Post-correction of OCR Errors Using PyEnchant Spelling Suggestions Selected Through a Modified Needleman–Wunsch Algorithm

Ewerton Cappelatti(✉), Regina De Oliveira Heidrich, Ricardo Oliveira, Cintia Monticelli, Ronaldo Rodrigues, Rodrigo Goulart, and Eduardo Velho

Feevale University, RS 239, 2755, Novo Hamburgo, RS, Brazil
ewertonac@feevale.br

Abstract. In this article, the efforts made by the Vocalizer project development team to correct errors from texts generated by OCR Tesseract are described. Vocalizer consists of a device that captures images from books, converts them into plain texts with the aid of an OCR (Optical Character Recognition) software. It also prepares the post-processing of the obtained text, and converts its textual content into voice. The whole process is performed autonomously. In the post-processing step, a modified Needleman-Wunsch algorithm was applied to select the suggestions made by the spellchecker PyEnchant. The results obtained were reasonable, which encourages further research.

Keywords: Optical character recognition software · Error correction
Character recognition

1 Introduction

Brazilian government made great efforts aiming at the inclusion of people with visual impairment in the Educational System. Since 2012, Brazilian government invested in voice scanners, which resulted in high costs, and did not fully satisfy the national requirements.

Hence, Brazilian government launched a public call for voice scanner projects that meet the national requirements. Feevale University (Novo Hamburgo city, Brazil) and Pináculo company (Taquara city, Brazil) participated and won the public bid for the development of a project called Vocalizer.

Vocalizer project aims to develop a device that provides autonomy for visually impaired people. The project proposal includes an equipment for capturing and the transforming images into a plain text using the Tesseract OCR software [1], a post-processing step, which handles errors, and a final step of speech synthesis, which uses a vocalization software. One of the goals which guide the development of the device is that must be economically affordable.

This article describes the post-processing module and the strategy used to correct common errors from the output of Tesseract OCR. The adopted error handling strategy

C. Stephanidis (Ed.): HCII Posters 2018, CCIS 850, pp. 3–10, 2018.
https://doi.org/10.1007/978-3-319-92270-6_1

uses a generic spell check library called PyEnchant [2]. This library provides a list of words as suggestions for misspellings, which combined with an algorithm selects the best word. This algorithm uses a heuristic selection based on the algorithm Needleman-Wunsch [3], which was adapted to consider the most frequent errors generated by the OCR. This article describes, also, technical difficulties encountered during the correction of errors, as well as the solutions adopted to reduce the error rate during the process of obtaining the plain text used as source for the vocalization software.

2 Related Work

Volk et al. [4] used a set of regular expressions which constituted substitution patterns. These expressions are constructed based on the analysis of an extensive corpus of literature. These standards are used under certain conditions to minimize false positives. This approach classifies substitutions into three categories: the first is if the substitution frequency is at least twice as high as the misspelled word in the corpus, then substitution can be made; the second considers the options gathered for the substitution in the corpus, hence the one with the highest frequency is chosen; and the third category is the unconditional substitution.

Kissos and Dershowitz [5] used a module designed to generate correction suggestions based on the modified Levenshtein distance, in which other primitive editing operations (merging and division of characters), also known as alignments of 2:1 and 1:2, are used. Such method inserts in the list of suggestions situations in which exchanges such as ‘ln’ for ‘h’, and ‘m’ for ‘nn’ occur.

Tong and Evans [6] used a bigram technique in statistical language models. The system gets information from multiple sources including letter n-grams, character confusion probabilities, and word bigram probabilities. Words with 4 or fewer letters are also indexed by their letter bigrams. For example, “that” produces the following bigrams {#th, tha, hat, at #, th, ha, t #}.

Nylander [7] used statistics to find unanticipated frequent trigrams and phonotactical rules. The author did not use lexicons. She aimed at identifying errors and not fixing them for later proofreading.

Bassil and Alwani [8] proposed the use of a post-processing context-based error correction algorithm for detecting and correcting OCR non-word and real-word errors. This algorithm was based on Google online spelling suggestions.

3 General Description

One of the project steps is the use of an OCR software to transform the images obtained by the image capturing equipment into plain texts. Tesseract is the OCR software adopted for this project. This is an open-source optical character recognition software under the Apache 2.0 License. There are many challenges to using OCR software for reading book pages. Thick and multi-page books can generate reading errors, in addition, different types of lighting can make it difficult to recognize certain regions of pages as well. Due to these factors, Tesseract cannot have 100% character recognition accuracy.

One of the recognition problems is the occasional swap between two letters, such as ‘a’ recognized as ‘o’, ‘f’ recognized as ‘t’, ‘c’ recognized as ‘e’, and so on. Other errors involve replacing a group of letters with a single letter or replacing one group of letters with another group of letters. This can be exemplified by the swapping of ‘r’ and ‘n’ by ‘m’, ‘f’ and ‘i’ by ‘h’, ‘m’ and ‘l’ by ‘n’ and ‘d’. Other recognition problems result in a diverse range of different characters. In these cases, recovery is very difficult or impossible.

The first effort to post-correct the OCR errors was to develop a simple routine which uses the PyEnchant spell checking software method. Such method provides a list of suggested words to repair a specific misspelling. However, two problems were found with this approach. Primarily, the first word suggestion is not usually correct, but in another position of the suggested words list. Thus, choosing the first word from the list has become out of question. Secondly, in others cases, the suggested words list did not even contain the correct word. Considering this, any word choice would be wrong. Therefore, two others problems had to be addressed: (1) identify which words in the list is most appropriate to replace the word mistakenly recognized by the OCR and (2) increase the chances of the correct word being present in the suggestions list.

3.1 Selecting the Best Word from the OCR Tesseract List of Suggestions

Each word recognized by the Tesseract OCR application was submitted to PyEnchant spell checking software. For each word that resulted in an error code by the software, a suggestion list was formed by the *suggest()* method of the checker object. Then it was necessary to choose which word from the list was the closest to the word identified as wrong by PyEnchant. Table 1 shows examples of lists suggested by PyEnchant for words identified as errors.

Table 1. Examples of word lists suggested by PyEnchant.

Wrong word	Correct word	List of suggested words
Deseubra	Descubra	‘desabra’, **‘descubra’**, ‘dessaibra’, …
Lmpõe	Impõe	‘lampie’, ‘limpe’, ‘limpo’, ‘lumpo’, ‘limpou’, ‘limpei’, ‘empoe’, **‘impõe’**, …

In order to establish the distance between each word in the list and the word with errors, the first hypothesis is the use of Levenshtein algorithm to calculate distances [9]. This algorithm defines ‘distance’ as the minimum number of unit operations required to transform one string into another. However, due to the high substitution frequency, such as ‘c’ by ‘e’ when compared to others such as ‘c’ by ‘:’, and the high substitution frequency between groups of letters by a single letter, such as ‘r’ and ‘n’ by ‘m’, in addition to substituting groups of letters by other groups of letters, this algorithm was discarded.

The second hypothesis is Needleman-Wunsch algorithm [3]. It has weights for positive and negative identifications, as well as for gaps. Nevertheless, the algorithm

had to be modified to accommodate different substitution probabilities, and to consider substituting groups of letters for one letter or groups of letters for other groups of letters. The issue of the substitution of letters groups had already been identified by Volk et al. [4]. A different approach which uses substitution patterns elaborated using regular expressions, was adopted by them.

The Needleman-Wunsch global alignment algorithm constructs a matrix of E scores using the dynamic programming technique for two sequences A and B. As the algorithm advances, the score matrix is filled with the optimal alignment score between the m first characters of A, and the n first of B. This score is determined as the largest value among the threefold calculated values: $E[i - 1, j] + g$, $E[i - 1, j - 1] + p(i, j)$, and $E[i, j - 1] + g$, where g is the punctuation for the insertion of a gap, and $p(i, j)$ is a function that returns a value, if the letters match, and a different value, if they mismatch [10]. A table of typical scores for the Needleman-Wunsch global alignment algorithm can be seen in Fig. 1.

match:	2
mismatch:	-2
gap:	-1

Fig. 1. Typical score table used in the Needleman-Wunsch algorithm

Thus, the alignment scores were elaborated considering the substitutions made, and taking into account the occurrence and frequency of the letter sets identified in the corpus analysis. Firstly, previous statistics of the most common substitutions were found. In order to do this, a set of pages from books was typed, so text files could be compared with the text returned by the OCR. A Python script for text comparison returned the words which differed from where the most common substitutions were extracted. The substitutions frequency determined the score value used in the algorithm. In addition, common substitutions among groups of letters were identified, and formed a scoring basis for these groups. Figure 2 shows an excerpt from the scoring table using the modified algorithm.

match (default)	128
mismatch (default)	-128
gap penalty	-64
mismatch (a, á)	48
....	
mismatch (b, l>)	31
...	
mismatch (d, cl)	54
...	
mismatch (fo, lb)	40
...	

Fig. 2. Scores used in the modified algorithm.

The alignment algorithm was modified to consider not only one letter but also 2-letter sequences in its score. As the algorithm advances the score matrix gets filled. It takes into account the current character, but in some cases the subsequent character as well. As in the original algorithm, once the scoring matrix is obtained, one moves from the lower right position towards the upper left position, inserting gaps when necessary. The score is obtained in accordance with three possible cases:

(1) if there is a match, the score for it is added to the total score of the alignment;
(2) if there is a gap, the score for it is added, however, if there are more than two gaps in a row, the scores from the second gap are considerably higher;
(3) if there is a match between a letter and a group of letters or a group of letters and another group of letters, the score is obtained from a scoring table, according to the occurrence frequency established for those letters.

The score for the word from the spell-checker suggestions list is then determined. The word with the highest score in the list is the one which will be chosen for the substitution.

For example, by applying the alignment in the word *'dinasua'* the following list is obtained:

[dina sua, dina-sua, dinas, Danúsa, dinasta, dinastia, dianas, dionas, donas, duinas, dunas, dionísia, donosa, násua, Dinís, danas, densa, densa-a, dinamia, dines, dinos, Dinísio, dônias, danesa, danosa, densai-a, diénias, duanas, Diana, danais, densai, diana, danasse, densou-a, dincas, dindas, dingas, disnas, diapausa, dinda-a, NASA, Nasa, densou, dienos, diense, dina, diniés, disa, difusa, rinusa, danasse-a, dinastas, disna, dissana, dana-as, dana-se, dassa, densada, densara, densava, dense-a, denso-a, diesa, diesa-a, diosa, disué, nassa, densa-o, dindou-a, danou-as, dana-a, dana-os, diesai-a, dindasse-a, danou-a, diesou-a, dinda-se, dane-as, dano-as]

In this case, word *'dinastia'* got the highest score and the correction is successful. However, this is not always the case. Sometimes the word poorly recognized by OCR also exists in Portuguese, hence the algorithm does not detect the need for correction. There are also cases in which a Portuguese word is more similar to the wrong one than the word in the original text. For instance, the word *'gorros'* recognized by the OCR as *'gurros'* achieves higher scoring in the element *'guarros'* of the list. This is because a gap scores less negatively than a mismatch.

3.2 Improving the List of Suggestions by PyEnchant

In some situations, the list of suggestions does not have the correct word. This is due to some reasons: the modified word recognized by Tesseract, or the initial letters word change, makes PyEnchant SpellChecker provides a set of suggestions far from the desired word. In order to solve this issue, a heuristic routine was implemented. First, it identifies sets of letters that can be unambiguously replaced by other word set from Tesseract identification. Then, the word identified by the modified Tesseract is submitted once more to PyEnchant, which results in a second list of suggestions. The first list of suggestions is then concatenated with the second and submitted to the modified Needleman-Wunsch algorithm. The highest scoring word is then chosen. Experiments

performed revealed significant improvement in the OCR error correction rate. Figure 3 shows the Vocalizer post-processing steps.

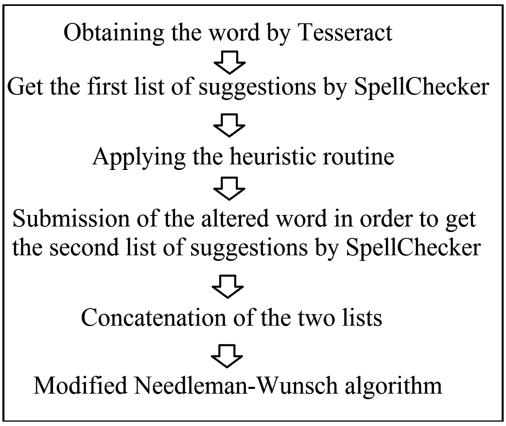

Fig. 3. Post-processing steps

The heuristic routine contains an identified set of likely modifications. These are usually at the beginning or at the end of the words and are performed to obtain a second list of the spelling checker. Examples of likely substitutions are 'gcnc' by 'gene' at the beginning of words, and 'enlo' by 'ento' at the end of words.

An example which illustrates the heuristic routine can be seen as follow. The word erroneously identified by OCR was 'Cncurrcgada'. The correct word should be '*encarregada*'. When submitted to the PyEnchant spellchecker the return is the following:

[Cancricida, Cancrocida, Cancericida, Concrecionada]

The correct word is not in the list and the reason is that both the beginning and the end of the word are incorrect. The heuristic routine changes *'rcgada'* by *'regada'* by supplying the spell-checker with a new word: *'Cncurregada'*. This resulted in a second list provided by PyEnchant:

[encarregada, encarregadão, encarregado, concretada, concurvada, encarregadas, encargada, encarregado-a, encarregadura, encarregar, encarrega-a, encarregara, encarregava, encorrugida, encarregai-a, encarregado-o]

The lists are concatenated, and the following list is obtained:

[encarregada, encarregadão, encarregado, concretada, concurvada, encarregadas, encargada, encarregado-a, encarregadura, encarregar, encarrega-a, encarregara, encarregava, encorrugida, encarregai-a, encarregado-o, Cancricida, Cancrocida, Cancericida, Concrecionada]

Once processed by the algorithm, it returns the word *'encarregada'*. In some cases, success is not achieved because words more similar syntactically may punctuate more, and its depends on how bad the word identified by OCR is damaged.

4 Results

Seven books were scanned, resulting in an initial set for testing algorithm. For each book a set of 20 pages was correctly typed so amount of errors could be evaluated. An evaluation of the text obtained by OCR was made comparing with the original text generating the number of different characters found. From these differences, the estimated difference was between 1.085% and 2.679% for texts in Portuguese and 4.77% for text with words in a foreign language.

Regarding the study, words separated by spaces or attached to each other were not considered because they were not treated by the algorithm. In addition, words which were so damaged that no recovery effort would be possible, such as '|:`lânicos' when the correct word would be 'britânicos', were not treated. Further evolution of the algorithm should be considered for these cases. Some tests results are shown in Table 2.

Table 2. Percentage of success in applying the algorithm.

Number of letters with error	Example Wrong word/correct word	Percentage of success
1	"deseubra'/'descubra'	48.96%
2	'dinasua'/'dinastia'	31.18%
3	'Cncurrcgada'/'Encarregada'	11.32%
>3	'biblioteca'/'biblimcca'	8.81%

If words considered extremely damaged were taken into account, the success percentages would decrease. Nevertheless, this issue must be addressed in a better reading and illumination images capturing. During the project, the reading improved considerably, nearly eliminating these situations.

5 Conclusions

Results obtained were reasonable, especially after inclusion of the heuristic routine in the algorithm to repair the words before a second submission to the spell checker. Considering the errors caused by the OCR were less than 3% for Brazilian literature texts, and 4.77% for texts with words in another language, the algorithm was capable of providing a text which was possible to be understood reasonably by visually impaired people. Further studies should be carried out aiming the treatment of words broken between spaces and words attached to others. The corpus used to create the substitution base should be expanded as more books are scanned and handled with the OCR tool.

References

1. Smith, R.: An overview of the Tesseract OCR engine. In: Proceedings of the Ninth International Conference on Document Analysis and Recognition, ICDAR 2007, vol. 2, pp. 629–633 (2007)
2. Ryan, K.: PyEnchant: a spellchecking library for Python. http://pythonhosted.org/pyenchant/faq.html. Accessed 05 Oct 2017
3. Needleman, S.B., Wunsch, C.D.: A general method applicable to the search for similarities in the amino acid sequence of two proteins. J. Mol. Biol. **48**(3), 443–453 (1970)
4. Volk, M., Furrer, L., Sennrich, R.: Strategies for reducing and correcting OCR error. In: Sporleder, C., van den Bosch, A., Zervanou, K. (eds.) Language Technology for Cultural Heritage, pp. 3–22. Springer, Berlin (2011). https://doi.org/10.1007/978-3-642-20227-8_1
5. Kissos, I., Dershowitz, N.: OCR error correction using character correction and feature-based word classification. In: Proceedings of the 12th IAPR Workshop on Document Analysis Systems, pp. 198–203 (2016)
6. Tong, X., Evans, D.A.: A statistical approach to automatic OCR error correction in context. In: Proceedings of the Fourth Workshop on Very Large Corpora, pp. 88–100 (1996)
7. Nylander, S.: Statistics and phonotactical rules in finding OCR errors. In: Proceedings of the NODALIDA, pp. 174–181 (1999)
8. Bassil, Y., Alwani, M.: OCR post-processing error correction algorithm using Google's online spelling suggestion. J. Emerg. Trends Comput. Inf. Sci. **3**(1), 90–99 (2012)
9. Levenshtein, V.I.: Binary codes capable of correcting deletions, insertions, and reversals. Sov. Phys. Dokl. **10**(8), 707–710 (1966)
10. Setúbal, J., Meidanis, J.: Introduction to Computational Molecular Biology. PWS Publishing Company, Boston (1997)