Diacritic-Based Matching of Arabic Words

MUSTAFA JARRAR, Birzeit University, Palestine
FADI ZARAKET, American University, Lebanon
RAMI ASIA, Birzeit University, Palestine
HAMZEH AMAYREH, Birzeit University, Palestine

Words in Arabic consist of letters and short vowel symbols called diacritics inscribed atop regular letters. Changing diacritics may change the syntax and semantics of a word; turning it into another. This results in difficulties when comparing words based solely on string matching. Typically, Arabic NLP applications resort to morphological analysis to battle ambiguity originating from this and other challenges. In this paper, we introduce three alternative algorithms to compare two words with possibly different diacritics. We propose the *Subsume* knowledge-based algorithm, the *Imply* rule-based algorithm, and the *Alike* machine-learning based algorithm. We evaluated the soundness, completeness and accuracy of the algorithms against a large dataset of 86,886 word pairs. Our evaluation shows that the accuracy of Subsume (100%), Imply (99.32%), and Alike (99.53%). Although accurate, Subsume was able to judge only 75% of the data. Both Subsume and Imply are sound, while Alike is not. We demonstrate the utility of the algorithms using a real-life use case – in lemma disambiguation and in linking hundreds of Arabic dictionaries.

Natural language processing, Phonology / morphology, Language resources

Additional Key Words and Phrases: Arabic; diacritics; disambiguation

1. INTRODUCTION

Diacritics are distinguishing features of the Arabic language. Arabic words consist of both letters and diacritics. Changing the diacritics may change the semantics of a word. The problem is that most Arabic text is typically written without diacritics, which makes it highly ambiguous. Although several approaches have recently been proposed to automatically restore diacritics to Arabic text (i.e., words in context), there exist no novel solutions to treat words without the context. People either fully ignore diacritics, or sensitively consider and treat them as different characters. This makes string matching techniques problematic if used in Arabic. For example, it is difficult to automatically compare words, or to calculate a distance metric between words such as (إلهام،الهام), (رضّ،رض), (جَوَّع،جوع), (جزَّر،جزَر), or (جزر،جزَر), specially when the context is not provided. Diacritic-based comparison framework is important in many basic application scenarios, such as searching in MS Word or using conditions in MS Excel, Google Sheets, SQL queries, and many others. As will be illustrated in the real-life use case on dictionary integration presented later in this paper, we found it very challenging to link and map between dictionary entries, due to different diacritization of these entries.

Diacritics in Arabic have two main roles: (i) they provide a phonetic guide, to help readers recite/articulate the text correctly, and (ii) they disambiguate the intended

Author's addresses: M. Jarrar and R. Asia, and H. Amayreh, Computer Science Department, Birzeit University, 1 Almarj St., Ramallah, West Bank 627, Palestine; F. Zaraket, American University of Beirut, 1107 Riad El-Solh St. Beirut 2020, Lebanon.

DOI:http://dx.doi.org/10.1145/0000000.0000000

meaning of otherwise ambiguous words. Table 1 shows a list of the most used diacritics of the Modern Standard Arabic (MSA). The *fatha* and *kasra* diacritics show as accents above and below the corresponding letter and indicate short 'a' and 'i' vowels, respectively. The *dhamma* diacritic shows as an accent with a small circle and denotes a short 'o' vowel. A *sukoon* shows a small circle atop and denotes a silent diacritic-sound on the letter. A *shadda* is a gemination marker seen above a letter. It denotes stressing the letter such that the letter is pronounced twice: first as a silent letter and second with a non-*sukoon* diacritic. A *tanween* diacritic is an indefiniteness mark and shows as a double *fatha*, *kasra*, or *dhamma* diacritic. It denotes the letter spelled with the marked diacritic followed by a silent 'n' diacritic-sound.

Table 1: Basic diacritic table in Buckwalter

Diacritic Name	Diacritic	Diacritic BW	Example
Fatha (short a)	ـَ	a	رسَم (ras**a**ma) drew
Dhamma (short o)	ـُ	u	سُنبلة (s**u**nobulap) spike (of grain)
Kasra (short y)	ـِ	i	سِهام (s**i**hAm) arrows
Sukoon (silent vowel)	ـْ	o	سعْر (siE**o**r) price
Shadda (stress mark)	ـّ	~	هدَّد (had~ad) threatened
Tanween-fatha	ـً	F	أبداً (abadA**F**) never
Tanween-dhamma	ـٌ	N	قلمٌ (kalamu**N**) pen
Tanween-kasra	ـٍ	K	شعبٍ ($aEobI**K**) people

It is common practice for Arabs to write without diacritics, which makes Arabic text highly ambiguous, see (Attia 2008). Ambiguity refers to the fact that the morphological, syntactic or semantic analysis of one word may lead to several possible word matches. That is, two words with the same non-diacritic letters, but with different and possibly omitted diacritic characters, are not necessarily the same. While morphological analysis is key in current automated analysis techniques for Arabic text (i.e., words in context), it is known also that morphological ambiguity is still a 'notorious' problem for the Arabic language, see (Kiraz 1998, Attia 2008).

The results of (Debili et al. 2002), cited in (Boujelben et al. 2008), state that non-diacritized words exhibit 8.7 syntactical ambiguity on the average, which drops to 5.6 for diacritized words. For instance, the word جزر (jzr) has different interpretations when it is not diacritized; e.g. جَزَر (jazar) means carrots, جُزُر (juzur) means islands, and جَزْر (jazr) means the fallback of the tide. The word جَزَرَ (jazara) is a past tense verb meaning "butchered".

The use of diacritics for disambiguation is not restricted to human readers. It applies also to automated tools such as morphological analyzers. Some morphological analyzers see (Attia 2008) and (Kammoun et al. 2010) use partial diacritics to resolve ambiguity, e.g. they filter solutions that are inconsistent with diacritics available in the input text.

Essential to the process of disambiguation is the comparison of two Arabic words with the same sequence of non-diacritic letters, but with different diacritics. A simple string comparison of two Arabic words such as word جزر (jzr) and جَزَر (jazar) returns a negative result due to the two additional diacritics, while readers identify them as the same word in a sentence if that is the appropriate meaning. In analogy with the Latin Alphabet languages, one may imagine the challenges that arise when removing vowels from existing words.

Research exists and uses sophisticated techniques to **(1)** promote the use of existing diacritics for automated understanding of Arabic text (Attia 2008, Boujelben et al. 2008, Debili et al. 2002, Vergyri et al. 2004, Alqrainy et al. 2008), **(2)** restore diacritics to non-diacriticized words (Zitouni et al. 2009, Bahanshal et al. 2012, Khorsheed 2013, Habash 2009, Hattab et al. 2012, Rashwan et al. 2011, Roth et al. 2008, Said et al. 2013, Mohamed

et al. 2014, Darwish et al. 2017), and **(3)** include diacritics in corpora (Attia 2008, Beesley 2000, Buckwalter 2002, Kammoun et al. 2010, Kulick et al. 2010). We review and comment on this related work in Section 2. To the best of our knowledge, no work exists that attempts to improve the accuracy of comparing two Arabic words with the same non-diacritic letters. Most existing NLP tools and applications typically ignore diacritics in searching and matching of Arabic words due to the complexity of handling them.

In practice, readers may google these two different Arabic words: جَزَرٌ (jazarun) means carrots, and جُزُرٌ (juzurun) means islands to get the same set of results; which illustrates that diacritics are not taken into account by Google. Bing, Yahoo, Facebook, and Twitter are other examples of search engines that ignore diacritics in the search process - diacritics are obviously removed during the indexing and query-parsing phases. Microsoft Word provides a partial treatment of diacritics when searching a document, but Google Docs, Google Sheets and Google Contacts, as well as Microsoft Excel, provide problematic support when treating diacritics. They use exact string-matching by considering diacritics sensitively as different characters, e.g., جُزُرٌ and its variant جُزرٌ with a missing *dhamma* in the middle do not match.

The importance of treating and ignoring diacritics vary from one application scenario to another. It might be less harmful if diacritics are ignored in internet search engines, as it only causes more irrelevant results to be retrieved (i.e., less precision). However, it is more challenging to treat diacritics when applying filters, as in Excel and Sheets, or when writing conditions in database queries. Other important challenges appear also when parsing and disambiguating fully or lightly-diacritized text. As we shall illustrate in the dictionary integration use case in Section 7, basic matching between dictionary entries, in order to map between these entries, is found to be a challenging difficult task, due to different diacritization of these entries.

To sum up, existing Arabic NLP tools either fully ignore diacritics, or sensitively consider and treat them as different characters; and there are no existing methods to smartly compare between diacritized Arabic words.

In this paper, we present a novel treatment for diacritic-aware Arabic word matching. We propose three alternative algorithms that compare two words with similar letters and possibly different diacritics, and study their accuracy. First, we propose the *Subsume* algorithm, which is a **knowledge-based algorithm** that uses a morphological analyzer in the background to check whether word w_1 morphologically subsumes word w_2. It computes a score metric that measures how much w_1 is a morphological replacement of w_2. Second, we propose the *Alike* algorithm, which is a **machine-learning based algorithm**, that uses a decision tree to classify a pair of words into "same" or "different". Third, we propose the *Imply* algorithm, which is a **rule-based algorithm** that determines an implication relationship (i.e., whether word w_1 implies word w_2), and computes the distance and conflicts between them based on a distance-map tuned over a long domain experience.

We evaluated the three algorithms against a large dataset that consists of 86,886 distinct word pairs. The dataset was constructed by collecting 35,201 distinct words from several Arabic dictionaries. Each word was then paired with potentially similar words generated from the SAMA 3.1 database that the ALMOR analyzer proposed to be potentially the same word. A linguist was employed to judge whether each pair is the "same" or "different", as explained in subsection 6.1. This dataset was used to evaluate the accuracy as well as the soundness and completeness of the three algorithms.

Our evaluation in Section 6, shows the accuracy of Subsume (100%), Imply (99.32%), and Alike (99.53%). Although Subsume was accurate, it was *incomplete* (i.e., was able to judge only 75% of the data). Both Subsume and Imply are *sound* (i.e., their judgement of whether a pair of words is the same, was always correct). This is not the case for Alike; although it was highly accurate and complete (returned results for all pairs), there were cases judged to be "same" that are really "different". Using the notions of soundness and completeness, formally defined in Section 6, is important to compare between the algorithms and to know whether to always trust their judgment or not.

A real-life use case is used to demonstrate the utility of the algorithms. We used the algorithms to integrate and link hundreds of Arabic dictionaries – by matching entries (i.e., verb lemmas) found across these dictionaries. We illustrated that basic string-matching techniques alone did not suffice to match dictionary entries. This is because these entries were partially- or non- diacritized, and morphological analyzers were unable to help in disambiguating them.

In Sections 7 and 8, we inspect the utility of the metrics provided by the algorithms. In particular Section 8 inspects the utility of the metrics in an automated clustering application. We analyzed a dataset of pairs of words and their distance metrics according to the Subsume and Imply algorithms and performed k-means clustering. We then inspected the generated clusters for insight. We observed that pairs in the same cluster shared common structural characteristics which is strong evidence of the utility of the distance metrics.

The rest of this paper proceeds as follows. In Section 2 we overview others' work and discuss how it relates to our proposed solutions. We present and discuss the Subsume, Alike, and Imply algorithms in sections 3, 4, and 5 respectively. In Section 6 we present the experimental setup and discuss our evaluation results. Then we discuss the utility of our work for practical case studies in Sections 7 and 8. Finally, we conclude and discuss future work in Section 9.

2. RELATED WORK

We first review related work that stresses the importance of considering diacritics for automated comprehension of Arabic text and for ambiguity reduction (Attia 2008, Boujelben et al. 2008, Debili et al. 2002, Vergyri et al. 2004, Alqrainy et al. 2008). Then we review works that attempt to restore diacritics to Arabic text (Bahanshal et al. 2012, Khorsheed 2013, Habash 2009, Hattab et al. 2012, Rashwan et al. 2011, Roth et al. 2008, Said et al. 2013). Then we review morphological resources that include diacritic information (Attia 2008, Beesley 2000, Buckwalter 2002, Kammoun et al. 2010, Kulick et al. 2010).

Diacritics and disambiguation. Several studies attested to the high ambiguity of un-vocalized text and the power of diacritics in ambiguity reduction. Programs that automatically add diacritics to Arabic text exist in the software industry, such as Sakhr and RDI. However, these commercial products are closed source. They use morphological analysis and syntax analysis to predict the diacritics. The work in (Vergyri et al. 2004) automatically adds diacritics for transcriptions of spoken Arabic text and employs existing acoustic information to predict the diacritics. The work in (Attia 2008) introduces an ambiguity-controlled morphological analyzer that employs a rule-based system and finite state machines. It found that some insignificant diacritics can be ignored, and it illustrates how diacritics might be used to filter vocalized solutions, especially for morphological analyzers.

Both (Vergyri et al. 2004) and (Attia 2008) quote from (Debili et al. 2002) that a dictionary word with no diacritics has on average 2.9 different possible vocalizations and that a sample text of 23 thousand words exhibited 11.6 different diacritic assignments per word on average. The same source reports that 74% of Arabic words have more than one possible vocalization. It also reports an average of 8.7 syntactical ambiguity for un-vocalized words, which drops to 5.6 for vocalized words. This is evidence of the role of diacritics in disambiguation of word forms, and that ignoring diacritics on Arabic words increases their polysemy, especially if no context (e.g., sentences) is used to help disambiguate them. We shall come back to this issue in Section 7, in which we further illustrated the challenges we faced with lemma disambiguation in the dictionary integration use case.

The work in (Boujelben et al. 2008) presents problems in DECORA-1 related to detection and correction of agreement errors in Arabic text. They introduced an enhancement, DECORA-2, that considers diacritics to help resolve the problems, and showed an improvement of about 10% over MS Word.) They claimed that most Arabic text omits diacritics, few Arabic teaching books have partial diacritics, and only the Quran and teaching books for early stages are fully vocalized. Studies in (Boujelben et al. 2008) show that an Arabic word usually has six varying vocalizations.

The work in (Alqrainy et al. 2008) takes an Arabic word, checks it against preset vocalization templates and returns the POS tags of the word. The approach works for a set of verbs and nouns and is reported to decrease ambiguity when diacritics that vocalize the last character of the pattern exist. This is evidence of both the utility of diacritics in disambiguation of POS tags and the importance of where to place the diacritic within the word.

Nevertheless, the study in (Seraye 2004) concludes that the presence of diacritics affects reading speed negatively while increasing text comprehension only when diacritics play a role in disambiguation. It advises that writers must provide diacritics economically when needed for disambiguation of the intended meaning.

Diacritic restoration. The vast majority of work on Arabic diacritics is concerned with restoring diacritics to Arabic text (Bahanshal et al. 2012, Khorsheed 2013, Habash 2009, Hattab et al. 2012, Rashwan et al. 2011, Roth et al. 2008, Said et al. 2013).

Among other morphological disambiguation tasks, the work in (Roth et al. 2008) explores two diacritics related tasks: DiacFull and DiacPart. DiacFull restores diacritics to all letters of the word and DiacPart restores them to all letters except the final letter. The work uses a linear optimization technique to select the best diacritization of the given word. They confirm two important hypotheses: (1) the use of lexemic features (e.g., POS, gender, number and others) help in determining the best diacritics, (2) tuning the parameters of the optimization algorithm to the task at hand helps the disambiguation task. In our work, we manually fine-tuned weights that characterize an arbitrary distance between diacritics to reduce comparison errors and we arrived at a similar result to hypothesis (1).

The work in (Habash 2009) is a follow-on to (Roth et al. 2008). It adds diacritics to words in context of morphological disambiguation and tokenization.

The work in (Zitouni et al. 2009) proposed the use of a maximum entropy framework for restoring diacritics, which allows combining diverse sources, such as lexical, segment-based, and POS features. This approach was compared later with another approach proposed by (Belinkov et al. 2015) who claim that diacritic restoration can be achieved without relying on external sources other than diacritized text. They use

recurrent neural networks to predict diacritics. They claim that this approach outperforms the state-of-the-art methods.

The work in (Rashwan et al. 2011) describes a commercial product by RDI to automate diacritization of Arabic text. It uses a stochastic process to decide the most likely diacritic map. Since the map is not necessarily grammatically correct and the word in question may be out of the vocabulary, a second stochastic process uses more features of the word including morphological features to select the most likely compatible solution out of a set of diacritic-based feature factorizations. This work is evidence of how much partial diacritics can reduce sophisticated work needed later to disambiguate the Arabic text. With our Subsume and Imply algorithms one can infer the diacritic that most reduces ambiguity and use it.

The work in (Khorsheed 2013) presents a system for Arabic language diacritization using Hidden Markov Models (HMMs). It represents each diacritic with an HMM and uses the context of the whole text to concatenate the HMM decisions and produce the final diacritic sequence.

The work in (Said et al. 2013) presents a hybrid system for Arabic diacritization based on rules and data driven techniques. It uses morphological features and out of vocabulary elimination techniques to reduce the solutions. They do better than (Habash 2009, Roth et al. 2008, Rashwan et al. 2011) by an absolute margin of 1.1% and still make an 11.4 full diacritization word error rate. This is evidence of how complex and sophisticated the diacritization process is.

The work in (Hattab et al. 2012) presents a system to diacritize Arabic text automatically using a statistical language model and morpho-syntactical language models. The work in (Bahanshal et al. 2012) presents an evaluation of three commercial Arabic diacritization systems using fully diacritized text from the Quran and short poems from the period of the advent of Islam.

A different system for automatic diacritic restoration of Arabic texts was proposed by (Mohamed et al. 2014). They proposed a hybrid approach which combines morphological analysis and hidden Markov. This approach differs from other hybrid approaches to linguistic and statistical models. It uses the Alkhalil analyzer with a lexical database containing the most frequent words in Arabic language.

The work in (Darwish et al. 2017) presents a recent state-of-the-art and publicly available Arabic diacritizer. A Viterbi decoder was used for word-level diacritization with back-offs to stems and morphological patterns. The authors illustrated better results compared with the work in (Belinkov et al. 2015) and (Habash 2009), and with work in (Rashwan et al. 2011) if case-ending is considered.

For more in-depth related work on Arabic diacritization, (Azmi et al. 2015) presents a comprehensive survey. Compared with our work, the above approaches aim at restoring diacritics to Arabic text, while our goal in this paper is to provide a framework for comparing and matching between Arabic words. Nevertheless, both kinds of approaches contribute to resolving the ambiguity and the challenges imposed due to missing diacritics in Arabic.

Existing morphological resources with diacritics. In addition to automatic diacritization tools, previous work includes tools that disambiguate based on input diacritics. Analyzers such as (Buckwalter 2002) and SAMA in (Kulick et al. 2010), contain the diacritization of lexicon entries in addition to other annotations. However, they ignore the partial diacritics in the analysis phase and the analysis makes little

benefit from lexicon vocalizations. MORPHO3 in (Attia 2008), Arabic Xerox in (Beesley 2000) and MORPH2 in (Kammoun et al. 2010) are other examples of morphological analyzers with the same capability. They later filter morphological solutions based on their consistency with the existing partial diacritics.

To summarize, we find that the vast majority of work discussing the challenges imposed due to missing diacritics is evidence of the utility of our work. Sophisticated and advanced technologies, coupled with advanced expert rules used to automate diacritization, lack in accuracy and make significant errors (11.4%). Morphological analyzers avoid partial diacritics in analysis and defer to interested NLP applications the task of filtering out inappropriate interpretations.

In conclusion, we are the first to approach the diacritic placement problem as a word-to-word comparison problem. Leveraging our algorithms, NLP tools can reduce ambiguity by placing the diacritics that matter. Users at the entry level also can be encouraged to introduce the minimal number of diacritics that matter to reduce ambiguity.

3. THE *SUBSUME* ALGORITHM

The Subsume algorithm takes two Arabic words w_1 and w_2 and returns a score between 0 and 1 that denotes how much w_1 can morphologically subsume w_2. The algorithm uses a morphological analyzer to compute the morphological distance. Before presenting how the algorithm works we define the following notions.

Definition 1 (Morpheme) A *morpheme* is the smallest unit of morphological structure of a given word. For Arabic, a morpheme is either an *affix* or a *stem*. The *affix* is either a *prefix* or a *suffix*. The stem could be a *root* or an *inflection* of the root. A word is the concatenation of *connecting* prefix, stem and suffix morphemes where the prefix and the suffix could be empty strings. Morpheme connectivity is a predefined relation. For example, the prefix يَ *ya* connects to the stem لعب *l'b* (play) to form the word يلعب *yal'b* (is playing). Each morpheme, whether inflectional or not, is associated with morphological features such as part of speech (POS), transliteration, lemma and gloss.

Definition 2 (Morphological analysis) The *morphological analysis* of a given word w is the set of all possible morpheme concatenations that form w. A word may have more than one morphological *solution* and one solution may have more than one set of features associated with it. Table 2 illustrates three different solutions for the word أحمدهم (Ahmdhm). The first solution differs from the other two in diacritics, morpheme segmentation and translation. The other two agree in diacritics and morpheme segmentation, however they differ in meaning.

Table 2: Three morphological solutions for أحمدهم

Meaning	Transliteration	Suffix	Stem	Prefix
Did he praise them?	aAhamidahum	هم	حَمِد	أ
Their Ahmad	aAhmadahum	هم	أَحْمَد	
The best of them	aAhmadahum	هم	أَحْمَد	

Definition 3 (Morphology *subsume* relation) We say word w_1 morphologically subsumes word w_2 if the morphological analysis of w_1 returns a set of solutions including the set of solutions returned by the morphological analysis of w_2. For example, the words أحمدهم without diacritics, or with one fatha on the first letter, morphologically subsumes أَحَمِدَهم and أَحْمَدَهم.

Definition 4 (Morphology distance metric) The morphological distance metric between two words w_1 and w_2 measures how much w_1 can morphologically disambiguate w_2. If w_1 and w_2 fully match in diacritized and non-diacritized characters, then the distance is 1; which denotes similarity. If w_1 and w_2 have different non-diacritic characters, then the metric is 0; which is the maximum distance and the words are deemed different. In case the words have the same non- diacritized characters, the metric is calculated as $|M2 - M1Minus| / |M2|$ where M1 and M2 are the sets of morphological solutions of w_1 and w_2 respectively, M1Minus is the complement of M1 in M, and |M| denotes the cardinality of M. Intuitively, the distance is a ratio measure of how many solutions of M2 can be eliminated by the diacritics of M1.

3.1 Description of the Subsume Algorithm

The Subsume algorithm (See **SubsumeMetric** in Figure 1) first makes use of a diacritic consistency algorithm **isDiacriticConsistent** shown in Figure 2.

Algorithm **isDiacriticConsistent** takes two words and makes sure the sequence of non-diacritic letters is an exact match and the partial diacritics included in the two words are consistent. Two diacritics are considered consistent if they are not conflicting diacritics. For example, the partially diacritized words شَاهد and شاهِد are diacritic-consistent as there is no conflict in diacritic assignment between them. However, the partially diacritized words شَاهَد and شَاهِد are not diacritic consistent since the *fatha* and the *kasra* on the third letter are conflicting.

If **isDiacriticConsistent** returns false, the Subsume algorithm reports that the two words are distinct by returning a score of 0. Otherwise, the Subsume algorithm uses a morphological analyzer to compute the morphological solutions M1 and M2 of w_1 and w_2, respectively. It also computes M, the morphological solutions of w, the undiacritized form of w_1 and w_2. The Subsume algorithm computes M1Minus, the complement of M1 in M, or intuitively the solutions that are eliminated due to the diacritics in w_1. Then, the algorithm computes M12 by subtracting M1Minus from M2. Intuitively, this is the set of solutions that would have been eliminated from M2 by the diacritics of w_1. The metric ($|M2 - M1Minus| / |M2|$) finally computes and returns the ratio of the eliminated solutions to the solutions of w_2.

Example: To compute the morphological distance between w_1 (بِن) and w_2 (بنّ), we first run the morphological analyzer to retrieve the solutions of the undiacritized word بن (w). Table 3 shows five different morphological solutions (|M| = 5) of بن (w). Notice that the same word with a *kasra* on the first letter بِن (w_1) has four solutions (|M1| = 4); and that the same word with a *shadda* on the second letter بنّ (w_2) has two solutions (|M2| = 2). Now, the set M1Minus has only one element بُنّ. The set M12 eliminates بُنّ and has consequently one solution بِنَّ.

Consequently, SubsumeMetric(بِن, بنّ) returns 0.5 and SubsumeMetric(بنّ, بِن) returns 0.25. Intuitively, since w_1 (بِن) and w_2 (بنّ) are diacritic consistent and have no conflicting diacritics, the *kasra* on w_1 implies half the solutions of w_2 while the *shadda* of w_2 implies one fourth the solutions of w_1.

The result of the Subsume algorithm is a score denoting how much word w_1 is a morphological replacement of word w_2.

As will be discussed in Section 6, although this algorithm is sound and its accuracy is 100%, it was unable to judge 25% of the evaluation dataset. This is due to the missing knowledge in the lexicon used by the morphological analyzer (SAMA 3.1).

Table 3: Morphological solutions for بن

	Transliteration	Semantics	POS
بُنَّ	bun~	Coffee	NOUN
بِن	bin/ben	Name of a person (like Benjamin)	NOUN_PROP
بِن	Bin	Son	Noun
بِنَّ	bin+na	They (female plural) appear	VERB_PERFECT+ PVSUFF_SUBJ:3F
بِن	bi+n	with Noon (name of a person)	PREP+NOUN_PROP

```
Double SubsumeMetric (Word w1 , Word w2 )
  // if w1 and w2 are diacritic-consistent continue
if (isDiacriticConsistent(w1, w2) == False)
  return 0;
   // w =  w1 but without diacritics
  Word w = removeDiacritics(w1);
   // find the morphological solutions for w, w1, and w2
  Solutions M = analyser(w);
  Solutions M1 = analyser(w1);
  Solutions M2 = analyser (w2);
   // M1Minus are the morphological solutions of w
   // minus the solutions of w1.
  Solutions M1Minus = M – M1;
   // M12 are the solutions w1 diacritics (that are not in
   // conflict with w2) would have eliminated from w2
 Solutions M12 = M2 – M1minus;
   // returns the ratio of the eliminated solutions
return (M12.size / M2.size);
```

Figure 1: SubsumeMetric Algorithm

```
Boolean isDiacriticConsistent(Word w1 , Word w2 )
//Returns True if all diacritic in w1 and w2 are
consistent
  int i1 =0, i2 =0;
  Diacritics d1, d2 ;
   while (i1 < w1.size && i2 < w2.size)
      if (!equals(w1[i1], w2[i2]))
        return false;
      i1 ++; i2 ++;
      d1.clear(); d2.clear();
      while (i1 < w1.size && isDiacritic(w1[i1]))
          d1.add(w1[i1]);
          i1 ++;
       endwhile // traverse w1 till next non-diacritic
       while (i2 < w2.size && isDiacritic(w2[i2]))
          d2.add(w2[i2]);
          i2 ++;
        endwhile // traverse w2 till next non-diacritic
        if (!isConsistent(d1 ,d2))
          return false;
   endwhile //traverse all of w1 and w2
return (i1 == w1.size && i2 == w2.size);
```

Figure 2: isDiacriticConsistent Algorithm

4. THE *ALIKE* MACHINE LEARNING BASED ALGORITHM

This section illustrates the use of supervised machine learning algorithms to compare between Arabic words. As explained in Section 6, we prepared and used a large dataset (86,886 pairs of words) classified by a linguist whether the pairs are the same or different.

We tried to extract and use many types of features related to comparing the diacritic characters to each other and the non-diacritic characters to each other. The features also include diacritic positioning and cardinality. Table 4 includes a list of the features that we found most useful to include in our feature vector with an example on each for the words pair: صَلَّبَ، صَلْبُ (Salubu, Sal~aba). We used the Information Gain algorithm (which measures the possible drop in entropy of the nodes in each level) to preprocess the features and select the most important ones among them. We passed the top five selected features as input to the decision tree-based classifier.

Table 4. Important features used in the Alike machine learning algorithm

Feature Vector	Feature Name	Feature Description	Example (Sal~aba, Salubu)
f_1	**W1NoOfLetters**	Number of letters in w_1	3
f_2	**W2NoOfLetters**	Number of letters in w_2	3
f_3	**D12A**	Concatenation of the diacritics on the first two letters of w_1 and w_2 (D11+D12+D21+D22)	a~aau
f_4	**D36A**	Concatenation of the diacritics on the rest of letters (letters 3 to 6)	a u

		(D13+D14+D15+D16+D23+D24+D25+D26)	
f_5	**D12B**	Concatenation of the diacritics on the first two letters of w_1 and w_2 (D11+D21+D12+D22)	aa~au
f_6	**D36B**	Concatenation of the diacritics on the rest of letters (letters 3 to 6) (D13+D23+D14+D24+D15+D25+D16+D26)	au_____
f_7	**D13**	Concatenation of diacritics from w_1 and w_2 between indices 1 and 3 (D11+D21+D12+D22+D13+D23)	aa~auau
f_8	**D46**	Concatenation of diacritics from w_1 and w_2 between indices 4 and 6 (D14+D24+D15+D25+D16+D26)	_ (no diacritics)
f_9	**D1**	Concatenation of the diacritics on letter 1 in w_1 and w_2 (D11+D21)	aa
f_{10}	**D2**	Concatenation of the diacritics on letter 2 in w_1 and w_2 (D12+D22)	~au
f_{11}	**D3**	Concatenation of the diacritics on letter 3 in w_1 and w_2 (D13+D23)	au
f_{12}	**D4**	Concatenation of the diacritics on letter 4 in w_1 and w_2 (D14+D24)	__
f_{13}	**D5**	Concatenation of the diacritics on letter 5 in w_1 and w_2 (D15+D25)	__
f_{14}	**D6**	Concatenation of the diacritics on letter 6 in w_1 and w_2 (D16+D26)	__
f_{15}	**D7**	Concatenation of the diacritics on letter 7 in w_1 and w_2 (D17+D27)	__
f_{16}	**Duplicate-12**	Flag=1 if 1st and 2nd letters are the same in either w_1 or w_2, 0 otherwise	0
f_{17}	**Duplicate-23**	Flag=1 if 2nd and 3rd letters are the same in either w_1 or w_2, 0 otherwise	0
f_{18}	**Duplicate-34**	Flag=1 if 3rd and 4th letters are the same in either w_1 or w_2, 0 otherwise	0
f_{19}	**Duplicate-24**	Flag=1 if 2nd and 4th letters are the same in either w_1 or w_2, 0 otherwise	0
f_{20}	**NonDiacriticConflict**	Flag=0 if there is a conflict in letters between w_1 and w_2,1 otherwise	0
f_{21}	**ShaddaDifferent**	Flag=1 if there is shadda conflict between w_1 and w_2, 0 otherwise	1
f_{22}	**Alef_flag**	Flag set to 1 if first letter is Allef (ا ,أ ,إ ,آ) in both words, 0 otherwise	0
f_{23}	**Li_Shadda**	Flag set to 1 if the diacritic of the ith letter is shadda and 0 otherwise	1 (for i=2)

The decision tree algorithm (Version C5.0, implemented in R "C50" package) was used as a classifier (Quinlan 1993). We progressively tried the pre-processing and the training with 30%, up to 70% of the data set in increments of 10%. A decision tree defines predicates that split the values of each feature into several branches. Then it orders the features in a manner to minimize the size of the tree and thus minimize branching in the classifier. The root of the tree is the feature with the highest order and the leaves of the tree are the decisions. Each decision is characterized with a decision support count and a local error estimation.

We validated the computed decision tree classifier across a fixed subset of the remaining dataset correspondingly. As will be discussed in Section 6, this decision tree-based classifier algorithm achieved above 99% precision and recall. Table 5 shows the top 5 features according to the Information Gain algorithm. The most important feature is whether the *shadda* diacritics were different across the two words. The second most important feature is whether the concatenation of the first three diacritics differed or not (D13). The conflict in non-diacritic characters was the third most important feature. The D36A and D36B come next in importance and are variations of the concatenation of the third to sixth characters from w_1 and w_2.

Table 5. Top five important features with the information gain metric

	Partial data set				
Feature	30%	40%	50%	60%	70%
ShaddaDifferent	0.479	0.478	0.475	0.474	0.473
D13	0.339	0.338	0.335	0.337	0.332
NonDiacriticConflict	0.304	0.307	0.304	0.304	0.303
D36A	0.219	0.218	0.214	0.215	0.214
D36B	0.219	0.217	0.214	0.215	0.214

We passed the top five features to the decision tree algorithm and we trained with 30%, 40%, 50%, 60% and 70% of the data set. These subsets were randomly selected using R's *runif* function, which is the most common method used to simulate independent uniform random numbers. Then, we obtained decision tree classifiers similar to the

one in Figure 3. We validated the classifiers on the same fixed subset of 30% and obtained consistently an accuracy of more than 99% as shown in Figure 4.

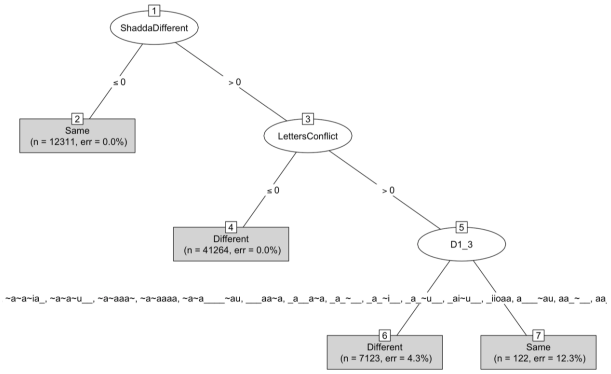

Figure 3. Decision tree classifier

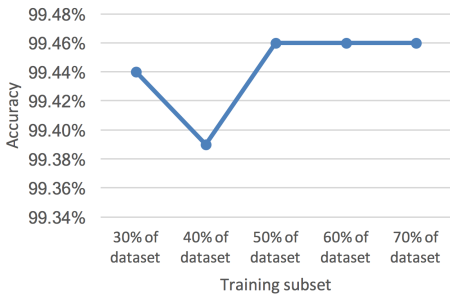

Figure 4. Accuracy with varying training data set.

5. THE *IMPLY* ALGORITHM

This section presents the Imply algorithm, which is a rule-based algorithm fine-tuned by language experts during several years of experience and use. This algorithm does not only judge whether two words are the same, but it also identifies which word implies the other and the distance between words, among other outputs. Before delving into the details of how the algorithm works, we present a few definitions.

Definition 5 (Implication Relation): Given two Arabic words w_1 and w_2, an *implication relationship* denotes that w_1 implies w_2 *iff* both words have the same letters and every letter in w_1 has the same order and same (or fewer) diacritics as the corresponding letter in w_2. For example, the word (فَعل) implies (فَعَل). If both w_1 implies w_2 and w_2 implies w_1, then the two words are called *compatible*; otherwise, in case of no implication, i.e., in case of letters or diacritics conflicts, then the two words are *incompatible*. Table 6 illustrates these cases. Notice that the implication relation concerns the sets of diacritics of words w_1 and w_2, unlike the subsume relation which reasons about the sets of morphological solutions (words) of w_1 and w_2.

Table 6: Implication directions with examples

Implication Direction	Meaning	Example	
		w_1	w_2
-2	Incompatible-letters	فَعَلَ	ثَعلَب
-1	Incompatible-diacritics	فَعل	فِعِل
0	Compatible-*imply each other*	فعَل	فَعل
1	Compatible-w_1 implies w_2	فعل	فَعَل
2	Compatible-w_2 implies w_1	فَعَل	فعل
3	Compatible-exactly equal	فَعل	فَعل

Definition 6 (Distance Map): A *distance map* denotes a matrix of all possible pairs of Arabic diacritics and a distance value between them (see Table 7). A distance map tuple <d_1, d_2, *delta*> denotes the two diacritics by d_1 and d_2 and the distance score by *delta*. As will be discussed later, the Imply algorithm uses this distance map to calculate the distance between two words. For example, the distance between the diacritics *fatha* and *kasra* is 1. When the *shadda* or *hamza* appears as either d_1 or d_2, and the compared diacritic is different, the distance equates 15 (but it can also equal

4 depending on which letter has the diacritic, as will be explained later). That is, the distance of 1 indicates a difference in one diacritic; but it becomes 15 (in case of *shadda* and *hamza* difference) to indicate that this is most probably another word, as the addition of *shadda* or *hamza* changes the semantics of a word in most cases. The number 15 was selected as we did not find any word in Arabic with more than 14 diacritics on it (see our use case in section 7). The maximum difference if *shadda* and *hamza* were not involved was found to be 8. In other words, if the total number of diacritic differences between two words is more than 14, then we know for sure that either *shadda* or *hamza,* or both are involved. Others are not.

Table 7: Diacritic Pair Distance Map

	Fatha	Dhamma	Kasra	Sukoon	Fathatan	Kasratan	Dhammatan	Shadda	Hamza
No Diacritic	1	1	1	0	1	1	1	1	1
Fatha	0	1	1	1	1	1	1	15	15
Dhamma	1	0	1	1	1	1	1	15	15
Kasra	1	1	0	1	1	1	1	15	15
Sukoon	1	1	1	0	1	1	1	15	15
Fathatan	1	1	1	1	0	1	1	15	15
Kasratan	1	1	1	1	1	0	1	15	15
Dhammatan	1	1	1	1	1	1	0	15	15
Shadda	15	15	15	15	15	15	15	0	15
Hamza	15	15	15	15	15	15	15	15	0

Definition 7 (Conflicting Diacritics): Two diacritics are called *conflicting diacritics* if they are distinct and appear on the same character of a given word. That is, given words w_1 and w_2, for a pair of diacritics d_1 and d_2, where d_1 is located in w_1 at the corresponding position (letter) of d_2 in w_2, if d_1 does not equal d_2, then d_1 and d_2 are conflicting diacritics and w_1 is incompatible with w_2.

Definition 8 (Words Matching): The matching between two words is defined as a tuple: <w_1, w_2, *implication direction, distance, conflicts, verdict*>. w_1 and w_2 are the two words to be compared; the *implication direction* is a number denoting the relationship between the two words (shown in Table 6); the *distance* denotes the overall similarity of the diacritization between the two words, which we compute based on the distance map; the *conflict* denotes the number of conflicting diacritics between the two words; finally, the *verdict* takes one of the values: 'Same', or 'Different', to state whether w_1 and w_2 are matching.

5.1 Description of *Imply* Algorithm

The Imply algorithm takes two words as input and produces the matching tuple defined by definition 8. For each input word, Imply generates two arrays, one for the letters (each letter receiving a cell) and one for diacritics (each diacritic in a cell). The words are then checked to find if they contain the same letters. If so, then for each pair of corresponding letters, an implication value and a distance is assigned. Figure 5 illustrates an example of comparing (فَعَلَ) and (فَعل).

Implication between letter pairs is determined as follows; if both letters have exactly the same diacritics (see Table 6), a direction-score of 3 is assigned to the corresponding position in the implication array. If the pair has conflicting diacritics, a direction-score of -1 is assigned to the pair in the array's corresponding position. If both letters have same diacritics, then a direction-score of 3 is assigned. If the first letter in the pair is missing a diacritic that is present on the second letter, then an implication direction-score of 1 is assigned to the pair in the array's corresponding position. If the second letter in the pair is missing a diacritic that is present on the first letter, an implication

direction-score of 2 is assigned to the pair in the array's corresponding position. That is, at this stage we only determine the implication direction between letters.

Implication between two words is determined as follows. Once an implication value is assigned for each pair of letters, the implication array is observed. First, the algorithm returns an overall implication value of -2 and quits if the two words have different letters. Otherwise, if *all* entries in the implication array contain 3 (i.e., same diacritics in all letters), then an overall implication direction 3 is returned. If all entries in the array contain 1 or 3, then an overall implication direction 1 is returned, i.e. w_1 implies w_2. If *all* entries contain either 2 or 3, then an overall implication direction 2 is returned, i.e. w_2 implies w_1. If all entries contain either 1 or 2 then an overall implication direction 0 is returned, meaning the words imply each other. If there exists at least one -1 entry, then an overall implication of -1 is returned, i.e. incompatible diacritics.

The implication distance between two words is determined as the following. While generating the implication array, the algorithm loops through the diacritic arrays. For each pair of diacritics, the algorithm returns a distance from the distance map (in Table 7) and adds it to the overall distance value. For example, the implication distance between the two words shown in Figure 5 is 2. This is because there are two letters with *fatha* in the first word but not in the second, and the *fatha-NoDiacritic* is given distance 1 in the distance map.

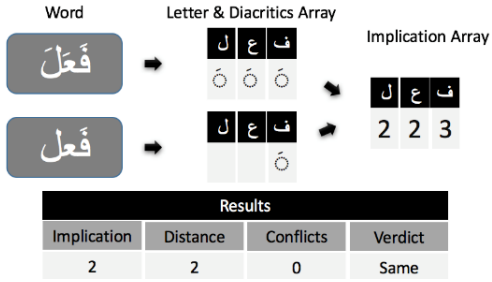

Figure 5: Implication Example

The number of conflicts between two words is determined as the following. The algorithm runs through the diacritics array and counts how many times -1 appears in the array. The algorithm does not calculate conflicts in case the two words have different letters.

5.2 Special Cases and Fine-Tuning

This subsection presents some important special cases used to fine-tune the Imply algorithm. We learned these special cases and how to handle them through intensive manual comparisons and investigations between similar words in our dataset.

Diacritics on the ***last letter of a word*** are a special case, as differences in diacritization do not change the meaning of the word. Considering this, the algorithm neglects the diacritics on the last letter of the words being compared.

The ***shadda*** is also special. We use different weights for calculating word distance based on the position of *shadda* in a word. When on the first letter of a word, the distance of a pair including the *shadda* is 4 if the diacritics are not the same, as people tend to ignore *shadda* in the beginning of a word most the time, e.g., رضّ should be رّضّ. In the case of *shadda* on any other letter in the middle of a word, this distance increases to 15.

This distance changes because the *shadda* plays a much larger role when found in the middle of the word, as opposed to the beginning. When on the first letter of a word (which is a rare case), it does not singlehandedly determine whether two words are the same, whereas it can certainly determine whether two words are the same or not when found in the middle of a word.

The ***hamza*** is similar to that of the *shadda*. When found atop the first letter of a word, it is considered a diacritic rather than a letter (e.g., إلهام/الهام). As found in the case of the *shadda*, when the *hamza* is on the first letter of a word, the distance of the diacritic pair is 4 if the diacritics are not the same. In all other cases, the *hamza* is considered a letter and treated as such. In other words, the *hamza* is treated as and considered a letter only when it appears after the first letter in a word.

The ***sukoon*** is a peculiar case. This is because it carries no sound. Because of this, it is usually neglected in writing, where a non-diacritized letter in a fully diacritized text is usually interpreted as a letter diacritized with a *sukoon*. As a result, in the Diacritic Map, the *sukoon* has a weight of 0 (i.e., considered no distance) in two cases: (i) *sukoon-sukoon*, and (ii) *sukoon-NoDiacritic*. Otherwise, in case the corresponding letter is diacritized with a diacritic other than the sukoon, it behaves like any other diacritic and a weight is given, as shown in Table 5.

As will be explained next in the section, the Imply algorithm is always sound as it always returns correct decisions. We established soundness by iterating many times over the dataset and refining for the non-deterministic results after each iteration. The refinement was based on input from a linguistic expert who carefully evaluated whether all critical cases are valid results, and what the correct result should have been. The refinement included changes to Distance Map entries and modifications of the special cases, while developing the algorithm.

The linguistic rules implemented in Imply also play a vital role in determining the weights within the Diacritic Map. Especially in the cases of the *shadda* and *hamza*, the heavy weights of the diacritics were largely influenced by the linguistic properties these diacritics have.

6. EVALUATION AND DISCUSSION

This section presents the evaluation of the three algorithms, whether they are sound and complete, as well as their precision, recall and F-measure, which we experimented over a large dataset of about 87,000 pairs of words. First, we define the notion of soundness and completeness, then we present our dataset, and discuss the results.

Definition 9 (Soundness) As known in logic (see Jarrar et al. 2008), we define the soundness of our algorithms as whether the "Same" results of the algorithm are correct or not. That is, if the algorithm judges two words to be the same, and this answer is *always correct* then the algorithm is called *sound*. If a single case is judged by an algorithm as "Same" but is actually "Different", then the algorithm is *not sound*. As will be described later, this measure is important for some application scenarios as we need to know whether to always trust the algorithm's judgment or not.

Definition 10 (Completeness) As is also known in logic, we define the completeness of our algorithms as whether the algorithm is *always able to judge* whether two words are the same. If so, then the algorithm is called *complete*. If a single case cannot be judged by the algorithm then the algorithm is *not complete*. Remark that an algorithm might be sound but not complete—able to correctly judge some, but not all cases.

The **accuracy** degree for each algorithm is also evaluated using the standard measures of **precision**, **recall** and **F-measure**.

6.1 Experiment Setup

To evaluate the algorithms using the above measures we needed to prepare a large dataset of pairs of words. We collected 35,201 distinct Arabic words extracted from 38 different dictionaries. We then used the ALMOR morphological analyzer (Habash and Rambow 2009) to retrieve possible matches of the collected 35,201 words. The retrieved matching words by ALMOR generated 86,886 word-pairs to compare.

The dictionaries we collected are at different levels of diacritization. For instance, the Al-Waseet, the Modern Arabic, Alghani Azaher, and Al-Maany, are each filled with highly diacritized verbs. Dictionaries of medium-level diacritization included, for example, the Biological Lexicon of Biology and Agricultural Sciences, the Dictionary of Economic Terms, and the Dictionary of Statistics, among others. Dictionaries providing no diacritics on their words include the Hydrology Glossary, the Lexicon of Chemicals and Pharmaceuticals, the Lexicon of Education and Psychology, and the Historical and Geological Lexicon. A high level of diacritization entails that all or most diacritics of a word are written on the word. A medium level of diacritization provides a word with few diacritics present on the word. Selecting dictionaries with different levels of diacritization is important for our experiment as this provides a granular scope of the performance and extent of the capabilities of our algorithms.

The collected words (without removing diacritics) were run through the ALMOR analyzer using the SAMA 3.0 database. ALMOR produced highly diacritized words that were *similar* (in appearance and spelling) to the input words, as will be explained in section 7. For each input word (of the 35,201 words), ALMOR returned a fully diacritized word that is possibly the same word as the input word. That is, the output of this process is a set of pairs where the first word is highly, partially, or none diacritized, and the second word, from ALMOR, is fully diacritized. In this process, after giving ALMOR our 35,201 words, it produced 86,886 distinct pairs of words. A sample of the dictionaries and the number of pairs that we were able to retrieve from each dictionary is presented in Table 8.

The 86,886 pairs of words were given to a linguist to manually classify them as “Same” or “Different”. We then ran the three algorithms and evaluated the results.

Table 8: Sample of the dictionaries experimented on

No	Dictionary		Diacritization Level	Words Used	Word Pairs Generated
1	Al-Ramooz	معجم الراموز للأفعال	High	8,734	19,296
2	Al-Waseet Verbs	المعجم الوسيط في تصريف الافعال	High	8,033	17,721
3	Al-Ma`any	قاموس المعاني	High	4,351	9,750
4	Scientific &Technical Terms	المصطلحات العلمية والفنية والهندسية	High	2,263	4,527
5	Pharmaceutical Dictionary	قاموس الصيدلة	None	572	1,211
6	Nubian Dictionary	القاموس النوبي	None	561	1,175

6.2 Results

Running the three algorithms returned the results in Table 9. For example: out of the total 18146 “same” pairs in the dataset, the Imply algorithm succeeded in judging 17554 of them as “same” while judging the rest 592 as “different”. Table 10 shows a comparative evaluation of the three algorithms.

Table 9: Returned Classification

		Same	Different	Total	
Manual classification	Same	18146	0	18146	86886
	Different	0	68740	68740	
Subsume	Same	17103	0	17103	86886
	Different	0	48643	48643	
	No-Answer	20097	1043	21140	
Imply	Same	17554	592	18146	86886
	Different	0	68740	68740	
Alike	Same	5363	97	5460	26066
	Different	25	20581	20606	

Table 10: Evaluation

	Sound	**Complete**	**Precision**	**Recall**	**F-measure**	**Accuracy**
Subsume	Yes	No (only 75%)	100%	100%	100%	100%
Imply	Yes	Yes	96.74%	100%	98.34%	99.32%
Alike	No (by 0.1%)	Yes	98.22%	99.54%	98.88%	99.53%

6.3 Discussion and Synthesis

The **Subsume algorithm** is sound and accurate but incomplete. The soundness was accomplished as all the pairs that were judged as "same" where really "same". It is also accurate as its Precision and recall were 100%. However, as expected, Subsume is incomplete as it was unable to judge a large number of pairs (24.33% of the dataset). This is because the database used by the morphological analyzer (SAMA 3.1) is not complete.

The **Imply algorithm** is sound, complete, and highly accurate (99.32%). Its soundness is accomplished as all "same" pairs were judged correctly. This is also evident from its 100% recall. The completeness of the algorithm is accomplished as it was able to judge all pairs of words.

Special Case: We found 592 pairs (3.23% of the dataset) that were classified as different by the Imply Algorithm, pairs like **(**بلّل، بلل), (جَوَّع، جوع), (جنَّس، جنس**)**; while being classified as same by the Subsume algorithm. After verifying these cases we found that they are "different" according to Arabic diacritization (because of strong *shadda* difference); Nevertheless since there is no other way to diacritize them, the Subsume algorithm (and our linguist expert) decided them to be the same. In other words, the absence of *shadda* in the first word is a critical difference, but there is no other possibility in Arabic to spell the word without *shadda*. This is why it was judged as "different" by Imply, but as "same" by Subsume given the background knowledge of Arabic.

The **Alike algorithm** is not sound, but complete and highly accurate (99.52%). Its accuracy is similar to the Imply algorithm. However, there are 25 pairs (0.1% of the test dataset) that are "different" but Alike classified them as "same". Although this is not a high number, it illustrates that Alike is not sound and cannot be fully trusted in the case of sensitive applications (as in Jarrar 2006 and Jarrar 2011).

Predictability: Unlike Alike, the Imply and Subsume algorithms are *predictable*. This is because their judgement can be tracked and explained, and thus it is easier to decide which algorithm to use given applications' sensitivity and requirements. This is not the case for Alike, since it is based on Entropy Statistics—the frequency of information produced by a source of data. It is worth noting that both the Imply and the Alike algorithms are computationally cheaper than the Subsume algorithm since they do not require access and background search in any database.

Measuring distance: The distance map used in the Imply algorithm was very useful in calculating a distance measure between words, which reflects the difference degree between words diacritization. However, the distance provided by Subsume reflects only how much a word morphologically disambiguates the other. Providing a distance measure using the Alike is complicated as it needs developing an additional model that solely calculates the distance between a pair of words without doing any classification into "same" or "different".

Hybrid approach: Combining the three algorithms to work together and synthesizing their results improve their overall performance. Since the Alike algorithm is not predictable, as it is not sound (returns "same" but might be "different") it cannot be combined with the other algorithms. Nevertheless, the Subsume and the Imply can be combined as the following: (i) ask the Imply algorithm, if it returns "Same" then it is the "Same", if not then (ii) ask the Subsume algorithm if the returned morphological distance is 0, then it is "Same". The intuition behind this hybrid approach is that we trust the judgment of the Imply algorithm, except when it says "different" but the morphological distance is 0 (by the Subsume algorithm). As explained in the special cases above, there are 592 pairs (among the 86K) that are judged as "different" by Imply because of *shadda* difference, but since there are no other possibilities in Arabic to spell them without *shadda* they are judged as "Same" (with morphological distance of 0) by Subsume.

Nevertheless, although combining these two algorithms will improve their overall performance, the computational complexity of their combination will also be higher. That is, since the Imply is already highly accurate (99.32%), the little improvement at the price of a higher computational cost is possible but might not be needed for most applications. In our dictionary integration use case (see next section), we preferred to use the Imply alone because of the high sensitivity of linking a huge number of dictionaries entries.

In the following section, we present a use case in which we used the Imply algorithm to match and link lemmas across dictionaries.

7. USE CASE (LEMMA DISAMBIGUATION AND DICTIONARY INTEGRATION)

This section presents a use case to demonstrate the utility of our algorithms. In our VerbMesh project, we aim to integrate dictionary entries of about 150 Arabic dictionaries that we collected and digitized at Birzeit University. At this stage, we are concerned with linking and integrating verb entries across dictionaries – in order to build a large graph of Arabic verbs and link it with Arabic Ontology (Jarrar 2011) and other sources (Abuaiadah et al. 2017, Jarrar et al. 2011, Jarrar et al. 2014, Jarrar et al. 2016). The major challenge we faced in this task is that basic string-matching techniques alone do not suffice to match and link two verbs. This is because same verbs across dictionaries (i) might be diacritized differently, and (ii) may come with different linguistic properties, which lead to incorrect and undesired matching. Our approach to tackle these issues was to use the notion of lemma to match between verbs. Verbs having the same lemma are considered same entries. In other words, same verbs across dictionaries are linked with their lemmas, although they might be diacritized differently.

To assign a lemma to each verb in every dictionary, we used the ALMOR morphological analyzer, utilizing the SAMA 3.1 database. Nevertheless, another major challenge we faced using this approach was that ALMOR returned too many lemmas for each dictionary entry. For example, ALMOR returns 13 different lemmas for the entry نَحُلَ, which are {نَحَلَ، نَحَلُ، نَحِلَ، حَلِّ، حَلِيَ، حَلَى، حَلُوُ، نَحُلُ، حالُ، حَلُّ، أحال، أَحَلّ، حَلَّ}. Another example is the undiacritized word سلب, for which ALMOR produced {سَلِبَ، سَلَبُ}. This is because ALMOR takes a word as input, segments it, and uses the stem to find other words with

a similar stem and their lemmas. As a result, multiple lemmas are returned for each ALMOR's word input. Therefore, a further disambiguation of lemmas was needed, which is where our algorithm plays an essential role.

By using the Imply algorithm, after ALMOR's lemmatization, we were able to filter and disambiguate the lemmas provided by ALMOR and keep only the correct lemmas. Taking each verb-lemma pair as input, our algorithms decided whether each pair had an implication relationship. For instance, after lemmatizing the 8,731 verbs found in Al-Waseet (No.2 in Table 9), 17,721 word pairs were returned by ALMOR -among them incorrect lemmas. After using our matching algorithm to filter out those incorrect lemmas (i.e., with different incompatible diacritics), we were able to reduce the number of matching pairs to 4,766 correctly matched pairs. In the case of the Al-Ramooz dictionary (No. 1), which includes 9,866 verbs, ALMOR returned 19,296 different pairs of words. After running the resulting pairs through the Imply algorithm, we were left with 4,575 correctly matched pairs.

Table 11: Sample of Lemma Reduction

No	Dictionary		Verbs	Initial Pairs	Correct Pairs
1	Al-Ramooz Verbs	معجم الراموز للأفعال	9,866	19,296	4,575
2	Al-Waseet Verbs	المعجم الوسيط في تصريف الافعال	8,731	17,721	4,766
3	Al-Ma`any	قاموس المعاني	4,425	9,750	2,504
4	Al-Mustalahat	المصطلحات العلمية والفنية والهندسية	7,021	4,523	915
5	Pharmaceutical Dictionary	قاموس الصيدلة	1,101	1,211	316
6	Nubian Dictionary	القاموس النوبي	1,118	1,175	299

Table 9 provides a sample of the dictionaries we matched, which provide an evidence of not only the algorithm's success, but also of the *added utility* when used on top of other programs, such as morphological analyzers. On average, the amount of word pairs originally proposed by ALMOR were reduced by 74.8% using the Imply algorithm. The remaining 25.2% of the word pairs were all pairs that are certain matches. This ensures the words being linked are, with certainty, the same words under the same lexeme. That is, the matching algorithm can be used with ALMOR or MADA in order to provide further filtration of the analysis and bring more desired results.

8. SYNTHESIS (UTILITY OF DISTANCES)

We evaluated the utility and meaning of the proposed distance metrics using unsupervised machine learning. We used the k-means clustering algorithm that partitions n vectors into k disjoint clusters, such that each vector belongs to the cluster with the nearest mean. We provided the k-means algorithm, which we implemented using R, with a set of word pairs with the distance metrics as part of the vector. Intuitively, we expect the resulting clusters to be meaningful in case our proposed metrics indeed possessed a utility to separate the words. The pairs we provided to the k-means clustering algorithm are Arabic words of similar letters and different diacritics.

We performed k-means clustering using the imply distance alone, the morphological distance alone, and both distances. We repeated the clustering with the number of non-diacritic letters in the word as an additional input feature. The unsupervised training over the word-pairs dataset (18,145 pairs) returned meaningful clusters indeed, which illustrates an evidence of the utility of the metrics we presented in previous sections.

Figure 6 reports on the results of the k=5 clustering using both metrics (Imply and Morph distances) and with the number of letters in a word. Figure 7 illustrates the number of pairs in each of the 5 clusters, grouped by the number of letters in each word. The x-axis

shows the morph distance from 0.0 to 1.0, where 0.0 means that the pair is a perfect match and 1.0 means that the pair is different. Some pairs had words with no morphological solutions and thus they show in the No-Results bin. The *y*-axis shows the Imply distance where a score of 0 denotes a perfect match and scores of 15 and above denotes a different pair of words.

Cluster 1 contains 1,186 pairs, among the total of the 18,145 pairs. Most words in this cluster are with more than 5 letters, and their Imply distance ranges from 3 to 6. There are no important correlations between the Imply and Morph distances in this cluster, however, there are only a few cases with Morph distance 0 and Imply distance above 4. All of these cases started with variants of Hamza but considered "Same" indeed. Furthermore, we found that all words in this cluster either started with Hamza or were continuous present tense derivation verbs that started with the ta (ت) prefix.

Cluster 2 contained 11,505 pairs. Interestingly, although all words in this cluster are with compatible *shadda,* except few cases with a *madda*, the extreme majority of them are with Morph distance 1 and Imply distance 0 to 3. That is, both distance metrics are confirming the decisions of each other. Both *shadda* and *madda* denote a stress of the letter that implies another letter. The rest either had *shadda*, had *madda*, differed only in one diacritic present in a word and missing in another, or followed one of *fa3ala*, *tafa3al*, *istaf3al* and *if3alala* derivation patterns.

Cluster 3 contained 973 pairs that are very close according to the Imply distance but had no Morph distance results. That is, these are words that are not found in the LDC's SAMA morphological database, thus have no Morph distance, but their Imply distance is between 0 and 2. This case illustrates further the usefulness of the Imply distance in deciding whether to words are "Same" though unknown to the morphological analyzer.

Cluster 4 has 3,889 pairs with an Imply distance 2 and variant morphological distance. Upon inspection these are (1) 60% of the pairs have Morph distance 1, which means that both metrics agree, (2) words with high difference in the count of diacritics so the implication from the word with fewer diacritics to the one with more diacritics returns a positive answer while the morphological solutions are much higher, and (3) words that are with similar number of diacritics, yet they are morphologically rich such that the morphological metric differs. That is, such morphologically rich words mean that many different solutions are retrieved in the morphological analyses phase when the diacritics were removed.

Cluster 5 has 592 pairs of words and illustrates an interesting finding by the k-means algorithm. These are indeed the special cases we reported in subsection 6.3, with Imply distance +15 (because of strong *shadda* difference) and with Imply distance 1 (according to the knowledge-based morphological analyses). This is why they were classified as "Different" by the Imply algorithm, and "Same" by the Subsume algorithm. As explained in subsection 6.3, though the absence of *shadda* is a critical (e.g., in جوع، جَوَّع), but, since there are no other possibilities in Arabic to spell such words without *shadda*, they were considered "Same", given the used morphological background database of Arabic.

The use of this unsupervised k-mean clustering was able to reveal meaningful clusters indeed. Clusters 3 and 5 illustrate the border cases of our distance metrics; respectively, the case of no Morph distances but found to be very similar pairs by low Imply distance, and the case of high Imply distance but found to be the same by the Morph distance of 1. The other three clusters illustrate that both distance metrics confirm each other as discussed above. Also, as discussed in section 6 and confirmed in this section, the Imply distance is a reliable metric and a framework for Arabic words matching. The case where this metric

returned "Different" pairs, in cluster 5, are not critical, not only because they are typically rare, but also they are problematic for linguists too, see subsection 6.3.

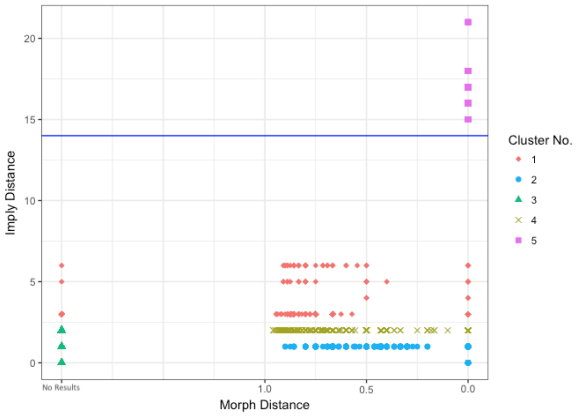

Figure 6: k-means clustering (k=5) based on Imply distance and Morph distance

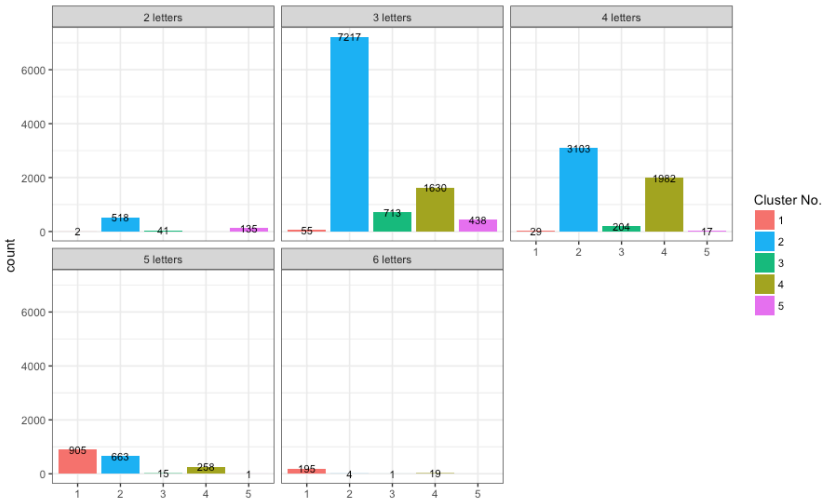

Figure 7: Frequency of the 5-means clustering (grouped by number of letters in words)

9. CONCLUSION AND FUTURE WORK

We presented three algorithms that compare between Arabic words. The Imply algorithm encodes expert knowledge in a set of expert rules and reports an implication direction, an implication measure and a matching verdict. The Subsume algorithm computes the morphological subsumption relation of one word with respect to the other, based on background knowledge. The Alike algorithm uses machine learning to classify words as same or different. We evaluated the algorithms, and experiments showed that all of them are highly accurate. We also compared between the algorithms based on their soundness, completeness and predictability. Finally, we illustrated the utility and the need for our algorithms in a dictionary integration scenario.

Future plans for the presented algorithms include further testing with an even larger dataset than the 86,886 pairs used and using word forms such as nouns and adjectives. Using larger datasets for learning will allow for even more accurate results, especially for the Alike algorithm. Also planned, is the expansion of the algorithms to include taking multi-word phrases as input and returning sets of words considered equal.

ACKNOWLEDGMENTS

This research was partially funded by Birzeit University (VerbMesh project, funded by BZU research committee), partially by Google's Faculty Research Award to Mustafa Jarrar and partially by grants from the Lebanese National Council for Scientific Research to Fadi Zaraket. The authors are very thankful to Mohamad Dwaikat, Faeq Alrimawi, and Reema Taha for helping in partial implementation and in the manual evaluation of the results. Finally, the authors would like to thank the anonymous reviewers for their valuable and significant comments that helped improve the paper.

REFERENCES

Abuaiadah, Diab, Dileep Rajendran, Mustafa Jarrar (2017): Clustering Arabic Tweets for Sentiment Analysis. IEEE/ACS 14th International Conference on Computer Systems and Applications.

Alqrainy, Shihadeh, Hasan AlSerhan, and Aladdin Ayesh. 2008. Pattern-based algorithm for part-of-speech tagging Arabic text. Proceedings of ICCES, IEEE, (pp.119–124).

Attia, Mohammed. 2008. Handling Arabic morphological and syntactic ambiguity within the LFG framework with a view to machine translation. PhD Dissertation. University of Manchester.

Azmi, A. M., & Almajed, R. S. (2015). A survey of automatic Arabic diacritization techniques. Natural Language Engineering, 21(3), 477-495.

Bahanshal, Alia and Hend Al-Khalifa. 2012. A first approach to the evaluation of Arabic diacritization systems. Proceedings of ICDIM, IEEE, (pp.155–158).

Beesley, Kenneth. 2001. Finite-state morphological analysis and generation of Arabic at Xerox research: Status and plans," In ACL Workshop on Arabic Language Processing: Status and Perspective (Vol. 1, pp. 1-8).

Belinkov, Y., Glass, J. 2015. Arabic diacritization with recurrent neural networks. In Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing.

Boujelben, Makram, Chafik Aloulou, and Lamia Hadrich Belguith. 2008. Toward a Robust Detection/Correction System for the Agreement Errors In Non-Voweled Arabic Texts." Proceedings of ACIT 2008.

Buckwalter, Tim. 2002. Buckwalter {Arabic} Morphological Analyzer Version 1.0. LDC catalog number LDC2002L49, Technical Report.

Darwish, K., Mubarak, H., & Abdelali, A. (2017). Arabic Diacritization: Stats, Rules, and Hacks. Proceedings of the 3rd Arabic Natural Language Processing Workshop (pp. 9-17).

Debili, Fathi, Hadhémi Achour, and E. Souissi. 2002. De l'étiquetage grammatical à la voyellation automatique de l'arabe. Technical Report.

Habash, Nizar. 2007. Arabic morphological representations for machine translation. book chapter. In Arabic computational morphology (pp. 263-285). Springer Netherlands.

Habash, Nizar, Owen Rambow, and Ryan Roth. 2009. MADA+ TOKAN: A toolkit for Arabic tokenization, diacritization, morphological disambiguation, POS tagging, stemming and lemmatization." Proceedings of MEDAR'09, Egypt.

Hattab, Abdullah, and Abdulameer Hussain. 2012. Hybrid Statistical and Morpho-Syntactical Arabic Language Diacritizing System. International Journal of Academic Research, 4(4).

Kammoun, Nouha Chaâben, Lamia Hadrich Belguith, Abdelmajid Ben Hamadou. 2010. The MORPH2 new version: A robust morphological analyzer for Arabic texts. In JADT 2010: 10th International Conference on Statistical Analysis of Textual Data.

Khorsheed, Mohammad. 2013. An HMM-based system to diacritize Arabic text. Journal of Software Engineering and Applications Vol (5): 124.

Kiraz, George Anton.1998. Arabic computational morphology in the West. Proceedings of the 6th

International Conference and Exhibition on Multi-lingual Computing (pp. 3-5).

Kulick, Seth, Ann Bies, and Mohamed Maamouri. 2010. Consistent and flexible integration of morphological annotation in the Arabic treebank. In Proceedings of LREC'2010, Malta.

Jarrar, Mustafa, Nizar Habash, Faeq Alrimawi, Diyam Akra, Nasser Zalmout (2016): Curras: An Annotated Corpus for the Palestinian Arabic Dialect. Journal Language Resources and Evaluation. 51(3):745-775.

Jarrar, Mustafa, Nizar Habash, Diyam Akra, Nasser Zalmout (2014): Building a Corpus for Palestinian Arabic: a Preliminary Study. The EMNLP Workshop on Arabic Natural Language Processing. ACL.

Jarrar, Mustafa (2011): Building a Formal Arabic Ontology. Proceedings of the Experts Meeting on Arabic Ontologies and Semantic Networks. ALESCO, Arab League.

Jarrar, Mustafa, Anton Deik, Bilal Faraj (2011): Ontology-Based Data and Process Governance Framework -The Case of E-Government Interoperability in Palestine. The IFIP International Symposium on Data-Driven Process Discovery and Analysis.

Jarrar, Mustafa (2006): Towards the Notion of Gloss, and the Adoption of Linguistic Resources in Formal Ontology Engineering. The 15th International World Wide Web Conference. ACM Press.

Jarrar, Mustafa, Stijn Heymans (2008): Towards Pattern-Based Reasoning for Friendly Ontology Debugging. Journal of Artificial Intelligence Tools. World Scientific Publishing. 17(4).

Mohamed B., Chennoufi A., Mazroui A., and Lakhouaja A. 2014. Hybrid approaches for the automatic vowelization of Arabic texts. Natural Language Computing. 3(4).Quinlan, J. R. 1993. C4.5: Programs for Machine Learning. Morgan Kaufmann Publishers.

Rashwan, Mohsen, Mohamed Al-Badrashiny, Mohamed Attia, Sherif Abdou, and Ahmed Rafea. 2011. A stochastic Arabic diacritizer based on a hybrid of factorized and unfactorized textual features. IEEE Transactions on Audio, Speech, and Language Processing, 19(1):166-175.

Roth, Ryan, Owen Rambow, Nizar Habash, Mona Diab, and Cynthia Rudin. 2008. Arabic morphological tagging, diacritization, and lemmatization using lexeme models and feature ranking. the 46th Annual Meeting of the ACL: Short Papers, ACL (pp. 117-120).

Said, Ahmed, Mohamed El-Sharqwi, Achraf Chalabi, and Eslam Kamal. 2013. A hybrid approach for Arabic diacritization. In International Conference on Application of Natural Language to Information Systems, (pp. 53-64). Springer Berlin Heidelberg.

Seraye, Abdullah. 2004. The Role of Short Vowels and Context in the Reading of Arabic, Comprehension and Word Recognition of Highly Skilled Readers. PhD Thesis. University of Pittsburgh.

Vergyri, Dimitra 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, ACL. (pp. 66–73).

Zitouni, I., Sarikaya, R. 2009. Arabic Diacritic Restoration Approach Based on Maximum Entropy Models. Computer Speech & Language, 23(3).