IRSTI 16.31.21

A. Tolegenova
Suleyman Demirel University, Kaskelen, Kazakhstan

AUTOMATIC ERROR CORRECTION: EVALUATING PERFORMANCE OF SPELL CHECKER TOOLS

Abstract. Spell checking is the task of detecting and correcting spelling errors in text and is one of the most sought-after processes in NLP. There are many open-source toolkits for checking and correcting errors in the text. To test how effective these tools are, in this article I have presented an evaluation of three types of tools as NeuSpell, SymSpell and Hunspell. SymSpell showed a high speed of 2480, this is an indicator of how fast it works than others. And NeuSpell achieved the lowest error rate of 0.80%. The results show the disadvantages and advantages of all algorithms, and that there is still room for improvement.

Keywords: NLP, open-source tools, spell checking, detect, correct.

***

Аңдатпа. Емлені тексеру мәтіндегі емле қателерін анықтау және түзету міндеті болып табылады және NLP-те сұранысқа ие процестердің бірі болып табылады. Мәтіндегі қателерді тексеруге және түзетуге арналған көптеген ашық бастапқы құралдар бар. Бұл құралдардың қаншалықты тиімді екенін тексеру үшін осы мақалада мен NeuSpell, SymSpell және Hunspell сияқты құралдардың үш түрін бағалауды ұсындым. SymSpell 2480 жоғары жылдамдықты көрсетті, бұл оның басқаларға қарағанда қаншалықты жылдам жұмыс істейтінінің көрсеткіші. Ал NeuSpell 0,80% қателіктердің ең төменгі деңгейіне қол жеткізді. Нәтижелер барлық алгоритмдердің кемшіліктері мен артықшылықтарын көрсетеді және әлі де жақсартуға мүмкіндік бар.

Түйін сөздер: NLP, ашық бастапқы құралдар, емлені тексеру, анықтау, түзету.

***

Аннотация. Проверка орфографии является задачей обнаружения и исправления орфографических ошибок в тексте и является один из востребованных процессов в НЛП. Для проверки и исправление ошибок в тексте есть множество инструментов с открытом исходным кодом. Чтобы проверить насколько эффективны эти инструменты, в этой статье я представила оценку трех типов инструментов, такие как NeuSpell, SymSpell и Hunspell. SymSpell показал высокий скорость 2480, это

показатель того насколько быстро оно работает чем другие. А NeuSpell добился самой низкий уровень в ошибках 0.80%. Результаты показывает недостатки и преимущества всех алгоритмов, и что есть еще пути для улучшения.

Ключевые слова: NLP, инструменты с открытым исходным кодом, проверка орфографии, обнаружение, исправление.

I. Introduction

Natural Language Processing (NLP) is a subset of AI that focuses on how natural language is processed. NLP uses machine learning algorithms to analyze text and speech. Automatic spelling correction is one of the important problems that many solve in NLP. This article walks through the algorithms and approaches that can be used to understand automatic spell checking, finding the errors, and correcting them. There are many software applications and tools here that have achieved high accuracy of spelling correction and are also used in many word processors and search engine applications. And the spell checker itself is mainly used as a major pre-processing step in NLP to clean up text data before it is passed to machine learning algorithms for classification.

Thus, for people who write and print text in different languages, spell checking in the process has become one of the most sought-after and indispensable. And also when writing search queries, many prints appear. And the lack of a mechanism can lead to incorrect searches since the search server displays information from the database using key phrases. The main aspects of spell checking are first detecting errors and then correcting those errors.

Before actually implementing any approach to working with building a spelling correction system, knowledge of spelling errors is required. There are many works on errors and their definitions. For example, in the book Wagner shows about four types of errors: errors of agreement, errors of superfluous words, errors or missing words, and spelling errors of real words. [1] And there are also Mitton's works, in which he analyzed the types of spelling errors, and this work describes approaches to building an automatic spelling correction system. [2] Spelling errors can be divided into different classes and categories, and the main ones are realword errors and non-word errors. This can be seen in the works of Pirinen [3], Kukich, [4] Tutanov and Moore [5].

Typographic errors (Non-Word Errors):

These errors occur when the correct spelling of a word is known, but the word is entered by mistake. These errors are keyboard-related and therefore do not meet certain linguistic criteria. Damerau[1] says that over 60% of spelling errors fall into one of the following four categories, such as Single letter insertion, Single-letter deletion, Single-letter substitution, and Transposition of two adjacent letters. Cognitive errors (Real Word Errors):

These errors are caused by the disability of the person who writes the text, i.e. the correct way of writing may not be known to the user. For example, a user

writing text may simply be learning a new language and not know the correct spelling. This set of errors is language and user-specific as it is more dependent on the use of language rules.

This work is a research work on the evaluation of development tools, namely a set of open source tools for automatically checking typed texts and their algorithms. The purpose of the research work is to study the already existing solutions that were created for spell checkers, to describe the strengths and weaknesses of these solutions. And also as an example, three currently used and widespread approaches to solving the problem of text identification and classification are considered. The first example is a model working with neural networks. A second model is a tool that works with the help of machine learning methods, namely using the n-gram model. The third is the models that were developed in java, namely hunspell and language tool since Java and C ++ are considered one of the most popular languages in product development.

The main idea is that by calculating the existences of all possible tools, you can analyze the algorithms that are used in these tools. Testing is carried out using data types from texts that were used in the work on spell-checking.

2. Methodology

2.1. NeuSpell

One methodology that was tested was NeuSpell, which is based on a neural network and is a spell-checking tool written in the Python language. Two of them are commercially available non-neural models, namely GNU Aspell and JamSpell. The following four models appear to be published neural models for spelling correction: SC-LSTM, CHAR-LSTM-LSTM, CHARCNN-LSTM, and BERT. And the remaining four are updates that have been added to existing ones for improvement. This is ELMo and BERT combined with a bi-LSTM semi-character model on input and bi-LSTM on output. That is, SC-LSTM has been enhanced with deep contextual representations from pre-trained ELMo/BERT to capture context around a token and add them to semi-character attachments. Here, Long Short Term Memory (LSTM) is a kind of neural network that is capable of learning algorithms for long-term dependencies. The LSTM module consists of cells and a gateway that learns, disables, or stores information from each of the modules. In NeuSpell, these neural networks are trained and, correcting errors, marks them in sequence. [6]

In Fig 1 we can see several text denoising strategies. They use a lookup table for this, and for character-level noise, they use a context-based character-level confusion dictionary. After running the "typo - correction" function from various database sources to fill in the table.

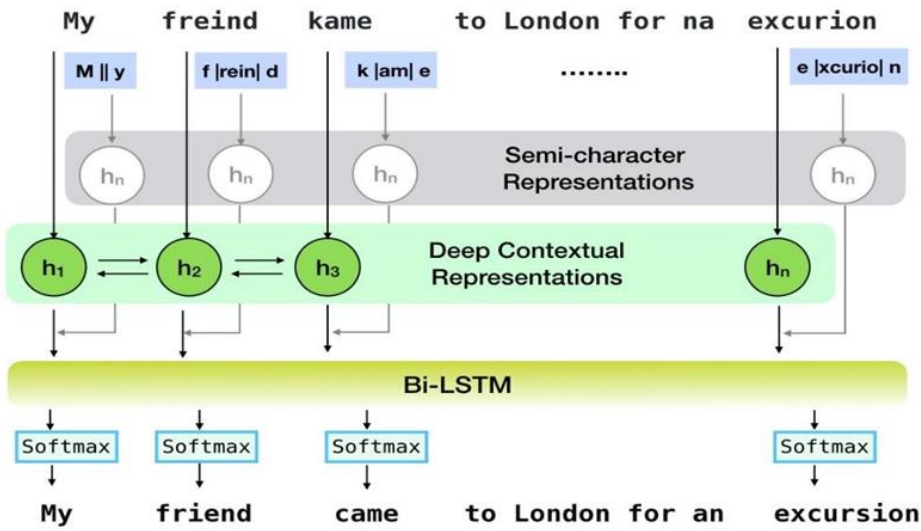

Fig 1. Structure of work Neuspell

2.2. SymSpell

The second model is the Symmetric Speech Correction Algorithm (SymSpell). Ye Kyaw

Thu show using studied the application of the algorithm SymSpell. [7] Wolf Garbe proposed SymSpell algorithms for implementation in C#. [8] Firstly, to detect errors in a word, the language model of N-grams is used to check the dictionary. SymSpell works according to the following principle: it generates words with a modification of the edit distance algorithm, namely, only a delete operation, no transpositions, replacements, or insertions, and then connects them to the original element. SymSpell's speed depends on the Symmetric Delete spelling algorithm, and memory requirements are controlled by prefix indexing. For example, using the Levenshtein algorithm, to generate words with 5 letters, we have to try approximately 3-4 million spelling errors within the edit distance. And because SymSpell only uses deletion, there will be 25 deletion options to search for. And also this tool uses a hash table for quick access to the index during the search. For example, we can take the words "test". When typing on the keyboard, we made a mistake in one letter, that is, instead of "test" we wrote "temt". In the hash table for the word "test" the words are stored by index: eat, tst, tet, tes. And for deletion in the word «temt", it will be "emt, tmt, tet, emt". We can find a similar "tet" here and the algorithm will change the word "temt" to "test".

2.3. HunSpell

The last tool that was tested was Hunspell implemented using the Java programming language. Here it works according to the following principles: [9]

- Parse the document by tokenizing, that is, extracting the words that will be checked

- Analyze each word by breaking it down into root and affix conjugations.
- Dictionary lookup if the combination of word and affix is valid for the language
- Suggest corrections by looking up similar or correct words in a dictionary for incorrect words.

3. Result

In this work, models of ready-made auto corrector solutions were considered that satisfy three criteria:

- implementation language,
- license type,
- update ability.

Some datasets were used to evaluate the model. The accuracy was tested both on several artificial data sets and real ones such as Wikipedia. The table below shows the results for English text.

Table 1: Result of an open-source toolkit

	Error Rate	Fix Rate	Precision	Recall	Speed
JamSpell	2.60%	81.16%	100.00%	81.16%	1764
Hunspell	9.80%	57.25%	70.83%	58.12%	284
NeuSpell	0.80%	89.00%	100.00%	88.58%	1200
SymSpell	5.70%	71.22%	92.00%	76.00%	2480

- Error Rate: the number of errors that were in the text after performing automatic correction
- Fix Rate: the number of fixed errors
- Precision, Recall: correctness
- Speed: Number of words per second

To check the correctness, namely through Recall, we should use this formula:

$$Recall = \frac{Spellchecker\_true}{Total\_list\_true}$$

Total_list_true is the total number of correct words and Spellchecker_true is the number of words that the autocorrect considers correct. To determine the Precision, you need the number of words that the autocorrect considers wrong (Spell_checker_false), divided by the total number of wrong words (Total_list_false).

$$Precision = \frac{Spellchecker\_false}{Total\_list\_false}$$

In Table 1, we can see that one of the options for speeding up performance is the SymSpell algorithm, it works 1.5-2 times faster than others. Because the algorithm searches for all lines within the maximum edit distance in a very short time. And the most effective for correcting errors is NeuSpell. Unlike other spell checkers that have been reviewed, this model accurately captures the context of misspelled words. According to the result of this experiment, the model with neural network proposed in this paper gains a higher precision than the other three spell checkers, with about 7.42% higher than JamSpell, but speed worse than JamSpell and SymSpell, because the algorithm uses ten models that consist JamSpell too.

IV. Conclusion

In this article, we evaluated the spell checker methods and reviewed the available out-ofthe-box English spell checker tools. When developing the ready error correction systems, such as t the reliability of the text, the frequency of errors, and the level of error were taken into account. The evaluation performance took advantage of the NeuSpell model and showed that this model is better protected against hostile attacks than other off-the-shelf tools that have been developed using machine learning for spell checking. And also to quickly find and fix errors, it is better to use the SymsSpell algorithm, since it finds all the lines in a very short time.

In future work, I want to take advantage of open-source spell-checking tools and create my algorithm in which the new algorithm will find fast and correct results. As well as using these methods to create an efficient database. In the future, I would also like to complement NLP (Natural Language Processing) with AI (Artificial Intelligence) to create a website and mobile app that does word prediction.

References

1 Joachim Wagner, Jennifer Foster, and Josef van Genabith, "A comparative evaluation of deep and shallow approaches to the automatic detection of common grammatical errors.", In Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL), pages 112–121, 2007.

2 Mohammed Nejja, Abdellah Yousfi, "Context's impact on the automatic spelling correction", In International Journal of Artificial Intelligence and Soft Computing, 2017.

3 Pirinen, T.A., Lindén, K, “State-of-the-art in weighted finite-state spell-checking”, In Computational Linguistics and Intelligent Text Processing, 2014.

4 Sai Muralidhar Jayanthi, Danish Pruthi, Graham Neubig, "NeuSpell: A Neural Spelling Correction Toolkit", In Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations, 2020.

5 Ei Phyu Phyu Mon, Ye Kyaw Thu, Than Than Yu, Aye Wai Oo, “SymSpell 4Burmese: Symmetric Delete Spelling Correction Algorithm (SymSpell) for Burmese Spelling Checking”, In 2021 16th International Joint Symposium on Artificial Intelligence and Natural Language Processing (iSAI-NLP), 2021.

6 Wolf Garbe, “SymSpell vs. BK-tree: 100x faster fuzzy string search & spell checking”. https://github.com/wolfgarbe/SymSpell, 2017

7 Cran.R-project[Online], “The hunspell package: High-Performance Stemmer, Tokenizer, and Spell Checker for R”, https://cran.r-project.org/web/packages/hunspell/vignettes/intro.html [Last update available 2020.12.09]