Review of language-independent techniques for error detection and correction of spellcheckers

Frida Mjaria
MJRFRI001
University of Cape Town

ABSTRACT

With constant adaptation of modern words that do not always follow the morphological rules of a language, spellcheckers have to be developed using language-independent techniques in order to correct misspelled words with great accuracy and efficiency. These techniques will be looked at in this review to find the best candidates for developing an error corrector for the isiZulu language. There are 3 components of which a spellchecker comprises, viz. a dictionary, an error model and a language model. Techniques such as n-gram models, the Levenshtein distance, Finite State Automata and noisy channel models can be jointly utilized to construct these components to provide the best candidate corrections for misspelled words. It is found that combining certain techniques together can improve the accuracy of a spellchecker. The language and corpora used, however, plays a significant role on the efficiency of these combinations and therefore, no technique or set of techniques can be used for all languages. It can be induced that utilizing n-grams and the Levenshtein distance model is beneficial in improving the accuracy rate of spellcheckers and in improving the ranking of candidate selections. Language-independent spellcheckers are able to achieve an accuracy rate of 85-89% for error detection and correction of misspelled words.

KEYWORDS

Spellcheckers, language-independent, error correction, n-grams, Levenshtein distance, NLP

1 INTRODUCTION

The Bantu languages spoken in South Africa can be categorized under 2 branches - the Sotho-Tswana branch, comprising of Sesotho, Sesotho sa Leboa (Northern Sotho) and Setswana, and the Nguni branch, which comprises of isiZulu, isiXhosa, isiNdebele and siSwati. From the Nguni languages, isiZulu is the most widely spoken language in South Africa with 22.7% of the population speaking isiZulu as a first/home language, followed by isiXhosa at 16%, whereas only 9.6% of South Africans identify with English as their first/home language[1]. Although isiZulu is more widely spoken than English, there is currently not enough support provided for writing in this language when it comes to spelling error detection and correction. Very few or no spellcheckers exist that can detect and correct typographical errors in Nguni. 2 spellcheckers currently exist for isiZulu that are freely available online for use – one developed by spellchecker.net[2] and the other developed by Ndaba et al.[3] [12]. The latter uses data-driven statistical models and n-grams and can only detect misspelled words. The spellchecker has an accuracy rate of 89%. The former can detect and correct misspelled words. The methodology and accuracy of this spellchecker is, however, not disclosed.

Spelling error correction is very relevant today due to many real-world Natural Language Processing applications, such as emails, blogs and social media messages. These applications require autocorrection in order to perform accurately and efficiently. A large amount of text is also generated online that is informal and unedited by nature, requiring spelling error detection and correction [4, 9, 15]. There are 2 main types of spelling errors that occur, which spellcheckers aim at detecting and correcting - nonword and real-word errors. Nonword errors are words that are not found in the dictionary and are therefore deemed to not exist, while real word errors are words that do exist in the dictionary, but the context of the sentence causes them to be incorrect. Real-word errors are very context sensitive, making nonword errors easier to detect and correct than real world errors, since the context of the language does not need to be taken into account. To detect this error, a dictionary lookup is performed to search for the input word. If the word matches a word in the dictionary, then the word is said to be correctly spelled. If no match is found in the dictionary, then the word is flagged as a misspelled word [12, 17].

Word errors can be categorized either as insertions, deletions, substitutions or transpositions [15]. An insertion error occurs when an extra character is unintentionally inserted in a word. This usually occurs with a user pressing 2 keys on the keyboard simultaneously, one being the intended key, while the other being erroneously pressed [1]. A deletion error occurs when a character is unintentionally omitted from the word. Substitution errors occur when correct characters are substituted by incorrect characters - this can sometimes be attributed to the pronunciation or phonetic sound of a word. Transposition errors occur when the position of 2 characters in a word are exchanged (ab -> ba) [1, 6]. 80% of misspelled words, which are nonword errors, result from a single insertion, deletion, substitution or transposition of characters [18].

[1] Nguni on Britannica Academic: 2017. *http://academic.eb.com/levels/collegiate/article/55655.* Accessed: 2017- 05- 08.

[2] https://www.spellchecker.net/africa_zulu_spell_checker.html

[3] https://keet.wordpress.com/2016/11/11/launch-of-the-isizulu-spellchecker/

There are many different techniques that can be utilized in developing a spellchecker. In this review, I will only focus on techniques that are language-independent. Section 2 highlights the importance of corpus selection for spellcheckers. Statistical models and a Bayesian approach to error detection and correction will be looked at in section 3. In section 4, the various methods for ranking candidate corrections for misspelled words will be discussed. Section 5 compares spellcheckers constructed using the models discussed in section 2 and concludes with a discussion on the efficiency of these models.

2 IMPORTANCE OF CORPUS SELECTION

A corpus is a collection of written texts used for linguistic analysis.[4] The quality of a spellchecker is affected by the type of corpus utilized [12]. The efficiency of the error model of a spellchecker is also affected by the corpus used. A corpus can contain misspelled words, which may cause spellcheckers to identify misspelled words in an input string as correctly spelled and this may affect the accuracy rate of the spellchecker. For instance, Whitelaw et al. [20] used the World Wide Web as a large noisy corpus without any human tweaking of the corpus, which included numerous misspelled words. This caused a decrease in the efficiency of the spellchecker's error detection module. A corpus which mostly contains obsolete words that are no longer used in the language can also reduce the accuracy of a spellchecker. The spellchecker might flag modern words which are correctly spelled as misspelled words [12].

The Language Model (LM) utilized can also be affected by the corpus. With the usage of an n-gram LM to determine the best candidate corrections, a corpus that is too small or contains outdated or misspelled words may affect which candidate corrections are selected as suggestions. The n-gram statistics used in the LM would be computed from the corpus and candidate corrections may receive an inaccurate higher or lower LM score [12, 14, 20].

The efficiency of error detection and correction in a spellchecker which uses a corpus can be increased by using multiple corpora. The combination of these corpora in the error and language model may, however, affect the accuracy and should be taken into consideration [12].

3 TECHNIQUES FOR LANGUAGE-INDEPENDENT ERROR DETECTION AND CORRECTION

3.1 Components of a spellchecker

A spellchecker has the following 3 components - a dictionary, an error model and an LM. A dictionary for a spellchecker is a list of words that are mostly correctly spelled that make up the dictionary of the spellchecker [14]. An LM in a spellchecker is used to determine how frequent a word occurs in a corpus. A dictionary and text data can be used to build an LM [6]. An error model is an algorithm for modelling spelling errors [14]. This section takes a look at the different techniques that could be used in constructing the components of a spellchecker.

3.2 N-gram analysis

An N-gram is an n-letter subsequence of words or a string, where n is usually one, two or three and can sometimes equal four [9]. N-grams can be represented as character n-grams or word n-grams and form the dictionary of a spellchecker. Traditional dictionaries are represented by full lexicon words or word n-grams. Instead of storing word n-grams in a dictionary, a corpus can be split into character n-grams and these can be utilized1 as the dictionary of the spellchecker [8]. The use of character n-grams instead of word n-grams might improve the number of input words that a spellchecker identifies as correctly spelled words. This can, however cause misspelled words to be identified as correctly spelled [3, 8].

Error models use n-grams to predict whether a word is misspelled by comparing an input word against n-grams in the dictionary. With character n-grams, each n-gram in the input word is compared with n-grams stored in the dictionary. If any n-gram in an input word is not found in the dictionary, the word is flagged as misspelled [12].

Higher-order n-grams are more context-sensitive, but have sparse counts, while lower-order n-grams have higher counts, but are less context sensitive. [12, 15]. N-grams can also be used in an n-gram LM to determine the best candidate corrections by computing the n-gram statistics of each candidate correction. An N-gram statistic is the probability of an N-gram occurring in a text and is computed from how frequent an n-gram occurs in words from a corpus. Ndaba et al. state that the efficiency of an n-gram model is dependent on the language used, based on their findings that trigrams have a higher accuracy in detecting and correcting errors in their isiZulu spellchecker compared to quadrigrams [12].

N-grams are therefore very useful in the construction of a dictionary for a spellchecker. They are also very useful in finding candidate corrections through the computation of the Levenshtein distance.

3.3 Levenshtein distance

The Levenshtein distance (LD), also known as the minimum edit distance or Levenshtein-Damerau distance, is an algorithm used to calculate the minimum edit distance required to transform one word into another [1]. Edit distance is the number of insertion, deletion, substitution and/or transposition operations that will have to be performed on the misspelled word to acquire the correctly spelled word [12, 15]. LD can be used together with n-grams in an error model to identify misspelled words. LD can also be used to find candidate corrections for the misspelled word(s) [12].

LD can be very costly to compute, especially when using a large corpus to formulate a dictionary using character n-grams. Using LD in combination with another algorithm might improve

[4] Robin. What Is Corpus? on Natural Language Processing: 2009. *http://language.worldofcomputing.net/linguistics/introduction/what-is-corpus.html.* Accessed: 2017- 05- 11.

the computation time and might also improve on the accuracy of the spellchecker [6].

3.4 Finite State Automata

This method represents the dictionary used with a spellchecker as a deterministic finite state automation. Given an input string, containing 1 or more words, a finite state machine (FSM) is constructed for the input string, with each path representing a word in the input string The FSM is then combined with the dictionary FSM, resulting in an FSM that contains the intersecting words from the input string FSM and dictionary FSM (FSM with words that are correctly spelled from input string). Calculating the difference between the dictionary FSM and the FSM containing the intersecting words yields an FSM containing all the words that are flagged as misspelled [6, 13, 16].

When it comes to traversing the dictionary automation and finding candidate corrections for these misspelled words, there are a few different algorithms that can be utilized. Oflazer uses the Levenshtein edit distance together with the Wagner-Fisher algorithm to traverse through the dictionary when finding candidate corrections [13, 19]. By using an FSM together with the Levenshtein distance and the Wagner-Fisher algorithm, Oflazer was able to control the traversal of the dictionary automation and avoid traversal of most of the dictionary states [6]. Shulz and Mihov and Hassan et al. developed their finite state automata techniques from Oflazer's approach, but with a few distinctions. Shulz and Mihov do not compute the Levenshtein edit distance when traversing the dictionary. They compute a deterministic Levenshtein automation of degree 1 [16]. The Levenshtein automation and the dictionary automation are then traversed in parallel to extract candidate corrections. A finite state acceptor is also constructed for each word in the input string. The acceptor accepts all words which are an edit distance k from the misspelled word to find candidate corrections [6, 16].

Hassan et al. [6] also use a deterministic dictionary automation. They do not compute the Levenshtein distance either. To generate candidate corrections, Hassan et al. compute a Levenshtein-transducer instead of computing a Levenshtein automation like Shulz and Mihov [16]. They do, however, construct a finite state acceptor and compose it with the Levenshtein-transducer.

Using FSM can improve the performance of a spellchecker, especially when combined with a Levenshtein automation or Levenshtein-transducer to find candidate corrections.

3.5 Noisy Channel Model

The noisy channel model can also be used to find candidate corrections for misspelled words. Each misspelled word in a text is treated as if it was correctly spelled, but became "distorted" or "noisy" from passing through a noisy channel, making the correct word difficult to recognize. The "noise" represents substitution, insertion, deletion and transposition changes. The noisy channel model is a type of Bayesian inference, where candidate corrections for a misspelled word are found by passing every word in the dictionary through the noisy channel model to find the word that comes closest to the misspelled word [7].

Candidate corrections can be found by calculating the probability scores of all the words in the dictionary using the noisy channel model. Whitelaw et al. [20] and Church and Gale [2] state that candidate correction s for an observed word w can be found by finding the word in the dictionary that maximizes P(s) P(w | s), where P(s) is the prior model of word probabilities and P(w | s) is the noisy channel model.

4 RANKING CANDIDATE CORRECTIONS

After locating spelling errors in an input string, a spellchecker will offer suggestions for correcting the misspelled words [11]. Using the Levenshtein distance can help with the selection of the best candidate corrections by calculating the minimum edit distance for each candidate correction with the misspelled word and scoring the candidate corrections. Candidate corrections with higher edit operations are less favoured than candidate corrections with lower edit operations [6, 11].

An LM score can be used to select the best candidate correction(s). This is done by substituting the misspelled word with the candidate correction in the input string. n N-grams are then extracted, which have the candidate correction in all possible positions in the n-gram [6]. A score is assigned to each n-gram according to the frequency or the likelihood that an n-gram occurs in the corpus used. A score is then assigned to the candidate correction, which is equal to the average score of all n-grams. By using the LM score assigned to the candidate corrections, words that are more frequently used in the language will be favoured as the best candidate corrections as opposed to words that are less commonly used. This algorithm also helps with ranking candidate words that may have the same edit distance away from the misspelled word [6, 11, 14].

After the best candidate corrections are chosen, they are displayed as suggestions. In the case where no candidate corrections can be found, a "no suggestion" message can be displayed and an option to add the word to a list of exception words can also be provided by the spellchecker [12, 1].

5 COMPARISON OF TECHNIQUES UTILIZED IN SPELLCHECKERS

The following table compares spellcheckers' performances in terms of accuracy achieved in detecting and correcting misspelled words. The spellcheckers were constructed using the techniques discussed in section 3. The languages and corpora used are also indicated.

Table 1: Comparison of spellcheckers' performance

	Langu age	**Corpora**	**Techniq ue(s)**	**Performance**	
Ndaba et al.	isiZulu	Ukwabel ana	LD, trigrams	INC	67% accurac

[12]		Corpus (UC); selection of isiZulu National Corpus (INC); small corpus of news items (NIC)	and quadrigrams and n-gram LM		y rate
				NIC	76% accuracy rate
				UC	Above 50% accuracy rate
Hassan et al. [6]	Arabic and English	-	FSM, Levenshtein transducer, n-gram LM	89% accuracy	
Whitelaw et al. [20]	English and German	Large corpus of crawled public web pages	Substring error model, n-gram LM, Confidence classifiers (constructed using noisy channel model),	English	Total error rate for best system = 2.62% Suggestion rate = 10%
				German	Correction error rate = 7.89% Total error rate = 9.8%
Samanta and Chaudhuri [15]	English	BYU corpus, bigram and trigram, text from Project Gutenberg	Confusion set constructed from Levenshtein distance, bigram and trigram model, stemming	85% accuracy rate for top-ranked suggestions, 93% for top 2 ranked suggestions	
Schryver and Prinsloo [3]	isiZulu	isiZulu Bona magazine, April 2003	N-grams and Levenshtein distance	68% accuracy	
Gupta and Sharma [5]	English	Brown corpus and set of commonly confused words	Trigrams and Bayesian approach	89.83% accuracy	

The above table shows different techniques used by different spellcheckers and the accuracy they achieved in providing candidate corrections to misspelled words. The most efficient spellcheckers have an accuracy rate of 85% and above. All of these spellcheckers utilize n-grams in their models and most of them use Levenshtein distance (or its variation). Ndaba et al. [12] were only able to achieve error detection with their spellchecker and could not perform any error corrections on misspelled words.

It should also be noted that the size of a corpus affects the accuracy of a spellchecker, where corpora which are too big or small can cause the accuracy to be limited or reduce [3, 12, 20]. The content of a corpus also affects the accuracy of a spellchecker, which can be noted from Whitelaw et al.'s [20] Gupta and Sharma's [5] spellchecker. Whitelaw et al. uses the web as a corpus, which is filled with a large amount of correctly spelled as well as misspelled words. Because of this, their spellchecker only achieved an accuracy rate of 68%. Whitelaw et al. [20] used n-grams, Levenshtein distance and 7 confidence classifiers, constructed using (the noisy channel model) to construct their spellchecker. Gupta and Sharma [5] also used n-grams and a Bayesian approach to construct their spellchecker. They achieved an accuracy rate of 89.83%. The content of the corpora used contributed to achieving this high accuracy.

It is difficult and would be inaccurate to compare all of the techniques together and decide which technique listed above is the best to utilize for the construction of all spellcheckers. This is due to the effects that the selected language(s) and corpora have on spellcheckers. However, it can be induced that using the Levenshtein distance (or its variation) and/or n-grams can help improve the accuracy level achieved by a spellchecker.

6 CONCLUSIONS

There are many language independent models that can be used for error detection and correction. These models are typically used in conjunction with each other to achieve a higher level of accuracy of spelling suggestions for misspelled words. It is, however, not possible to infer a technique or a set of techniques for error detection and correction that would be suited for all languages due to variations in language rules, the corpora used as well as the conjugative or agglutinative nature of the selected language(s) [13]. From these techniques, it can be deduced that a spellchecker with error detection and correction for Nguni languages can be constructed using statistical and/or Bayesian approaches with great accuracy. The best spellcheckers achieve an accuracy rate of 85% and above for error detection and correction and use n-grams in their models. Most of the spellcheckers also use Levenshtein distance (or its variation) in their models.

REFERENCES

[1] Chaudhuri, B. B. Reversed word dictionary and phonetically similar word grouping based spell-checker to Bangla text. In Proc. LESAL Workshop, (Mumbai, 2001).

[2] Church, K. W. and Gale, W. A. Probability scoring for spelling correction. Statistics and Computing. 1, 2 (1991), 93-103. DOI=10.1007/BF01889984.

[3] de Schryver, G. and Prinsloo, D. J. Spellcheckers for the South African languages, Part 1: The status quo and options for improvement. South African Journal of African Languages. 24, 1 (2004), 57-82.

[4] Farra, N., Tomeh, N., Rozovskaya, A. and Habash, N. Generalized Character-Level Spelling Error Correction. ACL (2). (2014), 161-167.

[5] Gupta, S. A., Sharma S. Correction Model for Real-word Errors. Procedia Computer Science, 70 (2015), 99-106.

[6] Hassan, A., Noeman, S. and Hassan, H. Language independent text correction using finite state automata. IJCNLP.Hyderabad. (2008).

[7] Jurafsky, D. and Martin, J. H. (2016). Spelling Correction and the Noisy Channel (draft, 3ed). Speech and Language Processing.

[8] Kanaris, I., Kanaris, K., Houvardas, I. and Stamatatos, E. Words versus character n-grams for anti-spam filtering. Int. J. Artif. Intell. Tools. 16, 06 (2007), 1047-1067.

[9] Kukich, K. Techniques for automatically correcting words in text. ACM Computing Surveys (CSUR). 24, 4 (1992), 377-439. DOI=10.1145/146370.146380.

[10] Mays, E., Damerau, F. J. and Mercer, R. L. Context based spelling correction. Information Processing & Management. 27, 5 (1991), 517-522.

[11] Mitton, R. Ordering the suggestions of a spellchecker without using context. Natural Language Engineering. 15, 02 (2009), 173-192. DOI=10.1017/S1351324908004804.

[12] Ndaba, B., Suleman, H., Keet, C. M. and Khumalo, L. The effects of a corpus on isiZulu spellcheckers based on N-grams. IIMC. (2016), 1-10.

[13] Oflazer, K. Error-tolerant finite-state recognition with applications to morphological analysis and spelling correction. Computational Linguistics. 22, 1 (1996), 73-89.

[14] Pirinen, T. A. and Hardwick, S. Effect of Language and Error Models on Efficiency of Finite-State Spell-Checking and Correction. In FSMNLP (2012), 1-9.

[15] Samanta, P. and Chaudhuri, B. B. A simple real-word error detection and correction using local word bigram and trigram. In ROCLING (2013).

[16] Schulz, K. U. and Mihov, S. Fast string correction with Levenshtein automata. International Journal on Document Analysis and Recognition. 5, 1 (2002), 67-85.

[17] Tong, X. and Evans, D. A. A statistical approach to automatic OCR error correction in context. In Proceedings of the fourth workshop on very large corpora, (1996), 88-100.

[18] Toutanova, K. and Moore, R. C. Pronunciation modeling for improved spelling correction. In Proceedings of the 40th Annual Meeting on Association for Computational Linguistics. (2002), Association for Computational Linguistics, 144-151.

[19] Wagner, R. A. and Fischer, M. J. The string-to-string correction problem. Journal of the ACM (JACM). 21, 1 (1974), 168-173.

[20] Whitelaw, C., Hutchinson, B., Chung, G. Y. and Ellis, G. Using the web for language independent spellchecking and autocorrection. In Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing: Volume 2-Volume 2. (2009), Association for Computational Linguistics, 890-899.