OCR and post-correction of historical Finnish texts

Drobac, Senka

Linköping University Electronic Press
2017

Drobac , S , Kauppinen , P S & Linden , B K J 2017 , OCR and post-correction of historical Finnish texts . in J Tiedemann (ed.) , Proceedings of the 21st Nordic Conference on Computational Linguistics, NoDaLiDa, 22-24 May 2017, Gothenburg, Sweden . Linköping Electronic Conference Proceedings , no. 131 , Linköping University Electronic Press , Linköping , pp. 70-76 , Nordic Conference of Computational Linguistics , Gothenburg , Sweden , 22/05/2017 . < http://www.ep.liu.se/ecp/131/ecp17131.pdf >

http://hdl.handle.net/10138/229864

publishedVersion

OCR and post-correction of historical Finnish texts

Senka Drobac and Pekka Kauppinen and Krister Lindén
University of Helsinki
Department of Modern Languages
{senka.drobac, pekka.kauppinen, krister.linden}@helsinki.fi

Abstract

This paper presents experiments on Optical character recognition (OCR) as a combination of Ocropy software and data-driven spelling correction that uses Weighted Finite-State Methods. Both model training and testing were done on Finnish corpora of historical newspaper text and the best combination of OCR and post-processing models give 95.21% character recognition accuracy.

1 Introduction

In recent years, optical character recognition of printed text has reached high accuracy rates for modern fonts. However, historical documents still pose a challenge for character recognition and OCR of those documents still does not yield satisfying results. This is a problem for all researchers who would like to use those documents as a part of their research.

The main reasons why historical documents still pose a challenge for OCR are: fonts differ in different materials, lack of orthographic standard (same words spelled differently), material quality (some documents can have deformations) and a lexicon of known historical spelling variants is not available (although if they were, they might not give any OCR advantage for morphologically rich languages as noted by Silfverberg and Rueter (2015), but they can be useful in the post-processing phase).

The leading software frameworks for OCR are commercial ABBYY FineReader[1] and two open source frameworks: Ocropy[2] (previously known as OCRopus) and Tesseract[3]. Springmann et al. (2014) experiment with these three and compare their performance on five pages of historical printings of Latin texts. The mean character accuracy they achieve is 81.66% for Ocropy, 80.57% for ABBYY FineReader, and 78.77% for Tesseract.

However, Finnish historical documents are mainly written in Gothic (Fraktur) font, which is harder to recognize. The National Library of Finland has scanned, segmented and performed OCR on their historical newspaper corpus with ABBYY FineReader. On a test set that is representative of the bulk of the Finnish material, ABBYY FineReader's recognition accuracy is only 90.16%.

In this work we test how Ocropy performs optical character recognition on historical Finnish documents. We achieve a character accuracy of 93.50% with Ocropy when training with Finnish historical data. Additionally, we also wanted to find out whether any further improvement in the OCR quality could be achieved by performing OCR post-correction with an unstructured classifier and a lexicon on the Ocropy output.

Our experiments show that already with a relatively small training set (around 10,000 lines) we can get over 93% accuracy with Ocropy and with additional post-correction, the accuracy goes beyond 94%. With two training sets combined (around 60,000 lines), we get accuracy even over 95%.

1.1 Related work

In Springmann et al. (2014), they apply different OCR methods to historical printings of Latin text and get the highest accuracies when using Ocropy. Some work on Fraktur fonts has been reported in Breuel et al. (2013) where models were trained on artificial training data and got high accuracies when tested on scanned books with Fraktur text.

In Shafait (2009), alongside with the overview of different OCR methods, they present the architecture of Ocropy and explain different steps of a

[1] https://www.abbyy.com
[2] https://github.com/tmbdev/ocropy
[3] https://github.com/tesseract-ocr

Proceedings of the 21st Nordic Conference of Computational Linguistics, pages 70–76,
Gothenburg, Sweden, 23-24 May 2017.

typical OCR process.

Approaches to OCR post-processing are numerous and commonly rely on an error model for generating correction candidates. A language model may be incorporated to model output-level character dependencies. A lexicon can be used to determine which suggestions are valid words of the language – historical OCR may pose a challenge here if lexical resources are scarce. The post-processing method used in our work is described by Silfverberg et al. (2016) and can be described as an unstructured classifier. While the method is relatively simple from both a theoretical and computational points of view as it lacks a language model and a segmentation model found in many recently proposed approaches (see e.g. Eger et al. (2016), Llobet et al. (2010)) the classifier nevertheless captures the regularities of character-level errors occurring in OCR output and demonstrably improves the quality of the processed text. A more detailed comparison with other OCR-post processing methods can be found in Silfverberg et al. (2016).

2 Data and resources

2.1 Data

In our experiments, we use two data sets hereafter referred to as DIGI and NATLIB. Both are part of a larger corpus of historical newspapers and magazines that has been digitized by the National Library of Finland and consists of image files of individual lines of printed text as well the contents of said lines as plain text.

We created the DIGI data set as follows: first, a total of approximately 12,000 non-punctuational tokens were picked at random from the entire corpus. For each token, a random sentence containing said token was retrieved from the corpus together with information pertaining to the publication as well as as the page on which the sentence appears. The ABBYY METS/ALTO file and the corresponding image file of the scanned page on which the sentence appeared were retrieved from a repository, and the latter was subsequently cropped to only contain the line(s) on which the sentence occurred (i.e. the line segmentation was done by ABBYY FineReader). The contents of the lines were then manually written into a plain text file, which serves as the ground truth for training and testing our methods. Images whose contents could not be made out were excluded from the final data set, which consists of a total of over 12,000 image files and lines of manually edited ground truth data. This also gave us an fairly good idea of the overall quality of the corpus: we compared the ABBYY FineReader output with the manually edited ground truth and calculated a character error rate of 90.16%.

The NATLIB data set contains 54,087 line images and corresponding ground truth texts segmented from 225 pages of historical documents. The data was provided to us by the National Library of Finland.

2.2 Ocropy

Ocropy (previously known as OCRopus - Breuel (2008), Breuel (2009), Breuel et al. (2013)) is a leading open source software toolkit for training OCR models using long short term memory networks. In addition to character recognition, Ocropy offers tools for pre-processing documents (line segmentation, binzarization/normalization), tools for creation and correction of ground truth and evaluation tools.

2.3 Unstructured Classifier and Lexicon

We chose to perform OCR post-correction by using the unstructured classifier described by Silfverberg et al. (2016), as it was fast and easy to implement. The system is originally designed for correcting individual input strings and makes use of an error model that can be formulated as a series of weighted context-sensitive parallel replace rules implemented as a weighted finite-state transducer. Lexical lookup is used to validate or discard suggestions generated by the error model.

For validation, we used a modified version of the lexicon also used by Silfverberg et al. (2016), which in turn is a modified and extended version of a finite-state morphology of modern Finnish that has been specifically tailored to accept forms and spellings found in 19th century Finnish. We further modified this lexicon to accept strings with leading and trailing punctuation, as we found that punctuation often provides important clues for finding the correct substitution, which could be lost if the data was tokenized and the punctuation removed.

[3] https://github.com/tmbdev/ocropy

3 Method

In this section we describe the OCR process in its entirety. The method consists of three major parts: Data preparation, OCR and finally post-processing.

1. Preparing the data

   We divided the DIGI data set into three parts: 9,345 images and lines were allocated to training, 1,038 served as development data and the remaining 2,046 lines were reserved for testing. The motivation for splitting the data this way comes from practical reasons: we initially had separate sets of 10,383 and 2,046 lines, so we decided to take 10% from the bigger set as the development data, 90% as the training data and to use the smaller set for testing.

   The NATLIB data was on the other hand completely randomly split into three parts: 43,704 lines was used for training, 100 was used as development set and 5,308 as test set. In this case we used a very small development set because from our previous experience with DIGI data, we learned that recognition of large amount of lines can be quite slow. And since we had to do recognition for all saved models to find the best one, we decided to save time by reducing the size of the development set.

2. OCR

   (a) Since Ocropy works with black and white images files, the first step was to binarize our data. For this, we used the `ocropus-nlib` program with default settings, which alongside with binarization performs vertical normalization of the text. Example line images before and after binarization are shown in Figure 1.

   (b) Once the images were binarized, we used them together with ground truth texts to train neural network models. Training was performed with the `ocropus-rtrain` program which saved a model after every 1,000 iterations. We tested all those models on the development set and when the recognition accuracy stopped improving, we stopped the training. The model that achieved the highest accuracy on the development set was used for testing on the test sets.

   (c) Finally, prediction was done with the `ocropus-rpred` function. We tested the best model from both data sets on all test sets.

3. Post-processing

   (a) In order to train the error models used in post-correction, we ran the best OCR models on their respective training data sets and aligned the output with the corresponding ground truth data at the character level. The alignment was performed iteratively, with each iteration yielding a better alignment compared with the previous one.

   (b) The aligned data was then used to train a number of error models of varying sizes. The different models were obtained by varying the value of the threshold (T) i.e. the number of times a substitution must occur in a given context in the aligned training data in order to be included in the rule set and the final model. The resulting models were then tested on the development data set in order to determine the optimal value of T.

   (c) Finally, the training data sets and the development data sets were combined.

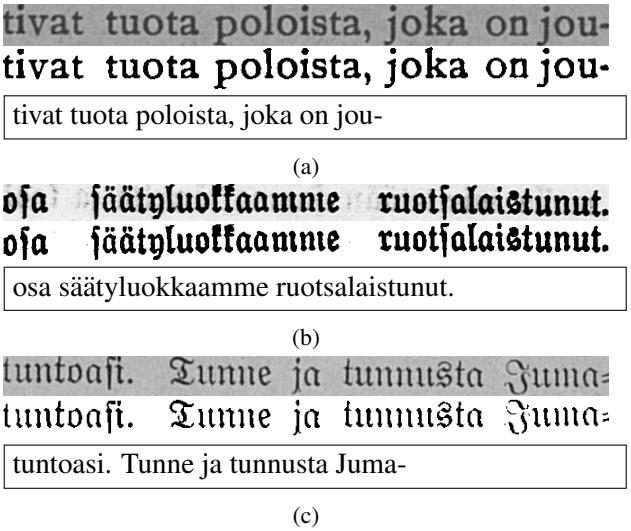


Figure 1: Example lines from DIGI test set. The first line shows the original scanned line, the second one is the binarized version, and the last one is the ground truth text.

The final error model was trained on this data with the threshold set at its optimal value. The resulting model was applied to the test data processed by Ocropy.

Due to the technical limitations of the post-correction method using lexical lookup, the input data was automatically split into strings which were post-processed individually. The input lines were split into tokens at blanks, and the output yielded by the post-correction system was then joined back into lines before calculating the character accuracy rate (see below).

4. Evaluation

   During both the prediction and evaluation phases, we measured the performance of the system by using character accuracy rate (CAR), which is essentially the percentage of correct characters in the system output and is a common metric in OCR-related tasks. It is the number of correct characters divided by the sum of correct characters and errors in the system output:

$$CAR = 100\% \times \frac{correct}{correct + errors} \quad (1)$$

   The number of errors is the overall Levenshtein distance (Levenshtein, 1966) between the system output and the ground truth and includes deletions and insertions.

   For languages with relatively long words such as Finnish, character accuracy rates and character error rates are arguably better indicators of the overall quality of the text than, for instance, word error rate, since longer words are more likely to contain multiple errors and are thus more difficult to correct. The legibility of a text may actually improve considerably without any notable change in its word error rate.

4 Results

For testing purposes we initially used two OCR models: the DIGI model and the NATLIB model. Both models were trained on their respective training sets (the DIGI training set with 451,000 iterations and the NATLIB training set with 177,000 iterations).

Table 1: Character accuracy rates after OCR and post-correction: the DIGI model (first column) was trained on DIGI training data with 451 000 iterations while the NATLIB model (second column) was trained on NATLIB training data with 177 000 iterations. The first two rows (DIGI-dev and NATLIB-dev) show the model CAR on the respective development sets. The DIGI-test row shows CAR for each model on the DIGI test set, with the following two rows showing results after post correction. The NATLIB-test row shows CAR for each model on the NATLIB test set and the final two rows show CAR after post-correction.

	DIGI model	NATLIB model
DIGI-dev	94.16%	-
NATLIB-dev	-	**94.73%**
DIGI-test	93.50%	89.05%
Post corr. (DIGI)	**93.68%**	89.12%
Post corr. (NATLIB)	93.59%	89.13%
NATLIB-test	93.82%	93.59%
Post corr. (DIGI)	94.05%	93.81%
Post corr. (NATLIB)	**94.13%**	93.94%

Table 1 shows character accuracy rates (CAR) for different combinations of trained models and the two test sets (DIGI-test and NATLIB-test) as well as post-correction results.

Afterwards, we also tested three models trained on combined data from both training sets: one trained from scratch on all lines from DIGI and NATLIB training sets, another trained on NATLIB train set with the DIGI model as the starting point and the third one was trained on the DIGI train set with the NATLIB model as the starting point. We chose the best models by calculating average CAR on both development sets.

Table 2 shows CAR for those three models tested on DIGI and NATLIB test sets as well as post-correction results.

Since the OCR accuracy on DIGI-test differs substantially depending on the model, we performed further analysis of the results on this test set using the `ocropus-econf` program. This is one of the Ocropy's evaluation programs which performs different kinds of evaluation tasks.

Tables 3 and 4 show the ten most common recognition mistakes on the DIGI test set. The first column in the tables gives the frequency of

Table 2: Character accuracy rates after OCR and post-correction of models with combined training data: (1) the model was trained from the beginning on both training data combined, (2) model was trained on NATLIB data with the DIGI model as starting point, (3) model was trained on DIGI data with the NATLIB model as starting point. The first row show the average character accuracy rate that the models scored on the two development sets. The following two rows show CAR on the DIGI test set before and after post correction. The last two rows show CAR on the NATLIB test set before and after post correction.

	(1)	(2)	(3)
Dev (avg)	**94.47%**	93.62%	93.62%
DIGI-test	93%	91.88%	92.32%
Post corr.	**93.27%**	92%	92.57%
NATLIB-test	94.83%	94.25%	93.68%
Post corr.	**95.21%**	94.56%	94.01%

the mistakes, the second column the recognition result and the third one the ground truth. Deletions are marked with "__" in the OCR column while insertions with "__" in the ground truth column. The most common mistake the DIGI model makes on the DIGI test set is insertion of spaces, which happened 122 times. Similarly, the most frequent mistake the NATLIB model made on the same test was deleting a hyphen symbol "-", which happened 663 times. One example when the hyphen was not recognized and simply left out is shown in Figure 1c. To better understand the severity of the mistakes, it is good to know that the DIGI test set has in total 78,116 characters.

Table 5 shows the frequency of Ocropy recognition mistakes per line for each model. The test set in both cases was DIGI-test. The DIGI-model does 100% correct OCR on 758 lines (37%) and the NATLIB-model on 351 lines (17%).

The best post-correction results were, due to the sparsity of the data, achieved by error models that were trained with high thresholds. This was especially the case with the NATLIB dataset, with threshold values between 60 and 70 yielding the best results for the development and test data sets. The resulting models could easily correct the most obvious cases without corrupting strings that were correct to begin with.

Table 3: A confusion matrix for the DIGI test set after recognition with the DIGI model (before post-correction)

Freq.	OCR	Ground truth
122		__
99	__	i
87	__	
78	u	n
60	i	__
43	__	-
41	__	t
41	.	,
36	l	i
31	e	o

Table 4: A confusion matrix for the DIGI test set after recognition with the NATLIB model (before post-correction)

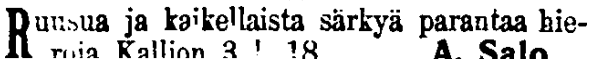
Freq.	OCR	Ground truth
663	__	-
324	__	
109	__	i
76	a	__
74	s	e
67	a	n
65	__	v
63	r	v
62	__	t
61	__	y

Table 5: Number of mistakes per line after recognition on DIGI test set (before post correction) with both models. The first column shows the number of mistakes per line, the second column the frequency of lines after recognition with the DIGI-model and the third column the frequency of lines after recognition with the NATLIB-model

Mistakes per line	DIGI-model n.° lines	NATLIB-model n.° lines
0	758 (37%)	351 (17%)
1	427 (21%)	509 (25%)
2	273 (13%)	292 (14%)
3	160 (8%)	180 (9%)
4	119 (6%)	162 (8%)
...	...	...

GT:	musta, nelisolkinen naisen nahkawyö
DIGI:	ew.O:
NATLIB:	e.":.:s'

(a)



GT:	Ruusua ja kaikellaista särkyä parantaa hie-
DIGI:	,:
NATLIB:	::,:

(b)

GT:	GIN LEIPURI- & KONDIITORI
DIGI:	.tt 1h110 Ml. t 00khron
NATLIB:	u 1.kL10Ml. K0tte4

(c)

Figure 2: Example lines with a large number of mistakes for both models

5 Discussion

A surprising result was that the NATLIB model performed poorly on the DIGI test set. Since the NATLIB model was trained on approximately five times more data than the DIGI model, we expected to get better results with this model. The main reason for the lower recognition rate was a large number of unrecognized hyphens, as shown in Table 4. Although both training sets were picked up from the same corpora, the reason for this phenomena could be that the NATLIB model represents 225 pages of consecutive text, while the DIGI model has a better distribution over the entire corpus.

After combining the two training data sets, we got significantly better results for the NATLIB test set, however the best results for the DIGI test set were still gained by the model trained on the DIGI data solely.

Since the models are trained on lines, a major problem for OCR is incorrect line segmentation. For example, Figure 2 shows three example lines on which both models performed extremely poorly. The first two images Figure 2a and Figure 2b have been incorrectly segmented - there is too much information from the following line caused by a large starting letter. Our future work is to put more focus on segmentation and preparation of representative data. We should use a neural network model that is trained on both artificial as well as real data to see if a bigger data set could incorporate correct artificial data for currently observed problems.

The other big problem are images with rare fonts (rare in the sense that they are not well represented in the training data). In the third image (Figure 2c), part of the text has been cut off by the incorrect segmentation, but since visible characters were not recognized, we believe that the main reason for the poor OCR result is the font. This kind of font is not very common in Finnish historical documents that the models have been trained on, so it is not recognized. This problem could be solved by adding more fonts to the training data.

The unstructured post-correction method gave a small improvement, which was interesting and by no means self-evident: it could have turned out that the remaining errors were so spurious that no further regularities could be extracted. This suggests that there is room for improving the neural network to incorporate the benefits provided by the post-correction method.

The biggest problem for our current post-correction is that it cannot influence spaces and word boundaries and they seem to be a major source of errors. A task for future work is therefore testing a structured post-correction method using more advanced line-oriented post correction.

6 Conclusions

Our experiments show that already with a relatively small but well-chosen training set (around 10,000 lines), we can get a character accuracy rate of more than 93% with Ocropy and with additional unstructured post-correction, the accuracy goes beyond 94%. With combination of the two training sets and with additional unstructured post-correction, the accuracy reaches 95.21%.

References

Thomas M Breuel, Adnan Ul-Hasan, Mayce Ali Al-Azawi, and Faisal Shafait. 2013. High-performance OCR for printed English and Fraktur using LSTM networks. In *2013 12th International Conference on Document Analysis and Recognition*, pages 683–687. IEEE.

Thomas M Breuel. 2008. The OCRopus open source OCR system. In *Electronic Imaging 2008*, pages 68150F–68150F. International Society for Optics and Photonics.

Thomas Breuel. 2009. Recent progress on the OCRopus OCR system. In *Proceedings of the International Workshop on Multilingual OCR*, page 2. ACM.

Steffen Eger, Tim vor der Brck, and Alexander Mehler. 2016. A comparison of four character-level string-to-string translation models for (OCR) spelling error correction. *The Prague Bulletin of Mathematical Linguistics*, 105:77–99.

Vladimir I. Levenshtein. 1966. Binary codes capable of correcting deletions, insertions and reversals. *Soviet Physics Doklady*, 10:707.

R. Llobet, J. R. Cerdan-Navarro, J. C. Perez-Cortes, and J. Arlandis. 2010. OCR post-processing using weighted finite-state transducers. In *2010 20th International Conference on Pattern Recognition*, pages 2021–2024, Aug.

Faisal Shafait. 2009. Document image analysis with OCRopus. In *Multitopic Conference, 2009. INMIC 2009. IEEE 13th International*, pages 1–6. IEEE.

Miikka Silfverberg and Jack Rueter. 2015. Can morphological analyzers improve the quality of optical character recognition? In *Septentrio Conference Series*, number 2, pages 45–56.

Miikka Silfverberg, Pekka Kauppinen, and Krister Lindén. 2016. Data-driven spelling correction using weighted finite-state methods. In *Proceedings of the SIGFSM Workshop on Statistical NLP and Weighted Automata*, pages 51–59, Berlin, Germany, August. Association for Computational Linguistics.

Uwe Springmann, Dietmar Najock, Hermann Morgenroth, Helmut Schmid, Annette Gotscharek, and Florian Fink. 2014. OCR of historical printings of latin texts: problems, prospects, progress. In *Proceedings of the First International Conference on Digital Access to Textual Cultural Heritage*, pages 71–75. ACM.