Proceedings of SIGUL2022 @LREC2022, pages 189–198
Marseille, 24-25 June 2022

A Language Model for Spell Checking of Educational Texts in Kurdish (Sorani)

Roshna Omer Abdulrahman and Hossein Hassani
University of Kurdistan Hewlêr
Kurdistan Region - Iraq
roshna.abdulrahman@ukh.edu.krd, hosseinh@ukh.edu.krd

Abstract

Spell checkers have become regular features of most word processing applications. They assist us in writing more correctly in various digital environments. However, this assistance does not exist for all languages equally. The Kurdish language, which still is considered a less-resourced language, currently, lacks well-known and well-tested spell checkers. We present a language model for the Kurdish (Sorani) based on educational texts written in the Persian/Arabic script. We also showcase a spell checker as a testing environment for the language model. Primarily, we use a probabilistic method and our language model with Stupid Backoff smoothing for the spell-checking algorithm. We test for spelling errors on a word and context basis. The spell checker suggests a list of corrections for misspelled words. The results show 88.54% accuracy on the texts in the related context, an *F1* score of 43.33%, and correct suggestions of an 85% chance of being in the top three positions of the corrections.

Keywords: Spell checker, Kurdish Language, Ngram Language Model, Low-Resourced, Error Detection.

1. Introduction

A spell checker is an application that detects grammatical and contextual spelling errors from a given text and tries to correct them according to an algorithm or a set of rules. Some spell checkers suggest a list of correct candidate words for a misspelled word or suggestions for a sequence of words. Automatic spell checking is a popular feature in word processors for most languages. Also, almost all web browsers provide built-in spell checkers.

Research on spell checkers dates back to the late 1950s, and they are now well established for many languages such as English, German, and Chinese. However, the Kurdish language is low-resourced, and the related research, data, and tools are still in their infancy.

About 19 to 28 million people speak the Kurdish language (Hassani and Medjedovic, 2016). Kurdish is written in different scripts, mostly in Persian/Arabic and Latin, and it includes several dialects (Hassani, 2018). The Kurdish language is less-resourced, and the Kurmanji dialect has had more research regarding spell checkers, for example, the Rastnivîs – Add-ons for Firefox while the Sorani dialect has recently had research regarding tools and corpora to be used for spell checkers. We discuss them in the following paragraphs.

We develop a spell checking application for the Sorani dialect focusing on scientific texts. Our application is based on a language model that is created over the segmented KTC corpus (Abdulrahman and Hassani, 2020) and uses the Stupid Backoff smoothing score (Brants et al., 2007) to find non-words and errors within the context of a sentence. We also use Edit Distance (Damerau, 1964) paired with the score for correction and then ranking the list of suggestions. The language model is publicly available at GNU V3.0 license at `https://github.com/KurdishBLARK/KTC-Language-Model`.

The rest of this paper is organized as follows. Section 2 reviews related work on Kurdish language models and spell checkers. Section 3 provides the methodology of the research. Section 4 presents the experiment environment and RastNus application that we created to use the language model. Section 5 presents the findings and results of the paper. Finally, in Section 6, we provide the conclusion.

2. Related Work

Even though the amount of research on the Kurdish language - for all of the dialects - was few and far between, we can say that recent research on the Kurdish language processing has gained popularity in the past decade.

A Spell checking system that already exists for Sorani is Renus, an error correction system that works on a word-level basis and uses lemmatization (Salavati and Ahmadi, 2018). Renus detects an error using lookup methods in a language model. Also, it corrects the erroneous word using Edit Distance. The system suggests a list of candidates' grams of the same position with an Edit Distance of less than three and ranks the suggested corrections based on the candidate's frequency and Edit Distance. Renus spell checker is evaluated by comparing the golden-standard word with first-ranked suggestions of the algorithm. The authors report that the lemmatizer has an accuracy of 86.7% while the spell checker's accuracy with a lexicon is 96.4% and without one is 87%. While most of their work revolves around the Peyv, the spell checker application in this paper is a step in the right direction.

Hawezi et al. (2019) create a spell checking algorithm for the Sorani dialect (Central Kurdish) with a focus on its morphological complexness - agglutinative. They store a list of base words in memory and use variants of a word in which the base stays the same but prefixes, suffixes, and infixes are changed according to a pattern. This method is similar to what spell checking libraries like Hunspell use. They report that 79.93% of the time, the first word is the correct word, 93.30% the correct word is in the top 3, and 97.01% the correct word is in the top 5 suggested words. They have also created a sample application to test the library, but the application is not open-source.

Ahmadi (2020) presents an open-source language processing toolkit for Kurdish (KLPT) that includes a spell checker based on Hunspell. The performance is not discussed since it was under review as of the writing of this article.

Hamarashid et al. (2021) present a word prediction system for Sorani and Kurmanji dialects of the Kurdish language. They use the ngram model where n=5. In other words, the system predicts the next five words after the input text. The authors suggest that the system is effective in correcting spelling errors. The authors develop an application that is not public nor is it open source.

AsoSoft text corpus is a Kurdish Sorani corpus that contains 188 million tokens. The corpus is mostly collected from websites, books, and magazines. The authors share a detailed look at the creation and cleaning process of the AsoSoft corpus. Veisi et al. (2020) create an ngram language model of the corpus to calculate perplexity. The corpus is available for usage on GitHub, while the language model was not shared publicly.

A spell checking web application that was published recently is by the AsoSoft Research Group is the Aso spell checker ئاسۆ هەڵەچنی that can be accessed through this website spell.kurdinus.com (Aso Mahmudi, 2022)

Our focus is on the Sorani dialect written using the Persian/Arabic script presented in table 1. In order to present a use case of the language model, we use the Python programming language for data processing and the spell checking algorithms, and finally developing a word processing environment that performs contextual spell checking on a word level.

ئ	ا	ب	پ	ت	ج	چ	ح
خ	د	ر	ڕ	ز	ژ	س	ش
ع	غ	ف	ڤ	ق	ک	گ	ل
ڵ	م	ن	هـ	و	ۆ	ی	ێ

Table 1: Kurdish Alphabet (from left to right)

As mentioned before, many languages have achieved acceptable accuracy in the spell checking task. We cannot use most of the spell checking algorithms for languages like Kurdish. Not only for script differences but also for inflection (Walther and Sagot, 2010) and grammatical rules. That makes the existing methods limiting and leads to needing different ones or modifying the existing methods to better suit the Kurdish language.

Compared with the Kurdish spell checkers mentioned previously, our work suggests a more robust language model for educational/scientific writing because it is built based on the textbooks edited by academic and professional editors for educational purposes. That allows the model to find errors and suggest corrections that are close to *de facto* writing standard that is currently formally followed in the Kurdistan Region of Iraq.

3. Methodology

In this section, we explain the steps we took for creating an ngram language model. The language model consists of lists of trigrams, bigrams, and unigrams with each ngram's frequency distribution. We look into the smoothing method that we used to make the language model more accurate when used in different scenarios. We also

Unigrams	1	<s>
	2	ئێوە
	3	هیوای
	4	دواڕۆژن
	5	.
	6	</s>
Bigrams	1	<s>, ئێوە
	2	ئێوە, هیوای
	3	هیوای, دواڕۆژن
	4	دواڕۆژن, .
	5	., </s>
Trigram	1	<s>, ئێوە, هیوای
	2	ئێوە, هیوای, دواڕۆژن
	3	هیوای, دواڕۆژن, .
	4	دواڕۆژن, ., </s>

Table 2: Unigrams, Bigrams, and Trigrams created from "<s>ئێوە هیوای دواڕۆژن.</s>".

present the methodology of creating and testing our spell checking algorithm with a simple environment that we develop to test the usage of our language model.

3.1. Developing the Language Model

Chen and Goodman (1999) explain a language model as "a probability distribution over strings P(s) that attempts to reflect the frequency with which each string s occurs as a sentence in natural text."

3.1.1. Ngram

When creating the ngram language model, we started by choosing the ready-made segmented Kurdish Textbook Corpus - KTC (Abdulrahman and Hassani, 2020). A Kurdish Sorani school textbook corpus with 31 books on twelve different subjects at the K-12 level including (Economics, Genocide, Geography, History, Human Rights, Kurdish, Kurdology, Philosophy, Physics, Theology, Sociology, Social Study). The (n) in ngram indicates a number that means it has n consecutive words. In our case, n=3, which is called a trigram. As an example, we take a word from the KTC corpus with two other words in a row in the context of a sentence "<s>ئێوە هیوای دواڕۆژن.</s>". We can create 4 trigrams, 5 bigrams, and 6 unigrams from the above sentence, as shown in table 3.1.1.

We use an ngram language model so that our spell checker can correct not only wrong words but also specify the errors made in the context of a sentence.

3.1.2. Smoothing

Smoothing techniques are used to improve performance in many cases, but when data is sparse, which is the case for Kurdish Sorani, smoothing is more necessary. "The term smoothing describes techniques for adjusting the maximum likelihood estimate to hopefully produce more accurate probabilities." (Chen and Goodman, 1999). There are many smoothing techniques, some are expensive and require a lot of training, such as Kneser-Ney Smoothing (Brants et al., 2007). We chose the funnily named Stupid Backoff Smoothing method by Brants et al. (2007) that most simply put multiplies the probability of a constant 0.4. We explain the smoothing method in more detail in the later sections. The point of using a smoothing method for our language model is to not get zeros too often when checking for a word or an ngram in our language model.

3.2. Testing Environment - A Spell Checking Application

In order to test the language model and have use cases for it, we develop a spell checking environment that uses the language model as its back-end. The spell checker consists of three main tasks that occur one after the other: error detection, error correction - which can be a list of candidate corrections, and ranking the correction list (Verberne, 2002). We look at the performance of of the mentioned tasks separately.

A significant component of our application is spell checking in context and not only a single word.

3.2.1. Algorithm

We build an algorithm that detects erroneous words and corrects them. The algorithm behind the application consists of many parts. We discuss them in the following subsections:

- Error detection: Given a body of text our algorithm uses a Trigram Language Model with Stupid Backoff smoothing (Brants et al., 2007) by checking the user's input text - which we refer to as (s) - for having more than three tokens, if (s) in less than three tokens we check the dictionary (lexicon or unigram list) lookup method, the method is more accurate when the RAM - random access memory - is not a problem (Kukich, 1992). In the case when (s) has more than three tokens, we check the Stupid Backoff score as shown in equation 1, where S is for Stupid Backoff score, w for word, and $\alpha = 0.4$. In equation 2, N is the total number of words in the unigram list (lexicon), and $f(w_i)$ is the frequency distribution of w_i, the current unigram. The researchers (Brants et al., 2007) chose 0.4 for the value of α based on good results in their experiments.

$$S(w_i|w_{i-k+1}^{i-1}) = \begin{cases} \frac{f(w_{i-k+i}^{i})}{f(w_{i-k+i}^{i-1})} & \text{if } f(w_{i-k+i}^{i}) > 0 \\ \alpha S(w_i|w_{i-k+1}^{i-1}) & \text{otherwise} \end{cases} \tag{1}$$

$$S(w_i) = \frac{f(w_i)}{N} \tag{2}$$

In our case of using trigrams, the score result is a relative frequency. As shown in equation 1 above, when the trigram has a frequency that is more than 0, the score is the trigram's frequency divided by the bigram frequency. This pattern continues until we reach the unigram level. When the unigram has a frequency more than 0, the score is the unigram's frequency divided by the total unigram frequency. Otherwise, the score is zero.

- List of candidate corrections: The erroneous word is modified by two Edit Distance (insertions, deletions, substitutions, and transpositions) by Damerau (1964) and Levenshtein and others (1966).
- Correction list ranking: We rank the suggestions based on the Stupid Backoff score using equation 1.
- Context correction: We use the Trigram Language Model within a sentence boundary.

To correct the errors the system found, it needs to find the types of errors starting from the smallest unit, letters, and more feasibly letters within a word boundary. We train our algorithm on what we consider the wrong word. We do not find "if" as a mistake unless it is used in the wrong context, such as "if course". Likewise, in Kurdish saying (بـه) is all well and good unless used in the wrong context, such as (بـه زانكۆ دەخوێنم).An incorrect letter substitute could have caused this type of spelling error. How many other types of errors can we find? Let us continue on the word level. Here is a list of error types our algorithm seeks to correct within a word boundary. We use the term *character* instead of using letters in the upcoming paragraphs:

1. Character substitution: زانكێ, the character □ is substituted with ێ.
2. Adding an extra character (insertion): زاينكۆ the character ى in the third position starting from the right.
3. A character missing or deleted: زاكۆ the character ن is missing
4. Neighboring character order transposition: زاكنۆ the characters ن and ك have changed positions.

Damerau–Levenshtein's (Bard, 2006) Edit Distance covers all of the word boundary errors we mentioned. Where Levenshtein's Distance (Levenshtein and others, 1966) measures the difference between two words by how many operations it is needed to turn one into the other, or in other words, correct the erroneous word. These operations include insertion, deletion, and substitution of a character.
Damerau (1964) adds a new operation, which is the transposition of neighboring characters within a word. The probability of the next word given the previous word is known as the chain rule (Samanta and Chaudhuri, 2013), and we are using Markov's assumption, the last n in the chain. We check for context within the sentence boundary. We take the start of the sentence token <s> and end of sentence token </s> into account to achieve a correctly normalized probability of the complete sentence.
Following equation 1, we take the user input of a set of serialized strings - a word, a sentence, or a paragraph. The algorithm segments the user input into sentences with the beginning of sentence tags <s> twice so that a trigram of (u, v, w) where w is the first word in a sentence u and v are the start of sentence indicators as well as the end of sentence tags </s>. Then the algorithm loops through each sentence and creates trigrams within its bounds, and then the index cursor starts at the beginning tag and gets the first trigram of (u, v, w) the tag included and continues until it reaches the end of the sentence tag. Within a sentence boundary, we

check each ngram's - trigram, bigram, and unigram - Stupid Backoff score. If the score is bigger than zero S(wi) > 0, we create a confusion set by using an Edit Distance of two and putting each correct candidate word back into the trigram, then recheck the trigram's score, and then put the trigram in the suggestion list with its score.
The Stupid Backoff technique, as shown in equation 1 checks the given trigram of (u, v, w) in the list of trigram frequencies. If the trigram has a frequency of zero, it checks the bigram of the given words of (v, w) multiplied by α - as previously mentioned, we chose 0.4 because it was suggested by the (Brants et al., 2007) to be the optimal value. If the bigram score is zero, it checks the unigram of (w) by calculating equation 2 - where N is the total number of unigrams (length of the lexicon) again multiplied by α and returns the score, or it returns zero where all the iteration resulted in zero. The suggestion list contains the top five suggestions of each trigram that are sorted by the score. We highlight the erroneous tokens from the suggestion list, and when the user selects a suggested item, we check the text one more time.

3.3. Testing Methods

We test and evaluate our algorithm's error detection and error correction with suggestion ranking with written tests by students that study the textbooks. The test data is educational and in the same style of writing as our corpus. The test data has not been used in developing the language model.
We collect testing data by having students who have studied the textbooks from the KTC corpus. The students take dictation from the remaining 10% of the selected corpus's test data. We chose the student randomly. A teacher with a supervisor reads the dictation material to them, and they type it in a basic text processor with all hints and help turned off.
We prepare the dictation material by looking at research on the typing speed of students in the same age range as our participants. (Horne et al., 2011) report that 11-year-olds have a typing speed of over 13 words per minute (wpm). 12-year-olds over 16 and 13-year-olds over 20 wpm. 14-year-olds over 24 wpm, and another research on elementary schools in Georgia, (Gillespie and Leader, 2005) report that an average kindergarten through fifth-grade students have a speed of 5.1 wpm with no prior practice and 5.91 wpm after 7 lessons taking the speed of typing by age into account and the time limit we categorized the dictation material for each grade.
We manually evaluate the performance of error detection the chosen method by computing precision, recall, and accuracy from calculating each test's true positive, false positive, true negative, and false negative. The equations are listed in equations (3, 4, 5, 6) respectively. Then we manually evaluate our test set and the collected dictation of the test set via the human evaluation of the tests.

$$Precision = \frac{TP}{TP + FP} \tag{3}$$

$$Recall = \frac{TP}{TP + FN} \tag{4}$$

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{5}$$

$$F1 = \frac{2 * TP}{2 * TP + FP + FN} \tag{6}$$

Moreover, similar to Samanta and Chaudhuri (2013), we rate the probability of the list of suggestions and flag where is suggestion is the correct candidate or a better wording in the context. We look into the top five positions in the suggestions that are retrieved from the algorithm.

4. The Test Environment - RastNus Application

In this section, we present the spell checking application created from the methodology of this paper. We show the application's user interface as well as the back-end of how the concept was implemented. We report the testing process, and we present the application's performance as well as other results in the results and discussion section.

4.1. User Interface

We named the spell checking application RastNus (RastNus ڕاستنوس - is a Kurdish noun that is composed of two other nouns; Rast means truthful and Nus means writing). When viewing the application as a user, RastNus has a simple interface of two components, the text editing area, and the "Spell Check" button. When the user inputs text in the text editing box and presses the spell check button, a table of erroneous words alongside its corrections is shown to the user with a select link next to each correct candidate word. The table of erroneous words and candidate selection is shown in figures 2 and 3, respectively.

After the user inputs text in the text area and selects the “Spell Check” button, the application follows a series of steps. A flow chart of the steps is shown in figure 1 - RastNus application process.

4.1.1. Algorithm

The algorithm behind the RastNus application first checks for non-words. If the given word is incorrect - using Edit Distance - the application shows the user a list of candidate words. RastNus’s algorithm contains the following components:

- A Language Model consists of lists of trigrams, bigrams, and unigrams with each ngram’s frequency distribution.
- A custom tokenizer trained on KTC’s test set using Punkt (Kiss and Strunk, 2006).

The first step in the main method of the RastNus application is loading the prepared data by starting with calling the LoadTokenizer method. When the method is called, the program loads KTC’s custom tokenizer. We manually added a list of abbreviations to the pickle file, and the abbreviations are: [‘د’, ‘م’, ‘د.خ’, ‘پ’, ‘پ.ز’]. Then the LoadNgramsInToDic method is called to load the ngram language model of trigrams, bigrams, and unigrams paired with frequency distribution created from our corpus. The SpellCheck method is triggered when the user clicks the “Spell Check” button. It starts by cleaning the user’s input text (T). It removes extra space and replaces the character ک with ك because the first form of the sound /k/ does not appear in the KTC we take this step to avoid unnecessary flagging.
The cleaned text (T) is sent to KTC’s custom tokenizer (trained using Punkt) so that (T) is made into trigrams. The resulting trigram is sent to get a Stupid Backoff smoothing score. If the score is zero, that ngram is appended to a list. Each word in that list is sent to the notKnown method to check whether that word exists in our dictionary list. If the word is unknown, it is sent to the candidatesSet2Prob method to get a list of words with an Edit Distance of two from the original unknown word. Each candidate word is put back into the ngram. We check for the Stupid Backoff score of each reconstructed ngram to check for the correction within the ngram context. If the score is more than zero, the ngram is considered as a candidate correction, and it is shown to the user in the form of a table shown in figure 2.
Once the user selects a suggested candidate, the updated text is once again checked, as presented in figure 3.

4.1.2. RastNus Testing Procedure

We manually tested the RastNus application by taking random paragraphs from the test set and running it through the application. We also checked the student dictation data and collected error types (tp, tn, fp, fn) while comparing RastNus’s spell checking manually with a human checker. We tested and evaluated the algorithm that our spell checker - RastNus, uses. The results are displayed and discussed in the following paragraphs.
We collected the test corpus of dictation to evaluate RastNus alongside the test set. Twenty-three students from 1st grade to 9th grade in total participated in the computer-based dictation. The first session with eight available computers, and in the second session of six participants, three laptops were available.
According to research, 4, 5, and 6-year-olds have a 5 to 13 word per minute speed, nine on average. In that case, a 250-word paragraph is sufficient for that age group and the time we have. However, 7th to 9th graders have a speed of 16 to 24 words per minute 20 on average, 1000 Word page dictation could be achieved in 30 minutes. In the first session, we selected the first n words (number of words needed per age group) for dictation from our test set, but for the second session, we selected a random set of sentences that made up n words or more.
We tested RastNus using the test set and the dictation data, by manually checking each word and flagging it as the corresponding error type (true positive, true negative, false positive, false negative). See an example in figure 4. Then we counted the error types and then calculated precision, recall, F1, and accuracy. We manually checked each suggestion of the test set and the dictation data by tagging the correct candidate in the top five suggestions the percentage of correct candidates suggested in the test set.

5. Results

In the following sections, we showcase the language model we created as well as present the results of our spell checking algorithm alongside it.
The trigram ngram language model of the KTC corpus consists of 94,188 unigrams, 372,903 bigrams, and 521,797 trigrams.

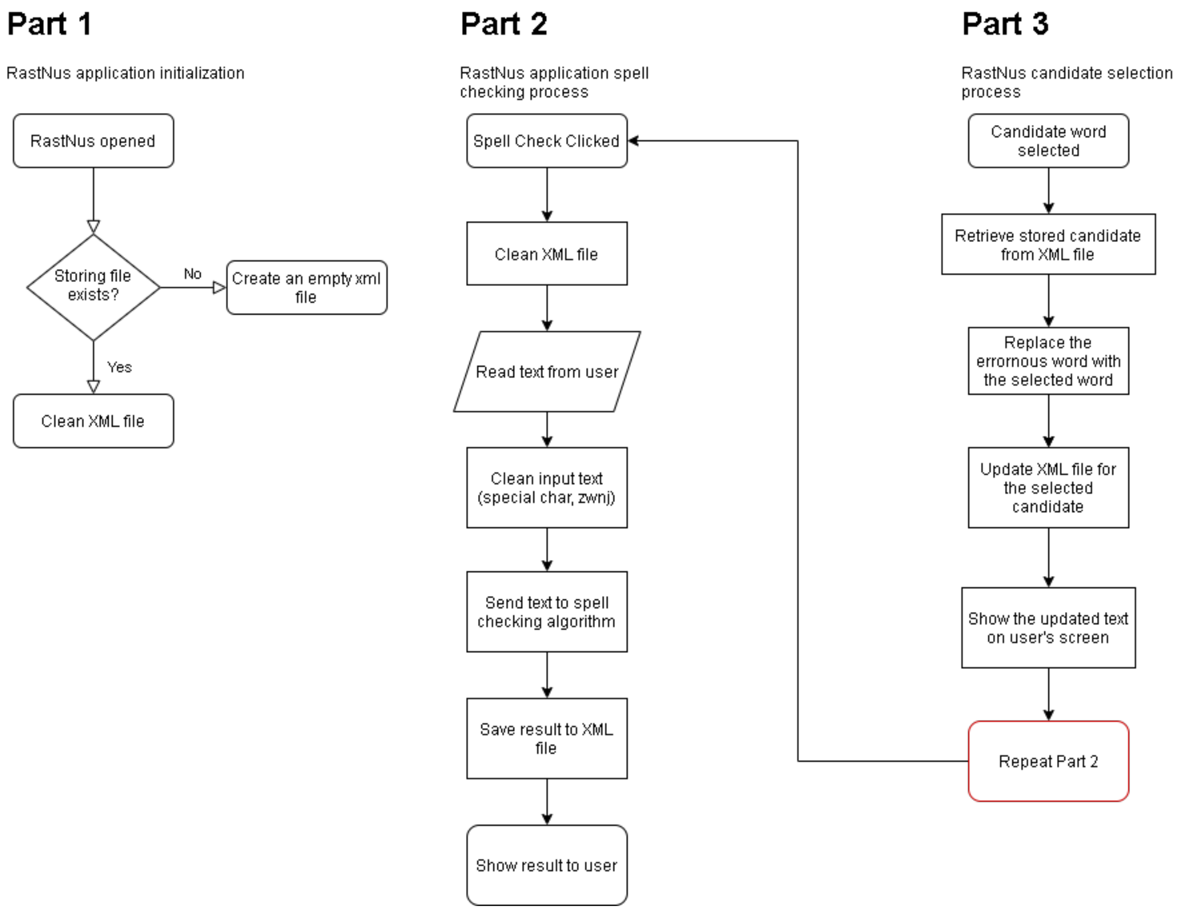

Part 1
RastNus application initialization
RastNus opened
Storing file exists?
No
Create an empty xml file
Yes
Clean XML file
Part 2
RastNus application spell checking process
Spell Check Clicked
Clean XML file
Read text from user
Clean input text (special char, zwnj)
Send text to spell checking algorithm
Save result to XML file
Show result to user
Part 3
RastNus candidate selection process
Candidate word selected
Retrieve stored candidate from XML file
Replace the erronous word with the selected word
Update XML file for the selected candidate
Show the updated text on user's screen
Repeat Part 2

Figure 1: RastNus application process.

#	Unigram	Bigram	Trigram
1	و	. </s>	) . </s>
2	<s>	) .	... ...
3	</s>	... ...	: ((
4	.	: (	) و (
5	لە	) و	(د.خ)
6	)	) ی	2 . </s>
7	(	کە لە	1 . </s>
8	:	) ،	3 . </s>
9	بە	<s>لە	<s>2 .
10	کە	<s>2-	پێغەمبەر (د.خ

Table 3: Top 10 ngrams.

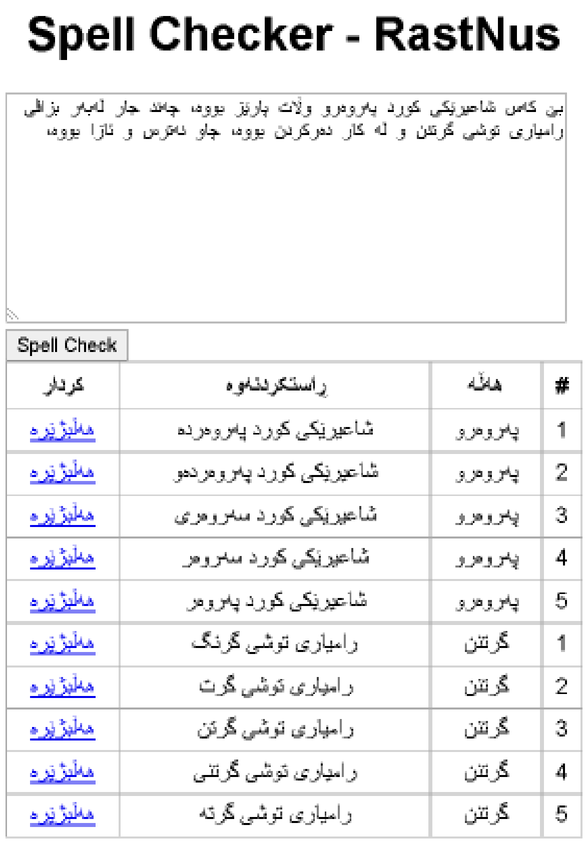
We present top the 10 ngrams of our language model that contains trigrams, bigrams, and unigrams in table 5.

The result of our spell checking algorithm is shown in table 5 it contains the error types (true positive, false positive, true negative, and false negative), and precision, recall, F1, and accuracy are calculated. From the first test using the dictation data, we can see that the F1 score is 65.94%, and the F1 score of testing the algorithm with our test set is 21.90%. The total F1 score of our method is 43.33%, while the accuracy is significantly higher. The dictation data has an accuracy of 82.27%, and the accuracy of testing the algorithm using our test set is 90%. Overall the accuracy of our method is 88.54%. The F1 score is a harmonic mean between precision and recall, while the accuracy measures all correctly flagged cases with equal importance.

The reason behind the notable difference in accuracy and F1 score is that unlike the F1 score, the accuracy takes true negatives into account.

Type	T.P	F.P	T.N	F.N	Precision	Recall	F1	Accuracy
Dictation	154	16	584	143	90.58%	51.85%	65.94%	82.27%
Test set	54	177	3,411	208	23.37%	20.61%	21.90%	90%
Total	208	193	3995	351	51.87%	37.20%	43.33%	88.54%

Table 4: RastNus spell checker performance.



Spell Checker - RastNus

بێ کەس شاعیرێکی کورد پەروەرو وڵات پارێز بووە، چەند جار لەبەر بزاڤی
رامیاری توشی گرتن و لە کار دەرکردن بووە، چاو نەترس و ئازا بووە،

#	هەڵە	ڕاستکردنەوە	کردار
1	پەروەرو	شاعیرێکی کورد پەروەردە	هەڵبژێرە
2	پەروەرو	شاعیرێکی کورد پەروەردەو	هەڵبژێرە
3	پەروەرو	شاعیرێکی کورد سەروەری	هەڵبژێرە
4	پەروەرو	شاعیرێکی کورد سەروەر	هەڵبژێرە
5	پەروەرو	شاعیرێکی کورد پەروەر	هەڵبژێرە
1	گرتن	رامیاری توشی گرنگ	هەڵبژێرە
2	گرتن	رامیاری توشی گرت	هەڵبژێرە
3	گرتن	رامیاری توشی گرتن	هەڵبژێرە
4	گرتن	رامیاری توشی گرتنی	هەڵبژێرە
5	گرتن	رامیاری توشی گرتە	هەڵبژێرە

Figure 2: RastNus Spell Checker testing.

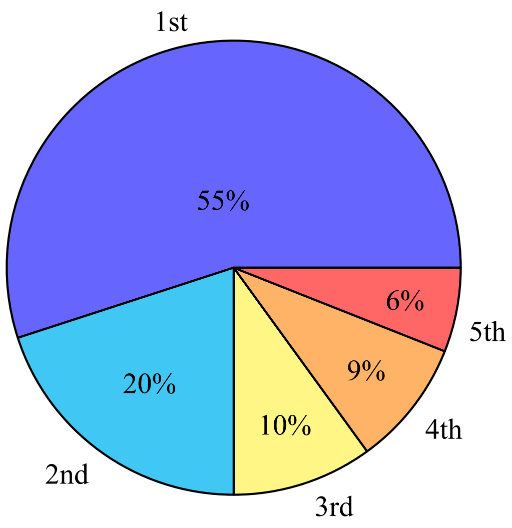
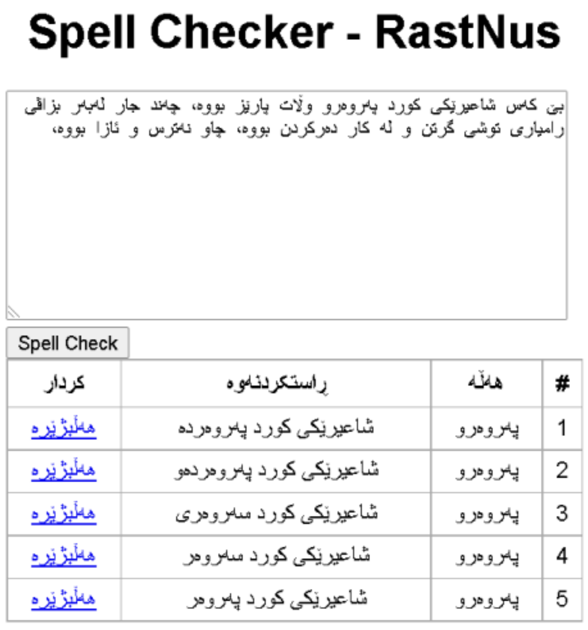


Figure 5: Total percentage of correct suggestion candidates in the top five positions.



Spell Checker - RastNus

بێ کەس شاعیرێکی کورد پەروەرو وڵات پارێز بووە، چەند جار لەبەر بزاڤی
رامیاری توشی گرتن و لە کار دەرکردن بووە، چاو نەترس و ئازا بووە،

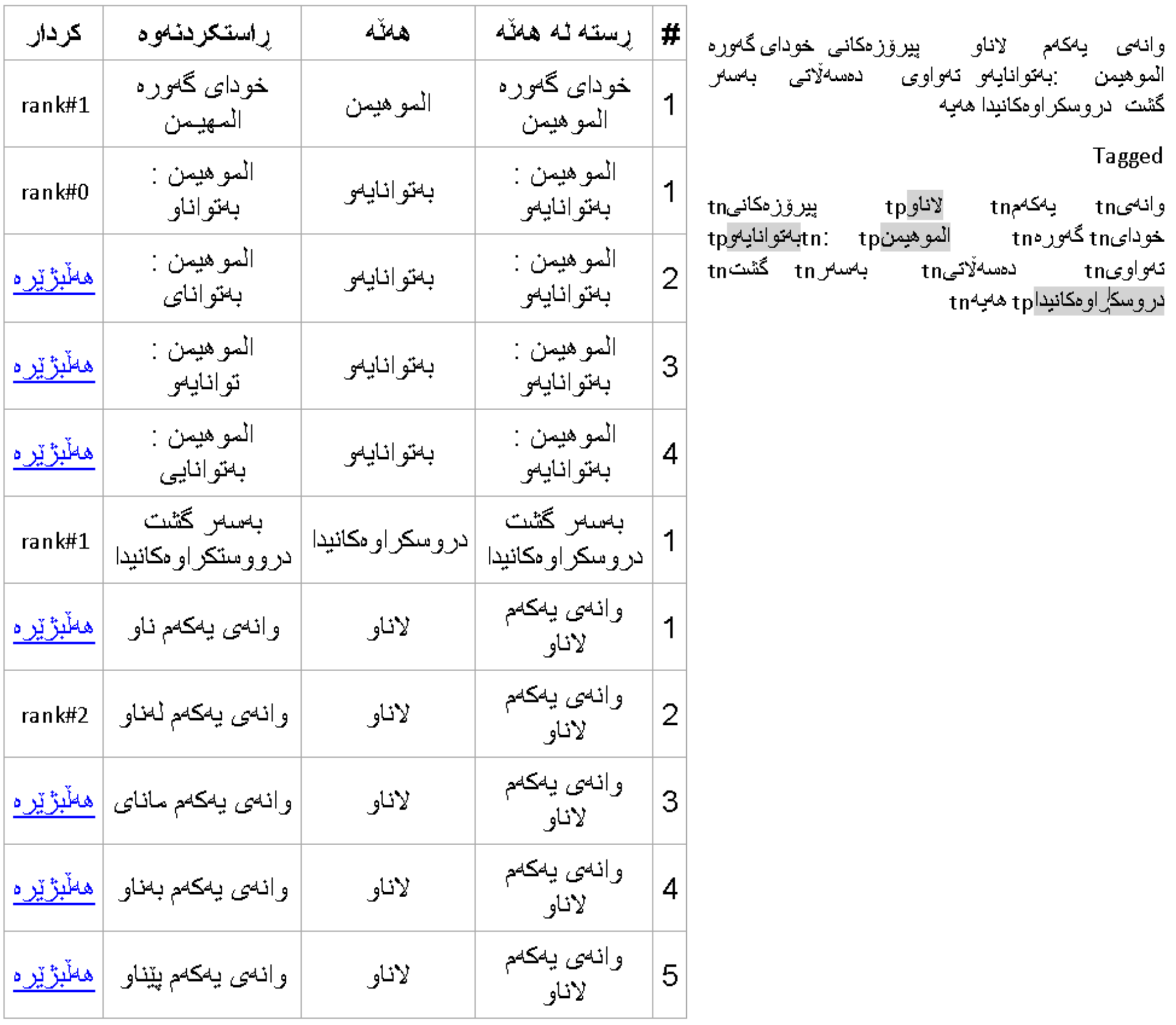
#	هەڵە	ڕاستکردنەوە	کردار
1	پەروەرو	شاعیرێکی کورد پەروەردە	هەڵبژێرە
2	پەروەرو	شاعیرێکی کورد پەروەردەو	هەڵبژێرە
3	پەروەرو	شاعیرێکی کورد سەروەری	هەڵبژێرە
4	پەروەرو	شاعیرێکی کورد سەروەر	هەڵبژێرە
5	پەروەرو	شاعیرێکی کورد پەروەر	هەڵبژێرە

Figure 3: RastNus Spell Checker: candidate word selected.

We manually checked each suggestion of the test set and the dictation data by tagging the correct candidate in the top five suggestions. The percentage of correct candidates suggested in the test set is presented in figure 5. The suggestion in the top three positions of the correct suggestion makes up over 85% of the correct suggestions.

6. Conclusion

We created an ngram language model made of lists of trigrams, bigrams, and unigrams with each ngram's frequency distribution, and we used the Stupid Backoff smoothing method.
We built a spell checking Web application RastNus to use the language that we aim at making it publicly available.
We used a desktop-based version of this application to test the error detection and correction of the language model using the developed spell checking algorithm. The spell checking algorithm uses a probabilistic method. Error detection uses a



کردار	ڕاستکردنەوە	هەڵە	ڕستە لە هەڵە	#
rank#1	خودای گەورە المهیـمن	الموهیمن	خودای گەورە الموهیمن	1
rank#0	الموهیمن : بەتواناو	بەتوانایەو	الموهیمن : بەتوانایەو	1
هەڵبژێرە	الموهیمن : بەتوانای	بەتوانایەو	الموهیمن : بەتوانایەو	2
هەڵبژێرە	الموهیمن : توانایەو	بەتوانایەو	الموهیمن : بەتوانایەو	3
هەڵبژێرە	الموهیمن : بەتوانایی	بەتوانایەو	الموهیمن : بەتوانایەو	4
rank#1	بەسەر گشت درووستکراوەکانیدا	دروسکراوەکانیدا	بەسەر گشت دروسکراوەکانیدا	1
هەڵبژێرە	وانەی یەکەم ناو	لاناو	وانەی یەکەم لاناو	1
rank#2	وانەی یەکەم لەناو	لاناو	وانەی یەکەم لاناو	2
هەڵبژێرە	وانەی یەکەم ماناى	لاناو	وانەی یەکەم لاناو	3
هەڵبژێرە	وانەی یەکەم بەناو	لاناو	وانەی یەکەم لاناو	4
هەڵبژێرە	وانەی یەکەم پێناو	لاناو	وانەی یەکەم لاناو	5

وانەی یەکەم لاناو پیرۆزەکانی خودای گەورە الموهیمن :بەتوانایەو تەواوی دەسەڵاتی بەسەر گشت دروسکراوەکانیدا هەیە

Tagged

وانەیtn یەکەمtn لاناوtp پیرۆزەکانیtn
خودایtn گەورەtn الموهیمنtp :tn بەتوانایەوtp
تەواویtn دەسەڵاتیtn بەسەرtn گشتtn
دروسکراوەکانیداtp هەیەtn

Figure 4: RastNus output manual tagging.

dictionary lookup to find non-word mistakes, and for real-word errors, we use the Stupid Backoff method for each trigram within a sentence boundary. For error correction, the algorithm uses the Edit Distance of two to create a confusion set. Where the word of the set has a Stupid Backoff score of over zero, it adds the word to the candidate list. The algorithm ranks the list of candidates using the score of relative frequency, which is the output of the Stupid Backoff smoothing method. The error detection has an F1 score of 43.33% and an 88.54% accuracy, and the correct suggestion is in the top three positions in 85% of cases.

The aim of the language model creation is that it can be used in official settings.

For future work, we are interested in expanding the language model and with it, the spell-checking environment to cover other Kurdish dialects and include different scripts.

The spell checker could be improved further as usually these kinds of applications could, and we would like to work on expanding the language model further by adding more (official and educational) documents.

We hope the researchers in the field to use the language model to enrich the data and tools for the Kurdish language.

7. Acknowledgements

We would like to apprecia the assistance of Hakary Basic School's staff in allowing us to use their materials and test their students.

8. Bibliographical References

Abdulrahman, R. O. and Hassani, H. (2020). Using Punkt for Sentence Segmentation in non-

Latin Scripts: Experiments on Kurdish (Sorani) Texts . *arXiv preprint arXiv:2004.14134*.

Ahmadi, S. (2020). KLPT – Kurdish Language Processing Toolkit. In *Proceedings of Second Workshop for NLP Open Source Software (NLP-OSS)*, pages 72–84.

Aso Mahmudi, A. R. G. (2022). Aso Spell Checker.

Bard, G. V. (2006). Spelling-Error Tolerant, Order-Independent Pass-Phrases via the Damerau-Levenshtein String-Edit Distance Metric. *Cryptology ePrint Archive*.

Brants, T., Popat, A. C., Xu, P., Och, F. J., and Dean, J. (2007). Large Language Models in Machine Translation.

Chen, S. F. and Goodman, J. (1999). An Empirical Study of Smoothing Techniques for Language Modeling. *Computer Speech & Language*, 13(4):359–394.

Damerau, F. J. (1964). A Technique for Computer Detection and Correction of Spelling Errors. *Communications of the ACM*, 7(3):171–176.

Gillespie, C. and Leader, L. (2005). *We can...can they? Touch Typing for First Graders*. Ph.D. thesis, Citeseer.

Hamarashid, H. K., Saeed, S. A., and Rashid, T. A. (2021). Next word prediction based on the N-gram model for Kurdish Sorani and Kurmanji. *Neural Computing and Applications*, 33(9):4547–4566.

Hassani, H. and Medjedovic, D. (2016). Automatic Kurdish Dialects Identification. *Computer Science & Information Technology*, 6(2):61–78.

Hassani, H. (2018). BLARK for multi-dialect languages: towards the Kurdish BLARK. *Language Resources and Evaluation*, 52(2):625–644.

Hawezi, R. S., Azeez, M. Y., and Qadir, A. A. (2019). Spell checking algorithm for agglutinative languages “Central Kurdish as an example”. In *2019 International Engineering Conference (IEC)*, pages 142–146. IEEE.

Horne, J., Ferrier, J., Singleton, C., and Read, C. (2011). Computerised assessment of handwriting and typing speed. *Educational and Child Psychology*, 28(2):52.

Kiss, T. and Strunk, J. (2006). Unsupervised Multilingual Sentence Boundary Detection. *Computational linguistics*, 32(4):485–525.

Kukich, K. (1992). Techniques for Automatically Correcting Words in Text. *Acm Computing Surveys (CSUR)*, 24(4):377–439.

Levenshtein, V. I. et al. (1966). Binary codes capable of correcting deletions, insertions, and reversals. In *Soviet physics doklady*, volume 10, pages 707–710. Soviet Union.

Salavati, S. and Ahmadi, S. (2018). Building a lemmatizer and a spell-checker for sorani kurdish. *arXiv preprint arXiv:1809.10763*.

Samanta, P. and Chaudhuri, B. B. (2013). A simple real-word error detection and correction using local word bigram and trigram. In *Proceedings of the 25th conference on computational linguistics and speech processing (ROCLING 2013)*, pages 211–220.

Veisi, H., MohammadAmini, M., and Hosseini, H. (2020). Toward Kurdish language processing: Experiments in collecting and processing the AsoSoft text corpus. *Digital Scholarship in the Humanities*, 35(1):176–193.

Verberne, S. (2002). Context-sensitive spellchecking based on word trigram probabilities. *Unpublished master's thesis, University of Nijmegen*.

Walther, G. and Sagot, B. (2010). Developing a Large-Scale Lexicon for a Less-Resourced Language: General Methodology and Preliminary Experiments on Sorani Kurdish. In *Proceedings of the 7th SaLTMiL Workshop on Creation and use of basic lexical resources for less-resourced languages (LREC 2010 Workshop)*.