Automatic restoration of diacritics based on word n-grams for Slovak texts

Conference Paper · November 2019

DOI: 10.1109/Informatics47936.2019.9119328

5 authors, including:

Štefan Toth
University of Žilina
18 PUBLICATIONS **47** CITATIONS
SEE PROFILE

Michal Ďuračík
University of Žilina
16 PUBLICATIONS **52** CITATIONS
SEE PROFILE

Matej Meško
University of Žilina
8 PUBLICATIONS **6** CITATIONS
SEE PROFILE

Patrik Hrkút
University of Žilina
19 PUBLICATIONS **74** CITATIONS
SEE PROFILE

Automatic restoration of diacritics based on word n-grams for Slovak texts

Štefan Toth, Emanuel Zaymus, Michal Ďuračík, Matej Meško, Patrik Hrkút
Department of Software Technologies
Faculty of Management Science and Informatics
University of Žilina
Žilina, Slovakia
{stefan.toth, michal.duracik, matej.mesko, patrik.hrkut}@fri.uniza.sk, emanuel.zaymus@gmail.com

***Abstract*—In the past and even now, many people still write texts without diacritics, especially in chat messages, e-mails or discussion posts. This issue evolved from historical reasons when people had a problem with text encoding in messages or wanted to write them faster. In this paper, we propose an algorithm based on word n-grams (contiguous sequence of n words) that restore diacritics of text written in the Slovak language. We also compare and evaluate our results with existing algorithms developed for Slovak texts.**

Keywords—diacritic, diacritics restoration, n-gram, Slovak language

I. Introduction

A diacritic, defined by Merriam-Webster dictionary [1], is a mark near or through an orthographic or phonetic character or combination of characters indicating a phonetic value different from that given the unmarked or otherwise marked element. Another simple definition is that a diacritic is a mark that is placed over, under or through a letter to show that the letter should be pronounced in a particular way.

Some people miss diacritical marks (diacritics) when typing, especially in instant chat communication, where users send to each other a lot of short messages by messengers, SMS or e-mails. This skipping is developed due to a need for a quick response or minimalization of time to send such a message. Practically this means to write a message as fast as possible, where diacritics omitting is an acceptable error.

There can also be a historical cause of this. In the past, character encoding was a problem. Various mail client applications have used different encodings for texts written in the specific language on different operating systems. It happened that a received mail from one mail client was not displayed correctly on another client. Therefore, people used to write texts without diacritics. When the era of mobile phones came, one of the main communication channels has become short text messages (called SMS by Short Message Service). The maximum length of one SMS is 140 bytes [2]. If the sender uses basic Latin characters without diacritics in a message, the default encoding is 7-bit (GSM-7). This means that the message can hold 160 characters. However, if a user uses non-standard or accented characters, the number of characters in a message will be truncated due to a change in the encoding (from 7-bit to 8bit or currently even 16bit for UCS2). For this reason, people preferred to write SMS messages without diacritics.

In sum up, omitting diacritics was a purely economic solution. But people soon realize, that this makes typing messages faster, too. Nowadays, the size of a message is not a problem (because of good access to the internet) and intelligent auto-correction can help to correct diacritic. But still, some people do not use diacritical marks in typing. Reason for this may be quick writing, habit, convenience, laziness or ignorance typing.

Occasionally, the contents of a mail or an SMS need to be used in another text. If its text is written without diacritics, then it is necessary to either manually overwrite or semi-automatically correct the text by using the spell checker. This is quite a laborious solution. To simplify and speed up the transformation, an automatic restoration system of diacritics would be a suitable solution.

In this paper, we are focusing on the Slovak language and its specific diacritical marks. Table I shows all lowercase and uppercase letters in the Slovak alphabet.

TABLE I. Diacritics in the Slovak language

Diacritical mark	Slovak letters with diacritical marks
´ (acute accent)	á, é, í, ĺ, ó, ŕ, ú, ý Á, É, Í, Ĺ, Ó, Ŕ, Ú, Ý
ˇ (caron)	č, ď, dž, ň, ľ, š, ť, ž Č, Ď, DŽ, Ň, Ľ, Š, Ť, Ž
¨ (diaeresis)	ä Ä
ˆ (circumflex accent)	ô Ô

Text can also contain diacritical marks that are no native to the current language, because there can be used words from foreign languages, containing letters such as: ö, ü, ů, Ö, Ü, Ů, ã, Ã, ẽ, Ẽ, ê, Ê, ć, Ć, ś, Ś, ç, Ç, ş, Ş, ţ, Ţ, … [3].

If a word without diacritics has only one meaning (there are no other meaning acquired by adding diacritics), the reconstruction can be done using a dictionary. Example of such word in the Slovak language is "bábätko" (translation: "a baby"). Removing diacritics, we obtain the word "babatko" as a result. There is no other word with other meaning in the Slovak language that can be created by adding a combination of diacritical marks. So, in this case, it is easy to restore correct diacritics.

The more difficult problem is a word which can have different meaning adding a various combination of diacritics. Example of such word is "boli" (meaning "we were" or "they were") and itself is a valid word. One of meaning we can reconstruct from this word is "bolí" (meaning "it hurts") and another one is "bôli" (meaning "in a grief"). Here must be the context understand to reconstruct diacritics correctly.

II. Current State

Several works have been published for the restoration of diacritics for the Slovak language. For example, in [4] authors used *Hidden Markov Model* and *Viterbi algorithm*. In [5] authors used *statistical language models* of Slovak language. Nowadays, *deep neural networks* have shown remarkable results as authors in [6] and [7] show state of the art solutions.

Reconstruction of diacritics is not only important for the Slovak language. There are many languages where diacritical marks are used. Publications dealing with this problem can be found for languages such as Czech [7], Hungarian [8], Romanian [9], Arabic [10], Vietnamese [11], and many others.

Now, we are mainly focusing on the Slovak language. There are three freely accessible services for the reconstruction of diacritics on the internet. These services will be described in the following sections.

A. Diakritik by SAS

The tool *Diakritik* was created by Ľ. Štúr Institute of Linguistics of the Slovak Academy of Sciences (SAS). It was open for public by 18.8.2014 [12]. It is based on a big corpus of Slovak text and has several options in how diacritics is reconstructed: first match, random match, most used and words n-gram (2, 3, 4, 5 or 6 words) and more. Authors of the Diakritik announce that it has a 0.21% error rate.

B. Statistical diacritic restoration by STUBA

The *statistical diacritic restoration* system was created by J. Geder as bachelor thesis [13] at the Faculty of Informatics and Information Technologies at the Slovak University of Technology in Bratislava (STUBA). The application creates all possible combination of n-words with correct diacritics. Next step calculates the probability of occurrence for all n-words combinations and one with the highest probability is chosen. The algorithm can reconstruct only words without a typo or missing letters. The announced success rate is 98%.

C. Diacritics restoration by BRM

The *diacritics restoration* (in Slovak "Doplĺňač diakritiky") (BRM) by R. Hraška is a web application open to the public [14]. The restoration is determined by most often word from diacritics form. The application does not use any sophisticated methods or algorithms or complicated linguistics methods.

III. Proposed Algorithm

Our algorithm is based on comparing the surrounding of the word to which we want to add diacritics with the n-grams associated with that word. We consider the words around 3 words before and 3 words after this word in context, if these words exist. The base algorithm scheme is shown in Fig. 1.

After reading the input text without diacritics, which is the text we want to reconstruct diacritics, we divide the text into separate words. In this step, we need to keep in mind that in our final text we want to preserve all the punctuation, white, and other characters that the user enters. Now we will process all the words one by one. If we still have a word, we take it and normalize it (convert it into a standardized form that we can process).

Let's convert all capital letters to lowercase. We do this because all words in n-grams are only lowercase, which simplifies the work and saves a lot of memory (if we were to work with data that also distinguishes between uppercase and lowercase letters, processing and searching such data would be much more complicated). We get the neighborhood word. When comparing, we will use no more than 4-grams. If the first word in a 4-gram is equal to our searching word, we need 3 following words after our word for comparison. If the last word in the 4-gram is equal to our word, we need 3 words before our word for comparison. Therefore, we consider the word context around 3 words before and 3 words after this word, of course, if these words exist.

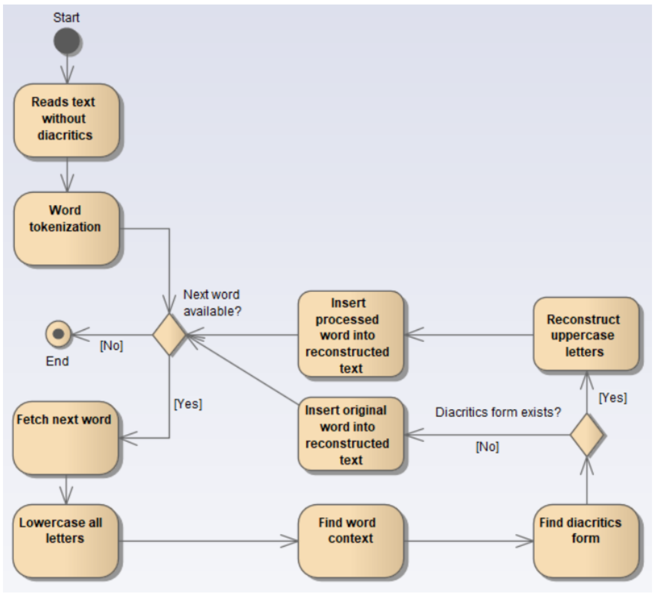

Fig. 1. Activity diagram – base algorithm scheme

Subsequently, we find diacritics form of the word (more information will be described in the next section, see Fig. 2). If we have successfully added diacritical marks, we change the lowercase letters to uppercase letters in the new word as they were in the original word without diacritics and write it into the resulting text. However, if we were not successful, we will add the original word to the resulting text, which we do not need to change to lowercase. When inserting words into the resulting text, we need to be careful to insert punctuation, white, and other characters correctly to preserve the original format of the text. Consequently, if we have another word, we begin by processing it, if not, our algorithm ends.

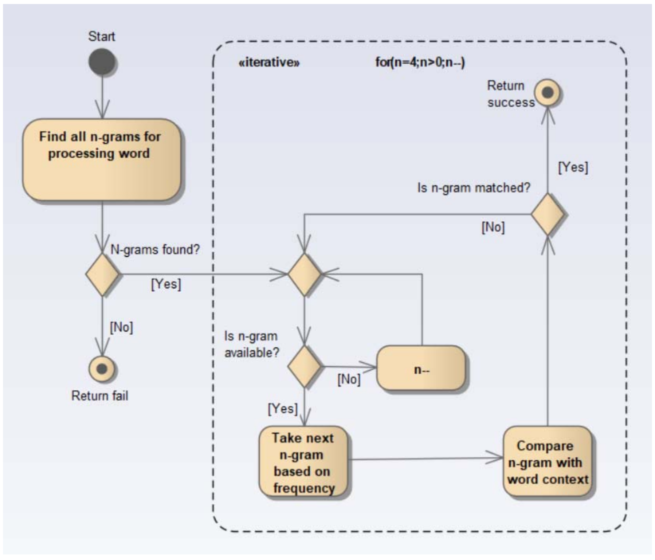

Fig. 2. Activity diagram – matching n-grams by the surrounding context

In the algorithm to find diacritics form of a word using its surroundings (see Fig. 2) we assume that we have a word for the restoration of diacritics and its surroundings. First, we find all the n-grams for processing word. By the corresponding n-grams, we mean every n-gram in which at least one of the *n* words corresponds to our reconstructed word (of course, if we remove the diacritics from the n-gram before comparing them). If we do not find any such n-gram, we end with returning fail, that is, we have not added any diacritics to that word. If we find at least one such n-gram, we can proceed to their gradual processing. In the above diagram, we see the for-cycle for (n = 4; n > 0; n--), which says that we will gradually process groups of the corresponding n-grams, ranging from 4-grams to 1-grams. If there is no n-gram in the current group, we decrement n. In this way, we iterate from the group of n-grams to the group of (n-1) -grams while n > 0.

If we have some n-gram in the current group, we take the one that has the highest absolute frequency. In the n-gram, we find where our word occurs (we have to keep in mind that the searched word may appear multiple times in the n-gram), and we compare the surrounding words of the n-gram with the surroundings of the reconstructed word (of course, we remove the diacritics from the n-gram before processing).

If we find a match, that is, the surrounding of the word agrees with the n-gram, we can return success and reconstruct diacritics by the found word from the n-gram). If not, let's see if we still have any n-gram in the current n-gram group. We can notice the fact that if we go into the for-cycle, our only possible exit is a return to success. This is because we assume that in the group of n-grams belonging to a given word there is necessarily at least one 1-gram. This condition ensures that if the cycle goes to 1-gram comparison, immediately at the first 1-gram, the cycle returns success because 1-gram has no neighborhood, and hence the match is certain.

If we find a match around a word with an n-gram with a higher n, we consider it more relevant than a match in a lower n, even if that n-gram had a higher absolute frequency. That's why we first search all 4-grams by frequency and then all 3-grams by frequency.

To help us quickly search through all the data associated with each word, we used the Trie data structure, which has an *O(n)* search complexity where n is number of items.

Because it is necessary to compromise between speed and accuracy (success) of the algorithm, we decided to implement two different ways, one focusing on speed (data structure in memory) and the other focusing on success (data structure on disk). It will always be crucial how and where we store the data (n-grams) that we want to search.

A. The data structure in memory

Implementation of the data structure in main memory is very simple. We follow the algorithm design as shown in Fig. 2, and when we find the *Find all n-grams for processing word*, the above-mentioned Trie structure comes into play. We chose the easiest approach – each word will have its corresponding n-grams at the top of the Trie tree. This ensures almost instant access to all necessary n-grams. In addition, the format of storing data in vertices can be adjusted so that we do not have to perform a for-cycle at all, nor do we search for the next n-gram from the *Take next n-gram based on frequency*. This means that all relevant 4-grams, then 3-grams, 2-grams and finally 1-grams will be on the leaf first. If we have n-grams stored in this memory, we don't have to hold information about their frequencies at all (which saves us memory). In result, the algorithm for finding n-gram is shown in Fig. 3.

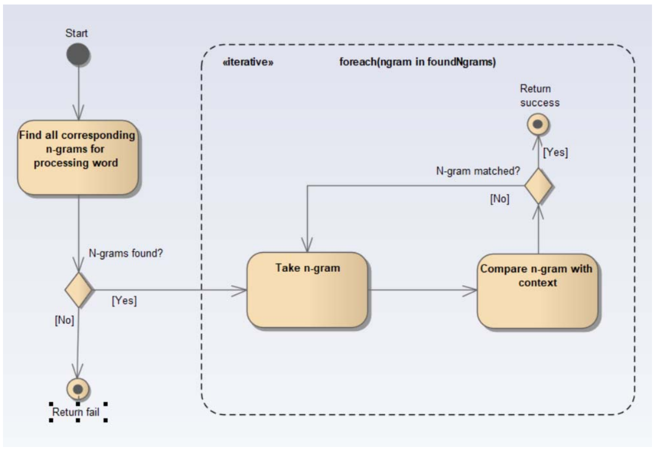
Start
«iterative»
foreach(ngram in foundNgrams)
Return success
Find all corresponding n-grams for processing word
[Yes]
N-gram matched?
[No]
N-grams found?
Take n-gram
Compare n-gram with context
[Yes]
[No]
Return fail

Fig. 3. Activity diagram – n-gram diacritic search using Trie

Based on our experiments, we can optimize the algorithm as follows:

1. There is no need to keep words without diacritics. They will be not found in the structure and skipped.
2. If there is exactly one word with diacritics for one word without diacritics, we can only store it in 1-gram database and remove all n-grams containing the word.
3. 1-gram cannot have multiple diacritics variations, because there is no context to use for the match. So, 1-gram contains only one the most often form.

We found, that the number of used n-grams are always approached in ratios showed in Table II.

TABLE II. PERCENTUAL AMOUNT OF N-GRAMS

	4-grams	**3-grams**	**2-grams**	**1-grams**
Percentual use of diacritics reconstruction	10%	24%	37%	29%

Table II shows the percentage of n-grams in a test database that was used in diacritics reconstruction. Using the memory only makes searching and whole algorithm very fast, but we want out to be runnable on common PCs (average 8 GB). We estimated, that we must not exceed 4-5 GB of RAM to be runnable on such PC. To approximately reach this limit maximum number of n-grams must be limited to 700 per one word. Limit of 700 n-grams involves all n-grams levels with percentual distribution described in Table II.

B. The data structure in disk

Our memory restriction makes need to use a disk to store data of n-grams. Adding this possibility greatly increases the size of the potential database. Using disk as storage for n-grams creates minor changes in our structure. Trie nodes contain the only reference to a binary file stored on disk as shown in Fig. 4. More in detail, nodes in Trie will have one reference to index in the binary file, the p position. This position in the file contains the count of n-gram k, which is a 4-byte number (integer). Next, all n-grams of p position are processed. N-grams in the binary file are also ordered by its occurrence frequency and by the count of n-grams.

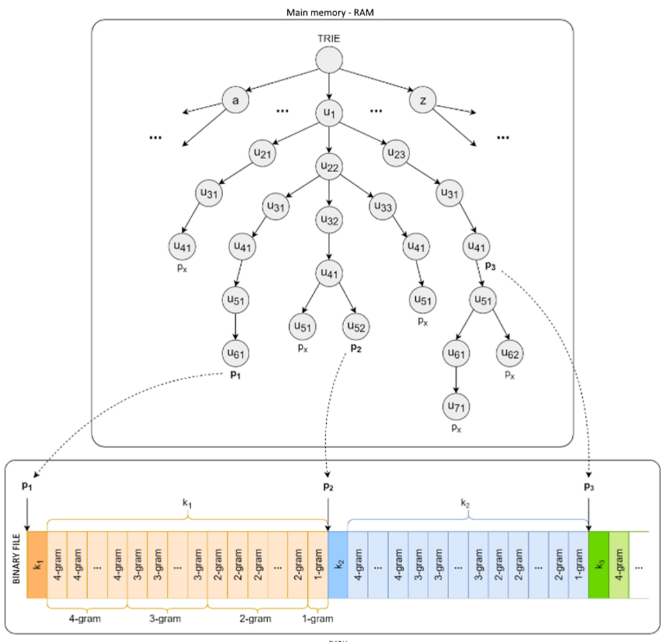

Fig. 4. The structure of n-gram database using the binary file

During our early test, we notice showiness of our implementation algorithm implementation as very slow. The issue was created by an enormous count of n-grams of one word reaching millions in some cases of word with high occurrence frequency. Our analysis provides, that most common n-gram count for one word is between 30,000-40,000. Based on our observation we set a limit of maximum 40,000 n-grams per word. This limitation speeds reconstruction of one word from 0.27s to 0.009s and success rate was decreased negligibly.

Another optimization step was in using cache memory for frequently repeating words or in a similar context. In our implementation, the cache stores last 1000 searched words with picked n-grams.

IV. Data

For building a good n-gram database is essential to use quality source. Our data was created from public-open texts from Slovak national corpus from the Slovak Academy of Sciences [15]. The corpus contains a primary large number of words from a huge range of categories (many literary genres, regions, fields of science, etc.) from the year 1995. All words in the corpus contain additional information about them generated by *IRSTLM Toolkit* [16]. Although we are aware that this data also contains incorrect n-grams, there is a very large number of them. We have used a version of the corpus labelled as *prim-8.0* released at 31.01.2018 with the token count at 1.5bln. For our algorithm, we are using sub-corpus prim-8.0-public-all.

V. Experiments

To verify the success of the proposed algorithms, we chose a set of our test texts. We divided them into three categories according to literary styles: professional, journalistic and artistic texts. After randomly selecting 25 texts by category, we obtained a set of more than 53,000 words, each category having approximately 17,000-18,000 words.

Subsequently, we removed diacritics from these texts and performed tests with our and mentioned algorithms. After that, we compared the reconstructed words with the original words. The overall results of the tests show Table III and Fig. 5.

TABLE III. Results of algorithms on our dataset sorted by success rate

Algorithm	**Error rate**	**Success rate**
Our algorithm (disk)	1.83%	98.17%
Our algorithm (memory)	1.93%	98.07%
BRM	3.00%	97.00%
SAS	3.05%	96.95%
STUBA	4.53%	95.47%

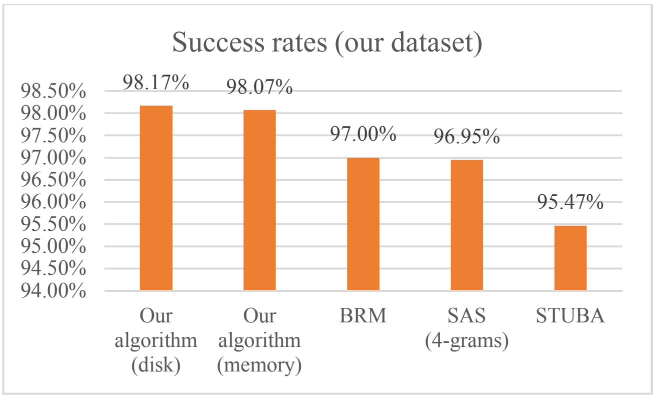

Fig. 5. The chart of overall success rates of the algorithms on our dataset

Also, we chose another experiment in which we used part of the dataset from the paper by Náplava et al. [7] published on [17]. Specifically, we chose a test set for the Slovak language obtained from Wikipedia articles that contains 524,694 words.

Since individual web diacritics restoration algorithm services have limitations on the length of input text, we have divided the test part of the dataset into subsets with words of max. 9000 characters. We reached 369 subsets, but only 350 of them were used because we had trouble getting the results from the STUBA algorithm. The reasons were unhandled exceptions and errors with empty results on 19 subsets of texts. Nevertheless, we further found that this reconstructor also changed the content of the resulting texts, so it was not possible to reliably compare its success based on words. For this reason, we did not include its results in the overall results. Besides, in the case of SAS, we also performed experiments for the 6-gram algorithm as shown in Table IV and Fig. 6.

TABLE IV. Results of algorithms on Wikipedia dataset sorted by success rate

Algorithm	**Error rate**	**Success rate**
Our algorithm (disk)	1.33%	98.67%
Our algorithm (memory)	1.53%	98.47%
BRM	1.70%	98.30%
SAS (6-grams)	2.38%	97.62%
SAS (4-grams)	2.47%	97.53%

We did not have the opportunity to carry out our tests with data using a neural network as described in [7], where the success rate for the Slovak language reached 98.84% and using the language model even 99.09%. This shows that the neural network achieved the best results at present. Despite this, our proposed algorithm achieves better results compared to other algorithms freely available and accessible on the web.

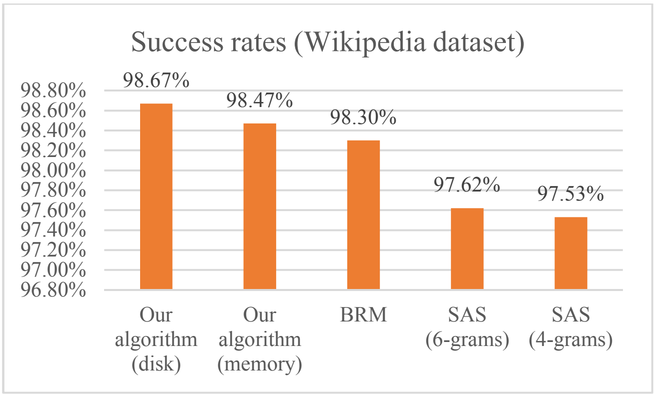

Fig. 6. The chart of overall success rates of the algorithms on Wikipedia dataset

A. *Our algorithm in memory*

The first algorithm with the data structure in the main memory reached a total success rate of 98.07%, using n-grams diacritics for 83.03% of words were reconstructed. The maximum size of data in main memory was 4.28 GB. The average speed of the reconstruction of one word was 0.24 ms. Table V shows the overall results.

TABLE V. TEST RESULTS – STATISTICS FOR OUR ALGORITHM IN MEMORY (OUR DATASET)

	Professional	**Journalistic**	**Artistic**	**Total**
Words in text	18437	17672	17187	53296
Use of 4-grams	373	576	198	1147
Use of 3-grams	2014	2884	1687	6585
Use of 2-grams	6475	7591	8027	22093
Use of 1-grams	5197	3917	5315	14429
Use of all n-grams	14059	14968	15227	44254
Time (ms)	3994	4363	4301	12658
Error words	25	223	513	1026
Success rate	99.86%	98.74%	97.02%	**98.07%**

B. *Our algorithm in disk*

The results of the second algorithm with data structure on the disk shown in Table VI. This algorithm has a better success rate than previous but is slower. It was used up to 44.56% of all n-grams from the cache. The maximum size of data in RAM was 1.16 GB and the average speed of the reconstruction of one word was 4.17 ms.

C. *SAS*

The Ľ. Štúr Institute of Linguistics of the Slovak Academy of Sciences has 10 methods for reconstruction diacritics: first (selection of the first word), random (selection of random word), naive (selection of the most frequent word), n-gram (selection of word based on n-grams where n can be 2, 3, 4, 5 or 6), surreal and maximalist. The default method is 4- grams.

We found a few shortcomings of the reconstructor:

- Ignore multiple spaces – if there are more than one white space or white space between words, the SAS ignores them and use only one.
- Adding diacritics to the Slovak word “a” (in English: *“and”*) – the word “a” without diacritics is the most common in the Slovak language. Nevertheless, the reconstructor very often added the accent "à", "á" or "ä".
- URLs – accents do not usually appear in URLs, but they are still added.

D. *BRM*

Although the BRM works only on adding the most likely word principle (or the highest occurrence), it achieves a very good success rate. It does not contain issues described in the previous text.

E. *STUBA*

Using our test texts and texts from Wikipedia we uncover some errors of this diacritics reconstructor:

- It ignores multiple spaces and in the result are replaced by one space.
- Upper and lower-case letters are not formatted as in input text. This makes a word as names and titles wrongly written. Such as MP3 -> mp3
- Issues with quotation marks - quotations marks can be presented with multiple symbols (" „ “ ” ' ` ‘ ’ … :). So whole words were missing.
- Numbering – if a word contains a number, this number is replaced by various letters: па1перт8 (18), я3зы3къ (33), в8та1йнэ (81), о3те1цъ (31), etc.
- Words “ty”, “TY”, “Ty” and “tY” throws an exception and make the application to crash.

TABLE VI. TEST RESULTS – STATISTICS FOR OUR ALGORITHM IN DISK (OUR DATASET)

	Professional	**Journalistic**	**Artistic**	**Total**
Words in text	18437	17672	17187	53296
Use of 4-grams	835	1476	559	2870
Use of 3-grams	3054	4538	3061	10653
Use of 2-grams	5338	5915	6091	17344
Use of 1-grams	4832	3039	5516	13387
Use of all n-grams	14059	14968	15227	44254
Words from cache	6402	6518	6801	19721
Words from cache (%)	45.54%	43.55%	44.66%	44.56%
Time (ms)	61708	79790	76311	217809
Error words	26	200	508	975
Success rate	99.86%	98.87%	97.04%	**98.17%**

VI. CONCLUSION

In this work, we dealt with the problem of diacritics reconstruction from texts without diacritics. We stated the reasons why such texts are created at all and what is the problem when adding diacritics. We described what algorithms exist for the Slovak language and propose our algorithm for reconstruction of diacritics with two implementations. We tested the algorithms, collected the results and compared them. Compared to other available algorithms and services, our proposed algorithm has achieved the best results and the lowest error rate. Nevertheless, there are other algorithms based on neural networks that perform much better results.

In the future, we would like to focus on improving the algorithm using neural networks, namely bidirectional LSTM with attention blocks.

VII. Acknowledgment

This publication was realized with support of the Operational Program Research and Innovation in frame of the project: ICT for smart society, code ITMS2014 +: 313011T462, co-financed by the European Regional Development Fund.

VIII. References

[1] "Merriam-Webster dictionary," 2019. [Online]. Available: https://www.merriam-webster.com/dictionary/diacritic.

[2] W. Jansen and A. Delaitre, Mobile Forensic Reference Materials: A Methodology and Reification, US Department of Commerce, National Institute of Standards and Technology, 2009.

[3] S. A. o. S. Ľ. Štúr Institute of Linguistics, Pravidlá slovenského pravopisu. 3., upravené a doplnené vyd., Bratislava: Veda, 2000.

[4] D. Hládek, J. Staš and J. Juhár, "Diacritics Restoration in the Slovak Texts Using Hidden Markov Model," in *Human Language Technology. Challenges for Computer Science and Linguistics. LTC 2013. Lecture Notes in Computer Science*, Springer, Cham, 2013.

[5] R. Krchnavy and M. Simko, "Sentiment analysis of social network posts in Slovak language," in *2017 12th International Workshop on Semantic and Social Media Adaptation and Personalization (SMAP)*, Bratislava, 2017.

[6] A. Hucko and P. Lacko, "Diacritics Restoration Using Deep Neural Networks," in *World Symposium on Digital Intelligence for Systems and Machines (DISA)*, Kosice, 2018.

[7] J. Náplava, M. Straka, P. Straňák and J. Hajic, "Diacritics restoration using neural networks," in *Proceedings of the Eleventh International Conference on Language Resources and Evaluation (LREC-2018)*, Miyazaki, Japan, 2018.

[8] A. Novak and B. Sikósi, "Automatic Diacritics Restoration for Hungarian," in *2015 Conference on Empirical Methods in Natural Language Processing*, Lisbon, Portugal, 2015.

[9] D. Tufiş and A. Ceauşu, "DIAC+: A Professional Diacritics Recovering System," in *Proceedings of LREC 2008*, Marrakech, Morocco, 2008.

[10] I. Zitouni, J. S. Sorensen and R. Sarikaya, "Maximum entropy based restoration of Arabic diacritics," in *ACL-44 Proceedings of the 21st International Conference on Computational Linguistics and the 44th annual meeting of the Association for Computational Linguistics*, Sydney, Australia, 2006.

[11] T. A. Luu and K. Zamamoto, "A Pointwise Approach for Vietnamese Diacritics Restoration," in *2012 International Conference on Asian Language Processing*, Hanoi, Vietnam, 2012.

[12] "DIAKRITIK – nástroj na rekonštrukciu diakritiky (tool for diacritics reconstruction)," Ľ. Štúr Institute of Linguistics, Slovak Academy of Sciences, [Online]. Available: https://www.juls.savba.sk/diakritik.html. [Accessed 2019].

[13] J. Gedera, "Doplňovač diakritiky," Faculty of Informatics and Information Technologies, Slovak Technical University, 2015. [Online]. Available: http://text.fiit.stuba.sk:8081/. [Accessed 2019].

[14] R. Hraška, "Dopĺňač diakritiky," [Online]. Available: https://diakritika.brm.sk/. [Accessed 2019].

[15] "Slovak National Corpus," Ľ. Štúr Institute of Linguistics, Slovak Academy of Sciences, [Online]. Available: https://korpus.sk/index_en.html. [Accessed 2019].

[16] M. Federico, N. Bertolti and M. Cettolo, "IRSTLM Toolkit," 2008. [Online]. Available: https://hlt-mt.fbk.eu/technologies/irstlm.

[17] J. Náplava, M. Straka, J. Hajič and P. Straňák, "Corpus for training and evaluating diacritics restoration systems," LINDAT/CLARIN digital library at the Institute of Formal and Applied Linguistics (ÚFAL), Faculty of Mathematics and Physics, Charles University, [Online]. Available: http://hdl.handle.net/11234/1-2607.