Comparison of Spelling Correction Approaches

Mateusz Wojciechowski

October 2024

Contents

1 Data Preparation

1.1 Single Word Data

As described in the README file I decided to used birkbeck dataset. To prepare data for further analysis I converted it to a .csv format with two columns: correct_spelling and wrong_spelling. There are repetitions in the first of those columns as many words were spelled incorrectly in many ways. This stage of preprocessing is implemented in the *generate_csv.py* file.

1.1.1 Splitting misspellings into categories

I decided that in order to properly analyse strengths and weakness of different approaches it is vital to divide this dataset into a few categories. After assigning a set of misspellings to one category I took them away from the main dataset and proceeded by checking if they belong to the following category.

First of all I obtained all **context based mistakes** (mistakes which were simply a correct english word used in place of other word). I had to separate those mistakes from all the other ones as the tools wouldn't be able to correct them (because without context they are indeed correct). To do so I used NLTK word corpus. The next category I created were **transposition mistakes**. Those were simple typing mistakes which often occur when writing on a computer, I distinguished them by checking if the Levenshtein distance between correctly and incorrectly spelled word was equal to 2 and if both sorted stings were equal. Another category I decided to create was **small typo mistakes**. They are usually a result of quick inaccurate typing, to classify them I checked if the Levenstein distance was equal or less than 3. I also thought it would be interesting to validate whether the remaining misspellings are **phonetic mistakes**. People make typing mistakes because they write the word how they hear it, and that sometimes results in those type of mistakes. Lastly all the other mistakes were classified as **other type** of spelling mistakes. The process of categorizing data is implemented in the *categorize_data.py*. I also converted the whole dataset to lowercase with *convert_to_lowercase.py* script.

1.2 Text Data

For text data I used a medium articles dataset from medium available on Kaggle.

The preparation of this dataset was much simpler and quicker. I dropped unnecessary columns, renamed the remaining ones and applied synthetic text augmentation. Then I used a simple self-implemented algorithm to make changes in the

text. Lastly I saved the dataset into a .csv file. The whole process is implemented in *synthetically_augment_articles.py*.

2 Generating corrections

2.1 Single word correction

To generate corrections I applied the tools I have chosen on each subset of the dataset and obtained results in form of .csv files with columns corresponding to corrections performed by each approach. Script used for this task: *generate_corrections.py*

2.2 Text correction

To generate corrections I once again used the same tools as before. I obtained the corrections by running the *generate_text_corrections.py* script.

3 Results

The results for both single word and text correction are presented in the *analyse_results.ipynb* notebook

3.1 Single Word Correction Results

When we look at the values of all the metrics we will automaticaly notice that the results differ depending on which sub-dataset we are analysing. In case of **transposition errors** we can see that all models did similarly, both BERTChecker and SymSpell seem to have achieved great results, slightly surpassing LSTMCNN checker. This might be considered the easiest type of error as all the letters required to build a correct word are already in the misspelled word.

The more complex type of misspellings we analyse the more it is noticeable that different tools do indeed perform with different precision. In case of **small typo mistakes** SymSpell is performing worse than the other two neural network based models, the differences become even more visible when we look at the most complex types of errors: **phonetic and all the other mistakes**. The accuracy of SymSpell is in those cases close to 0 as well as cosine similarity. The performance differences also start to show between BERTChecker and LSTMCNN checker, it seems as BERT is clearly the strongest of tools for misspelling correction as it achieves high accuracy

and cosine similarity (also great values of Levenshtein and Jaro-Winckler) when correcting various types of errors

3.1.1 Why the results look as they do?

When we examine the metric values across sub-datasets, it becomes evident that each spelling correction model (BERT, LSTMCNN, and SymSpell) has distinct strengths and weaknesses, which vary depending on the type of misspelling being corrected.

SymSpell, a dictionary-based algorithm, excels with simpler errors, such as transposition errors where the correct letters are already present but misplaced. This type of error aligns well with SymSpell's approach, as the algorithm efficiently searches for minor adjustments in letter positions to find dictionary matches.

SymSpell performs poorly with more complex spelling errors, such as phonetic errors and uncommon typos, where the correct letters are missing or significantly altered. These types of errors require a deeper understanding of language patterns, which SymSpell lacks. As a result, its accuracy, cosine similarity, and other similarity metrics drop significantly for these types of errors.

LSTMCNN combines Convolutional Neural Networks (CNN) and Long Short-Term Memory (LSTM) networks, allowing it to capture sequence patterns in spelling. This enables it to handle more complex errors, such as small typos and some phonetic mistakes, with better accuracy than SymSpell. It performs well on a range of error types but shows limitations in more nuanced errors where semantic understanding is required.

Despite its neural network architecture, LSTMCNN lacks the advanced pattern recognition capabilities that BERT offers, particularly with very complex errors. In cases of phonetically complex or rare misspellings, BERT tends to outperform LSTMCNN in terms of accuracy, cosine similarity, and Levenshtein distance, as BERT's architecture allows it to make more nuanced corrections based on letter patterns.

BERT's Transformer-based architecture provides it with a highly advanced capacity for recognizing letter and sub-word patterns, allowing it to handle even highly complex misspellings, including phonetic errors and rare spelling mistakes. BERT Checker generally outperforms the other tools on complex errors, as reflected in metrics such as accuracy, cosine similarity, Levenshtein distance, and Jaro-Winkler similarity. Its ability to interpret intricate letter sequences and subword structures makes it especially robust for advanced misspellings.

3.2 Text correction results

The results came to me as a surprise, this time BERT Checker was clearly the worst performing tool achieving weak values of both Jaccard and Levenshtein metrics. LSTMCNN checker achieved the best results and SymSpell took the second place. We have to note there that the mistakes i applied using my own way of augmentation were quite simple (mostly transpositions and small typos).

3.2.1 Why the results look as they do?

In my opinion such results might have occurred because both LSTMCNN and SymSpell are better at focusing on specific words, on the other hand BERT based on the Transformers architecture uses among others the attention mechanism which is strongly compromised because of frequent misspellings. BERT is not able to understand context as it usually does and makes errors while trying to repair the text.

4 How those tools can be improved?

LSTMCNN - to enhance LSTMCNN's handling of complex errors, especially phonetic or unfamiliar typos, the model could be pre-trained on a larger dataset with a broader variety of misspelling patterns. Adding a mechanism to process sentences as a whole, in addition to isolated words, could improve consistency in text correction and prevent errors that arise from word-by-word isolation.

SymSpell - enhancing SymSpell with a phonetically-aware dictionary or adding common misspelling patterns could improve its accuracy on phonetic and typographic errors, broadening its applicability to more error types.

BERT - could be further improved for text correction by adding a character-level noise robustness layer, allowing it to better handle noisy text and minor typos that affect tokenization.

5 Challenges encountered

The first challenge I encountered was finding suitable datasets and coming up with ways how to use them to obtain interesting results. I also have spent a lot of time researching correction tools and metrics that would be able to present their strengths and weaknesses. Unfortunately I had to put a lot of effort into getting Neuspell library to work as the available version was lacking some key elements, I went through a lot of forums and tried many solutions before it worked. In general i feel like I have

learned a lot while doing this project and I hope its results and approaches I used will prove to be interesting to a person who will be reading about my work.