OOV Pashto Spell Checker with Morphological Operations

M. Taimoor Khan[1], Nasarullah Jan Wazir[2], Imran Khan[3], Muhammad Khan Afridi[4] and Omar Usman Khan[5]
[1] *GESIS - Leibniz Institute for the Social Sciences, Germany*
[2]*Pashto Academy, University of Peshawar, Pakistan*
[3] *Independent researcher*
[4]*Abasyn University, Pakistan*
[5] *National University of Computer and Emerging Sciences, Pakistan*

***Abstract*—A spell-checking model detects spelling errors in the input text, generates possible corrections for each error, and organizes them based on their relevance. Such tools are essential for writing, editing, and publishing in a language. In literature, different variants of edit distance and language models are used for dealing with spelling errors. The existing approaches are improved for efficient use of computational and memory resources without compromising accuracy. Exploiting the efforts made for other low-resource languages, we have proposed the first spell-checking model for Pashto language. A common issue with the existing spell checker models of low-resourced languages is having many false positives due to limited vocabulary. Our proposed approach makes use of morphological operations to generate out-of-vocabulary words and is coupled with a filtering mechanism to retain the words with more repeating syllable patterns only. The n-gram probabilities are combined through linear interpolation to detect and correct real-word errors. The test samples with spelling errors are generated by randomizing characters for non-word errors and words in higher-order n-gram for real-word errors. The proposed approach achieves detection accuracy of 85.5% and correction accuracy of 75%, respectively.**

***Index Terms*—Spell checker, linear interpolation, edit distance, language models**

I. Introduction

Spell-checking is one of the fundamental tools in text-based applications. It ensures that the user has written the correct information in the input text [1]. Spell checking tools are part of everyday use in text editors, search engines, publication portals, and assessments. Most of the spell-checking research is focused on prominent languages and has considerable success. Pashto is a low-resourced language that does not have a spell checker despite having tens of millions of native speakers in Afghanistan and the adjacent regions of Pakistan. Content in the Pashto language is produced in the form of newspapers, books, magazines, news channels, and web pages. Older books are also digitized with the help of optical character recognition tools, thus making them available for applying the latest artificial intelligence techniques. Prominent social media platforms support the Pashto language allowing amateur authors to express themselves in their language. It emphasizes the need for a spell checker for the Pashto language to ensure error-free content. Building such a basic tool will also encourage research in other areas of natural language processing (NLP) for the Pashto language.

Manuscript received Aug 21, 2021

TABLE I: Out-of-vocabulary words generated from the vocabulary words using prefix or suffix morphological operations. The last column shows annotation whether the generated word is a correct word.

Vocabulary word	Generated word	Type	Label
کول	راکول	prefix	correct
عدد	عددي	suffix	correct
چې	ګچې	prefix	incorrect
خو	خوسر	suffix	incorrect

Spell checking is also very helpful in preprocessing datasets for building smart applications. In the case of training NLP models for automation, there is a huge reliance on the quantity and quality of data. Although the quantity of data available hasn't been an issue for some tasks, its quality does pose a challenge particularly when the data is collected from online portals and social media. Some of the frequently found mistakes in data compiled from online sources include spelling errors, over or underuse of capitalization, domain and platform-specific noise, grammatical errors, and use of regional slang. To train an AI-based model for specific NLP tasks, these issues are to be addressed at the preprocessing step to ensure the quality of data. Bypassing these corrections leads to building inconsistent and unreliable models [2]. Therefore, spell checking is an essential utility to improve the performance of sentiment analysis, text categorization, text summarization, spam detection, and other prediction tasks.

A spell checker may encounter two types of errors as spelling mistakes. They are called non-word errors and real-word errors. A non-word error is a case when a word does not exist in the trained model and is, therefore, not considered a word of the language. A comprehensive dataset is required to ensure this. It generally arises due to typos or a lack of understanding of the language. On the other hand, the real

word error is due to a wrongly placed known word. Such a word is used out of its intended purpose and is therefore unfit in the context of the sentence. The real word errors are more challenging in detection as they require an understanding of the context of each word [1]. Generally, the spell checker model needs huge data with enough diversity to avoid false positives i.e., detecting correct words as errors due under representation.

A spell checker model has two components. The first component detects errors in the spelling of each word. Language models are often used for this purpose that varies from a list of permitted words to finite-state graphs. The second component is the error model that has the purpose of rectifying the detected mistakes with possible correct suggestions. It performs two tasks i.e., generating candidate replacements for the misspelled word and prioritizing them to help the end-user with more likely suggestions [3]. It makes use of edit distance or its variants and n-gram frequencies for both tasks. In case of suggesting correct words for a non-word error, edit distance is applied on unigrams at the character level. Whereas, in the case of real-word errors, it is applied on higher-order n-grams at the word level. If there is a tie in edit distance of multiple words, they are ordered based on their frequencies of normalized frequencies. Other string similarity measures and phonetic measures may also be used for this purpose.

The existing research in spell checking for low-resourced languages focuses on the prime issue of dealing with the lack of a comprehensive language dictionary and large consistent datasets. They are essential for achieving high detection and correction accuracy. A secondary concern of such tools for any language is the efficient use of computational and memory resources, considering the volume of words to deal with. With a brute force attempt candidate words are generated with Levenshtein edit distance [4]. If a word is not found in the dictionary, it is indicated as an error. The candidate words that have the shortest Levenshtein edit distance are provided with possible corrections for it. A drawback with the Levenshtein edit distance for spell checking is that it supports the insert, delete and substitute operations only but does not support transportation. Therefore, the suggested correction will be irrelevant for words having transportation-based errors. Damerau Levenshtein edit distance which supports transportation operations is used as a more promising solution [5]. To improve the efficiency of the model, the suggestions are precomputed by identifying all the possible errors of a word using edit distance. Different hash functions are used to facilitate the retrieval of these suggestions. However, the downside is that it increases memory usage by manifolds. The use of only the delete operation in edit distance for all the corpus words helps in dropping memory consumption by manifold without compromising on accuracy [6].

In this research, we address the issue of not having access to a comprehensive vocabulary for building a spell checker model. The proposed approach uses morphological operations to generate newer out-of-vocabulary words from the existing list of words. Frequent morphemes at the start and end of words are identified and used as prefixes and suffixes to generate new out-of-vocabulary words. Table I provides samples of out-of-vocabulary words generated from frequent morphemes of existing words. The first two are correct words while the latter two are incorrect and removed in filtering. Two filtering mechanisms are used to remove newly generated non-words. The first filter is based on a word synonym dictionary of limited words where an out-of-vocabulary word having a mention in the dictionary is considered as the correct word. The second filter is having a high linear interpolation score of character-level n-gram probabilities with add-k smoothing. It helps to retain words with more repeating syllable patterns only. Similarly, the insufficient higher-order n-grams are supplemented with out-of-vocabulary higher-order n-grams generated from the frequent n-grams through applying edit distance with substitution only. The linear interpolation filter of n-gram probabilities with add-k smoothing is applied at the word level to weed off the unlikely higher-order n-grams. The frequency of an out-of-vocabulary n-gram is computed by averaging the frequencies of the nearest in corpus n-grams. These words serve as the ground truth for detecting and correcting spelling errors. Thus, our proposed approach empowers the spell checker model to detect and correct errors for words that are beyond the available list of words. It results in dropping the false positive cases which are reflected by the accuracy of the model. Using such an approach that can compensate for the limited vocabulary can help with the other tasks of low-resourced languages as well.

We believe that this is the first Pashto spell checker for which recent approaches in spell checking research are considered. The non-word errors are detected with an accuracy of 80.5% while the non-word error correction accuracy is 75.5%. While, the real word error detection accuracy is 90.5% and correction accuracy is 74.5%, respectively. Figure 1 provides an interface of the Pashto spell checker application that has identified 13 mistakes in a text of 66 words. Some of the errors are provided with possible corrections while others have no relevant corrections in the trained model. We have also compiled a dataset from Pashto journal and BBC Pashto articles having a total of 87,253 unique words. The dataset is made publicly available at (http://pwr.nu.edu.pk/coCL/).

II. Literature Review

Spell checker applications are developed for many languages and are a frequently used tool in everyday usage. Therefore, considerable research has been done in the field of spell checking [1], [5], [7]. Error detection is the first task in spell checking that helps in detecting non-word and real-word errors. The correction tasks are comprised of two subtasks that are candidate words generation and their sorting in decreasing order of relevance. Edit distance and its variants can be used for both operations. In the case of having limited language vocabulary, morphological operations with linguistic rules help in generating out of corpus candidate words. For the subtask of sorting candidate words, edit distance is coupled with n-gram frequencies to resolve ties. An important aspect of spell checking is the use of efficient data structures that ensure minimum use of memory and high retrieval efficiency.

By and large, language models are used as simple dictionaries to deal with error detection. Unigrams help in detecting

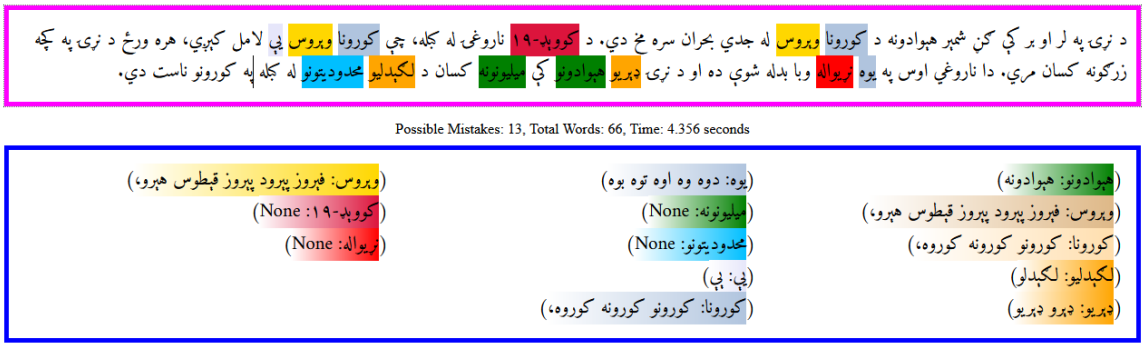

Fig. 1: Pashto spell checker interface that has identified the possible mistakes in the given text and has suggested their possible corrections.

non-word errors while bigrams, trigrams, and 4-grams are used in detecting real-word errors. Finite-state graphs are used for error detection with morphological operations as permitted by the rules of the language. This is a very powerful approach as it allows to facilitate words that are not present in the dictionary or the accompanying corpus. With this approach, both non-word and real-word errors are detected without requiring a complete listing of all the words [1], [8]. Generally, a spell-checking model relies on two resources that are a comprehensive dictionary and a standard corpus. The dictionary is crucial in detecting non-word errors because when the dictionary is comprehensive, any word that does not exist in the dictionary can be discarded as an error word. On the other hand, the real word errors are the words mistakenly used in a context. Large datasets are used to help in finding them through higher-order n-grams [7]. If a word is not used in a context more often in the corpus as indicated by its frequency lower than the acceptable threshold then the word is indicated as a real word error. Considering the diversity of a language, it is possible that a word may not be used or is rarely used in a context in the available corpus. This will lead to producing many false negatives. Non-word errors are generally decided based on lookup in the hash table. A lookup table or language vocabulary that is not comprehensive will also result in having many false negatives. Low resources languages mainly suffer from these problems due to lack of such resources. On the other hand, incorporating additional methods for compensating the problem of having many false negatives, false positives are introduced.

A common mistake in spelling is wrongly using its vowels. Therefore, a hash function that treats all vowels as the same letter will result in the higher similarity between the words having differences in their vowels. The word that has common consonants will have more chances of being suggested as possible corrections than otherwise [9]. An efficient solution for this case is provided by normalizing all the words by dropping their vowels. It allows calculating distances among words with respect to their consonants only. The normalized results are arranged through a typical hash function. The buckets in the hash are expected to have more collisions due to the dropping of their vowels, however, the collisions are precomputed. A misspelled word is mapped to a bucket where its top n words are suggested as possible corrections. The suggested corrections are organized with edit distance. In case of a tie in edit distance, the suggested words are arranged in decreasing order of their frequency. Generally, edit distance after two is not explored.

A good response time is another concern for the spell-checking models. It is often not feasible to detect errors and generate possible corrections on the fly due to bad response time. The response time is improved by training the model offline with possible mistakes that a user may make. Using Levenshtein edit distance, all possible candidate words of a word are computed with edit distances one and two. The edit distance with delete operations only is 3 times faster and has reduced memory consumption while retaining the same accuracy [6]. Bk-trees are also used for searching and retrieving related candidate words, however, it is computationally more expensive [10]. Tries have comparable search performance but by being a prefix tree it requires a common prefix which makes it more suitable for auto-complete and search suggestions but not for spell checking [11]. Minimal perfect hash function compactly stores n-grams with full frequency counts consuming just over two bytes per n-gram and is efficient in retrieval [12]. In [13], the authors deal with spelling errors in tagging by constructing co-occurrence graphs, where two tags are connected by an edge if they have some non-zero probability of occurring together. The co-occurrence graph is constructed having tags as nodes and their edges with nonzero co-occurrence probabilities as their weights [8]. Higher-order n-grams like trigrams and 4-grams are used as contextual information in detecting and correcting real word errors [7]. The n-gram probabilities are also precomputed and stored in hash tables for efficient retrieval. Generating the errors and computing their relevances is part of the model training which results in a high reduction of computational cost for test cases. However, the memory consumption is increased by manifolds. Weeding the redundant error cases, generating only the highly likely errors, and merging the similar error forms of different words can help in reducing memory consumption. Considering the existing research for other languages, a particular challenge for the Pashto language is to deal with an incomplete language vocabulary and smaller dataset.

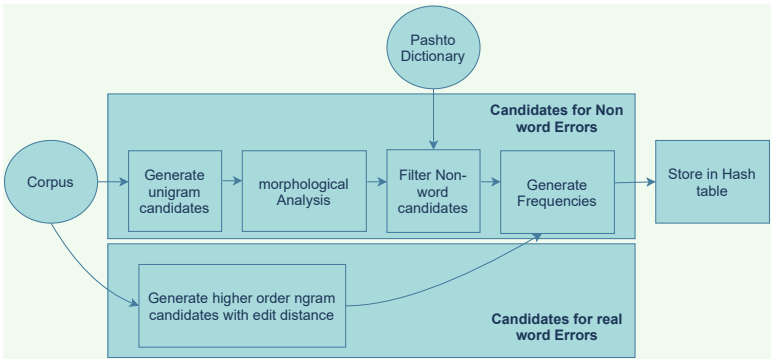

Fig. 2: Block diagram of model training to store possible corrections in a hash for improved efficiency.

III. Proposed Work

We propose a spell-checking model for the Pashto language that utilizes a small dataset to learn about the types of spelling errors discussed earlier, identify them and suggest a correction for them. The proposed model undergoes a training phase to pre-compute all candidate words and their frequencies which results in improving the computational efficiency at the expense of memory consumption. There are two resources available for conducting this research which is the Pashto corpus and a small word synonym dictionary. Both resources are not comprehensive and are, therefore, supported with mechanisms for generating newer words. Figure 2 provides a workflow for training the model. A separate set of tasks are followed for dealing with non-word and real-word errors which provide candidates for both non-word errors and real-word errors in their respective hash tables. When a word in the input text does not appear in the non-word hash, it is considered a non-word error. Whereas, when a higher-order n-gram is not found in the real word candidates hash, a real word error is highlighted. The correction mechanism activates with the detection of an error that estimates the closest corrections from the candidate list and provides them in decreasing order of their relevance.

Unigrams are generated from the available corpus that is going to serve as the basis for the detection and correction of non-word errors. An issue faced is the limited number of unique words due to smaller corpus which results in a large number of false negatives through word lookup. In other words, many words are considered non-word errors because they were not part of the unigram word list generated from the corpus. To compensate for the insufficient candidate words, morphological operations are performed to generate out of dictionary words. These generated words are also added to the list of candidate unigrams. To avoid, false positives i.e., having non-words considered as correct words, the candidate unigrams undergo a filtering process. The first filter is applied with the help of a word synonym dictionary consisting of around two thousand words. The candidate unigrams having a mention in the dictionary are considered acceptable. The second filter is applied through linear interpolation of the n-gram scores of the candidate word at the character level. So, the words that are not part of the two resources but its sequence of characters in parts is more often repeated in other words, chances are that this is a correct word. A threshold is applied to decide only the high scoring out of dictionary words as candidate unigrams. Their frequencies are calculated and are stored in the hash table. Real word errors are detected through higher-order n-grams i.e., bigrams, trigrams, and 4-grams. They are generated from the corpus and are added to the candidate list for real-word errors. Since a word could be used in many different contexts but we only have access to limited data therefore, newer contexts of a word are generated using edit distance with substitution operation only [14]. It helps to generate out of dictionary higher-order n-grams that are added to the list of candidate words as well, therefore, supporting the use of words in newer related contexts. A filter is applied using a threshold on linear interpolation scores of the higher-order n-grams at the word level. Frequencies are calculated for the selected n-grams and are added to the real word hash table.

The training module of the proposed approach is elaborated in of Algorithm 1. It consumes Pashto journal articles and a dictionary of words and their synonyms as input. The algorithm outputs hash tables of words and frequencies for non-word and real-word errors. Lines 1 - 14 maintain a list of corrections for non-word errors. In line 1, the corpus is tokenized into a list of words. The loop from line 2 to 4 expands on the available list of words generates new words using morphological operations and adds them to the list of candidate words. The loop from lines 5 - 12 applies the filtering mechanism on the newly generated words. Linear interpolation scores of out-of-vocabulary words are computed at character level on line 6. For an out-of-vocabulary word to stay in the list, it may either exist in the $pDict$ or has a linear interpolation score above the given threshold t_n, lines 7 - 9. Linear interpolation is calculated as,

$$\hat{p}(w_i, w_{i-1}, w_{i-2}) = \lambda_1 \times p(w_i|w_{i-1}, w_{i-2}) + \lambda_2 \times p(w_i|w_{i-1}) + \lambda_3 \times p(w_i) \quad (1)$$

where,

$$\sum_i \lambda_i = 1. \quad (2)$$

It is the weighted mix of n-gram probabilities that result in a better estimation of the unseen word. It is applied at the

Algorithm 1 Building Pashto Spell Checker Model

Input: corpus, pDict
Output: non-word candidates hash, real word candidates hash

1: Tokenize corpus into unique words
2: **for** word word: words **do**
3: new words = morphologicalforms(word)
4: **end for**
5: **for** new word nw: new words **do**
6: Calculate lis(nw) score at the char level, using Eq.(1)
7: **if** nw $\in$ pDict or lis(nw) $\geq t_n$ **then**
8: Retain nw
9: **else**
10: Remove nw from new words
11: **end if**
12: **end for**
13: non-word candidates = words $\cup$ new words
14: Calculate frequencies of non-word candidates and add to hash
15: Generate higher order ngrams from corpus
16: **for** ngrams hon: higher order ngrams **do**
17: new hons = GenerateHigherOrderNgrams(hon) using edit distance with substitution only
18: **end for**
19: **for** new hon nhon: new hons **do**
20: Calculate lis(nhon) score at word level, using Eq.(1)
21: **if** lis(nhon) $\geq t_r$ **then**
22: Retain nhon
23: **else**
24: Remove nhon from real word candidates
25: **end if**
26: **end for**
27: real word candidates = hon $\cup$ new hon
28: Calculate frequencies of real word candidates and add to hash

character level to capture the structure of characters within a word which are expected to have repeated syllables. Thus, favoring unknown words that have a sequence of characters more often repeated in other words. The probability of n-grams is calculated using add-k smoothing using Eq.(3) which is widely considered as the most effective method of smoothing [15]. It uses absolute discounting by subtracting a fixed δ value from the probability's lower order terms to drop n-grams with lower frequencies as,

$$P^*_{Add-k}(w_i|w_{i-1}) = \frac{C(w_{i-1}w_i) + k}{C(w_{i-1}) + kV} \tag{3}$$

A word that fails both the conditions is filtered off on line 10. Lines 13 and 14 merge the words from the corpus with the new words that are retained, calculate their frequencies and add them to the non-word hash. The second half of the algorithm from lines 15 - 28 maintains a list of corrections for the real-word errors. In line 15, bigrams, trigrams, and 4-grams are generated as higher-order n-grams from the corpus. The loop on lines 16 - 18 generates newer higher-order n-grams from each of the hon as nhons using edit distance with substitution operation only. It samples words from the corpus and substitutes them with a word in the higher-order n-grams. The loop from lines 19 - 26, iterates through all the newly generated higher-order n-grams and filters them based on their linear interpolation score at the word level, calculated in line 24. A condition is applied on it in lines 25 - 27 against a threshold t_r to retain nhons given in line 22 otherwise removed

Algorithm 2 Spell Check Analysis

Input: Input text, non-word hash, real word hash
Output: Misspelled words, sorted list of corrections

1: Tokenize input text string as words
2: **for** word tw : words **do**
3: **if** tw $\notin$ non-word hash **then**
4: Indicate as non-word error
5: corrections = edit distance(tw, non-word hash)
6: sort(corrections, by edit distance and frequency)
7: Provide top n corrections in descending order
8: **end if**
9: **if** hongram(tw) $\notin$ real word hash **then**
10: Indicate as real word error
11: corrections = edit distance(tw, real word hash)
12: sort(corrections, by edit distance and frequency)
13: Provide top n corrections in descending order
14: **end if**
15: **end for**

on line 24. Lines 27 and 28 combine the two lists of higher-order n-grams, calculate their frequencies and store them in real word hash. Frequencies of n-grams that exist in the corpus are calculated directly, whereas for out of dictionary n-grams it is the average of frequencies of the closest n-grams having an edit distance of one.

Algorithm 2 provides the working mechanism of the proposed approach for processing input text. It consults the hash tables generated in the training phase to detect error words and provide a list of corrected words sorted in decreasing order of their relevance. The precomputed candidate n-grams and their frequencies result in improving the response time manifolds as a trade-off for some extra memory. When an input text is provided, the spelling error detection mechanism activates. In line 1, it tokenizes the input text into words. In lines 2 - 15, it iterates through all the words and detects a possible error. Lines 3 - 8 are responsible for detecting and correcting non-word errors while lines 9 - 14 deals with detecting real-word errors. A word is considered a non-word error or a real word error if it does not exist in the non-word hash or real word hash, respectively as verified in lines 3 and 9. If a match is not found a non-word or real word error is indicated as shown in lines 4 and 10, respectively. Similarly, corrections are identified through edit distance from their respective hash tables as given in lines 5 and 11. The list of candidate corrections for either error is sorted through edit distance where a match in edit distance is resolved through frequencies, lines 6 and 12. Finally lines 7 and 13 provide the top n corrections for each type of error. The proposed approach detects and corrects both types of errors and is particularly effective in identifying and correcting out-of-vocabulary words. The balance between the computational cost and memory consumption can be adjusted according to the need of the application.

IV. Results and Discussion

We have compiled a dataset of 15 articles from Pashto journal and BBC Pashto. It is a very small dataset considering the nature of the problem we are dealing with. However,

TABLE II: N-grams and their frequencies

N-gram	Total	f_1	f_2	f_5	f_{10}
Unigrams	87,253	47362	23,553	7,167	9,171
Bigrams	1,104,314	436,362	80,034	14,573	9,104
Trigrams	1,104,313	791,115	67,248	7,583	3,785
4-grams	1,016,696	974,216	38,560	2,762	1,057

since a comprehensive list of words for the Pashto language is not available, therefore, the assumption of expecting out-of-vocabulary words holds. It has a total of 87,253 unique words with only a few words having high frequencies while the majority of the words have low frequencies following power-law distribution. The count of n-grams above certain frequencies is presented in Table II. The first column has the total number of n-grams. The second column f_1 has n-grams with exactly one occurrence. The column f_2 is the n-grams with frequency from 2 to 4, f_5 has n-grams with frequencies from 5 to 9 while f_{10} has n-grams with frequencies above 10. The second resource consists of a word synonym dictionary that has a total of two thousand words. The dictionary is also utilized as an important resource, but represents only a small fraction of the total words. Since this is the first attempt to build a spell checker for Pashto therefore, no public dataset is available to evaluate our approach.

TABLE III: Results of non-words spelling errors detection

Freq.	Total	Detected	Not Detected	Accuracy
f_0	100	50	50	50.0%
f_1	180	162	18	90.0%
f_2	71	68	3	95.6%
f_5	23	21	2	91.3%
f_{10}	11	9	2	81.8%
f_{20}	15	12	3	80.0%
Total	400	322	78	80.5%

TABLE IV: Results of non-words spelling errors correction

Freq.	Total	Corrected	Not Corrected	Accuracy
f_0	50	8	42	16.0%
f_1	95	89	6	93.7%
f_2	32	31	1	96.9%
f_5	14	14	0	100%
f_{10}	5	5	0	100%
f_{20}	4	4	0	100%
Total	200	151	49	75.5%

A test dataset is compiled consisting of 800 words. The 400 instances are for the detection of non-word errors where 200 words are correct while the other 200 have non-word errors. The correct words have 25% of out-of-vocabulary words to make sure that the model does not distort out-of-vocabulary correct words. The incorrect words having non-word errors are generated by randomizing characters in the vocabulary words. Again 25% of the non-word errors are passed through the randomization process twice and are considered hard cases. The later 400 test words are higher-order n-grams compiled for testing the detection and correction of real-word errors. It has 200 correct real words while the other 200 words have real-word errors. To generate real-word errors the randomization is applied at the word level on the higher-order n-grams.

TABLE V: Results of real-word spelling errors detection

Freq.	Total	Detected	Not Detected	Accuracy
f_0	100	52	48	51%
f_1	219	219	0	100%
f_2	42	42	0	100%
f_5	15	15	0	100%
f_{10}	5	5	0	100%
f_{20}	19	19	0	100%
Total	400	362	48	90.5%

TABLE VI: Results of real-word spelling errors correction

Freq.	Total	Corrected	Not Corrected	Accuracy
f_1	141	101	40	71.6%
f_2	37	29	8	78.4%
f_5	5	5	0	100%
f_{10}	7	6	1	85.7%
f_{20}	10	8	2	90%
Total	200	149	51	74.5%

The word frequencies play an important role in the detection and correction process. The words that are more frequently have higher chances of their errors being detected and corrected. Therefore, the out-of-vocabulary spelling mistakes has the least chance of being corrected. In fact, the out-of-vocabulary correct words also has a higher chance of being distorted. Therefore, the words detected and corrected are presented along with their frequencies to highlight this point. Tables III and IV have the results of detection and correction of non-word errors using the first 400 test cases. The row f_0 represents the out-of-vocabulary words that have zero frequency in the training corpus, while the row f_{20} is the number of words having a frequency above 20. The model succeeds in correcting detecting half of the out-of-vocabulary errors and could fix the only 16% of them. This attributes to the generation of out-of-vocabulary words. It can be observed that the detection and correction accuracy of the model improves as the correct form of the error word get more frequent. There are only a few exceptions to this statement as a few of the frequent words could not be detected.

Tables V and VI highlight the results of detection and correction of real-word errors. It correctly detects 51% of the real word errors which goes to 100% for words that are found in the corpus. The total detection accuracy of the model for real-word errors is 90.5%. The correction accuracy improves as the context of an error word is more frequent. Its correction accuracy is lowest when the context of a word has a frequency of one. Its values go up to 100% for words that have context frequency between 5 and 9. There are few mistakes in the frequent n-grams that are due to polysemy i.e., these context words are frequent with other words as well. The total correction accuracy of real-word errors is 74.5%.

Figure 3 shows the detection accuracy of the model (Figure 3 Left) and correction accuracy of the model (Figure 3

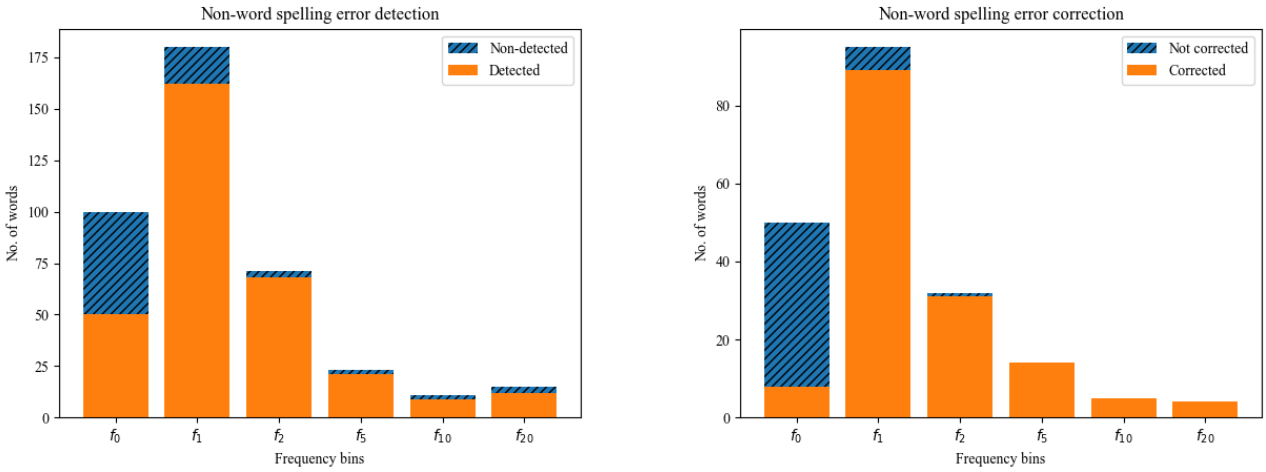

Fig. 3: Non-word error detection and correction across frequency bins

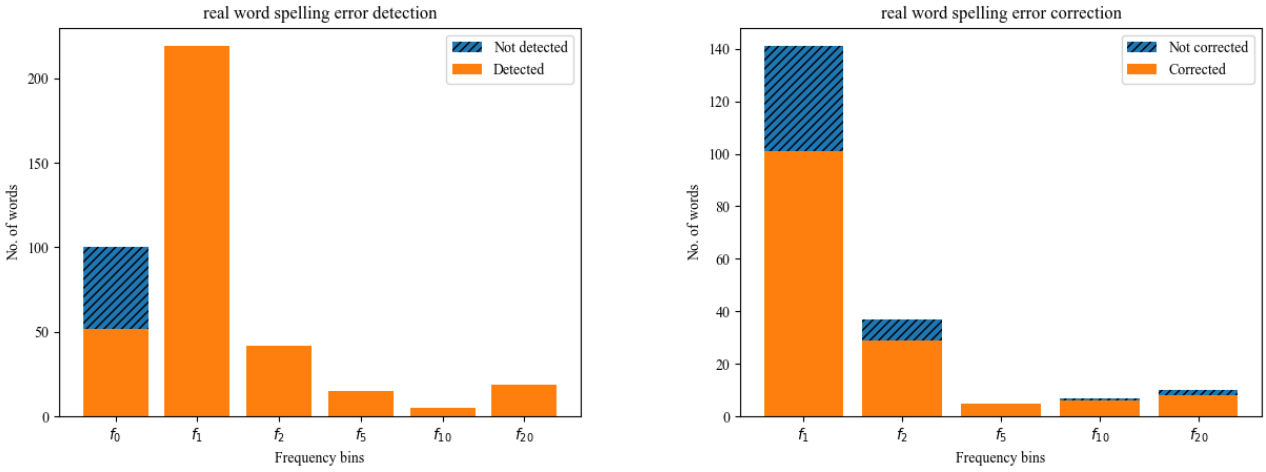

Fig. 4: Real word error detection and correction across frequency bins

Right) for non-word errors across frequency bins. Frequent words and their higher-order n-grams play an important role in detecting non-word errors. The accuracy drops with the frequency of the correct form of the non-word error. In Figure 4 the real word error detection (Figure 4 Left) and real word error corrections (Figure 4 Right) are plotted against the frequency bins of the words having spelling errors. A similar trend could be observed where the error detection and correction rate drops considerably as the frequency of a word decreases.

The error correction is based on the criteria that the correct form of the word exists in the corpus or newly generated words and has a linear interpolation score above the threshold. Our proposed approach has the capability of detecting and correcting out-of-vocabulary words and therefore, out-of-vocabulary words are also added into the test data to evaluate the model's performance. It correctly detects 50% of the out-of-vocabulary spelling errors and corrected 16% of them. This shows the use of morphological operations for generating newer words as a promising direction for detecting and correcting out-of-vocabulary words. The use of morphological operations incurs an extra computational cost that is linear to the number of existing words in the vocabulary. The memory consumption also increases by 15 - 20%. However, it is very effective in detecting and correcting out-of-vocabulary words that are similar to the words in the corpus in terms of their syllable structures. The detection and correction accuracy of known words against both non-word and real-word errors is also encouraging.

V. Conclusion

Spell checking models of low-resourced languages face a common problem of to limited vocabulary and noisy datasets. To compensate for the incomplete Pashto vocabulary, morphological operations are performed on words to generate newer out-of-vocabulary words. The proposed approach shows positive results in detecting and correcting out-of-vocabulary words while achieving very high accuracy for known words. The use of linear interpolation at the character level helped in detecting and correcting non-word errors. It also has few

exceptions of correct words whose character sequence is not very common resulting in false negatives. Edit distance with substitution allowed to generate newer contexts of existing words in higher-order n-grams. However, despite the filtering process, it introduces some noise word sequences. The proposed approach has achieved detection accuracy of 85.5% and correction accuracy of 75%. In future, it can be extended to to apply morphological operations within the word stems as well. The issue of polysemy for real-word errors also need additional context consideration.

REFERENCES

[1] A. Sharma and P. Jain, "Hindi spell checker," *Indian Institute of Technology Kanpur*, 2013.

[2] F. Bakanov, "Spelling correction: How to make an accurate and fast corrector. https://medium.com/@fbakanov,"

[3] P. Norvig, "How to write a spelling corrector. http://norvig.com/spell-correct.html,"

[4] L. Yujian and L. Bo, "A normalized levenshtein distance metric," *IEEE transactions on pattern analysis and machine intelligence*, vol. 29, no. 6, pp. 1091–1095, 2007.

[5] C. Zhao and S. Sahni, "String correction using the damerau-levenshtein distance," *BMC bioinformatics*, vol. 20, no. 11, pp. 1–28, 2019.

[6] W. Garbe, "1000x faster spelling correction algorithm (2012) https://wolfgarbe.medium.com/1000x-faster-spelling-correction-algorithm-2012-8701fcd87a5f,"

[7] S. Verberne, "Context-sensitive spell checking based on word trigram probabilities," *Unpublished master's thesis, University of Nijmegen*, 2002.

[8] T. Pirinen, K. Lindén, *et al.*, "Finite-state spell-checking with weighted language and error models: Building and evaluating spell-checkers with wikipedia as corpus," in *Proceedings of LREC 2010 Workshop on creation and use of basic lexical resources for less-resourced languages*, 2010.

[9] D. Hládek, J. Staš, and M. Pleva, "Survey of automatic spelling correction," *Electronics*, vol. 9, no. 10, p. 1670, 2020.

[10] G. Wu, *Approaches to Compound Splitting in German Spoken Term Detection*. PhD thesis, National Research Center, 2012.

[11] H. Shang and T. Merrettal, "Tries for approximate string matching," *IEEE Transactions on Knowledge and Data Engineering*, vol. 8, no. 4, pp. 540–547, 1996.

[12] D. Guthrie, M. Hepple, and W. Liu, "Efficient minimal perfect hash language models.," in *LREC*, Citeseer, 2010.

[13] F. Bonchi, O. Frieder, F. M. Nardini, F. Silvestri, and H. Vahabi, "Interactive and context-aware tag spell check and correction," in *Proceedings of the 21st ACM international conference on Information and knowledge management*, pp. 1869–1873, 2012.

[14] S. Murugan, T. A. Bakthavatchalam, and M. Sankarasubbu, "Symspell and lstm based spell-checkers for tamil," 2020.

[15] V. Siivola, T. Hirsimaki, and S. Virpioja, "On growing and pruning kneser–ney smoothed n-gram models," *IEEE Transactions on Audio, Speech, and Language Processing*, vol. 15, no. 5, pp. 1617–1624, 2007.