not preclude other cases, it is high enough to make other cases rare.
- If the POS of the stem is VERB then restrict to {a, o, u, ∼a, ∼u, or null}.
- If stem POS is VERB and VERB is not present tense, then restrict to {a, o, ∼a, or null}.
- If first suffix is "wn" or "yn" (either plural noun suffix or plural pronoun), then restrict to {a}.
- If first suffix is "An" (dual noun suffix), then restrict to {i}.
- If stem POS is NOUN and more than 80% of time the case ending was "o" in the training corpus, then restrict to {o}. This is meant to capture foreign named entities, which by the convention in the training corpus always get a case ending of "o".
- If word diacritized using the aforementioned Transliteration mining and sequence labeling method, then it is assumed to be a foreign named entity and the case ending is restricted to {a} or {null} if it ends with a "A, y, w, or Y".

[2]prepositions don't have a gender or number

- If suffix POS is CASE, then restrict to {F, ∼F}.
- If word contains non-Arabic letters, then restrict to {null}.
- If the last letter in the stem is a long vowel "A, y, w, or Y", then restrict to {null}.
- If last letter in the stem is not a long vowel and is not followed by a long vowel, then disallow {null} to insure a case ending is chosen.
- If prefix POS has DET, then disallow tanween {K, N, F, ∼K, ∼N, ∼F}.
- If word or stem POS appeared in training more than 50 times, then restrict case ending to those seen for the POS during training.
- If word or stem template appeared in training, then restrict case ending to those seen for the template during training.
- If stem-template matches "fwAEl, mfAEyl, or mfAEl", then disallow tanween.
- If a bigram appeared more than 1,000 in training with a single case ending for both words in the bigram, then restrict to the seen case endings.
- If the prefix does not have a DET and the following word does, then assume an idafa construct and disallow tanween.
- If the word matches ">n, <n, or lkn", then restrict case ending to {o} only if followed by a verb, and to {∼a} otherwise.

3.4.3 Results

Table 2 shows the results of full diacritization including case ending with and without filtering heuristics. As can be seen, filtering heuristics led to 4.3% and 11.1% relative reduction in word and character error rates respectively. We are doing better than the other three systems. Though the system of Rashwan et al. (2015) was more accurate at core-word level, our system was doing a better job in case ending recovery leading to better overall diacritization.

System	% WER	% DER
SVM^{Rank}	13.38	3.98
SVM^{Rank}+Heuristics	**12.76**	**3.54**
MADAMIRA	19.02	5.42
Rashwan et al. (2015)	15.95	4.29
Belinkov and Glass (2015)	30.50	7.89

Table 2: Full word diacritization results of our system using SVM^{Rank} only (SVM^{Rank}) and SVM^{Rank} after using heuristics to include/exclude possible/impossible case endings (SVM^{Rank}+Heuristics).

3.5 Error Analysis

For error analysis, we inspected 500 random errors from the WikiNews test set. The following are those constituting more than 5% of the errors:
1. Core word diacritization errors: (21%) wrong form – ex. "Eadol" (justice) instead of "Ead∼al" (he adjusted); (8%) out of vocabulary – ex. "Eab∼ady" (Abbady – proper name).
2. Case ending errors: (12%) wrong ADJ-NOUN attachment – ex. "wzArp AlSHp AlSEwdyp" (Saudi Health Ministry) where Saudi was attached to health instead of Ministry; (14%) misidentification of SUBJ or OBJ – ex. OBJ is mistaken for a SUBJ because SUBJ is omitted, or SUBJ appears several words after VERB; (11%) words following conjunctions where their dependency is not clear; (7%) appositions; (6%) substitution of tanween with kasra or damma with kasra and damma respectively {u ↔ N, i ↔ K}; (6%) attachments in NOUN phrases with multiple subsequent nouns; and (5%) the SUBJ of a nominal sentence switches place with the predicate.

3.6 Implementation and Speed

The diacritizer is implemented entirely in Java making it platform independent and is backwards compatible to JDK 1.5. The diacritizer is packaged as a single jar file that is 100 megabytes in size. It is able to diacritize a test set composed of 500k words on an Intel i7 laptop with 16 gigabytes of RAM in 3 minutes and 44 seconds (including 42 sec. loading time) with memory footprint of 2.2 gigabytes. The current implementation is single threaded, so it does not make use of the multiple cores. We are providing it for free for research purposes.

4 Conclusion

In this paper we present a new state-of-the-art publicly available Arabic diacritizer. It uses a Viterbi decoder for word-level diacritization with backoffs to stems and morphological patterns. It also uses transliteration mining in conjunction with sequence labeling to diacritize named entities for which we have English transliterations. For case ending, it uses SVM^{Rank} coupled with filtering heuristics. The diacritizer achieves 12.76% WER and 3.54% DER on a new multi-genre free of copyright test set.

References

Gheith A Abandah, Alex Graves, Balkees Al-Shagoor, Alaa Arabiyat, Fuad Jamour, and Majid Al-Taee. 2015. Automatic diacritization of arabic text using recurrent neural networks. *International Journal on Document Analysis and Recognition (IJDAR)*, 18(2):183–197.

Aqil M Azmi and Reham S Almajed. 2015. A survey of automatic arabic diacritization techniques. *Natural Language Engineering*, 21(03):477–495.

Mohamed Bebah, Chennoufi Amine, Mazroui Azzeddine, and Lakhouaja Abdelhak. 2014. Hybrid approaches for automatic vowelization of arabic texts. *arXiv preprint arXiv:1410.2646*.

Yonatan Belinkov and James Glass. 2015. Arabic diacritization with recurrent neural networks. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 2281–2285, Lisbon, Portugal.

Tim Buckwalter. 2004. Buckwalter arabic morphological analyzer version 2.0. ldc catalog number ldc2004l02, isbn 1-58563-324-0.

Andy Cole, David Graff, and Kevin Walker. 2001. Arabic newswire part 1 corpus (1-58563-190-6). *Linguistic Data Consortium (LDC)*.

Kareem Darwish and Hamdy Mubarak. 2016. Farasa: A new fast and accurate arabic word segmenter. In *Proceedings of the Tenth International Conference on Language Resources and Evaluation (LREC 2016)*, Paris, France, may. European Language Resources Association (ELRA).

Ali El-Kahky, Kareem Darwish, Ahmed Saad Aldein, Mohamed Abd El-Wahab, Ahmed Hefny, and Waleed Ammar. 2011. Improved transliteration mining using graph reinforcement. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 1384–1393.

Ya'akov Gal. 2002. An hmm approach to vowel restoration in arabic and hebrew. In *Proceedings of the ACL-02 workshop on Computational approaches to Semitic languages*, pages 1–7. Association for Computational Linguistics.

Nizar Habash and Owen Rambow. 2007. Arabic diacritization through full morphological tagging. In *Human Language Technologies 2007: The Conference of the North American Chapter of the Association for Computational Linguistics; Companion Volume, Short Papers*, pages 53–56. Association for Computational Linguistics.

Nizar Habash, Owen Rambow, and Ryan Roth. 2009. Mada+ tokan: A toolkit for arabic tokenization, diacritization, morphological disambiguation, pos tagging, stemming and lemmatization. In *Proceedings of the 2nd International Conference on Arabic Language Resources and Tools (MEDAR), Cairo, Egypt*, pages 102–109.

Thorsten Joachims. 2006. Training linear svms in linear time. In *ACM SIGKDD-2006*, pages 217–226. ACM.

Philipp Koehn, Hieu Hoang, Alexandra Birch, Chris Callison-Burch, Marcello Federico, Nicola Bertoldi, Brooke Cowan, Wade Shen, Christine Moran, Richard Zens, et al. 2007. Moses: Open source toolkit for statistical machine translation. In *Proceedings of the 45th annual meeting of the ACL on interactive poster and demonstration sessions*, pages 177–180. Association for Computational Linguistics.

Taku Kudo. 2005. Crf++: Yet another crf toolkit. *Software available at http://crfpp. sourceforge. net*.

Bebah Mohamed Ould Abdallahi Ould, Abdelouafi Meziane, Azzeddine Mazroui, and Abdelhak Lakhouaja. 2011. Alkhalil morphosys. In *7th International Computing Conference in Arabic*, pages 66–73, Riyadh, Saudi Arabia.

Rani Nelken and Stuart M Shieber. 2005. Arabic diacritization using weighted finite-state transducers. In *Proceedings of the ACL Workshop on Computational Approaches to Semitic Languages*, pages 79–86. Association for Computational Linguistics.

Arfath Pasha, Mohamed Al-Badrashiny, Mona Diab, Ahmed El Kholy, Ramy Eskander, Nizar Habash, Manoj Pooleery, Owen Rambow, and Ryan M Roth. 2014. Madamira: A fast, comprehensive tool for morphological analysis and disambiguation of arabic. In *LREC-2014*, Reykjavik, Iceland.

Mohsen Rashwan, Mohammad Al-Badrashiny, Mohamed Attia, and Sherif Abdou. 2009. A hybrid system for automatic arabic diacritization. In *The 2nd International Conference on Arabic Language Resources and Tools*, pages 54–60.

Mohsen Rashwan, Ahmad Al Sallab, M. Raafat, and Ahmed Rafea. 2015. Deep learning framework with confused sub-set resolution architecture for automatic arabic diacritization. In *IEEE Transactions on Audio, Speech, and Language Processing*, pages 505–516.

Dimitra Vergyri and Katrin Kirchhoff. 2004. Automatic diacritization of arabic for acoustic modeling in speech recognition. In *Proceedings of the workshop on computational approaches to Arabic script-based languages, COLING'04*, pages 66–73, Geneva, Switzerland. Association for Computational Linguistics.

Imed Zitouni, Jeffrey S Sorensen, and Ruhi Sarikaya. 2006. Maximum entropy based restoration of arabic diacritics. In *Proceedings of the 21st International Conference on Computational Linguistics and the 44th annual meeting of the Association for Computational Linguistics*, pages 577–584. Association for Computational Linguistics.