École Polytechnique Fédérale de Lausanne

Bachelor Semester Project

Pushing the limits of OCR on complex multilingual documents : OCR post-correction

Supervisors
Sven NAJEM-MEYER
Matteo ROMANELLO

Author
Jérôme CECCALDI

Digital Humanities Laboratory
Lausanne, 10 June 2022

Table des matières

1. Introduction

1.1 OCR Introduction

Nowadays, Optical Character Recognition (OCR) is one of the most popular method to convert printed document into digital ones. There are numerous OCR techniques that have been studied which all have their own performance and application domain, but the overall conclusion is that most modern texts are easily digitize whereas historical material make it harder to get satisfying results [13]. This loss of performance on historical documents is the main reason why post-correction is required to get better outputs as this work focuses on classical commentaries.

1.1.1 Commentaries

Commentaries are widely used in scholarship to elucidate languages or to get a more detailed understanding of the culture that produced it and the context in which it was written. Additionally, in the case of classical commentaries, its purpose includes explaining historical figure, events, laws, daily life or clarifying vocabulary and grammar. Hence, the benefits for classical scholarship to get correctly digitized commentaries but it comes with a cost, as it becomes harder to get an accurate output.

1.1.2 Limitations

As stated before, character recognition on commentaries is tedious for various reasons. First of all, historical documents come with a major drawback which is the quality of the input. Even if proper care was given to preserve those, paper and ink may deteriorate seriously with time making it hard to get a consistent representation of letters. Additionally, historical documents often comes with exotic fonts or bold characters, as showed in Figure 1.1, that makes it even harder for the OCR. For instance, errors introduced by the low quality of OCRed text can have an impact on information retrieval as demonstrated by van Strien *et al.* [19].

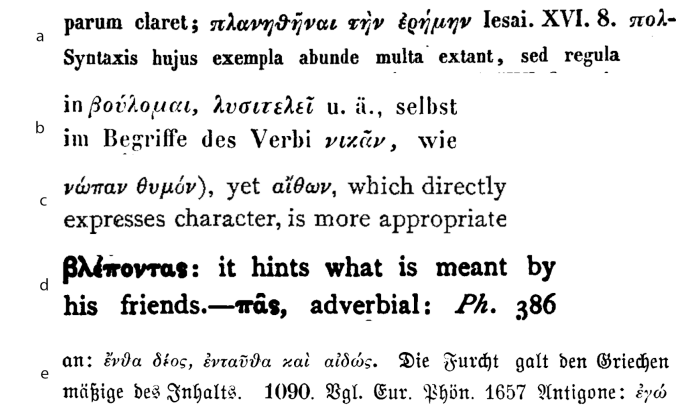

FIGURE 1.1 – Multiple lines with different fonts and quality

Secondly, because we are working with commentaries, the complexity of the layout make it harder for the OCR to get a proper alignment of the text as seen on Figure 1.2. On the output side, this may end with a wrong sequence of sentences or mixed ones which should be detected afterwards.

Thirdly, in the present time, the amount of ground truth data to train an OCR or a post-correction model is still low, hence it makes it impossible to use specific techniques for post-correction [13].

Finally, another aspect concerning the commentaries of this work is the multilingual feature of those documents which are a mix of Latin and polytonic Greek scripts. Thus, the output may mix some of the alphabet, especially on similar letters.

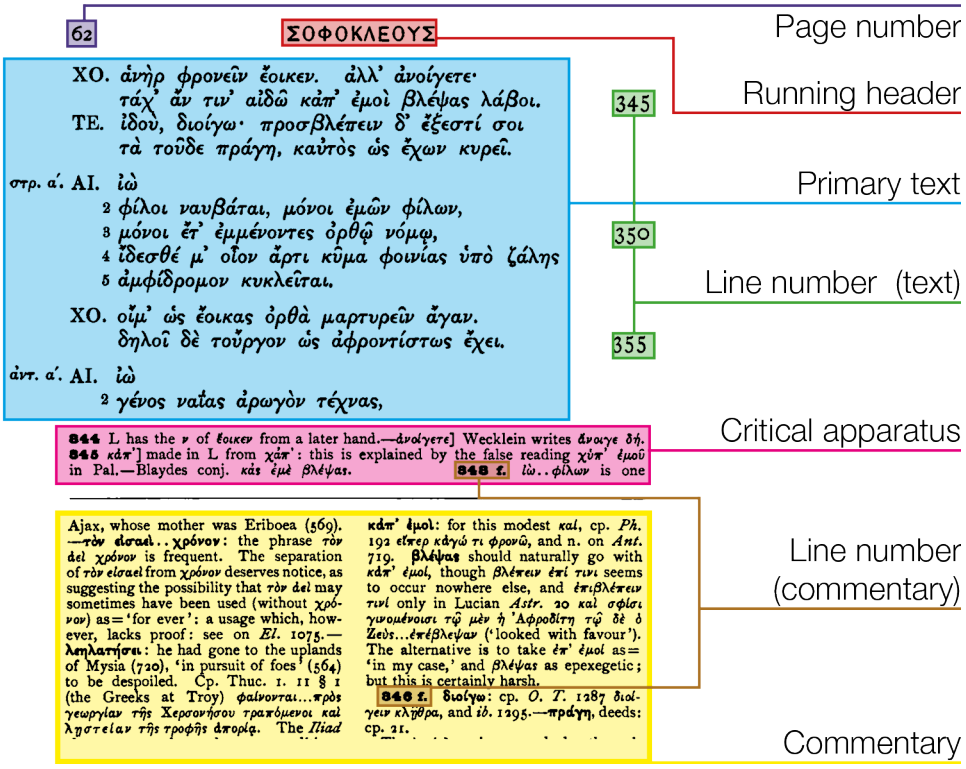

FIGURE 1.2 – Classical commentary layout with legend

1.2 Goal

The goal of this project was to explore and get a first try with possible OCR post-correction techniques and evaluate which could lead to potential candidates for multilingual classical commentaries.

1.3 Summary

To begin with, multiple approaches and techniques were consider to chose the most suitable ones for this project. Then, one supervised and unsupervised approach were chosen to experiment with. Finally, some evaluations were performed on trained model and techniques to get a first estimate on the performance. This concludes on an initial guess on which approach may be the most promising for OCR post-correction.

2. Background

The first part of this project was to gather information on the performance and application domain of different post-correction techniques. Hence, this section summarize the most popular techniques to give an overall idea of what options were taken into account. These approaches falls down into two categories : an isolated-word approach which relies on characteristics from single token to generate candidates for correction and a context dependent approach which uses neighbouring tokens to guess potential corrections.

2.1 Isolated-word approaches

These are the standard ways to make post-correction by operating on a single token at a time, in this case a token may be a character, a word or part of a word depending on the approach. One major drawback of these techniques are their inability to detect real-word errors, words that are part of the target language but are not matching the source material. For instance, "It hat been over a year" instead of "It has been over a year", here "hat" is part of the English vocabulary but is incorrect given the rest of the sentence.

2.1.1 Merging outputs

One way to generate candidates for correction is to get them directly from the OCR output. This means using multiple OCR engine on the same document or to create multiple versions of the original document at pre-processing to give them to the same OCR engine. Then alignment methods are used to group candidates for a token and finally the best candidate is chosen using one of several possible techniques (decision list, max entropy classification, CRF).

2.1.2 Lexical approaches

Another popular way to find candidates for correction makes use of lexicons and distance metrics. These methods relies on two significant parts, one is the generation of the lexicon and the other is the distance metrics used to find the most suitable token in the lexicon as replacement.

The most popular measures until now are the Damerau-Levenshtein edit distance (LV distance) and its variations which corresponds to the minimum number of single-character edits (i.e. insertions, deletions or substitutions) required to get from one token of the lexicon to another. The LV distance is generally used to get a list of alternatives from which a single best replacement is chosen using decision list based on typographical characteristics or token frequency.

Now, regarding lexicons it may range from a simple dictionary to a list of dynamic generated lexicons. The main decision factor is to know which is the best option given the application domain. Indeed, conventional dictionaries will lack of words for a specific domain and it also includes specific historical period as word spelling may vary given the era.

2.1.3 Error models

Several approaches focus on error from the OCR output to approximate a correction either by using distance metrics or by statistical methods. Some of these techniques also rely on character language models to fix erroneous tokens, which has the major benefit of considering context to guess a better correction for non-word errors.

Error models have seen many implementations but some draw more attention than others. Earlier versions would take spelling in consideration by associating probabilities with character transformation, like Church and Gale [3] or Brill and Moore [2] did. Thus, insertion and deletion probabilities would be conditioned on the last character, resulting in a weighted character edit to find the best correction based on its context. This was further improve to associate probabilities to multiple edit operations which was then used by Taghva and Stofsky [20] back in 2001 to implement a first interactive post-correction process named OCRSpell. The process would generate a final list of candidate tokens to the user as possible corrections for any erroneous token in the text.

Other interesting implementation make use of ngram word frequencies to detect errors as in the work from Kettunen *et al.* [9]. Here, they combine trigram frequencies and compare it to a threshold to detect errors, then they select the best match using edit distance and word frequency or by using Weighted Finite-State Transducer (WFST).

Lastly, one particularly interesting method using error model is the one from Reffle and Ringlstetter [17] which focuses on information from spelling variations and OCR errors to improve the output from digitized historical texts. In this case, they analyze the OCRed document to build profiles adapted to the text. On one hand, a local profile, composed by a ranked list of possible candidate for each token in the text and generated using ground truth and spelling variations. On the other hand, a global profile containing a list of error types and historical token sequences with their frequencies. This global profile can then be used to detect errors from similar texts or text processed by the same OCR engine. Finally, the best possible replacement is found in the local profile for the given token. Additionally, this automated process was used in another interactive tool for post-processing named PoCoTo [21] which utilise these profiles to target errors and suggest potential correction to the user.

2.2 Context dependent approaches

An efficient way to handle real-word errors calls on context dependant approaches, which utilises characteristics from surrounding tokens (characters or words). The two most popular categories of this kind are language models and sequence-to-sequence models (Seq2Seq) which are further detailed below.

2.2.1 Language models

Similarly to error models, many post-OCR processing methods make use of language models to generate and rank correction candidates. The only difference as explained beforehand, is the consideration of the surrounding tokens, this includes word ngrams and not only character ngrams as in error models.

On one hand, statistical language models either build a probability distribution of word sequences from frequency counts or utilises word ngram frequency to rank candidates. Similar works were performed by detecting erroneous token with a lexicon-lookup and by selecting candidates based on the LV distance. For instance, Hládek *et al.* [8] used this method and ranked suggestions exploiting a Hidden Markov Model (HMM), the state transition probabilities modeled word bigram language model probability and observation probabilities where used as smoothing string distance to find the best replacement. This example illustrates well how it is possible to take a common technique and refine it at one step with a language model.

Similarly, some clever addition are done by Evershed *et al.* [5]. They utilise an error model and gap trigrams, which is the probability of a word given its right and left neighbours, to generate candidates at character level and word level respectively. Additionally, they exploit weighted character edits and correlation between the bitmap of a correct and erroneous token, once "reversed OCR", to build the error model. Finally, they rank candidates by confusion weight and word trigram model.

On the other hand, neural network based language models learn to associates tokens to a vector representation, called word embedding and are trained as classifiers to predict the most likely next token given its neighbours. In this case, a great benefit is the ability to perform training either at word or character level. Typically, those methods are implemented on character-level bidirectional LSTM, like D'hondt *et al.* [4] did on digitized French clinical texts where they generated training materials by applying random character edit.

2.2.2 Sequence-to-sequence models

This approach relies on machine translation (MT), which performs translation from one language to another. Traditionally MT is statistical (SMT), it takes a source language (s) and a target language (t) as input, a probability distribution is then estimated using the probability $p(s|t)$ that the source s is a translation of the target t and the probability $p(t)$ that t is in the target language. A slight twist, when working on OCR post-processing is the fact that the source language is the OCR output and the target language is the ground truth. In this field, some remarkable works are those from Afli *et al.* [1] which trained an SMT successfully to reduce OCR error of historical French texts but it was done with a large training set of over 60 million words which is not representative of most OCR post-processing dataset sizes. Other significant works were done by comparing Seq2Seq models against traditional one, here a Conditional Random Fields (CRF) model. Schnober *et al.* [18] conducted an experiment that proved Seq2Seq models could outperformed a CRF one on small datasets.

A last model to consider was the neural network Seq2Seq (NMT) model, which gained an increasing amount of popularity recently and have seen some great open source projects like openNMT [11]. Compared to SMT, NMT will directly performs translation of a source sentence to a target sentence using a specified neural network without having to generate

a probability distribution, in this case the training is required to optimize parameters of the network. The most relevant work in this field for our use case is the one from Mokhtar *et al.* [14] that exploit NMT model and some enhancement technique on historical datasets and reveals that the character-level model performs better than the word-level one as the first can deal with unseen words.

3. Models

3.1 Challenges and Choice

The merging output approach was first consider as it could be easily implemented using a decision list for instance, unfortunately it was decided at first to focus on a unique OCR engine and it was unclear at the beginning of the project if it would be possible to get sufficiently different candidates from multiple pre-processing of the original commentaries.

Next the lexicon-lookup option seemed very flexible as it could be implemented from an existing model and then modified for a specific purpose either by changing the lexicon or the distance metrics. The independence from the OCR engine made it also the best alternative for an unsupervised approach or supervised approach when using the ground truth to generate the dictionaries.

Every approaches using error models have some benefits that could not be overlooked but they also comes with a higher cost at implementation. Weighted character edit can lead to more accurate correction as it takes character context into account but it requires associating probabilities for insertion and deletion to each character. As we have very little ground truth from these kind of commentaries it makes it harder to generate meaningful probabilities based on similar documents. Using trigram frequencies against a set threshold comes with the same drawback, as estimating a correct threshold requires enough data to not end with a threshold specialized into only some kind of words. At last, the profiling method requires a lot of care on establishing spelling variations and token frequencies may not be sufficient from our corpus to rank correctly the possible correction in local profiles.

As clarified in the error model, statistical approach make it harder for small training material to perform correctly, which is why a bidirectional character-level LSTM was chosen at first as an interesting model for neural network language model but NMT model were given priority later on the project. Furthermore, as showed in Figure 3.1 the use of word ngrams to rank candidates seemed like a wrong option with multilingual commentaries as the probability distribution would not make any sense.

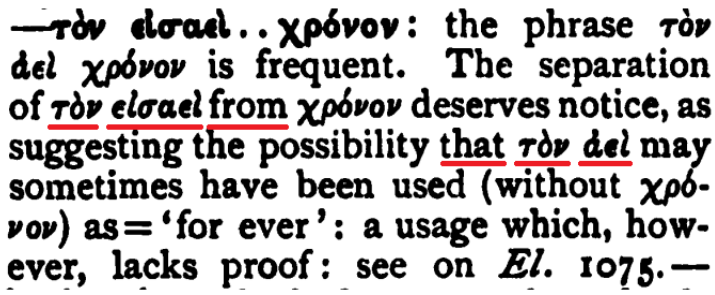

FIGURE 3.1 – Mixed latin and greek word ngrams

In the case of Seq2Seq model, it seemed reasonable to focus on neural network first as the amount of recent open source resources along with their documentation makes it easy to implement and try such alternatives. Another great benefit from the neural network model was its ability to deal with unseen words at character level as our dataset is multilingual and historical, it seemed like a successful choice.

3.2 Implementations

In the end, two models were tested at first, a classical lexicon-lookup approach and a more recent NMT model. Both implementations are detailed in the following sections.

3.2.1 Lexicon-lookup

First the lexicon-lookup model was mainly inspired by the polytonic Greek OCR tool suite Ciaconna, developed by Bruce Robertson [26]. Initially this tool provides a complete OCR pipeline from image pre-processing to OCR output post-processing based on OCRopus [25], a collection of document analysis programs. The reason why Ciaconna framework is not directly used in this part of the project is because it was designed as a complete OCR pipeline, hence making it impossible to only use the post-processing part.

The model that performed the lexicon-lookup works on line-separated file and is made up of several python modules grouped in two categories ; firstly a pre-processing one which comports :

— **remove_empty_lines.py** : To removes empty lines from the corpus with the corresponding lines from the ground truth and vice versa.
— **lowercase.py** : To lowercase all lines from corpus and ground truth.
— **dehyphenate.py** : To fuse back words split by hyphen.
— **remove_special_token.py** : To remove special character/words that are useless when performing lexicon check (i.e. parenthesis, roman number, etc...).

Secondly, a bin module which contains all files necessary for the lookup :

— **levenshtein.py** : Which relies on the *python-Levenshtein* module [22] to compute the edit distance from a target keyword to another given word.
— **spellcheck.py** : To perform correction candidates generation and selection. Generation is performed by selecting word with edit distance under a certain threshold and selection is done by taking the candidates with the less edit distance or with the best frequency.

— **word_error_rate.py** : which relies on the *evaluate* module [24] from hugging face to compute the word error rate of the entire corpus compared to the ground truth.
— **char_error_rate.py** : which relies on the *evaluate* module [24] from hugging face to compute the character error rate of the entire corpus compared to the ground truth.

All files in the pre-processing module perform I/O operations by taking as input a line separated corpus (with ground truth sometimes) and output a the refactored file after transformations are applied. A bash script **prepare_data.sh** performs all pre-processing steps and rename the file at the end to facilitate usage. Spellcheck on the file can simply be performed by using :

```
python3 bin/spellcheck.py [corpus file] [greek dict] [english dict]
```

then it can be evaluate using :

```
python3 bin/word_error_rate.py [corrected corpus file] [ground truth]
```

and

```
python3 bin/char_error_rate.py [corrected corpus file] [ground truth]
```

3.2.2 Neural Network Machine Translation

The NMT model was performed using the open source neural machine translation framework openNMT-py, a PyTorch version of the OpenNMT project from MIT [11]. The normal usage takes a source file and a target file for training to train the neural network into translating from the source language to the target language. In the context of OCR post-correction, the source language is the OCR output and the target language is the ground truth.

As this model relies heavily on big dataset size to train efficiently it was required to extends an already trained model and fine-tune it for our particular usage. The chosen model **ocr.pt** originate from a work of Hämäläinen and Hengchen [6] named NATAS, a library having methods for processing historical English corpora with the functionality to normalize historical spelling and to perform OCR post-correction. The model was first trained on openNMT at character level with the default parameters, except for the encoder where they use a BRNN to increase performance. The default of two layers for both the encoder and the decoder is used as well as the default attention model, which is the general global attention presented by Luong *et al.* [12]. Finally, the model was trained for the default number of 100'000 training steps. From here a **config.yaml** file was produced to continue the training from the NATAS model by mean of the "train_from" and "reset_optim". These flags offers the option to resume training from a checkpoint and to update the vocabulary at any point without resetting all optimisation parameters.

In the end the project structure is similar to the one used for lexicon-lookup, the folder is split into three main modules ; firstly a "preparation" module which contains :

— **remove_empty_lines.py** : To remove empty lines from the source and target corpus.
— **dehyphenate.py** : To fuse back tokens separated on two lines by a hyphen.
— **lowercase.py** : To lowercase the source file and target file.

— **char_tokenizer.py** : Which relies on pyonmttok, a customizable text tokenization library with BPE and SentencePiece support [23] to perform character tokenization of each line.
— **char_detokenizer.py** : To detokenize each lines from the source and target file.
— **compute_bleu.py** : Which relies on the sacrebleu module to compute BLEU score [16] using the translated source file and the target file.

Secondly, a "models" folder containing all tested models : ocr[version]_states_[steps nbr], where the version differentiate between the first one that were made without lowercase and without dehyphenate and the second one that implemented both. Thirdly, a "run" folder containing the generated vocabulary files from openNMT.

As in the lexicon-lookup case, a bash script to perform all preparation transformations was implemented. The script used for the first version is **prepare_data.sh** and the script used for second version is **prepare_data2.sh**. Once pre-processing is done, one can begin the training by using :

```
!onmt_train -config config.yaml
```

and then translate using :

```
!onmt_translate -model [model] -src [src file] -output [out file] -gpu 0
  -min_length 1
```

4. Results

All results are compiled in this section and were trained on a single corpus which is the *Commentary on Sophocles : Antigone* from Sir Richard C. Jebb, this corpus was the only one with ground truth data obtained by manually correcting remaining errors from the OCR output. It was then split in a training set of 4000 tokens and a testing set of 2000 tokens.

4.1 Lexicon-lookup

Overall, three versions of the lexicon-lookup were evaluated. Lexicon 1 to 3 corresponds to the equivalent version described in the Discussion section.

Lexicons	#tokens	WER	CER	Time $[hh:mm:ss]$
Lexicon 1	1925	0.145	0.056	00 : 55 : 48
Lexicon 2	1908	0.138	0.053	00 : 47 : 26
Lexicon 3	1908	0.088	0.037	00 : 37 : 11
testing set	2126	0.103	0.041	$--:--:--$

Table 4.1 – Evaluations for different lexicon-lookup versions

4.2 Neural Network Machine Translation

Evaluations were compiled in two tables, one for *Version 1* which corresponds to the first tries on the corpus without using **lowercase.py** and **dehyphenate.py**. The second table corresponds to *Version 2* which uses both.

Version 1	BLEU score	WER	CER	Time $[hh:mm:ss]$
ocr_states_108000	10.240	0.464	0.284	00 : 28 : 38
ocr_states_120000	29.74	0.397	0.190	01 : 24 : 12
ocr_states_150000	31.86	0.360	0.169	03 : 56 : 40
ocr_states_200000	35.75	0.350	0.164	07 : 15 : 21
test.source	74.27	0.129	0.049	$--:--:--$

Table 4.2 – Version 1 evaluations for the NMT model

Version 2	BLEU score	WER	CER	Time $[hh:mm:ss]$
ocr2_states_108000	24.05	0.384	0.237	00 : 28 : 34
ocr2_states_120000	36.67	0.332	0.144	01 : 12 : 10
ocr2_states_150000	43.69	0.286	0.114	03 : 01 : 26
ocr2_states_200000	45.50	0.279	0.110	06 : 04 : 01
test2.source	73.74	0.129	0.048	−− : −− : −−

TABLE 4.3 – Version 2 evaluations for the NMT model

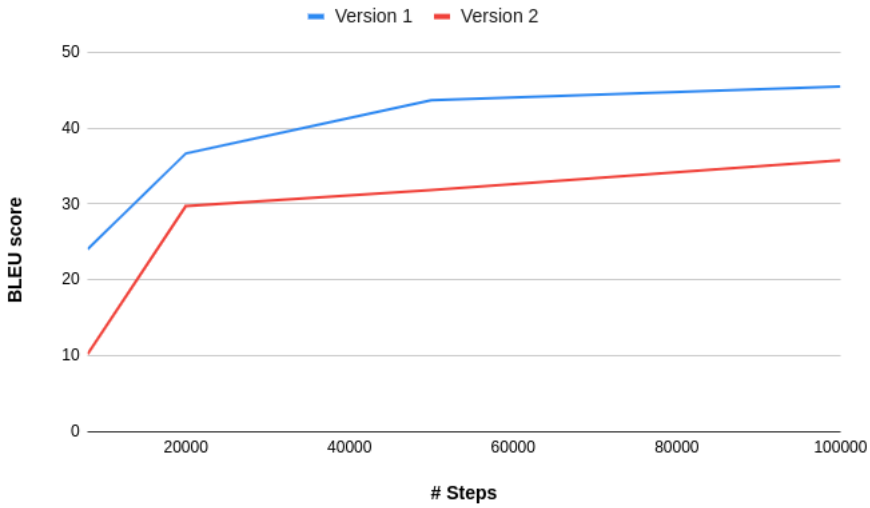

FIGURE 4.1 – BLEU score compared to training length

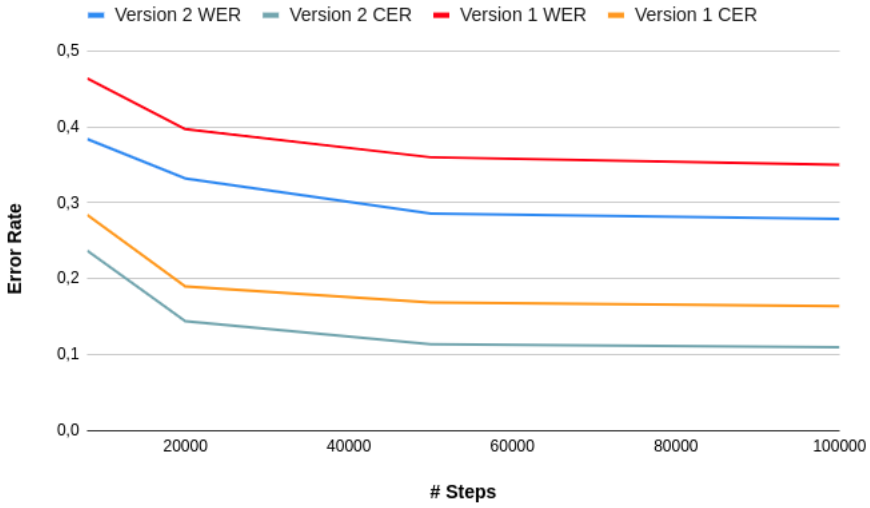

FIGURE 4.2 – WER and CER compared to training length

5. Discussion

5.1 Lexicon-lookup

5.1.1 Observations

The first version performed with a 0.145 WER and a 0.056 CER and took 56 minutes to execute for 1925 lines. This first version only took care of removing constraining character for the lexicon lookup (i.e. .,; : at the end of sentences) as well as removing parenthesis. Because some kind of tokens were still problematic and also because candidates were generated multiple times for the same keyword, some improvements were made to clean corpus before correction and to add a cache mechanism to speed up process on duplicate.

The second version performed better with a 0.138 WER and a 0.053 CER in around 47 minutes for a very similar corpus size. Hence the caching was effective which does not seem surprising as the OCR is generally consistent when generating erroneous tokens. For instance, the token "ð" is often a wrong output for the corresponding token "§". Unfortunately it is still hard to improve the WER or CER as many tokens, especially Greek ones, are recognized as erroneous because of their spelling variations and are then searching for incorrect candidates or does not found any at all.

Thus, the third and last version makes use of the ground truth to generate the lexicons and performed with a better evaluation than the OCR output which is already a small improvement. Obviously, using the ground truth in the lexicon resulted in an non significant improvement at running time as more tokens can find a match in the lexicon. This is especially true for complex tokens that were recognized correctly but would have little chance to be taken into account in a dictionary. It also proves useful for names as they would not appear in any dictionary either.

5.1.2 Improvements

Following the observations, the lexicon-lookup remains a safe bet but the level of attention and care that need to be done when generating lexicons, candidates or ranking them is a significant time consuming task and it requires some further improvements to perform correctly. The main enhancement would be to take into account spelling variations of Greek words. This could be done by analyzing similar texts or documents of the same era. Furthermore, by generating spelling variations in this manner it could also append frequencies to each words to help with the ranking procedure. At last, it would be a good idea to build a list of known classical names of the corresponding era or mythology as it would greatly reduces false positive.

5.2 Neural Network Machine Translation

5.2.1 Observations

On the NMT part the results were not delightful at first but further experimentation by increasing the number of steps revealed that the NMT alternative might not be totally useless. In fact, the BLEU score of *Version 2* increased by 10 at each doubling of the number of steps at the beginning but it became clear that it would reach an upper limit as we further augment the number of steps. At this point continuing to increase the training time would probably lead to overfitting and not increase performance at all. A similar observation can be made when watching the WER and CER as they drop faster at the beginning and then reached a lower limit in this case.

In the end, one of the most significant observation is the efficiency of the dehyphenate and lowercase module which granted an average increase of 11.07 on the BLUE score and an average decrease of 0.073 and 0.05 on the WER and CER respectively. The other one, was the significant drop, even with sufficient training time, of the evaluation metrics compared to the original OCR output.

5.2.2 Improvements

Even if results seems not satisfying at first sight, it must be reminded that it was a first try at using NMT for this kind of corpus. The implementation relies on an already trained model which is not adapted for this use and it could be further improved by using a model trained on multilingual documents or historical texts. Furthermore, fine-tuning a model which was trained for a more specific usage also comes with the benefit of modifying the architecture and parameters of the model. In this case, it was not possible to modify the architecture when continuing the training on the ocr.pt model but it would be interesting to test other configuration. Finally, it seems reasonable to think that increasing the amount of ground truth might enable to train the model further, thus pushing this upper and lower limits as later as possible. This could be made either by manual correction like done with the Jebb commentary or by using data augmentation techniques which were not explored in this project.

6. Conclusion

To summarize, both techniques experimented during this project resulted in interesting results. On one hand the classic lexicon-lookup performs quickly with good results but it requires more work and attention on problematic patterns and spelling variations. On the other hand, the NMT approach performed poorly but was experimented as a quick test and could profit from a more thorough implementation, especially by augmenting the training corpus or by fine-tuning a more specialized model.

Bibliographie

Articles

[1] Haithem Afli et al. « Using SMT for OCR Error Correction of Historical Texts ». In : (mai 2016), p. 962-966. URL : https://aclanthology.org/L16-1153.

[2] Eric Brill et Robert C. Moore. « An Improved Error Model for Noisy Channel Spelling Correction ». In : (oct. 2000), p. 286-293. DOI : 10.3115/1075218.1075255. URL : https://aclanthology.org/P00-1037.

[3] Kenneth W. Church et William A. Gale. « Probability scoring for spelling correction ». In : *Statistics and Computing* 1 (2 déc. 1991), p. 93-103. ISSN : 09603174. DOI : 10.1007/BF01889984. URL : https://www.semanticscholar.org/paper/Probability-scoring-for-spelling-correction-Church-Gale/d8d59ed%20bfde36fd640cec30b2537630f3e30ff7.

[4] Eva D'hondt, Cyril Grouin et Brigitte Grau. « Generating a Training Corpus for OCR Post-Correction Using Encoder-Decoder Model ». In : (nov. 2017), p. 1006-1014. URL : https://aclanthology.org/I17-1101.

[5] John Evershed et Kent Fitch. « Correcting noisy OCR : Context beats confusion ». In : *ACM International Conference Proceeding Series* (2014), p. 45-51. DOI : 10.1145/2595188.2595200.

[6] Mika Hämäläinen et Simon Hengchen. « From the Paft to the Fiiture : a Fully Automatic NMT and Word Embeddings Method for OCR Post-Correction ». In : (sept. 2019), p. 431-436. DOI : 10.26615/978-954-452-056-4_051. URL : https://www.aclweb.org/anthology/R19-1051.

[7] Ahmed Hamdi et al. « In-depth analysis of the impact of OCR errors on named entity recognition and linking ». In : *Natural Language Engineering* (mars 2022), p. 1-24. ISSN : 1351-3249. DOI : 10.1017/S1351324922000110. URL : https://www.cambridge.org/core/product/identifier/S1351324922000110/type/journal_article.

[8] Daniel Hládek et al. « Learning string distance with smoothing for OCR spelling correction ». In : *Multimedia Tools and Applications* 76 (22 nov. 2017), p. 24549-24567. ISSN : 15737721. DOI : 10.1007/S11042-016-4185-5/TABLES/3. URL : https://link.springer.com/article/10.1007/s11042-016-4185-5.

[9] Kimmo Kettunen et al. « Analyzing and Improving the Quality of a Historical News Collection using Language Technology and Statistical Machine Learning Methods ». In : (août 2014). URL : https://www.researchgate.net/publication/264858824_Analyzing_and_Improving_the_Quality_of_a_Historical_News_Collection_using_Language_Technology_and_Statistical_Machine_Learning_Methods.

[10] Ido Kissos et Nachum Dershowitz. « OCR Error Correction Using Character Correction and Feature-Based Word Classification ». In : (avr. 2016). doi : 10.48550/arxiv.1604.06225. url : https://arxiv.org/abs/1604.06225v1.

[11] Guillaume Klein et al. « OpenNMT : Open-Source Toolkit for Neural Machine Translation ». In : (juill. 2017), p. 67-72. url : https://www.aclweb.org/anthology/P17-4012.

[12] Minh-Thang Luong, Hieu Pham et Christopher D. Manning. « Effective Approaches to Attention-based Neural Machine Translation ». In : *CoRR* abs/1508.04025 (2015). arXiv : 1508.04025. url : http://arxiv.org/abs/1508.04025.

[13] Jiří Martínek, Ladislav Lenc et Pavel Král. « Building an efficient OCR system for historical documents with little training data ». In : *Neural Computing and Applications* 32 (23 déc. 2020), p. 17209-17227. issn : 14333058. doi : 10.1007/S00521-020-04910-X/TABLES/9. url : https://link.springer.com/article/10.1007/s00521-020-04910-x.

[14] Kareem Mokhtar, Syed Saqib Bukhari et Andreas Dengel. « OCR error correction : State-of-The-Art vs an NMT-based Approach ». In : *Proceedings - 13th IAPR International Workshop on Document Analysis Systems, DAS 2018* (juin 2018), p. 429-434. doi : 10.1109/DAS.2018.63. url : https://www.semanticscholar.org/paper/OCR-Error-Correction%5C%3A-State-of-the-Art-vs-an-Mokhtar-Bukhari/a4b47b46519e01115e697178770d3c27a7c99678.

[15] Thi-Tuyet-Hai Nguyen et al. « Survey of Post-OCR Processing Approaches ». In : (mars 2021). doi : 10.5281/ZENODO.4635569. url : https://zenodo.org/record/4635569.

[16] Matt Post. « A Call for Clarity in Reporting BLEU Scores ». In : (oct. 2018), p. 186-191. doi : 10.18653/v1/W18-6319. url : https://aclanthology.org/W18-6319.

[17] Ulrich Reffle et Christoph Ringlstetter. « Unsupervised profiling of OCRed historical documents ». In : *Pattern Recognition* 46 (5 mai 2013), p. 1346-1357. issn : 00313203. doi : 10.1016/J.PATCOG.2012.10.002. url : https://www.researchgate.net/publication/256822282_Unsupervised_profiling_of_OCRed_historical_documents.

[18] Carsten Schnober et al. « Still not there ? Comparing Traditional Sequence-to-Sequence Models to Encoder-Decoder Neural Networks on Monotone String Translation Tasks ». In : (déc. 2016), p. 1703-1714. url : https://aclanthology.org/C16-1160.

[19] Daniel van Strien et al. « Assessing the impact of OCR quality on downstream NLP tasks ». In : *ICAART 2020 - Proceedings of the 12th International Conference on Agents and Artificial Intelligence* 1 (2020), p. 484-496. doi : 10.5220/0009169004840496. url : https://dare.uva.nl/search?identifier=fd295310-eb9e-4271-9dad-862bd3f4eb57.

[20] Kazem Taghva et Eric Stofsky. « OCRSpell : An interactive spelling correction system for OCR errors in text ». In : *International Journal on Document Analysis and Recognition* 3 (3 2001), p. 125-137. issn : 14332833. doi : 10.1007/PL00013558. url : https://www.researchgate.net/publication/220163531_OCRSpell_An_interactive_spelling_correction_system_for_OCR_errors_in_text.

[21] Thorsten Vobl et al. « PoCoTo - An open source system for efficient interactive postcorrection of ocred historical texts ». In : *ACM International Conference Proceeding Series* (2014), p. 57-61. doi : 10.1145/2595188.2595197. url : https://www.researchgate.net/publication/266657594_PoCoTo_-_an_open_source_system_for_efficient_interactive_postcorrection_of_OCRed_historical_texts.

Web pages

[22] dorianj, LukeImhoff et ztane. *python-Levenshtein 0.12.2*. url : https://pypi.org/project/python-Levenshtein/. (accessed : 10.06.2022).

[23] guillaumekln et jsenellart. *pyonmttok 1.31.0*. url : https://pypi.org/project/pyonmttok/. (accessed : 08.06.2022).

[24] lvwerra et lysandre. *evaluate 0.1.1*. url : https://pypi.org/project/evaluate/. (accessed : 08.06.2022).

[25] *ocropus/ocropy*. url : https://github.com/ocropus/ocropy. (accessed : 10.06.2022).

[26] Bruce Robertson. *brobertson/ciaconna*. url : https://github.com/brobertson/ciaconna. (accessed : 10.06.2022).