UNIVERSITY OF CAPE TOWN
DEPARTMENT OF COMPUTER SCIENCE

Computer Science Honours Final Paper 2017

Title: A Statistical Approach to Error Correction for an isiZulu Spellchecker

Author: Frida Mjaria

Project Abbreviation: ALSPEL

Supervisor(s): Maria Keet

Category	Min	Max	Chosen
Requirement Analysis and Design	0	20	0
Theoretical Analysis	0	25	0
Experiment Design and Execution	0	20	20
System Development and Implementation	0	15	10
Results, Findings and Conclusion	10	20	20
Aim Formulation and Background Work	10	15	10
Quality of Paper Writing and Presentation	10		10
Quality of Deliverables	10		10
Overall General Project Evaluation (*this section allowed only with motivation letter from supervisor*)	0	10	
Total marks		**80**	**80**

A Statistical Approach to Error Correction for an isiZulu Spellchecker

Frida Mjaria
Department of Computer Science
University of Cape Town
fmjaria@gmail.com

ABSTRACT

Spellcheckers have become significantly important with the increase of text-based communication. There is however, very little support provided for spell checking in Nguni languages, of which two (isiZulu and isiXhosa) are the most spoken languages in South Africa. The aim of this research paper is to improve on spell checking for the Nguni language, isiZulu, by providing an error corrector that can accurately provide candidate corrections to misspelled isiZulu words using a statistical approach. The error corrector is then integrated with the isiZulu error detector developed by Ndaba et al. [12] and implemented by Norman Pilusa.

This paper focusses on non-word error correction. Trigrams, minimum edit distance and probabilities of trigrams based on occurrence in corpus are used to construct the error corrector. The error corrector is tested on 6 000 auto-generated isiZulu spelling errors. These errors represent the 4 types of non-word errors that occur (substitution, insertions, deletions and transpositions). The performance of the error corrector is tested using confusion matrices.

The experiment in section 4.3 was then performed on the whole system utilizing the auto-generated spelling errors. The system achieved a language recall rate of 89%, an error recall of 84%, a language precision of 85% and an error precision of 88%. The error corrector was found to have an accuracy rate of 94%, which is well above the expected accuracy rate of 85%. The suggestion adequacy of the system was found to be 62%, which is also above the expected value of 60%. From the following results, the spell checker is found to be successful at detecting and correcting spelling errors using a statistical error model.

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 spelling error detection and correction. Two spellcheckers currently exist for isiZulu. One was developed by Ndaba et al. [12] and implemented by Norman Pilusa [2] and the other by spellchecker.net[3]. The former uses data-driven statistical models and n-grams and can only detect misspelled words. According to Ndaba et al. [12], the spellchecker has an accuracy rate of 89%. The latter can detect and correct misspelled words. The methodology and accuracy of this spellchecker is, however, not disclosed.

The aim of this paper is to improve on spellchecking for the isiZulu language by developing an error corrector that can accurately provide candidate corrections to incorrectly spelled isiZulu words in a given input text. This error corrector is integrated with Ndaba et al.'s error detector to form a fully functional isiZulu spellchecker. Accuracy of the error corrector is measured by the number of words detected to be incorrect by the error detector, that the error corrector can provide candidate corrections to.

A large amount of text is generated online on a daily basis that is informal and unedited by nature, requiring spelling error detection and correction [4, 9, 15]. Spellcheckers aim to detect two types of spelling errors that occur – non-word and real world (or context-based) spelling errors. Non-words errors are words that do not occur in a given language. These errors are usually caused by typographical errors made by the user when typing or by spelling a given word according to its pronunciation (phonetical errors). There are 4 types of non-word errors, viz. substitutions, insertions, deletions and transpositions [15]. Substitution is when a letter in a word is replaced with another, insertion is when a letter is added to a word, deletion is when a letter is omitted from a word and transposition is when a swap occurs between 2 adjacent letters in a given word [1, 6].

While non-word errors are categorized as non-existent words in a language, real-word errors denote words that are correctly spelled, but are used in the wrong context within a sentence [12, 17].

[1] Nguni on Britannica Academic: 2017. http://academic.eb.com/levels/collegiate/article/55655. Accessed: 2017- 05- 08
[2] https://github.com/normanpilusa/Isizulu_Spellchecker
[3] https://www.spellchecker.net/africa_zulu_spell_checker.html

This paper presents a method for creating an isiZulu spelling error corrector for non-word errors using a statistical approach. The error corrector utilizes probabilities and trigrams to produce corrections for input words that are flagged as incorrect spelled.

The paper consists of the following sections: related work, system design and implementation, experiment design and execution, results and discussion, conclusions and future works. Related work highlights the different statistical methods and components that have been used to construct existing spell checkers; system design and implementation details the methods and components used to develop the error corrector; experiment design and execution details the design of the experiment and how it was executed. This section also lists the evaluation metrics and hypotheses used to determine the success of the experiment. The results and discussion section highlights the results obtained from the experiment and what they indicate. The paper concludes with the conclusions and future works sections, which highlight the accuracy of the error corrector and the system as a whole and what can be done in future for spellchecking in isiZulu,

2 RELATED WORK

This section looks at the components of a spellchecker as well as n-grams and the minimum edit distance and their uses in developing spell checkers. This section also looks at various spell checkers that have been developed using a statistical approach.

2.1 Components of a spell checker

A spellchecker has the following 3 components - a body of text representing the language (corpus), an Error model (EM) and a Language model (LM) [14]. An LM in a spellchecker is used to determine how frequent a word occurs in a dictionary. A corpus can be used to build an LM [6]. An EM is an algorithm used for modelling spelling errors [14].

2.2 Corpus

A corpus is a collection of written texts used for linguistic analysis[4]. The efficiency of the error model of a spellchecker is 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, [20] used the World Wide Web as a large noisy corpus without any human tweaking of the corpus, which included many 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 model 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 of the spellchecker [12].

2.3 N-grams

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 words grams. Instead of storing word grams in a dictionary, a corpus can be split into character n-grams and these used 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. [12] state that the efficiency of an n-gram model is dependent on the language used, finding that trigrams have a higher accuracy in detecting and correcting errors in their isiZulu spellchecker compared to quadrigrams.

2.4 Damerau-Levenshtein distance

The Damerau-Levenshtein distance (DL), also known as the minimum edit 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]. DL can be used together with n-grams in an error model to identify a misspelled word. DL can also be used to find candidate corrections for the misspelled word [12].

[4] http://language.worldofcomputing.net/linguistics/introduction/what-is-corpus.html

2.5 Statistical Approaches for Error Correction

Appendix A1 details the various statistical methods used in developing existing spellcheckers. The 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). 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 reduced [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. 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.

2.6 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].

A language model 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 language model 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].

3 SYSTEM DESIGN AND IMPLEMENTATION

This section details the design and implementation of the techniques used to construct the error corrector. The system provides candidate corrections for words flagged as incorrect by the error detector developed by Ndaba et al. [12]. According to Damerau, 80% of spelling errors occur from a single substitution, insertion, deletion or transposition error in a word [18]. The error corrector will focus on non-word errors originating from a one-character change from the intended word. The design of the error corrector was implemented using the Java programming language. Java is a platform independent, portable, object oriented programming language that offers vast libraries to ease development. Java was used to develop the error detector and for integration purposes, the error corrector is also developed using this language.

3.1 Corpus

An isiZulu corpus was obtained from Dr. Langa Khumalo from the Language Department of the University of KwaZulu-Natal. The corpus comprises of isiZulu articles and novels stored in text files. The corpus was cleaned by removing punctuation (excluding apostrophes and hyphens occurring in isiZulu) and non-isiZulu words occurring in the text files. The text files were then used to construct trigrams.

3.2 Trigram Construction

A text file to store trigrams is created. A trigram is a group of three consecutive characters. Trigrams are constructed by taking each word in the corpus and using the following method to obtain the trigram constituents of a word:

Starting from index position 0 of the given word:

- o Find the next three consecutive characters of the word from the index position to index + 2 and store them in the trigrams text file.
- o Increment the index by one.
- o Repeat the above steps until index = length of word - 3.

Unique trigrams from the trigram text files are stored in a trigram list, which is used to calculate the various probabilities mentioned in section 3.3.

3.3 Probabilities

Once the trigrams are constructed, a list of unique trigrams occurring in the corpus is constructed. The frequency at which each trigram, occurs in the corpus is counted and stored in a text file together with the trigram from the trigram list. The frequencies are used in the search algorithm to determine the probability of a trigram being correct if it occurs in the trigram list. Another set of probabilities calculated is the probability of a specific trigram occurring after its precedent trigram. These probabilities serve 2 purposes in the error corrector: to identify a correct trigram that does not belong in the given word and to determine the order of candidate corrections.

3.4 Search Algorithm

When a word from the input text is flagged as incorrect by the error detector, the word is broken up into its trigram constituents. The list of unique trigrams is then traversed to identify every incorrect trigram in the word. A HashMap is used to store the trigrams from the trigram list and their respective frequencies. Only trigrams with frequencies greater than the frequency threshold are stored in the HashMap. This improves the performance of the system, since file accesses only occur once when the program is initialized. An ArrayList is also created to store trigrams stored in the HashMap. This is done for easy traversal of all trigrams instead of using an Iterator on the HashMap.

A trigram is viewed as incorrect if the trigram does not appear in the trigram list or if the trigram occurs in the list, but its frequency is below the assigned frequency threshold. Once a trigram is identified as incorrect, the Damerau-Levenshtein (DL) or minimum edit distance of the trigram and each unique trigram is calculated. The DL distance is the minimum number of operations that need to be performed in order to transform one string into another [12]. All the trigrams that yield an edit distance of 2 or lower are stored in an array. This method is repeated for each trigram that is identified as incorrect.

DL is used to obtain candidate trigrams. The ArrayList is traversed and the DL distance of each incorrect trigram and each unique trigram is calculated. All trigrams from the ArrayList that yield an edit distance of two or lower are stored in an ArrayList. This method is repeated for each trigram that is identified as incorrect.

3.5 Storing Candidate Trigrams

The ArrayList containing candidate trigrams needs to be stored together with the incorrect trigram. This is achieved by creating a Trigram object. The Trigram object consists of an ArrayList to store candidate corrections and a String variable to store the incorrect trigram. A Trigram object is created for each trigram of the incorrect word and stored in a Trigram ArrayList. Trigrams that are viewed as correct contain an empty suggestions ArrayList.

3.6 Candidate Corrections Algorithm

Once all the incorrect trigrams are identified and candidate trigrams are obtained by the search algorithm, the trigrams are joined to form possible isiZulu words to display as candidate corrections to the user. The candidate corrections are formed using a brute force algorithm, where possible combinations of candidate trigrams stored in ArrayLists are computed using string manipulation. These combinations are also combined with the correctly spelt trigrams of the flagged word to form candidate corrections.

Due to the computational intensity of the algorithm, the ArrayLists containing candidate trigrams are ordered and the BinarySearch Algorithm is used to determine the start and end indices to perform combinations on. The BinarySearch algorithm determines the first and last strings in the ArrayList where a combination between two strings is possible. This is done to improve on the running time of the algorithm. By using BinarySearch, the complexity is improved from O(n^2) to O(nlogn).

Due to the agglutinative nature of isiZulu, some of the candidate corrections obtained from this algorithm might be non-existent in the isiZulu language. To mitigate this, the list of unique words from the corpus is traversed. All candidate corrections that do not occur in this list are discarded. To further improve on performance, all unique words occurring in the word list are stored in a HashSet for quick access.

3.7 Optimization of Error Corrector

Two techniques are utilized to further improve on the performance of the error corrector in finding candidate corrections. The first technique, which is a novel technique, is to check if the error that occurred is a transposition error. This is done by checking if the trigram contains 2 adjacent consonants. If there are 2 adjacent consonants, the consonants are swapped and the HashMap containing the unique trigram is checked to see if the trigram exists. If it does exist, string manipulation is used to formulate a new word containing the trigram with the swapped adjacent consonants. The HashSet containing unique words is checked to see if the new word is a correct isiZulu word. If it is correct, the word is displayed as a candidate correction and the suggestions algorithm is terminated for this word. This drastically optimizes the performance of the error corrector, since candidate corrections do not need to be searched for if the new word is found to be correct.

Probabilities are used in the second method to improve on the accuracy of the error corrector in finding candidate corrections. Due to the agglutinative nature of isiZulu, a trigram can be viewed as correct by the error detector, but it is incorrect in the given word. When obtaining candidate corrections through string manipulation, the correct trigram is joined to the candidate trigram to form a new string. If the trigram occurring before the incorrect trigram does not belong in the word and the previous trigram is recognized as correct, the intended word will not appear as a candidate correction. To mitigate this, the probabilities of trigram occurring after a specific trigram is calculated. This is done by traversing through the corpus, where every word is in its trigram form, and calculating the frequency of the trigrams occurring next to each other in a given word. This is done for all combinations of unique trigrams that have a frequency >= 35 that can be combined to form a new string. These probabilities are stored in a text file, where they are accessed and stored in a HashMap when the system is initialised. The trigram is used as a key and a TriNext object is stored as the value. The TriNext object contains an ArrayList with all possible trigrams that can occur next as well as a HashMap containing the probabilities of these trigrams.

When a trigram is viewed as correct, if the trigram is not the first trigram of the word, the HashMap containing the probabilitiees of the trigram occurring next after the previous trigram is accessed to see if the previous trigram is stored as a key. If the previous trigram is stored, the ArrayList containing the possible trigrams that can occur next is set as the suggestions ArrayList for this trigram.

3.8 Ordering of candidate corrections

In order to show the most relevant suggestions to users, candidate corrections are ordered according to their probability score in the following manner:

- Each candidate correction is broken up into its trigram constituents and stored in an ArrayList.
- For each trigram (excluding the trigram occurring at index 0), the probability of the trigram occurring after the previous trigram is obtained.
- The probabilities are summed up and divided by the total number of trigrams. The sum is stored in a HashMap with the candidate correction as the key.
- The candidate corrections are then sorted according to their probability score and at most 10 suggestions are displayed to the user.
- Figure 1 depicts candidate corrections provided by the system.

Incorrect word	Candidate Corrections
Substitution	
wasemausa esedazuljka	*wase, waso* *esedazuluka*
Transposition	
bezibguza ngewzindlela	*bezibuza* *ngesizini, ngethi, ngenze, ngenza*
Insertion	
zigqamia ungasagcwli	*zigqamisa* *ungasho, ungase, unga*
Deletion	
ephephandabeni uhpakanyiswe	*ephephandabeni, ephephandaba, ephe* *uphakanyiswe*

Figure 1: Candidate corrections for incorrectly spelled

4 EXPERIMENT DESIGN AND EXECUTION

This section details the experiment design and execution followed to determine the performance of the spell checker based on different evaluation metrics.

4.1 Dataset

The corpora provided by Dr. Khumalo was used for training the system and an isiZulu wordlist ("correct.txt") obtained from Norman Pilusa's GitHub account[5] was used as the testing dataset to test the spell checker. We assume that each word in the wordlist is correctly spelled. Four text files ("substitutions.txt", "transpositions.txt", "insertions.txt" and "deletions.txt") were created to store spelling errors according to the type of nonword error occurring. 6000 words from the wordlist were fed to an error generating module[6] to generate the various nonword spelling errors. The module takes each word and randomly injects a spelling error into the word and then stores the word in the appropriate text file according to the spelling error that was injected. Each text file contains 1500 incorrectly spelled words.

This method is used to create spelling errors as no resource containing common misspellings occurring in isiZulu could be found.

4.2 Evaluation Metrics

According to van Huyssteen et al [a], spellcheckers should have a high proficiency of the language, in terms of flagging very few correctly spelled words as incorrectly spelled. Ideally the spell checker should flag all incorrectly spelled words and provide candidate corrections to these words as well as recognize all correctly spelled words in a given text. In this section, we will look at the evaluation metrics that will be used to evaluate the performance of the system.

4.2.1 Confusion Matrix

A confusion matrix (figure 2) is used to classify the results into 4 categories: True Positives (TP), True Negatives (TN), False Negatives (FN) and False Positives (FP). TP denotes the number of words that are known to be correctly spelled that the system recognizes as correctly spelled. TN are the number of correctly spelled words that the system flags as incorrect. FP denotes the number of incorrectly spelled words that the system flags as incorrectly spelled. FN are the number of incorrectly spelled words that the system recognizes as correctly spelled [12].

	Recognized as Correctly Spelled	*Flagged as incorrectly spelled*
Correctly spelled word	TP	FN
Incorrectly spelled word	FP	TN

Figure 2: Confusion Matrix

4.2.2 Recall Measures

[5] https://github.com/normanpilusa/Isizulu_Spellchecker/blob/master/correct.txt
[6]https://github.com/benjholla/spellwrecker/blob/master/SpellWrecker/src/spellwrecker/components/spellwreckers/QwertySpellWrecker.java

Recall measurements are used to denote how representative the lexicon of the spellchecker is of the language as well as how free is the lexicon used of incorrectly spelled words [a]. In order to determine how representative the corpus is of the isiZulu language, the lexical recall (LR) of the spellchecker is calculated. LR is the number of correctly spelled words that are recognized as correct by the spell checker (TP) in relation to the total number of correctly spelled words in the text (TP + FN) [a, b]. LR is calculated as follows:

$$LR = \frac{TP}{TP + FN}$$

To determine how error-free the corpus is, we calculate the Error Recall (ER) of the spell checker. ER is the number of incorrectly spelled words flagged by the spell checker (TN) in relation to the total number of incorrectly spelled words (TN + FP) [a, b]. This is calculated using the following equation:

$$ER = \frac{TN}{TN + FP}$$

4.2.3 Precision Measures

Precision measures are used to denote how accurate the spell checker is at recognizing correctly spelled words (lexical precision (LP)) and flagging incorrectly spelled words (error precision (EP)) [a]. LP and EP calculated using the following equations:

$$LP = \frac{TP}{TP+FP} \text{ and } EP = \frac{TN}{FN+TN}$$

4.2.4 Suggestion Adequacy and Accuracy

Suggestion Adequacy (SA) denotes the ability of the spell checker to provide accurate suggestions that are relevant to the user [a]. Accuracy (Cs) is represented by the number of TN that the error model was able to provide candidate corrections to. This however, does not determine the relevance (Cv) of the suggestions. In order to determine the SA of the error model, the Cv of the suggestions is calculated. Cv is represented by the number of corrections provided for TN that contain the intended word that was incorrectly spelled.

Cs is determined by the number of words that obtained suggestions from the error model. In order to determine Cv, a scoring system is used based on the combination of scoring systems used by Paggio and Underwood [c] and Van Zaanen and Van Huyssteen [d]. The scoring system is as follows:

- If the suggestions contain the intended word: Cv = 1
- If the suggestions do not contain the word: Cv = 0.5
- If no suggestions provided: correct suggestion = 0

This scoring system is performed on each word and the total sum for all TN determines Cv. To get the percentage for accuracy and relevance, Cv and Cs are divided by TN.

4.3 Design

The error detector uses a separate set of unique trigrams and wordlist from the error corrector to perform error detection on input text. Since the corpus utilized affects the performance of the spell checker, the experiment began by testing which set of unique trigrams and wordlist will provide better performance. The system was tested using correctly spelled words stored in a text file ("Accuracy.txt"). The set that yielded the highest TP values was utilized for both error correction and error detection. Once the trigramlist and wordlist was determined, the frequency threshold (T_F) for the system was determined.

T_F is defined as the minimum frequency that a trigram can occur in a corpus to be used in the system. Frequency denotes the number of times a trigram occurs in a corpus. The T_F of the system was determined first by feeding the system with 1895 incorrectly spelled words stored in a text file ("threshold.txt"). The file was iteratively tested using different T_F values in each iteration. The TN, FP, Cs and Cv values were recorded for each iteration. The T_F value that yielded the best results was utilized for the remainder of the experiment.

After testing for the threshold frequency, the performance of the system was tested with and without the use of Cv optimization probabilities discussed in section 3.4.

In order to determine the performance of the system for each type of spelling error, each text file containing the various spelling errors was divided into three sections, with each section containing 500 words. This is done to ascertain that the system provides consistent results for each type of spelling error across all phases. Each section was fed to the system and the resulting TN, FP, Cs and Cv values were stored. This method was repeated for each type of spelling error. Finally, the system was tested using the testing dataset containing correctly spelled words. The resulting TP and FP values were used together with the cumulative TN and FN values to generate a confusion matrix. The confusion matrix together with the cumulative Cv and Cs values were used to measure the recall, precision, accuracy and SA of the system to determine the accuracy of the spellchecker. The accuracy of the spellchecker is determined using the following hypothesis in section 4.4.

4.4 Hypotheses

The following hypotheses are used to determine if the spellchecker is successful at detecting and correcting nonword spelling errors in isiZulu:

- *Hypothesis 1:* The system achieves an LR rate >= 85%
- *Hypothesis 2:* The system achieves an ER rate >= 80%
- *Hypothesis 3:* The system achieves an LP and EP rate >=85%
- *Hypothesis 4:* The system achieves an accuracy rate >= 85%
- *Hypothesis 5:* The system achieves an overall suggestion rate >= 60%

5. RESULTS AND DISCUSSION

This section describes the results obtained from the experiment. Bar graphs and a confusion matrix are used to analyze the results.

5.1 Corpus and Threshold

Table 1(a) contains the results obtained from testing which trigram and word list yielded the highest TP value.

Table 1(a): TP and FP results for trigram and word lists

	TP	FN	Accuracy
Trigram and word list from error detector module	10328	2125	83%
Trigram and wordlist from error corrector module	10566	1887	85%

From the results obtained in table 1, the error detector has a higher accuracy when the trigram and word list used to develop the error corrector is used. Therefore, the error corrector and error detector will utilize the same trigram and word list.

Table 1(b) shows the results after testing the system with different T_F values. The results show that a higher T_F values yield higher TP values and lower F_P values. Although though TP values for $T_F > 45$ increase and FP values decrease, the Cv values for $T_F > 45$ decrease significantly as T_F increases, while $T_F <= 45$ does not alter the Cv value significantly. Therefore, the chosen T_F that will be used to determine if a trigram is correct is 45. This will be the T_F used to evaluate the system.

Table 1(b): threshold results using "threshold.txt"

T_F	TP	FN	Cs	Cv
35	1584	311	1392	957
40	1599	296	1412	956
45	1608	287	1420	957
50	1615	280	1423	938
55	1625	270	1435	936
60	1633	262	1440	916

5.2 Alternatives

Table 2 denotes the performance of the system with and without the use of optimization probabilities discussed in section 3.7.

Table 2: performance system with and without optimization probabilities

	TN	FP	Cs	Cv
Without probabilities	1608	287	1420	957
With probabilities	1608	287	1530	1014

The system performance in terms of accuracy and relevance increases significantly when the optimization probabilities are utilized. This proves that using the optimization probabilities improves the overall accuracy and relevance of the error corrector.

5.3 Nonword Error Performance Evaluation

The following results were obtained for each type of spelling error.

5.3.1 *Substitutions*

The results obtained for each phase of testing for substitution errors are listed below.

Table 3: substitution results

Phase	TN	FP	Cs	Cv
1	398	102	366	240
2	401	99	376	235
3	415	85	381	243
Total	**1214**	**286**	**1123**	**718**

From the 1500 incorrectly spelled words with a substitution error, 1214 words were flagged as incorrectly spelled and 286 words were recognized as correctly spelled. For each phase of testing, the spell checker provided corrections to over 90% of TN, with an average of 92% for accuracy. The Cv values obtained were not significantly lower than the Cs values, with an average difference of approximately 135 candidate corrections not containing the intended word.

5.3.2 *Insertions*

Table 4 depicts results obtained for insertion errors.

Table 4: insertion results

Phase	TN	FP	Cs	Cv
1	480	20	453	153
2	481	19	457	136
3	487	13	459	145
Total	**1448**	**52**	**1369**	**434**

For insertion errors, the spellchecker was able to significantly detect incorrectly spelled words, yielding very low FN values for each phase of testing. From the flagged words, the spell checker achieved a high accuracy rate for each phase with an average accuracy rate of 95%. Cv was, however, significantly low for each phase. This low Cv rate could be caused from the extra trigram created when a character is erroneously injected into a word. The error corrector does not exclude this trigram when providing candidate corrections and this might be the contributing factor to the low Cv values obtained.

5.3.3 *Deletions*

The Cs results obtained for deletion errors are similar to those achieved by the spellchecker for substitution and deletion errors, with an average of 90% of TN being provided candidate corrections. The TN values for each phase are however much lower and the FP values increase significantly.

Table 5: deletion results

Phase	TN	FP	Cs	Cv
1	332	168	320	232
2	321	179	310	232
3	296	204	227	229
Total	**949**	**551**	**857**	**693**

The low TN and high FP values indicate that when a character is omitted from a word, the trigram constituents of the incorrect word have a high enough frequency that they are all recognized as correct by the error detector. The TN and FP values are an indication that the threshold frequency is too low for deletion errors. This could also indicate that a much larger corpus is required for the spell checker.

5.1.4 *Transpositions*

Looking at table 6, the spell checker's performance for transposition errors is the best from all the spelling errors. All TN, Cs and Cv values are significantly high and all FP values are significantly low.

Table 6: transposition results

Phase	TN	FP	Cs	Cv
1	479	21	472	424
2	481	19	476	433
3	484	16	478	433
Total	**1444**	**56**	**1426**	**1290**

The spell checker's high performance in correcting transposition errors results from the optimization method for transposition errors, stated in section 3.7. Taking phase 1 and running the spell checker without the optimization method yields the following results in table 7

Table 7: Phase 1 performance with and without optimization method for transposition errors

	TN	FP	Cs	Cv
Optimized	479	21	472	424
Not Optimized	479	21	452	215

The results from table 7 shows that Cs marginally decreases. However, there is a significant decrease in the Cv value obtained without the use of the optimization method.

5.4.5 *Cumulative results*

This section details the cumulative performance results for each spelling error. Figure 3 denotes the comparison of the spelling errors according to TN, FP, Cs and Cv values.

Table 8: cumulative results for all spelling errors

Phase	TN	FP	Cs	Cv
Substitutions	1214	286	1123	718
Insertions	1448	52	1369	434
Deletions	949	551	857	693

Transpositions	1444	56	1426	1290
Total	**5055**	**945**	**4775**	**3135**

Table 8 shows the overall performance of the system. From the cumulative results, it can be seen that the error detector is able to detect transposition, substitution and insertion errors adequately. The system, however, does not achieve the same level of error detection for deletion errors. The system is also able to adequately provide candidate corrections for all spelling errors. When it comes to the intended word being a part of the provided candidate corrections, the system has a low Cv value for insertion errors compared to the other spelling errors. Refer to figure 3 for a visual depiction of the results from table 8.

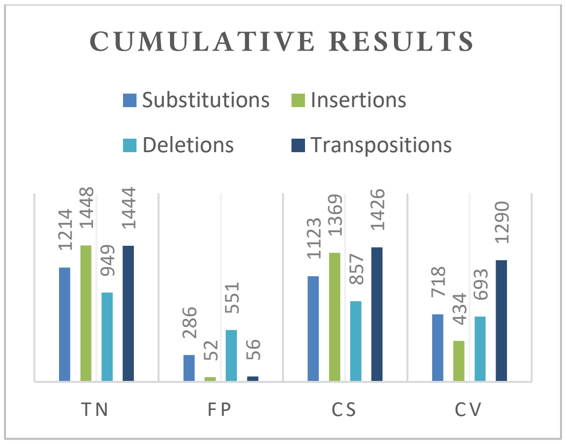

Figure 3: Column chart representing cumulative results for each spelling error

Overall, the system performs adequately at detecting and correcting nonword errors. In order for a spell checker to be successful, the system needs to be able to recognize correctly spelled words as correct. To evaluate this, we take a look at the system's performance when it is fed with a text file containing the correct words used to generate the nonword spelling errors.

5.5 System performance using correctly spelled words

Figure 4 denotes the results for TP, FN, Cs and Cv obtained from running the spell checker using the 6000 correctly spelled words that were used for generating spelling errors.

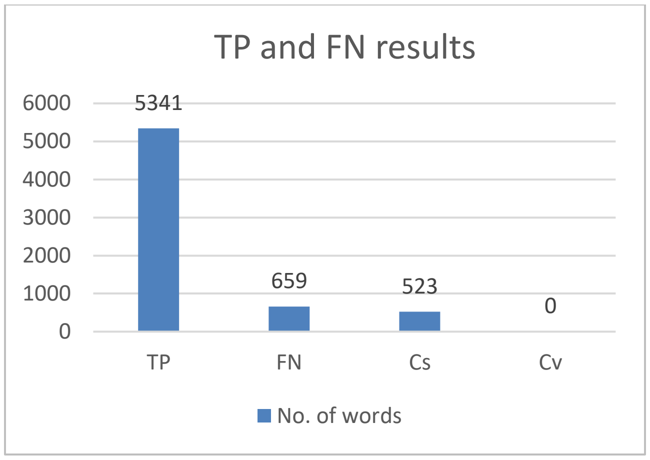

Figure 4: Column chart representing cumulative results for each spelling error

Figure 4 shows that the TP values are significantly higher than the FN values. From the 6000 words, the system was able to recognize 5341 words (89%) as correctly spelled. This shows that the system can adequately recognize correctly spelled words.

5.6 Recall and Precision Measures

Figure 5 denotes the confusion matrix containing the overall TN, FN, TP and TN values. TN and FP values are obtained from table 7, and the TP and FN values are obtained from figure 4.

	Recognized as Correctly Spelled	*Flagged as incorrectly spelled*
Correctly spelled word	TP = 5341	FN = 659
Incorrectly spelled word	FP = 945	TN = 5055

Figure 5: Confusion Matrix

The confusion matrix from figure 5 is used to calculate the recall and precision measures of the system.

5.6.1 Recall Measures

Using the equation from section 4.2.2 and the confusion matrix, the following results are obtained for LR and ER:

$$LR = 89\% \text{ and } ER = 84\%$$

The result for LR proves that the system is representative of the isiZulu language. Hypothesis 1 is proven to be correct.

Hypothesis 2 is also proven to be true, with ER >= 80%. This shows that the corpus used for the system is adequately free of errors.

5.6.2 Precision Measures

Using the equation from section 4.2.3 and the confusion matrix, the following results are obtained for LR and ER:

$$LP = 85\% \text{ and } EP = 88\%$$

The LP and EP values shows that the error detector is can recognize correctly spelled and flag incorrectly spelled words. Thus, hypothesis 3 is proven to be correct.

5.7 Suggestion Adequacy and Accuracy Measures

This section focuses on the suggestion adequacy and accuracy of the error corrector for each type of spelling error. The overall suggestion adequacy and accuracy is also discussed in this section.

Table 9 depicts the accuracy results for each spelling error using the overall Cs and TN values for each type of spelling error.

Table 9: Accuracy results for each spelling error

	Cs	TN	Percentage (%)
Substitutions	1123	1214	93
Insertions	1369	1448	95
Deletions	857	949	90
Transpositions	1426	1444	99
Total	**4775**	**5055**	**94**

The error corrector achieves an accuracy rate above 90% for each type of spelling error. The overall accuracy of the system is 94%. This is an accuracy rate well above the expected accuracy rate of 85%, proving hypothesis 4 to be correct.

Table 10 depicts the relevance results for each type of spelling error as well as the combined relevance result.

Table 10: Relevance results for each spelling error

	Cv	TN	Percentage (%)
Substitutions	718	1214	59
Insertions	434	1448	30
Deletions	693	949	73
Transpositions	1290	1444	89
Total	**3135**	**5055**	**62**

For deletion errors, the system falls short in providing corrections with the intended word as part of the candidate corrections. The system achieves a relevance rate of only 30%. Although the relevance rate for deletion errors is 30%, the system achieves an overall relevance rate of 62%, which is greater than the expected rate of 65%. Therefore, hypothesis 5 is proven to be true. For a visual representation of the accuracy and relevance rates achieved, refer to figure 6(a) and 6(b) in Appendix A2.

6. CONCLUSIONS

Error detection and correction are important spell checking aspects. In order for a spellchecker to be viewed as successful, it should be representative of the language it is correcting, it's corpus or dictionary should be free of errors and it should be able to recognize correctly spelled words and flag incorrectly spelled words at a high accuracy rate. On top of this, the spellchecker should be able to provide candidate corrections for words that are flagged as incorrect using an error corrector. For error correction to be viewed as successful, the error corrector should be able to provide accurate and relevant suggestions to a user [a].

In this paper, we implemented a statistical approach to finding candidate corrections for isiZulu, which is the most widely spoke languages in South Africa. An isiZulu spellchecker that can only perform error detection was used. The isiZulu language was chosen, since it is the only Bantu language with a standalone error detector. This error corrector is the first step towards providing error correction for more Nguni languages. The error corrector was developed and integrated with the error detector. The experiment was then performed on the complete system. The system achieved a language recall rate of 89%, an error recall of 84%, a language precision of 85% and an error precision of 88%. The error corrector was found to have an accuracy rate of 94%, which is well above the expected accuracy rate of 85%. The suggestion adequacy of the system was found to be 62%, which is also above the expected value of 60%. From the following results, the spell checker is found to be successful at detecting and correcting spelling errors.

7. FUTURE WORKS

This error corrector sets the first step towards achieving an isiZulu spellchecker that can provide users with accurate and relevant error correction. The error corrector performs expertly at providing relevant candidate corrections for transposition errors and it can adequately provide relevant candidate corrections for substitution and insertion errors. The system however falls short when it comes to deletion errors. A method to improve on the suggestion adequacy for deletion errors can be a future addition to the spellchecker. The size of the corpus and how error-free it is also plays a role in the overall performance of the system. Looking at the effects of the corpus on the performance of the system and implementing a larger corpus can be looked at in future. Presently, the spell checker cannot provide real-time error detection and autocorrection to the user. This is another feature that can be implemented in future works.

APPENDIX A

A1 Table showing the language, corpora and techniques used to develop existing spellcheckers and how they performed overall

	Language	Corpora	Technique(s)	Performance	
Ndaba et al. [12]	isiZulu	Ukwabelana Corpus (UC); selection of isiZulu National Corpus (INC); small corpus of news items (NIC)	trigrams and quadrigrams and n-gram LM	INC	67% accuracy rate
				NIC	76% accuracy rate
				UC	Above 50% accuracy rate
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	

A2 Column graphs depicting the cumulative results for each spelling error.

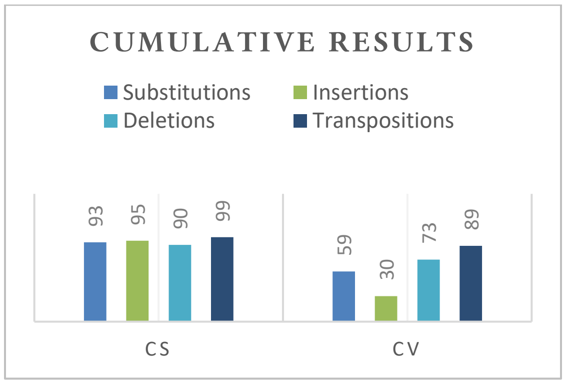

Figure 6 (a): Column chart representing cumulative results for each spelling error

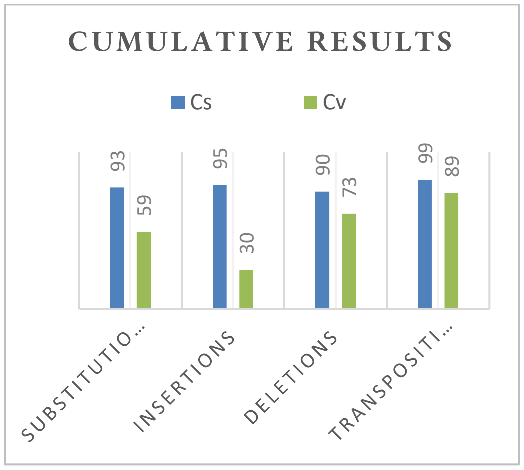

Figure 6 (b): Column chart representing cumulative results for each spelling error

ACKNOWLEDGMENTS

I would like to thank Dr. Maria Keet for her guidance and supervision through the development of the error corrector. I would also like to thank Dr Langa Khumalo for providing the corpus used in this research.

REFERENCES

[1] Bidyut, B. and Chaudhuri, A. A simple real-word error detection and correction using local word bigram and trigram. In Proceedings of the 25th. *Computational Linguistics and Speech Processing (ROCLING 2013).* (Oct 4-5, 2013). ACLCLP, Taiwan, 2013.

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

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

[4] Farra, N., Tomeh, N., Rozovskaya, A. and Habash, N. Generalized Character-Level Spelling Error Correction. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics.* (June 22-27, 2014). ACL, Stroudsburg, Pennsylvania, 2014, 161-167.

[5] Hassan, A., Noeman, S. and Hassan, H. Language independent text correction using finite state automata. In *Proceedings of the 3rd International Joint Conference on Natural Language Processing.* (Jan 7-12, 2008). AFNLP, Hyderabad, India, 2008.

[6] Jurafsky, D. and Martin, J. H. Spelling Correction and the Noisy Channel; 5, 1.

[7] 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. DOI=10.1142/S0218213007003692.

[8] Kernighan, M. D., Church, K. W. and Gale, W. A. A spelling correction program based on a noisy channel model. In *Proceedings of the International Conference on Computational Linguistics.* (August 20, 1990). Association for Computational Linguistics, Stroudsburg, Pennsylvania, 1990, 205-210.

[9] Kukich, K. Techniques for automatically correcting words in text. ACM Computing Surveys (CSUR), 24, 4 (Dec 1, 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. In *ISTAFRICA.2016.* (May 11-13, 2016). IIMC, Durban, South Africa, 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] Paggio, P. and Underwood, N. L. Validating the TEMAA LE evaluation methodology: a case study on Danish spelling checkers. Natural Language Engineering, 4, 3 (1998), 211-228.

[15] Pirinen, T. A. and Hardwick, S. Effect of Language and Error Models on Efficiency of Finite-State Spell-Checking and Correction. In Anonymous *Finite State Methods and Natural Language Processing.* (July 23-25, 2012). Association for Computational Linguistics, Stroudsburg, Pennsylvania, 2012, 1-9.

[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. DOI=10.1007/s10032-002-0082-8.

[17] Sharma, S. and Gupta, S. A Correction Model for Real-word Errors. Procedia Computer Science, 70, Supplement C (2015), 99-106. DOI=//doi.org/10.1016/j.procs.2015.10.047.

[18] Starlander, M. and Popescu-Belis, A. Corpus-based Evaluation of a French Spelling and Grammar Checker. In Anonymous *International Conference on Language Resources and Evaluation (LREC).* (May 29-31, 2002). 2002.

[19] Tong, X. and Evans, D. A. A statistical approach to automatic OCR error correction in context. In Anonymous (August 4, 1996). University of Copenhagen, Copenhagen, Denmark, 88-100.

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

[21] van Huyssteen, G. B., Eiselen, E. R. and Puttkammer, M. J. Re-evaluating evaluation metrics for spelling checker evaluations. In *Proceedings of the 25th International Conference on Computational Linguistics (COLING 2014).* (August 24, 2014). ACL, Stroudsburg, Pennsylvania, 2004, 91-99.

[22] Van Zaanen, M. and Van Huyssteen, G. Improving a spelling checker for Afrikaans. Language and Computers, 47, 1 (2003), 143-156.

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

[24] 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.* Association for Computational Linguistics, 2009, 890-899.