Natural Language Engineering **22** (5): 751–773. © Cambridge University Press 2015
doi:10.1017/S1351324915000030

Arabic spelling error detection and correction†

MOHAMMED ATTIA[1,2], PAVEL PECINA[3], YOUNES SAMIH[4], KHALED SHAALAN[2] and JOSEF VAN GENABITH[1]

[1]*School of Computing, Dublin City University, Ireland,*
e-mail: mattia@computing.dcu.ie, josef@computing.dcu.ie
[2]*Faculty of Engineering and IT, The British University in Dubai, UAE*
e-mail: khaled.shaalan@buid.ac.ae
[3]*Faculty of Mathematics and Physics, Charles University in Prague, Czech Republic*
e-mail: pecina@ufal.mff.cuni.cz
[4]*Department of Linguistics and Information Science, Heinrich-Heine-Universität Düsseldorf, Germany*
e-mail: samih@phil.uni-duesseldorf.de

(*Received 31 October 2013; revised 8 February 2015; accepted 12 February 2015; first published online 18 March 2015*)

Abstract

A spelling error detection and correction application is typically based on three main components: a dictionary (or reference word list), an error model and a language model. While most of the attention in the literature has been directed to the language model, we show how improvements in any of the three components can lead to significant cumulative improvements in the overall performance of the system. We develop our dictionary of 9.2 million fully-inflected Arabic words (types) from a morphological transducer and a large corpus, validated and manually revised. We improve the error model by analyzing error types and creating an edit distance re-ranker. We also improve the language model by analyzing the level of noise in different data sources and selecting an optimal subset to train the system on. Testing and evaluation experiments show that our system significantly outperforms Microsoft Word 2013, OpenOffice Ayaspell 3.4 and Google Docs.

1 Introduction

Spelling correction solutions have significant importance for a variety of applications and NLP tools including text authoring, OCR (Tong and Evans 1996), search query processing (Gao *et al.* 2010), pre-editing or post-editing for parsing and machine translation (El Kholy and Habash 2010; Och and Genzel 2013), intelligent tutoring systems (Heift and Rimrott 2008), etc. In this introduction, we define the spelling error detection and correction problem, present a brief account of relevant work,

† We are grateful to our anonymous reviewers whose comments and suggestions have helped us to improve the paper considerably. This research is funded by the Irish Research Council for Science Engineering and Technology (IRCSET), the UAE National Research Foundation (NRF) (Grant No. 0514/2011), the Czech Science Foundation (grant no. P103/12/G084), DFG Collaborative Research Centre 991: The Structure of Representations in Language, Cognition, and Science (http://www.sfb991.uni-duesseldorf.de/sfb991), and the Science Foundation Ireland (Grant No. 07/CE/I1142) as part of the Centre for Next Generation Localisation (www.cngl.ie) at Dublin City University.

https://doi.org/10.1017/S1351324915000030 Published online by Cambridge University Press

outline core aspects of Arabic morphology and orthography, and provide a summary of our research methodology.

1.1 Problem definition

The spelling correction problem is formally defined (Brill and Moore 2000) as: given an alphabet Σ, a dictionary D consisting of strings in $\Sigma^\star$, and a spelling error s, where $s \notin D$ and $s \in \Sigma^\star$, find the correction c, where $c \in D$, and c is most likely to have been erroneously typed as s. This is treated as a probabilistic problem formulated as in (1) (Kernigan, Church and Gale 1990; Brill and Moore 2000; Norvig, 2009):

$$argmax_c P(s|c)P(c) \tag{1}$$

Here c is the correction, s is the spelling error, $P(c)$ is the probability that c is the correct word (or the language model), and $P(s|c)$ is the probability that s is typed when c is intended (this is called the error model or noisy channel model), $argmax_c$ is the scoring mechanism that computes the correction c that maximizes the probability $P(s|c)P(c)$.

Based on this definition, we assume that a good spelling correction system needs a balanced division of labor between the three main components: the dictionary, error model and language model. In this paper, we show that in the error model there is a direct relationship between the number of correction candidates and the likelihood of finding the correct correction: the larger the number of candidates, the more likely the error model is to find the best correction. At the same time, in the language model there is an inverse relationship between the number of candidates and the ability of the model to decide on the desired correction: the larger the number of candidates, the less likely the language model will be successful in making the right choice. A language model is negatively affected by a high dimensional search space. A language model is also negatively affected by noise in the data when the size of the data is not large.

In the error model, we deploy the dictionary in a finite-state automaton to propose candidate corrections for misspelled words within a specified edit distance (Ukkonen 1983; Hulden 2009b) from the correct words. Based on an empirical analysis of the types of errors, we devise a set of frequency-based rules for re-ranking the candidates generated via edit distance operations so that when the list of candidates is pruned, we do not lose many plausible correction candidates.

For the n-gram language model, we use the Arabic Gigaword Corpus 5th Edition (Parker *et al.* 2011), the largest available so far for Arabic, and an additional corpus of news articles crawled from the Al-Jazeera web site. The Gigaword corpus is divided into nine data sets according to the data sources, such as Agence France-Presse, Xinhua News Agency, An Nahar, Al Hayat, etc. We analyze the various data sets to estimate the amount of noise (the ratio of spelling errors against correct text), and our n-gram language modeling experiments show that there is clear association between the amount of noise and the disambiguation quality of the model.

To sum up, the system architecture is a pipeline of three components where the output of one component serves as the input to the next, and these components are:

https://doi.org/10.1017/S1351324915000030 Published online by Cambridge University Press

(1) Error detection through a dictionary (or a reference word list).
(2) Candidate generation through edit distance as implemented in a finite state compiler.
(3) Best candidate selection using an n-gram language model.

1.2 Arabic morphology and orthography

Arabic has a rich and complex morphology as it applies both concatenative and non-concatenative morphotactics (Beesley 1998; Ratcliffe 1998). Concatenative morphotactics models the addition of clitics and affixes to the word stem without affecting the internal structure of the word, such as شَكَرَ '$akara'[1] 'to thank', which can be inflected as, وَشَكَرَتهُ 'wa-$akara-to-hu' 'and she thanked him'. On the other hand, non-concatenative morphotactics models the employment of internal alterations to a word in order to express both inflectional and derivational phenomena, such as دَرَّسَ 'dar~asa' 'to teach', which can be inflected as دُرِّسَ 'dur~isa' in the passive and دَرِّس 'dar~is' in the imperative. In Arabic morphology this is typically modeled by a group of morphological templates, or patterns.

Both concatenative and non-concatenative morphotactics are frequently seen working together in words such as وَسَيُستَدعَونَ 'wa-sa-yusotadoEa-wona' 'and they will be summoned' where non-concatenative morphotactics is used to form the passive, while concatenative morphotactics are used to produce the tense, number and person, as well as the affixes of the conjunction and future particles. Arabic has a wealth of morphemes that express various morpho-syntactic features, such as tense, person, number, gender, voice and mood for verbs, and number, gender, and definiteness for nouns, in additional to a varied outer layer of clitics. This is the basis of the considerable generative power of the Arabic morphology. This is illustrated by the fact that a verb, such as شَكَرَ '$sakara', generates 2,552 valid forms, and a noun, such as مُعَلِّم 'muEal~im' 'teacher', generates 519 valid forms (Attia 2006).

Arabic orthography has a unified and standardized set of rules that are almost unanimously agreed upon by traditional grammarians. However, on the one hand, some of these rules are too complicated to be grasped and followed by everybody, and on the other hand, writers will sometimes opt for speed, when writing on a keyboard, and become reluctant to press the shift key, leading them to use one character to represent many others. For example the bare *alif* ا 'A' is written without the shift key, but the other hamzated forms, such as أ '>', إ '<', and آ 'ا' will need the shift key. This led to what is called 'typographic errors' or 'orthographic variations' (Buckwalter 2004a). These orthographic variations, which can sometimes be referred to as sub-standard spellings, or soft spelling errors, are basically related to the possible overlap between orthographically similar letters in three categories:

[1] Throughout this paper, we use the Buckwalter transliteration system: http://www.qamus.org/transliteration.htm

https://doi.org/10.1017/S1351324915000030 Published online by Cambridge University Press

(a) the various shapes of *hamzah*s (ا 'A', أ '>', إ '<', آ '|', ئ '}', ء ''', ؤ '&'), (b) *taa marboutah* and *haa* (ة 'p', ه 'h'), and (c) *yaa* and *alif maqsoura* (ي 'y', ى 'Y'). It should also be noted that, in modern writing, vowel marks (or diacritics) are normally omitted, which means that شَكَرَ is merely written as شكر. This leads to a substantial amount of ambiguity when deciding on the correct vowelization, an issue that has a considerable impact on NLP tasks related to POS tagging and speech applications. This problem, however, is not relevant to the current task as we only deal with unvowelized text as it appears in the newspapers.

1.3 Relevant work

Detecting and correcting spelling errors is one of the problems that intrigued NLP researchers from an early stage. Damerau (1964) was among the first researchers to address this issue. He developed a rule-based string-matching technique for error correction, based on four edit operations (substitution, insertion, deletion, and transposition), but his work was limited by memory and computation constraints at the time. Church and Gale (1991) were the first to rank the list of spelling candidates by probability scores (considering word bigram probabilities) based on a noisy channel model. Kukich (1992), in her survey, classified work on spelling errors into three categories: (a) error detection, (b) isolated word correction, and (c) context-dependent (or context-sensitive) correction. Brill and Moore (2000) tried to improve the noisy channel model by learning generic string-to-string edits, along with the probabilities of each of these edits. Van Delden, Bracewell and Gomez (2004) used machine-learning methods (supervised and unsupervised) for handling spelling errors, including errors related to word merging and splitting.

Beside n-gram language modeling, statistical machine t\ranslation (SMT) has also been used for the task of spelling correction. Han and Baldwin (2011) perform normalization of ill-formed words in Twitter short messages. They generate a text normalization data set, and then use a phrase-based SMT for the selection of candidates. Wu, Chiu and Chang (2013) use a similar method in building a spelling error correction and detection system for Chinese using a decoder based on the SMT model for correction.

In our research, we address the spelling error detection and correction problem for Arabic, a morphologically rich language with a large array of orthographic variation. We focus on isolated word errors, i.e. non-word spelling errors, or strings that do not form valid words in the language. At the current stage, we do not handle context-sensitive errors.

The problem of spell checking and spelling error correction for Arabic has been investigated in a number of papers. Shaalan, Allam and Gomah (2003), Shaalan, Magdy and Fahmy (2013), and Alfaifi and Atwell (2012) provide characterization and classification of spelling errors in Arabic. Haddad and Yaseen (2007) propose a hybrid approach that utilizes morphological knowledge to formulate morpho-graphemic rules to specify the word recognition and non-word correction process. For correction, they use two probabilistic measures: Root-Pattern Predictive Value

https://doi.org/10.1017/S1351324915000030 Published online by Cambridge University Press

and Pattern-Root Predictive Value. They also consider keyboard effect and letter–sound similarity. No testing of the system performance has been reported. Hassan, Noeman and Hassan (2008) develop a language independent system that uses finite state automata to propose candidate corrections within a specified edit distance from the misspelled word. After generating candidates, a word-based language model is used to assign scores to the candidates and choose the best correction in the given context. They use an Arabic dictionary of 526,492 full form entries and test it on 556 errors. However, they do not specify the data the language model is trained on or the order of the n-gram model. They also do not indicate whether the test errors are actual errors extracted from real texts or artificially generated. Furthermore, their system is not compared to any other existing systems.

Shaalan *et al.* (2012) use the Noisy Channel Model trained on word-based unigrams for spelling correction, but their system performs poorly against the Microsoft Spell Checker. Alkanhal *et al.* (2012) developed a spelling error detection and correction system for Arabic directed mainly towards data entry errors, but the problem of their work is that they test on the development set which could make their work subject to overfitting. Moreover, the small size of their dictionary (427,000 words) questions the coverage of their model when applied to other domains.

In recent years, there has been a surge of interest in spelling correction for Arabic. The QALB (Qatar Arabic Language Bank) project[2] has started as a joint venture between CMU-Qatar and Columbia University, with the aim of building a corpus of manually corrected Arabic text for building automatic correction tools for Arabic text. They released the guidelines in Zaghouani *et al.* (2014). The group has also participated in the EMNLP 2014 Conference with a shared task on Automatic Arabic Error Correction[3]. However, the domain in the QALB shared task is user comments (or unedited text), while the domain of our project is edited news articles. The type of errors handled in the QALB data is punctuation errors (accounting for 40% of all errors), grammar errors, real-spelling errors and non-word spelling errors, beside normalization of numbers and colloquial words, but our data is focused only on formal non-word spelling errors.

1.4 Our methodology

Our research differs from the previous work on Arabic in a number of respects: we use an n-gram language model (mainly bigrams) trained on the largest available corpus to date, the Arabic Gigaword Corpus 5th Edition, supplemented by news data crawled automatically from the Al-Jazeera web site. In addition, we provide frequency-based typification of the spelling errors by comparing the errors with the gold correction and characterizing the edit operations involved. Based on this classification, we develop frequency-based re-ranking rules for reordering and constraining the number of candidates generated via edit distance and integrate them into the overall model. Furthermore, we show that careful selection of the

[2] http://nlp.qatar.cmu.edu/qalb/
[3] http://emnlp2014.org/workshops/anlp/shared_task.html

https://doi.org/10.1017/S1351324915000030 Published online by Cambridge University Press

language model training data based on the amount of noise present in the data, has the potential to further improve the overall results. Moreover, we focus on the importance of the dictionary (word list) in the processes of spell checking and candidate generation. We show how our word list is created and how it is more accurate in error detection than what is used in other systems.

In order to test and evaluate the various components of our system, we create a development set and a test set, and both are manually annotated by a language expert. The development set consists of 444,196 tokens (words with repetitions), and 59,979 types (unique words), collected from documents from Arabic news web sites. Of this development set, 2,027 misspelt types are manually identified and provided with gold corrections. For the test set, we collect 471,302 tokens (50,515 types) from the Watan-2004 corpus by Mourad Abbas,[4] selecting the first 1,000 articles of the International section. In the test set, 53,965 tokens (7,669 types) are manually annotated as errors, and of these errors, 49,690 tokens (5,398 types) are provided with corrections. Misspelt words that do not receive corrections are marked as 'unknown' either because they are colloquial or classical words, foreign or rare words, infrequent proper nouns, or simply unknown. To save time, the annotator worked on types for spelling error tagging. However, in order to assign corrections, the annotator worked on tokens; reviewing each word in context in the corpus. The reason behind this is that it is not always possible to determine without context what the correction should be. For example, the misspelt word احداث AHdAv, can be corrected either as أحداث >HdAv 'events' or إحداث <HdAv 'effecting', depending on the context. Here are the guidelines given to the annotator:

(1) Misspelt words need to be corrected in context in the corpus. Bear in mind that a misspelt word can have more than one possible correction depending on the context.
(2) If a proper noun is familiar or frequent (by consulting frequency counts on Google and Al-Jazeera web site), then it should be considered correct, otherwise it should be corrected or tagged 'UNK' (unknown).
(3) Words should be tagged UNK if they are:
    (a) not known
    (b) purely colloquial or classical
    (c) foreign and unfamiliar
    (d) extremely rare.

We use the development set for analyzing the types of errors and fine-tuning the parameters of the candidate re-ranking component described in Section 4 and summarized in Table 4. The blind test set is used to evaluate our system and compare it to Microsoft Word 2013, OpenOffice Ayaspell version 3.4 (released 1st of March 2014), and Google Docs (tested in April 2014). Our system performs significantly better than these three systems both in the tasks of spell checking and automatic correction (or first-order ranking).

[4] http://sites.google.com/site/mouradabbas9/corpora

https://doi.org/10.1017/S1351324915000030 Published online by Cambridge University Press

The remainder of this paper is structured as follows: Section 2 shows how our dictionary (or word list) is created from the AraComLex finite-state morphological analyzer and generator (Attia *et al.* 2011). This dictionary is compared with other available resources. Section 3 illustrates how spelling errors are detected and explains our methods of using character-based language modeling to predict valid words versus invalid words. Section 4 explains how the error model is improved by analyzing error types and deducing rules to improve the ranking produced through finite-state edit distance. Section 5 shows how the language model can be improved by selecting the right type of data to be trained on. Various data sections are analyzed to detect the amount of noise they contain, then suitable subsets of data are chosen for the n-gram language model training and the evaluation experiments. Finally, Section 6 concludes.

2 Improving the dictionary

The dictionary (or word list) is an essential component of a spell checker/corrector, as it is the reference against which the decision can be made whether a given word is correct or misspelled. It is also the reference against which correction candidates are filtered. There are various options for creating a word list for spell checking. It can be created from a corpus, a morphological analyzer/generator, or both. The quality of the word list will inevitably affect the quality of the application whether in checking errors or generating valid and plausible candidates.

For Arabic, one of the earliest word lists created for the purpose of spell checking is the Arabic Spell[5] open-source project (designed for Aspell), which relies on the Buckwalter morphological analyzer (Buckwalter 2004b). This list generates about 900,000 fully inflected words. Another dictionary is the Ayaspell[6] word list, which is the official resource used in OpenOffice applications. Developers of this word list created their own morphological generator, and their word list contains about 300,000 inflected words. In this paper, we use the term 'word' to designate fully inflected surface word forms, while the term 'lemma' is used to indicate the uninflected base form of the word without affixes or clitics.

In our research, we create a very large word list for Arabic using AraComLex[7] (Attia *et al.* 2011), an open-source large-scale morphological transducer. AraComLex contains 30,587 lemmas and is developed using finite state technology. There are a number of advantages of finite-state technology that makes it especially attractive in dealing with human language morphologies (Wintner 2008). They include bidirectionality and the ability to generate as straightforwardly as to analyze.

AraComLex generates about 13 million surface word forms, of which 9 million are found to be valid forms when checked by the Microsoft Spell Checker (Office 2013). For the sake of comparison, we also use a list of 2,662,780 surface word types created from a text corpus (from the Arabic Gigaword corpus and data crawled

[5] http://sourceforge.net/projects/arabic-spell/files/arabic-spell
[6] http://ayaspell.sourceforge.net
[7] http://aracomlex.sourceforge.net

https://doi.org/10.1017/S1351324915000030 Published online by Cambridge University Press

Table 1. *Arabic word lists matched against Microsoft spell checker*

	Word types	MS accepted	MS rejected
AraComLex	12,951,042	8,783,856	4,167,186
Arabic-Spell for Aspell (using Buckwalter)	938,977	673,874	265,103
1 billion-word corpus (Gigaword and Al-Jazeera)	2,662,780	1,202,333	1,460,447
Ayaspell for Hunspell 3.1	292,464	230,506	61,958
Total (duplicates removed)	15,147,199	9,306,138	5,841,061

from the Al-Jazeera web site) of 1,034,257,113 tokens. At one stage of the validation processes, we automatically match the word lists against the Microsoft Spell Checker to determine which words are accepted and which are rejected. It is to be noted that we relied on MS Spell Checker at this initial stage for the purpose of bootstrapping our dictionary, because it was the best performing software at the time. The results are shown in Table 1. We take the combined (AraComLex and corpus data) and filtered (through Microsoft Spell Checker) list of 9,306,138 words types as our initial list and name it 'AraComLex Extended 1.0'. It is to be pointed out that AraComLex (due to the fact that it is a morphological analyzer) has a relatively poor coverage of named entities, but this deficiency is handled in AraComLex Extended 1.0 through the augmentation from the combined Gigaword and crawled Al-Jazeera corpus data.

A second round of validation has been conducted by checking our word list against the Buckwalter morphological analyzer, and later rounds have been conducted manually on high-frequency words. The output of these series of checking and validation is the latest version of AraComLex Extended, that is version 1.5[8]. Table 2 presents the results of the evaluation of the different word lists against AraComLex Extended 1.5 using the test set, and it shows that AraComLex Extended 1.5 significantly outperforms the other word lists in precision, recall and f-measure. It must be noted, however, that Ayaspell for Hunspell, as is the standard with Hunspell dictionaries, comes in two files: the .dic file which is the list of words, and the .aff file which is a list of rules and other options. Table 2 evaluates only the Ayaspell word list file, but the system as a whole is evaluated in the next section.

By comparing our word list to those available for other languages, we find that for English there are, among other word lists, AGID[9], which contains 281,921 types, and SCOWL[10], containing 708,125; for French, there is a word list that contains

[8] http://sourceforge.net/projects/arabic-wordlist/files/Arabic-Wordlist-1.5.zip
[9] http://sourceforge.net/projects/wordlist/files/AGID/Rev%204/agid-4.zip/download
[10] http://sourceforge.net/projects/wordlist/files/SCOWL/Rev%207.1/scowl-7.1.zip/download

https://doi.org/10.1017/S1351324915000030 Published online by Cambridge University Press

Table 2. *Evaluation of Arabic word lists matched against Microsoft Spell Checker*

	Word types	Precision	Recall	F-measure
AraComLex	12,951,042	98.42	95.69	97.04
Arabic-spell for aspell (using Buckwalter)	938,977	89.47	42.57	57.69
1 billion-word corpus (Gigaword and Al-Jazeera)	2,662,780	85.64	99.79	92.18
Ayaspell for Hunspell 3.1	292,464	97.64	28.13	43.68
AraComLex Extended 1.5	9,199,554	**99.30**	**99.09**	**99.19**

338,989 types[11]. The largest word list we find on the web is a Polish word list for Aspell containing 3,024,852[12]. This makes our word list one of the largest for a human language so far. Finnish and Turkish are agglutinative languages with rich morphology that can lead to an explosion in the number of words, similar to Arabic, but word lists for these two languages are not available to us yet. The large number of word types in our list is further testimony to the morphological productivity of the Arabic language (Kiraz 2001; Watson 2002; Beesley and Karttunen 2003; ; Hajič and Jin, 2005).

3 Error detection

For spelling error detection, we use two methods, the direct method, that is matching against the dictionary (or word list), and a character-based language modeling method in case such a word list is not available.

3.1 Direct detection

The direct way for detecting spelling errors is to match words in an input text against a dictionary, or list of correct words. Such a dictionary for Arabic can run into several million surface forms as shown earlier. This is why it is more efficient to use finite state automata to store words in a more compact manner. An input string can then be composed against the valid word list paths and spelling errors will merely be the difference between the two word lists (Hassan *et al.* 2008, Hulden 2009a).

We evaluate the task of error detection as binary (two class) classification on the test set and compare it with three major text authoring tools: Ayaspell version 3.4, Microsoft Office 2013, and Google Docs. For each word in the test set, the method

[11] http://www.winedt.org/Dict/
[12] Ibid.

https://doi.org/10.1017/S1351324915000030 Published online by Cambridge University Press

Table 3. *Comparison of accuracy, recall, precision, and f-measure of AraComLex Extended 1.5 against other applications*

	Accuracy	Recall	Precision	f-measure
Ayaspell for Hunspell v. 3.4	95.74	96.69	98.26	97.47
Microsoft Word 13	97.68	**99.14**	98.14	98.64
Google Docs (April 2014)	87.91	96.02	90.33	93.09
AraComLex Extended 1.5	**98.63**	99.09	**99.30**	**99.19**

under evaluation predicts if the word is correct (class one) or not (class two). Based on the prediction and the manual annotation, we calculate tp as the number of words correctly predicted as erroneous ('true positives'), fp as the number of words incorrectly predicted as erroneous ('false positives'), tn as the number of words correctly predicted as correct ('true negatives'), and fn as the number of words incorrectly predicted as correct ('false negatives').

Then, we employ the standard binary classification evaluation metrics, calculated as in (2)–(5). *Accuracy* is the ratio of correct predictions (words correctly predicted as erroneous or correct), *precision* is the ratio of correctly predicted items against all predicted items, *recall* is the ratio of all correctly predicted items against all items that need to be found, and *f-1 measure* as the geometric mean of precision and recall.

$$accuracy : \frac{tp + tn}{tp + tn + fp + fn} \tag{2}$$

$$recall : \frac{tp}{tp + fn} \tag{3}$$

$$precision : \frac{tp}{tp + fp} \tag{4}$$

$$f\text{-}measure : 2 \times \frac{precision \times recall}{precision + recall}. \tag{5}$$

As the results in Table 3 show, our system outperforms the other systems in accuracy, precision, and f-measure.

https://doi.org/10.1017/S1351324915000030 Published online by Cambridge University Press

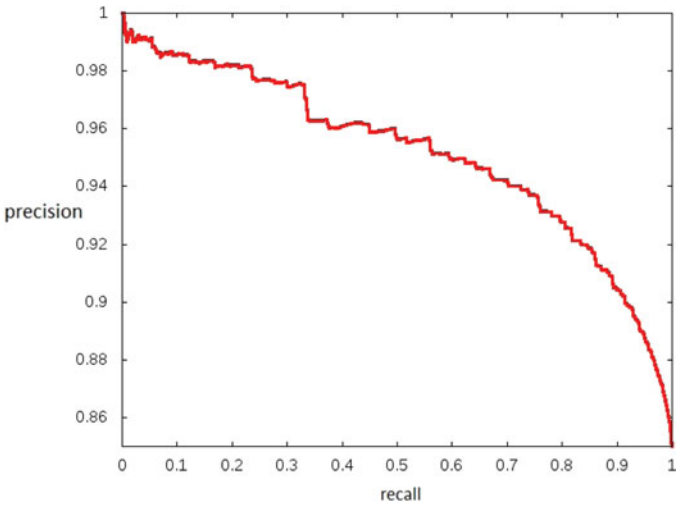


Fig. 1. (Colour online) Results of the LM classifier identifying valid and invalid Arabic word forms.

3.2 Detection through language modeling

Language modeling has been used frequently for the purpose of spelling correction (Brill, Eric and Moore 2000; Magdy and Darwish 2006; Choudhury *et al.* 2007). However, here we build a language model in order to help the validation and classification of Arabic words either in the existing word list or for new words that may be encountered at later stages. Arabic is challenging for language modeling due to the high graphemic similarity of Arabic words. This is shown by Zribi and Ben Ahmed (2003) who conducted an experiment automatically using four edit operations (addition, substitution, deletion, and transposition) to change words, calculating the number of correct forms among the number of automatically built forms (or lexically neighboring words) resulting from these edit operations. They found that the average number of neighboring forms for Arabic is 26.5, which is significantly higher than that for French, 3.5 and English, 3.0.

In this experiment, we build a character-based tri-gram language model using SRILM (Stolcke *et al.* 2011) in order to classify words as valid and invalid. We split each word into characters, and create two language models: one for the total list of words accepted as valid (9,306,138 words), and one for the total list rejected as invalid (5,841,061 words) as filtered through the MS Spell Checker, as shown in Table 1 above. The maximum word length attested in the data is found to be 19 characters. We test the model against our test set, and the results are presented in Figure 1 which shows the precision-recall curve of the classifier. The curve represents precision and recall scores of the detection of spelling errors based on the difference between the perplexity obtained by the accept model and the perplexity of the reject model. The downward movement of the curve indicates that the model is working quite reasonably giving a precision of 85% at a recall of 100%. The model also achieves a precision of around 98% at a recall of 35%. We can identify 60% of all errors with a precision of 95%, i.e. with 5% false alarms only.

https://doi.org/10.1017/S1351324915000030 Published online by Cambridge University Press

4 Improving the error model: candidate generation

For a spelling error s and a dictionary D, the purpose of the error model is to generate the correction c, or list of corrections c_1^n where $c_1^n \in D$, and c_1^n are most likely to have been erroneously typed as s. In order to do this, the error model generates a list of candidate corrections $c_1, c_2, \ldots, c_n$ that bear the highest similarity to the spelling error s.

We deploy finite-state automata to propose candidate corrections within edit distance 1 and 2 measured by the edit distance from the misspelled word (Mitton 1996; Oflazer 1996; Hulden 2009b; Norvig 2009). The automata works basically as a character-based generator that replaces each character with all possible characters in the alphabet as well as deleting, inserting, and transposing neighboring characters. There is also the problem of merged (or run-on) words that need to be split, such as أوأي >w>y 'or any'. These are cases where two words are joined together and the space between them is omitted, such as 'to the' in English when written as 'tothe'.

Candidate generation using edit distance is a brute-force process that ends up with a huge list of candidates. Given that there are 35 alphabetic letters in Arabic, for a word of length n, there will be n deletions, $n-1$ transpositions, $35n$ replaces, $35(n+1)$ insertions and $n-3$ splits, totaling $73n+31$. For example, a misspelt word consisting of six characters will have 469 candidates (with possible repetitions). This large number of candidates needs to be filtered and reordered in such a way that the correct correction comes top or as near the top of the list as possible. To filter out unnecessary forms, candidates that are not found in the dictionary are discarded. The ranking of the candidates is explained in the following subsection.

4.1 Candidate ranking

The ranking of the candidates is initially based on a crude minimum edit distance where the cost assignment is based on arbitrary letter change. In order to improve the ranking, we analyze error types in the development set (containing 2,027 misspelled types with their corrections) to see how they are distributed in order to devise ranking rules for the various edit operations. Table 4 shows the top 20 most common spelling error types.

Based on these frequency observations, we develop a re-ranker to order edit distance operations according their likelihood to generate the most plausible correction. Table 4 shows that *soft errors* are the most frequent type of errors in the data, these are the errors related to *hamzahs* (أ '>', إ '<', ؤ '&', ا 'A', ئ }, ء ''', and آ '|'), the pair of *yaa* (y) and *alif maqsoura* (Y), and the pair of *taa marboutah* (p) and *haa* (h). According to the data analyzed, soft errors account for 71.76% of all the spelling errors. Our re-ranker translates these facts and primes the edit distance scoring mechanism with rules based on the frequency of error patterns in Arabic. It assigns a lower cost score to the most frequently confused character sets (which are often graphemically similar), and higher score for other operations. For speed and efficiency, we use the finite-state compiler Foma (Hulden 2009b) in

https://doi.org/10.1017/S1351324915000030 Published online by Cambridge University Press

Table 4. *Most frequent spelling error types in Arabic*

#	Error type	Ratio %
1.	أ ‘>’ mistaken as ا ‘A’	24.17
2.	splits	16.38
3.	ي ‘y’ mistaken as ى ‘Y’	15.54
4.	إ ‘<’ mistaken as ا ‘A’	15.34
5.	ى ‘Y’ mistaken as ي ‘y’	7.25
6.	deletes	4.44
7.	inserts	3.70
8.	ا ‘A’ mistaken as إ ‘<’	3.26
9.	ة ‘p’ mistaken as ه ‘h’	1.28
10.	transpositions	1.18
11.	أ ‘>’ mistaken as ا ‘A’ and ى ‘Y’ mistaken as ي ‘y’	0.69
12.	ا ‘A’ mistaken as أ ‘>’	0.69
13.	إ ‘<’ mistaken as أ ‘>’	0.64
14.	ؤ ‘&’ mistaken as ء ‘’’	0.54
15.	ه ‘h’ mistaken as ة ‘p’	0.49
16.	أ ‘>’ mistaken as ؤ ‘&’	0.49
17.	أ ‘>’ mistaken as إ ‘<’	0.44
18.	آ ‘|’ mistaken as ا ‘A’	0.39
19.	آ ‘|. mistaken as أ ‘>’	0.30
20.	ا ‘A’ mistaken as آ ‘|’	0.25



Fig. 2. Crude edit distance.

finding candidates within certain edit distances. Figures 2 and 3 show the different configuration files for the crude and re-ranked edit distance.

A similar approach has been followed by Shaalan *et al.* (2003) who defined rules for substituting letters belonging to the same groups (based on graphemic similarity) as shown here.

```
{A, >, <, |}, {b, t, v, n, y}, {j, H, x}, {d, *}, {r, z},
{s, $}, {S, D}, {T, Z}, {E, g}, {f, q}, {p, h}, {w, &}, {y, Y}.
```

As also noticed from Table 4, split words constitute 16% of the spelling errors in the development set, such as عبدالدايم ‘EbdAldAym’ ‘Abdul-Dayem’,

ولاتريد ‘wlAtryd’ ‘and does not want’, and

مايحدث ‘mAyHdv’ ‘what happens’. There are seven words and particles that are commonly found in the joint word forms, and they are: عبد Ebd, يا yA, أبو Abw, ولا wlA, لا lA, وما wmA, ما mA.

https://doi.org/10.1017/S1351324915000030 Published online by Cambridge University Press

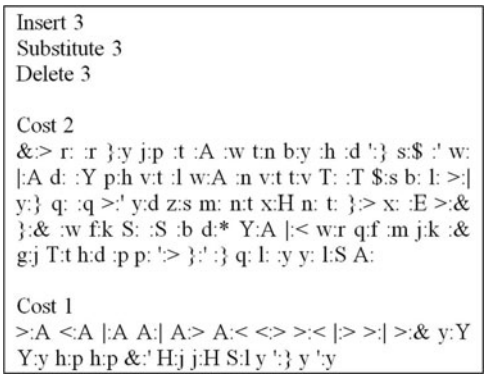


Fig. 3. Re-ranked edit distance.

It is worth mentioning that although the majority of cases with joined words occur with orthographically non-linking letters (such as 'A', 'd', 'w'), there are a few instances where the merge occurs with linking characters as well, such as تحسنملحوظ 'tHsnmlHwZ' 'noticeable improvement' and
حازتأغلبية 'HAzt>glbyp' 'got majority'.

The problem with split words is that they are not handled by the edit distance operation. Therefore, we add a post process for automatically inserting spaces between the various parts of the string. However, this is prone to overgeneration: a word of length n will have $n-3$ candidates, given that the minimum word length in Arabic is two characters. For example *thebag* will have: *th ebag*, *the bag*, and *theb ag*. To filter out bad candidates, the two parts generated from splitting conjoined words are spell checked against the reference dictionary, and if both or either of the two parts is not found, the candidate pair is discarded.

Generating split words for all spelling errors is not a good strategy as this will increase the search space when trying to disambiguate later for the purpose of choosing a single best correction. Therefore, we need to find a method to spot misspelled words that are likely to be an instance of merged words. In order to decide which words should be considered as possibly having a merged word error, we rely on two criteria: word length and lowest edit score. When we analyze the merged words in our development set, we notice that they have an average length of 7.09 characters, with the smallest word consisting of 4 characters and the longest consisting of 15. The average lowest edit score is 2.11. Compared to normal words, we see that the average length is 6.49, the smallest word is 2 and the longest word is 14, with an average lowest edit score of 1.19. We evaluate three criteria for detecting split words on the development set, as shown in Table 5 with w standing for 'word length' and l for 'lowest edit score'. The criteria of 'word length > 3 characters and lowest edit score > 1' has the best f-measure, and therefore we choose it for deciding which words to split.

4.2 Evaluation of the candidate ranking technique

Our purpose in ranking candidates is to allow the correct candidate (the gold correction) to appear at the top or as near the top of the list as possible, so that

https://doi.org/10.1017/S1351324915000030 Published online by Cambridge University Press

Table 5. *Evaluating criteria for deciding split words*

Criteria	Precision	Recall	f-measure	Accuracy
$w > 2$ & $l > 0$	0.17	1	0.28	0.17
$w > 3$ & $l > 1$	0.54	0.88	0.67	0.86
$w > 4$ & $l > 2$	0.73	0.39	0.51	0.88

Table 6. *Comparing crude edit distance with the re-ranker using the development set*

Cut-off limit	Crude edit distance score gold found in candidates		Re-ranked edit distance score gold found in candidates	
	without split words %	after adding split words %	without split words %	after adding split words %
100	79.97	90.97	82.09	93.09
90	79.87	90.87	82.04	93.04
80	79.72	90.73	82.04	93.04
70	79.33	90.33	82.04	93.04
60	78.93	89.94	81.85	92.85
50	78.34	89.34	81.85	92.85
40	77.16	88.16	81.65	92.65
30	75.04	86.04	81.55	92.55
20	71.88	82.88	81.01	92.01
10	64.58	75.58	79.92	90.92
9	62.90	73.90	79.72	90.73
8	61.77	72.77	79.63	90.63
7	59.60	70.60	79.13	90.13
6	56.83	67.83	78.93	89.94
5	53.33	64.33	78.59	89.59
4	48.99	59.99	78.10	89.10
3	44.06	55.06	77.70	88.70
2	37.15	48.15	75.78	86.78
1	23.88	34.88	65.66	76.67

when we reduce the list of candidates, we do not lose many correct ones. We test the ranking mechanism on both the development set (2,027 errors types with corrections) and the test set (5,398 errors types with corrections) as shown in Tables 6 and 7 respectively.

We compare crude edit distance with our revised edit distance re-ranking scorer, and both testing experiments show that the re-ranking scorer performs better at all levels. We notice that when the number of candidates is large the difference between the crude edit distance and the re-ranked edit distance is not big (about 2% absolute for the development set and 0.28% absolute for the test set at the 100 cut-off limit without splits), but when the limit for the number of candidates is lowered the difference increases quite considerably (about 42% absolute for development set and 67% absolute at 1 cut-off limit without splits). This indicates that our frequency-based re-ranker has been successful in pushing good candidates up the top of the list. We also notice that adding splits for merged words has a beneficial effect on all counts.

https://doi.org/10.1017/S1351324915000030 Published online by Cambridge University Press

Table 7. *Comparing crude edit distance with the re-ranker using the test set*

Cut-off limit	Crude edit distance score gold found in candidates		Re-ranked edit distance score gold found in candidates	
	without split words %	after adding split words %	without split split words %	after adding words %
100	97.21	97.80	97.49	97.96
90	97.20	97.79	97.49	97.96
80	97.16	97.75	97.49	97.96
70	97.14	97.73	97.48	97.95
60	97.01	97.60	97.47	97.94
50	96.52	97.11	97.46	97.93
40	94.13	94.72	97.43	97.90
30	82.82	83.41	97.40	97.87
20	75.85	76.44	97.39	97.86
10	54.86	55.45	97.28	97.75
9	52.59	53.18	97.25	97.72
8	50.40	50.99	97.24	97.71
7	48.60	49.19	97.22	97.69
6	46.18	46.76	97.19	97.66
5	43.21	43.80	97.14	97.61
4	39.22	39.81	97.11	97.58
3	35.57	36.16	97.03	97.50
2	29.67	30.26	96.51	96.98
1	20.01	20.60	87.46	87.93

5 Spelling correction

Having generated correction candidates and improved their ranking based on the study of the frequency of the error types, we now use language models trained on different corpora to finally choose the single best correction. We compare the results against the Microsoft Spell Checker in Office 2013, Ayaspell 3.4 used in OpenOffice, and Google Docs (April 2014).

5.1 Correction procedure

For automatic spelling correction (or first-order ranking), we use the n-gram language model. Language modeling assumes that the production of a human language text is characterized by a set of conditional probabilities, $P(w_k|w_1^{(k-1)})$, where $w_1^{(k-1)}$ is the history and w_k is the prediction, so that the probability of a sequence of k words$P(w1,\ldots,w_k)$ is formulated as a product using the Chain Rule for conditional probabilities as in (6) (Brown *et al.* 1992):

$$P(w_1^k) = P(w_1)P(w_2|w_1)\ldots P(w_k|w_1^{(k-1)}). \tag{6}$$

We use the SRILM toolkit (Stolcke *et al.* 2011) to train 2-, 3-, 4-, and 5-gram language models on our data sets. As we have two types of candidates, normal words and split words, we use two SRILM tools: *disambig* and *n-gram*. We use the *disambig* tool to choose among the normal candidates. Handling split words is done

https://doi.org/10.1017/S1351324915000030 Published online by Cambridge University Press

as a posterior step, where we use the *n-gram* tool to score the chosen candidate from the first round and the various split-word options. Then the candidate with the least perplexity score is selected. The *perplexity* of a language model is the reciprocal of the geometric average of the probabilities. So, if a sample text S has $|S|$ words, then the perplexity is $P(S)^{(-1/|S|)}$ (Brown *et al.* 1992). This is why the language model with the smaller perplexity is in fact the one with the higher probability with respect to S.

5.2 Analysing the training data

Our language model is based on raw data from two sources: the Arabic Gigaword Corpus 5th Edition and a corpus of news articles crawled from the Al-Jazeera web site. The Gigaword corpus is a collection of news articles from nine news sources: Agence France-Presse, Xinhua News Agency, An Nahar, Al-Hayat, Al-Quds Al-Arabi, Al-Ahram, Assabah, Asharq Al-Awsat, and Ummah Press.

Before we start using our available corpora in training the language model, we analyze the data to measure the amount of noise in each subset of the data. The concept of data cleanliness and its impact on machine learning processes has been discussed in the literature (Mooney and Buescu 2005; Han and Kamber 2006), with emphasis on the fact that real-world data will tend to be noisy, incomplete and inconsistent, and will need to undergo some sort of cleaning or preparing.

In order to measure the level of cleanliness of our training data, we create a list of the most common spelling errors. This list of spelling errors is created by analyzing the data using MADA (Habash and Rambow 2005; Roth *et al.* 2008) and checking instances where words have been normalized. This is done by matching the analyzed word with the original word, and if there is a literal mismatch, then we know that normalization has taken place. In this case, the original word is considered to be a suboptimal variation of the spelling of the output form. MADA performs normalization on the soft spelling error related to the different shapes of *hamzah*s, *taa marboutah* and *haa*, and *yaa* and *alif maqsoura* explained in more detail in Section 1.2 earlier. We collect these suboptimal forms and sort them by frequency. Then, we select the top 100 misspelled forms and see how frequent they are in the different subsets of data in relation to the word count in each data set. Since soft errors account for 71.76% of all the spelling errors, we believe we have strong ground to assume that the presence of these suboptimal forms is an evidence of the lack of careful editing of the data, giving an indication to the amount noise/cleanliness of the data. It is to be noted that Arabic text denormalization is a subproblem of automated text error correction and a prerequisite for some NLP applications (Moussa, Fakhr and Darwish 2012; El Kholy and Habash 2010). Figure 4 and Table 8 show the varying level of noise in the different subsets of data.

The analysis shows that the data has a varying degree of cleanliness, ranging from the very clean to the very noisy. Data in the Agence France-Presse (AFP) (containing 217,300,912 words) is the noisiest while Ummah Press (3,976,268 words) is the cleanest, and Al-Jazeera (151,329,247 words) is the second cleanest. Due to the fact that the Ummah Press data is not comparable in size to the AFP data, we

https://doi.org/10.1017/S1351324915000030 Published online by Cambridge University Press

Table 8. *Corpus subsets with word count and ratio of noise*

Data set	Word count (tokens)	Ratio of spelling errors to word count (%)
Gigaword 5th edition	1,034,257,113	7.56
Agence France-Presse	217,300,912	11.40
Xinhua news agency	107,280,700	9.96
An Nahar	253,833,020	8.16
Al Hayat	233,666,870	6.10
Al-Quds Al-Arabi	50,279,354	4.92
Al-Ahram	72,681,195	3.85
Assabah	22,858,611	3.80
Asharq Al-Awsat	72,380,183	2.23
Ummah press	3,976,268	0.18
Al-Jazeera	151,329,247	0.22

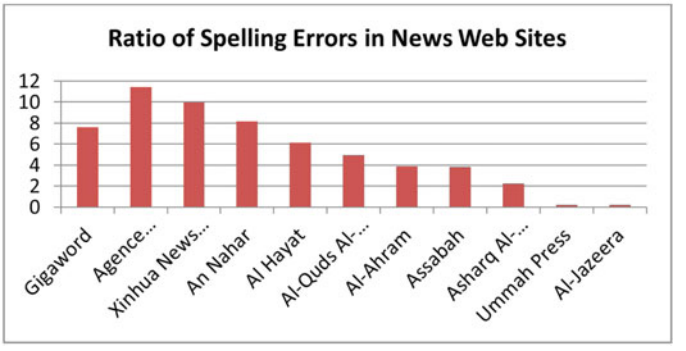


Fig. 4. (Colour online) Ratio of noise in corpus data.

ignore it in our experiments and use instead the Al-Jazeera data for representing the cleanest data set.

5.3 Automatic correction evaluation

For comparison, we first evaluate the automatic correction (or first-order ranking) of three industrial text authoring applications: Google Docs[13], OpenOffice Ayaspell 3.4, and Microsoft Word 2013. Using our test set of 49,690 spelling error tokens with corrections, we test the automatic correction of these systems. The results in Table 9 are reported in terms of accuracy (number of correct corrections divided by the number of all errors).

Next, we evaluate our approach on the test set using language models trained on the AFP data (as representing the noisiest type of data), the Al-Jazeera data (as representing the cleanest type of data) and the entire Gigaword corpus (as representing a huge data set with a moderate amount of noise). We run

[13] Tested in April 2014

https://doi.org/10.1017/S1351324915000030 Published online by Cambridge University Press

Table 9. *Evaluation of first-order ranking of spelling correction of Google Docs, Ayaspell and MS Word 2013*

	Google Docs accuracy %	OpenOffice Ayaspell accuracy %	MS Word accuracy %
Tested on word tokens	2.57	67.43	76.43

our experiments on the candidates generated through the re-ranked edit distance processing explained in Section 4 with varying candidate cut-off limits. We choose the best correction from among the normal candidates using the SRILM *disambig* tool, and for the split words using *n-gram* tool.

As Table 10 shows, the best score achieved for the automatic correction is 93.64% using the bigram language model trained on the Arabic Gigaword Corpus with a candidate cut-off limit of 2, and with the split words added. Table 10 also shows that the system performance deteriorates as the number of candidates increases, which means that the n-gram language model needs a compact number of candidates to disambiguate from among them.

Comparing the LM trained on the two data sets which are comparable in size, the AFP and Al-Jazeera data sets, we find that the LM trained on the AFP has consistently lower scores than those for the LM trained on the Al-Jazeera data. The Al-Jazeera data is relatively clean while the AFP data has a large amount of suboptimal misspellings. We assume that the relatively low performance of the language model trained on the AFP data is due to the amount of noise in the data. However, this assumption is not conclusive, and it can be argued that the difference could simply be due to the different genres or the dialects that are predominant in this data set.

Table 10 shows that the extremely large Gigaword corpus makes up for the effect of noise and produces the best results among all the data sets. The best score achieved for the Gigaword corpus (93.64%) is 0.86% absolute better than the score for Al-Jazeera (92.78%). This could be a further indication in favor of the argument that more data is better than clean data. However, we must notice that the Gigaword data is one order of magnitude larger than the Al-Jazeera data, and in some applications, for efficiency reasons, it could be better to work with the language model trained on a smaller data set. We notice that the addition of the split word component has a positive effect on all test results.

We conducted further experiments with other language models trained on higher order n-grams, going from 2- to 3-, 4- and 5-grams, but the higher n-gram order did not lead to any statistically significant improvement of the results, and sometimes the accuracy even slightly deteriorates, which leads us to believe that the 2-gram language model is good enough for conducting this type of task.

Compared to other spelling error detection and correction systems, we notice that our best accuracy score (93.64%) is significantly higher than that for Google Docs (2.57%), Ayaspell 3.4 for OpenOffice (67.43%), and Microsoft Word 2013 (76.43%) as stated in in Table 9 above.

https://doi.org/10.1017/S1351324915000030 Published online by Cambridge University Press

Table 10. *First-order correction accuracy using the 2-gram LM trained on data from AFT, Al-Jazeera, and the entire Gigaword corpus on the test set*

	Normal candidates accuracy 2-gram			Normal candidates + splitwords accuracy 2-gram		
Candidate cut-off limit	AFP	Jazeera	Gigaword	AFP	Jazeera	Gigaword
100	49.24	74.91	74.09	49.69	75.33	74.53
90	49.84	75.80	75.02	50.28	76.22	75.46
80	50.35	76.15	75.37	50.79	76.58	75.80
70	56.18	79.89	80.46	56.62	80.32	80.90
60	57.17	80.38	81.01	57.62	80.80	81.45
50	58.15	81.02	81.59	58.60	81.45	82.03
40	59.62	81.70	82.32	60.06	82.13	82.76
30	61.78	82.93	83.37	62.22	83.36	83.81
20	64.14	84.66	84.83	64.58	85.08	85.27
10	75.78	87.31	87.91	76.23	87.73	88.36
9	76.68	87.74	88.32	77.13	88.16	88.77
8	77.80	88.25	88.78	78.25	88.67	89.23
7	78.85	88.70	89.27	79.29	89.12	89.71
6	80.12	89.23	89.76	80.57	89.65	90.21
5	81.30	89.88	90.43	81.74	90.29	90.88
4	82.90	90.54	91.16	83.34	90.96	91.60
3	87.15	91.40	92.43	87.59	91.82	92.88
2	90.63	92.36	93.19	91.07	92.78	**93.64**

6 Conclusion

We described our methods for improving the three main components in a spelling error correction application: the dictionary (or word list), the error model and the language model. The contribution of this paper is showing empirically that these three components are highly interconnected and interrelated and they have direct impact on the overall quality and coverage of the spelling correction application. The dictionary needs to be an exhaustive and accurate representation of the language word space. The error model needs to generate a plausible and compact list of candidates. The language model, in its turn, needs to be trained on either clean data or an extremely large amount of data. For spelling error detection, we develop a novel method by training a tri-gram language model on strings of allowable and unallowable sequences of Arabic characters, which can help in the validation of existing word lists and making decisions on new unseen words. Our spelling correction significantly outperforms the three industrial applications of Ayaspell 3.4, MS Word 2013, and Google Docs (tested April 2014) in first-order ranking of candidates.

References

Alfaifi, A., and Atwell, E. 2012. Arabic learner corpora (ALC): a taxonomy of coding errors. In *Proceedings of the 8th International Computing Conference in Arabic (ICCA 2012)*, Cairo, Egypt.

Alkanhal, M. I., Al-Badrashiny, M. A., Alghamdi, M. M., and Al-Qabbany, A. O. 2012. Automatic stochastic arabic spelling correction with emphasis on space insertions and deletions. *IEEE Transactions on Audio, Speech, and Language Processing* **20**(7): 2111–2122.

https://doi.org/10.1017/S1351324915000030 Published online by Cambridge University Press

Attia, M. 2006. An ambiguity-controlled morphological analyzer for modern standard arabic modelling finite state networks. In *The Challenge of Arabic for NLP/MT Conference*, The British Computer Society. London, UK, pp. 48–67.

Attia, M., Pecina, P., Tounsi, L., Toral, A., and van Genabith, J. 2011. An Open-source finite state morphological transducer for modern standard arabic. In *International Workshop on Finite State Methods and Natural Language Processing (FSMNLP)*, Blois, France, pp. 125–133.

Beesley, K. 1998. Arabic morphology using only finite-state operations. In *The Workshop on Computational Approaches to Semitic Languages*, Montreal, Quebec, pp. 50–57.

Beesley, K., and Karttunen, L. 2003. *Finite State Morphology*. CSLI Studies in Computational Linguistics. Stanford, California: CSLI.

Brill, E., and Moore, R. C. 2000. An improved error model for noisy channel spelling correction. In *Proceedings of the 38th Annual Meeting of the Association for Computational Linguistics*, Hong Kong, pp. 286–293.

Brown, P. F., Della Pietra, V. J., de Souza, P. V., Lai, J. C., and Mercer, R. L. 1992. Class-based n-gram models of natural language. *Computational Linguistics* **18**(4): 467–479.

Buckwalter, T. 2004a. Issues in Arabic orthography and morphology analysis. In *Proceedings of the Workshop on Computational Approaches to Arabic Script-based Languages, Association for Computational Linguistics*, Stroudsburg, PA, USA, pp. 31–34.

Buckwalter, T. 2004b. *Buckwalter Arabic Morphological Analyzer (BAMA) Version 2.0*. Linguistic Data Consortium (LDC) catalogue number: LDC2004L02.

Choudhury, M., Saraf, R., Jain, V., Mukherjee, A., Sarkar, S., and Basu, A. 2007. Investigation and modeling of the structure of texting language. *International Journal on Document Analysis and Recognition* **10**(3–4): 157–174.

Church, K. W., and Gale, W. A. 1991. Probability scoring for spelling correction. *Statistics and Computing* **1**: 93–103.

Damerau, F. J. 1964. A technique for computer detection and correction of spelling errors. *Communications of the ACM* **7**(3): 171–176.

El Kholy, A., and Habash, N. 2010. Techniques for Arabic morphological detokenization and orthographic denormalization. In *Proceedings of the Workshop on Semitic Languages in the Seventh International Conference on Language Resources and Evaluation (LREC)*, Valletta, Malta, pp. 45–51.

Gao, J., Li, X., Micol, D., Quirk, C., and Sun, X. 2010. A large scale ranker-based system for search query spelling correction. In *Proceedings of the 23rd International Conference on Computational Linguistics*, Beijing, China, pp. 358–366.

Habash, N., and Rambow, O. 2005. Arabic tokenization, part-of-speech tagging and morphological disambiguation in one fell swoop. In *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics*, Ann Arbor, Michigan, US, pp. 573–580.

Haddad, B., and Yaseen, M. 2007. Detection and correction of non-words in Arabic: a hybrid approach. *International Journal of Computer Processing of Oriental Languages* **20**: 237–257.

Hajič, J., Smrž, O., Buckwalter, T., and Jin, H. 2005. Feature-based tagger of approximations of functional arabic morphology. In *Proceedings of the 4th Workshop on Treebanks and Linguistic Theories (TLT)*, Barcelona, Spain, pp. 53–64.

Han, B., and Baldwin, T. 2011. Lexical normalisation of short text messages: makn sens a #twitter. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics*, Portland, OR, pp. 368–378.

Han, J., and Kamber, M. 2006. *Data Mining, Southeast Asia Edition: Concepts and Techniques*. San Francisco, CA: Morgan Kaufmann Publishers.

Hassan, A., Noeman, S., and Hassan, H. 2008. Language independent text correction using finite state automata. In *IJCNLP*, Hyderabad, India, pp. 913–918.

Heift, T., and Rimrott, A. 2008. Learner responses to corrective feedback for spelling errors in CALL. *System* **36**(2): 196–213.

https://doi.org/10.1017/S1351324915000030 Published online by Cambridge University Press

Hulden, M. 2009a. Fast approximate string matching with finite automata. In *Proceedings of the 25th Conference of the Spanish Society for Natural Language Processing (SEPLN)*, San Sebastian, Spain, pp. 57–64.

Hulden, M. 2009b. Foma: a finite-state compiler and library. In *Proceedings of the 12th Conference of the European Chapter of the Association for Computational Linguistics*, Association for Computational Linguistics. Stroudsburg, PA, USA, pp. 29–32.

Kernigan, M., Church, K., and Gale, W. 1990. A spelling correction program based on a noisy channel model. AT & T Laboratories, 600 Mountain Ave., Murray Hill, NJ, pp. 205–210.

Kiraz, G. A. 2001. *Computational Nonlinear Morphology: With Emphasis on Semitic Languages*, Cambridge University. Cambridge, United Kingdom.

Kukich, K. 1992. Techniques for automatically correcting words in text. *Computing Surveys* **24**(4): 377–439.

Levenshtein, V. I. 1966. Binary codes capable of correcting deletions, insertions, and reversals. *Soviet Physics Doklady* **10**(8): 707–710.

Magdy, W., and Darwish, K. 2006. Arabic OCR error correction using character segment correction, language modeling, and shallow morphology. In *Proceedings of the 2006 Conference on Empirical Methods in Natural Language Processing*, Sydney, Australia, pp. 408–414.

Mitton, R. 1996. *English Spelling and the Computer*. Harlow, Essex: Longman Group.

Mooney, R. J., and Bunescu, R. 2005. ACM SIGKDD explorations newsletter. *Natural Language Processing and Text Mining* **7**(1): 3–10.

Moussa, M., Fakhr, M. W., and Darwish, K. 2012. Statistical denormalization for arabic text. In *Proceedings of KONVENS 2012*, Vienna, pp. 228–232.

Norvig, P. 2009. Natural language corpus data. In T. Segaran and J. Hammerbacher (eds.), *Beautiful Data*, pp. 219–242. Sebastopol, California: O'Reilly.

Och, F. J., and Genzel, D. 2013. Automatic spelling correction for machine translation. Patent US 20130144592 A1. June 6, 2013.

Oflazer, K. 1996. Error-tolerant finite-state recognition with applications to morphological analysis and spelling correction. *Computational Linguistics* **22**(1): 73–90.

Parker, R., Graff, D., Chen, K., Kong, J., and Maeda, K. 2011. Arabic Gigaword Fifth Edition. LDC Catalog No.: LDC2011T11.

Ratcliffe, R. R. 1998. *The Broken Plural Problem in Arabic and Comparative Semitic: Allomorphy and Analogy in Non-concatenative Morphology*, Amsterdam Studies in the Theory and History of Linguistic Science, Series IV, Current issues in linguistic theory, vol. 168. Amsterdam, Philadelphia: J. Benjamins.

Roth, R., Rambow, O., Habash, N., Diab, M., and Rudin, C. 2008. Arabic morphological tagging, diacritization, and lemmatization using lexeme models and feature ranking. In *Proceedings of ACL-08: HLT*, Columbus, Ohio, US, pp. 117–120.

Shaalan K., Allam, A., and Gomah, A. 2003. Towards automatic spell checking for arabic. In *Proceedings of the 4th Conference on Language Engineering, Egyptian Society of Language Engineering (ELSE)*, Cairo, Egypt, pp. 240–247.

Shaalan, K., Magdy, M., and Fahmy, A. 2013. Analysis and feedback of erroneous arabic verbs. *Journal of Natural Language Engineering*, Cambridge University, UK. FirstView: 1–53.

Shaalan, K., Samih, Y., Attia, M., Pecina, P., and van Genabith, J. 2012. Arabic word generation and modelling for spell checking. In *Language Resources and Evaluation (LREC)*, Istanbul, Turkey. pp. 719–725.

Stolcke, A., Zheng, J., Wang, W., and Abrash, V. 2011. SRILM at sixteen: update and outlook. In *Proceedings of the IEEE Automatic Speech Recognition and Understanding Workshop*, Waikoloa, Hawaii.

Tong, X., and Evans, D. A. 1996. A statistical approach to automatic OCR error correction in context. In *Proceedings of the 4th Workshop on Very Large Corpora*, Copenhagen, Denmark, pp. 88–100.

https://doi.org/10.1017/S1351324915000030 Published online by Cambridge University Press

Ukkonen, E. 1983. On approximate string matching. In *Foundations of Computation Theory*, vol. 158, pp. 487–495. Lecture Notes in Computer Science, Berlin: Springer.

van Delden, S., Bracewell, D. B., and Gomez, F. 2004. Supervised and unsupervised automatic spelling correction algorithms. In *Proceedings of the 2004 IEEE International Conference on Web Services*, pp. 530–535.

Watson, J. 2002. *The Phonology and Morphology of Arabic*, New York: Oxford University.

Wintner, S. 2008. Strengths and weaknesses of finite-state technology: a case study in morphological grammar development. *Natural Language Engineering* **14**(4): 457–469.

Wu, J., Chiu, H., and Chang, J. S. 2013. Integrating dictionary and web N-grams for chinese spell checking. *Computational Linguistics and Chinese Language Processing* **18**(4): 17–30.

Zaghouani, W., Mohit, B., Habash, N., Obeid, O., Tomeh, N., Rozovskaya, A., Farra, N., Alkuhlani, S., and Oflazer, K. 2014. Large scale arabic error annotation: guidelines and framework. In *The 9th Edition of the Language Resources and Evaluation Conference (LREC)*, Reykjavik, Iceland, pp. 26–31.

Zribi, C. B. O., and Ben Ahmed, M. 2003. Efficient automatic correction of misspelled arabic words based on contextual information. Lecture Notes in Computer Science, Springer, **2773**: 770–777.

https://doi.org/10.1017/S1351324915000030 Published online by Cambridge University Press