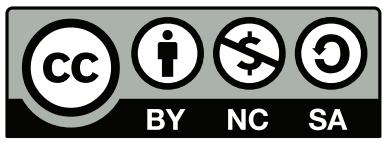
Module 9
The CIS error profiling technology

Florian Fink

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



2015-09-15

Introduction to postcorrection

OCR on historical documents

- Even state-of-the-art OCR engines introduce detection errors to the digitalization process.
- The detection rate of OCR engines on historical documents is in general worse than on modern documents due to:
  - bad quality of the original documents
  - bad quality of the scans
  - unusual fonts
  - unusual characters
  - historical spelling
- For a later scientific work on the documents, the results of the digitalization must be further improved
- The results of the OCR must be manually verified and corrected

Error detection

- Modern word processors support include powerful spellchecker to help the user to produce (mostly) error-free text
  - They detect misspelled words and mark them in the text
  - They automatically generate a list of corrections for the user
- Error correction systems use dictionaries in order to detect misspelled words and to generate correction lists
- Through the inclusion of a word context, even correct dictionary entries in a wrong context can be detected

The spell checker poem

The spell checker poem shows the weakness of word based error detection without context:

```
I have a spelling checker
It came with my PC
It highlights for my review
Mistakes I cannot sea.

I ran this poem thru it
I'm sure your pleased to no
Its letter perfect in it's weigh
My checker told me sew.
```

Error detection on OCRed historical documents

- Spellchecker as used in word processors can be used to detect misspelled words.
- They need at least dictionary of the document's language
- A language model of the document further improves the error detection.
- For historical documents dictionaries and language models are scarce
- The OCR further complicates the detection errors into the documents:
  - is a unknown word a historical spelling variant?
  - was the unknown word introduced by a erroneous character recognition?
  - do both factors overlap?

Basic error detection and correction

Word based error detection and correction

Given a dictionary of all words in a language the spell checker searches every word in the dictionary

- If an according dictionary entry is found, the word is correct
- If an according dictionary entry is *not* found, the word is marked as a possible error
  - Each word of the dictionary is compared to the misspelled word
  - Dictionary entries that are *similar* to this word are selected
  - The selected dictionary entries are provided as correction suggestions for the misspelled word.
- To compare words different kinds of word distance measures – like the *Levenshtein distance* are used.

Levenshtein distance

- The Levenshtein distance is a common metric to measure the distance between two words.
- It is defined as the *minimal number of character level edits* that convert one word into another.
- Character level edits include:
  - Substitution of one character to another
  - Insertion of one character
  - Deletion of one character

For example the Levenshtein distance between *kitten* and *sitting* is 3:

- kitten → sitting (substitution of *k* with *s*)
- sitten → sitting (substitution of *e* with *i*)
- sittin → sitting (insertion of *g* at the end)

Context sensitive error detection

- The context of words is represented, using so-called *N-Gram* language models for the different languages.
- N-Gram models count overlapping sequences of N words in big language corpora to calculate the probability of word contexts.
- These probabilities are then used to identify unlikely word sequences in the input documents.

The language profiler

Overview

- Dictionaries and language models for historical languages are scarce.
- They are needed to do error detection and correction on OCRed historical documents, though.
- The language profiler detects errors and generates correction suggestions for misspelled words.
- It mainly just needs a modern dictionary and a list of spelling patterns to work.
- The profiler can be supplied over the web as web service.
- The profiler and the profiler web service are is documented in the profiler manual.

Language profiles

As any spellchecker, the profiler must be configured for a specific language – the so-called language profile. You can generate such language profiles for your documents. You will need:

- At least one dictionary of modern words
- A list of patterns that describe the differences between modern spelled and historical spelled words
- A (small) historical ground truth training file

Basic workings

- For any given language profile and input file, the profiler identifies unknown words using:
  - the modern dictionary of the language profile.
  - hints from the OCR engine in the input document.
- To find correction suggestions, it uses the Levenshtein distance of misspelled words and dictionary entries.
- For all unknown words in the input, it tries then to apply pattern rules to generate valid dictionary entries and calculates the weights for the different patterns.
- It iterative optimizes these weight calculations onto the whole document.
- It gives out a list of the most common pattern encountered in the document
- It gives out the input document augmented with a list of correction suggestions.

Getting your hands dirty

Overview

- I will give you just the basics here – You should read the documentation if you intend to use the profiler.
- The profiler is developed on/for the Linux Operation System – you should have access to such an OS.
- The profiler is a command line tool – you should be able to use the command line.
- The profiler is not provided in a package but as source code – you should be able to use a C++ compiler and the `make` utility.
- The profiler relies on external tools an libraries – you should be able to install these requirements accordingly.

Installing the profiler

- The source profiler is available through its github repository
- You will need a C++ compiler the additional Xerces-c XML, java and boost libraries on your system.
- Compile the source code of the profiler using `make` on the provided Makefile
- Install the profiler and its additional tools locally to your home directory.
- The installation will install:
  - The profiler executable `profiler`
  - The dictionary compiler `compileFBDIC`
  - The language profile training executable `trainFrequencyList`

Building a language profile

- You need one (sorted) modern dictionary, one pattern file and one historical training file
- Compile your dictionary:

```
$ compileFBDIC dict.txt dict.fbdic
```

- generate the language profile configuration file:

```
$ profiler --generateConfig > language-profile.ini
```

- edit the configuration according to the documentation in the profiler manual and add your compiled dictionary
- generate the initial weights for your language model:

```
$ trainFrequencyList --config language-profile.ini
```

The pattern file

```
# each line represents exactly one pattern rule
# each pattern rule consists of a
# modern pattern (left side)
# and a historical pattern (right side)
ä:ae
ü:ue
ö:oe
ss:s
u:v
n:nn
ß:s
ss:ß
f:ff
# ...
```

Using the profiler

- To use the the profiler on a plain text file use the command:

```
$ profiler --config language-profile.ini --sourceFile input.txt \
  --sourceFormat TXT --out_doc output.xml
```

- The profiler understands different input file formats:

```
$ profiler --config language-profile.ini --sourceFile input.xml \
--sourceFormat DocXML --out_doc output.xml
```

- The profiler produces different output files (at once):

```
$ profiler [...] --out_doc output.xml --out_xml corrections.xml \
  --out_html overview.html
```

- See the profiler manual for all possible command line options

Input and output file formats

The profiler understands among others two different input formats:

- The format `--sourceFormat TXT` specifies plain text files.
- The format `--sourceFormat DocXML` specifies DocXML (AbbyyXML) formatted files.

The profiler can produce three different output formats:

- The `--out_doc` option generates an output of your document in a format that `PoCoTo` uses.
- The `--out_xml` option generates a profile file, that contains the correction suggestions.
- The `--out_html` option generates a nice HTML file that displays the document, the errors and the correction suggestions.

The DocOut format

```
<!-- [...] -->
 <token token_id="16" isNormal="true">
   <ext_id>18</ext_id>
   <wOCR>Bibliothec</wOCR>
   <wOCR_lc>bibliothec</wOCR_lc>
   <wCorr></wCorr>
   <cand>Bibliotheca:{bibliotheca+[]}+ocr[(a:,10)],voteWeight=0.984376,\
         levDistance=1</cand>
   <cand>Bibliothece:{bibliothece+[]}+ocr[(e:,10)],voteWeight=0.0156243,\
         levDistance=1</cand>
   <cand>Bibliotheces:{bibliotheces+[]}+ocr[(c:,9)(es:c,10)],\
         voteWeight=2.99037e-27,levDistance=2</cand>
   <coord l="761" t="481" r="1413" b="557"/>
   <abbyy_suspicious value="true"/>
 </token>
<!-- [...] -->
```

The profile_out format

```
<!-- [...] -->
 <dictionary_distribution>
  <item dict="dict_modernexact" frequency="13978" proportion="0.65385"/>
  <item dict="dict_modernhypotheticerror" frequency="7399.75"
        proportion="0.346138"/>
 </dictionary_distribution>
 <spelling_variants>
  <pattern left="iu" right="ju" pat_string="iu_ju" relFreq="0.236162"
           absFreq="251.729"/>
  <pattern left="e" right="ae" pat_string="e_ae" relFreq="0.000882386"
           absFreq="13.2681"/>
 </spelling_variants>
 <ocr_errors>
  <pattern left="s" right="f" pat_string="s_f" relFreq="0.372328"
           absFreq="3248.2">
  <pattern_occurrences>
  <type wOCR_lc="poteftate" wSuggest="potestate" freq="9"/>
  <type wOCR_lc="nobiliflimum" wSuggest="nobilissimum" freq="1"/>
  <type wOCR_lc="duriflimis" wSuggest="durissimis" freq="1"/>
<!-- [...] -->
```

Thanks for your attention!