

DATeCH 2017 – PoCoTo Workshop – Profiler

Florian Fink & Uwe Springmann

Centrum für Informations- und Sprachverarbeitung (CIS)
Ludwig-Maximilians-Universität München (LMU)



May 31, 2017

- OCR'ed historical Documents contain errors.
- Historical documents contain lots of spelling variation.
- If you want to find errors in OCR'ed documents, you need a fitting historical dictionary.
- If you only have a modern dictionary, you will get a lot of false positives (*vnnd*, *Thurm*, ...).

The language Profiler was created to help to find OCR-errors in OCR'ed historical documents and to generate correction candidates for suspected errors¹.

- The profiler tries to distinguish real OCR-errors from historical spelling variants.
- The profiler can detect OCR-errors in historical patterns.
- The profiler uses various modern dictionaries.
- The profiler uses a (language-dependent) pattern list, that describes historical spelling variations.
- The profiler generates a document-dependent profile (the *document profile*).

¹Ulrich Reffle, *Algorithmen und Methoden zur dokumentenspezifischen Analyse historischer und OCR-erfasster Texte*, 2011

- The profiler is documented in the [profiler manual](#) (included in the workshop's data package).
- Its source is available on [github](#).
- The profiling web service, that is used by PoCoTo is also available on [github](#).
- It needs Linux and different C++ development tools (cmake, make, g++, boost, xerces, ...).
- The profiler contains various tools that compile different forms of dictionaries, perform lookup in different dictionaries and generate correction candidates for unknown words.

The profiler uses a so called *language back-end*, that contains the language dependent resources for the profiler. A minimal language back-end contains at least:

- A configuration file
- A collection of historical and modern compiled dictionaries.
- A historical pattern file
- A frequency list of a historical patterns from a ground truth²

²This is a bug, since this resource should be purely optional.

- Dictionaries must be compiled from plain text files using the `compileFBDic`.
- The historical pattern file is a plain text file that lists various spelling variation patterns in the form: `modern:historical`.
- The frequency list must be compiled from a historical ground truth using the `trainFrequencyList` tool³

³You can use a small garbage file if you do not have a appropriate historical ground truth.

- The configuration file sets some variables for the profiler.
- Its main purpose is to set up the profiling process.
- It defines which pattern file to use
- It defines the dictionaries to use and the order of their evaluation.
- There is a simple configuration file on [github](#).

```
...  
# Dictionary and Pattern settings  
[language_model]  
patternFile = "${:PATH}/patterns.txt"  
...  
# RANK 0  
[dict_modernExact]  
path = "${:PATH}/modern.fbdic"  
histPatterns = 0  
ocrErrors = 0  
ocrErrorsOnHypothetic = 0  
cascadeRank = 0  
# RANK 2  
[dict_modernHypotheticError]  
path = "${:PATH}/modern.fbdic"  
histPatterns = 3  
ocrErrors = 2  
ocrErrorsOnHypothetic = 1  
cascadeRank = 2
```


- If you work with the profiler you will often recognize missing historical patterns.
- The simplest resource to update is the historical pattern list.
- It is a plain text file that can be edited.
- New patterns can be added easily to the pattern file.
- There are some example pattern files on [github](#).

The pattern file is a plain text file with one pattern per line. Each pattern must be of the form `modern:hist`. You can use `$` to mark end of words. Lines that start with `#` are ignored:

```
# patterns.txt
# cases are ignored!
# turm was often spelled thurm
t:th
# teil was often spelled theyl
ei:ey
# $ marks the end of words
lich$:lig$
bar$:lich$
```

- Dictionaries are compiled from plain text files.
- Each token is on its own line.
- The text files must be sorted in ascending order.
- To add entries to a dictionary, you
 - add the dictionary entry to the file
 - sort the file
 - compile the dictionary from the sorted file
- There are some example dictionaries on [github](#).

```
+-- my-language
|   +-- freqlist.binfrq
|   +-- modern.fbdic
|   +-- patterns.txt
|   +-- weights.txt
+-- my-language.ini
```

- With the profiler you can now profile your own documents.
- First of all you need a minimal language back-end.
- The language back-end contains at least:
 - A configuration file
 - A compiled modern dictionary
 - A pattern file
 - Two files that describe the historical ground truth (`weights.txt`, `freqlist.binfrq`)

- The profiler can profile plain text or DocXML documents.
- The command `profiler --config my-language.ini --sourceFile my-doc.xml --out_xml patterns.xml --out_doc my-doc-out.xml` starts the profiling.
- The profiler produces two output files:
 - 1 `patterns.xml` lists the assumed historical and OCR-error patterns and their occurrences in the document.
 - 2 `my-doc-out.xml` contains a DocXML file with correction suggestions for the unknown tokens.
- The profiler has some more command options (`profiler --help`).

```
<token token_id="16" isNormal="true">
  <ext_id>16</ext_id>
  <wOCR>vber</wOCR>
  <wOCR_lc>vber</wOCR_lc>
  <wCorr></wCorr>
  <cand>vber:{über+[(ü:v,0)]}+ocr[],voteWeight=0.9503,
    levDistance=0</cand>
  <cand>über:{über+[]}+ocr[(ü:v,0)],voteWeight=0.0489158,
    levDistance=1</cand>
  <cand>uber:{über+[(ü:u,0)]}+ocr[(u:v,0)],voteWeight=0.00...
    levDistance=1</cand>
  <!-- ... -->
</token>
```

```

(ocr:vber)
uber:{über+[(ü:u,0)]}+ocr[(u:v,0)]
^      ^      ^      ^      ^      ^
|      |      |      |      |      |
|      |      |      |      |      +- error pattern position
|      |      |      |      + ---- error pattern (correct:ocr)
|      |      |      +----- hist pattern position
|      |      +----- hist pattern (mod:hist)
|      +----- correct modern version
+----- correction candidate

```

- The profiler web-service is a wrapper around the profiler with various language back-ends.
- It offers a SOAP-based web interface to profile documents.
- Its documentation is in the [profiler manual](#)
- Its source code is on [github](#).
- PoCoTo can connect to such a web-service to profile a document.

- You can configure the URL of the web-service going to Tools->Options->Profiler
- The default URL points to a profiling web-service hosted by CLARIN-D.
- The CLARIN-D web-service has language-back-ends for German, Latin and Greek.

- You can profile your current project by clicking Profiler->Order document profiler in the menu area.
- You will be asked which language back-end to use
- Select a language and click Order document profile.
- depending on your document and the settings of the profiler the profiling can take some time.
- After the profiling has stopped, you now will have access to the common error pattern tab in the error area and you will get a list of correction suggestions if you try to correct a token.

If the profiler web-service does not support your language or if you want to improve the language resources, you can profile the document manually and import it back into PoCoTo.

- ❶ Export your project File->Export->Export as DocXML.
- ❷ Run the profiler on the DocXML file.
- ❸ Import the two output files of the profiler back into PoCoTo.

Thanks for your attention!