Exploring Historical Document Collections with HistoRadar

Alberto González Palomo[1]

Abstract. We present our use of Natural Language Processing techniques to guide the exploration of historical document collections and assist in note taking. Our application is polished enough for actual use by historians without special knowledge of NLP or computer programming.

We provide ready-to-use Java executable files for mainstream operative systems (Linux, Microsoft Windows, MacOS X), and the source code is freely available under the General Public License[5].

1 INTRODUCTION

Digitisation of historical document collections has enabled historians to search in their contents efficiently, provided that one knows before hand what to look for.

The next step is to use the increasing computing power available to improve the task of finding interesting topics without the time-consuming and distracting task of reading the whole content first.

One example of using Natural Language Processing techniques to improve discovery of interesting content is query expansion, where the search is extended to terms related to the ones given by the user. With our application we seek to go beyond that, to the point where the computer itself suggests search terms in a graphical way.

In our discussions with historians we identified the following basic workflow when working with a document collection:

1. Read documents in the collection
2. Collect interesting topics
3. Snowball method:
   (a) Read again, collecting notes about selected topics
   (b) Add findings to "snowball"
   (c) Follow leads
   (d) Iterate

We use Natural Language Processing to identify possible topics that are displayed in a custom user interface using a "radar screen" metaphor, and provide a user interface for one-click annotation into the snowball document.

The first version of this application was implemented as part of the seminar "Unlocking the Secrets of the Past: Text Mining for Historical Documents (WS 2009/10)" at the University of Saarland, and documented in the report [7]. It is now maintained and developed further by this author.

The source collection we chose was the British Cabinet Papers, made available by the British National Archives at `http://www.nationalarchives.gov.uk/cabinetpapers/`

[1] Sentido Research, email: alberto@s14h.com

Visiting that web page illustrates the motivation for our work: there is a search form that allows searching for keywords and restricting the date range, but where to start? What keywords could we expect to find in those papers? After all, unexpected findings would be the most interesting.

2 SOURCE TEXT ACQUISITION

The Cabinet Papers are in PDF format, and include both the scanned image and their own automatic transcription using OCR. We extracted the text transcription and worked with the resulting text files.

It is interesting to note that the oldest documents dating from 1910 were typeset and printed for the British Government so they had good quality, while more recent documents were often typewritten, occasionally with mistakes crossed-over and hand-written annotations and corrections which affected the OCR negatively, with the result of older documents being actually easier to work with than recent ones contrary to our expectations.

Most files contained several actual documents, so we needed to split them. To find the first line of each collated document, we took the notice they have in their first pages:

```
This Document is the Property of
His Britannic Majesty Government
```

Due to the high incidence of OCR errors, we took a probabilistic approach for matching that line of text. Many OCR errors are due to local damage to the document, such as faded ink, stains, or interference from annotations or stamps. Therefore we assume that the error rate is not uniform in the notice and some parts of it will match literally. One example of an actual line with errors is the following:

```
*iTfois Document is the Property of
Eis Britannic Majestyˆ Goyernmˆ tX
```

We build several regular expression patterns that match different parts of the notice, words in this case: "`\b`" means that the word begins or ends at that point, "`.*`" that anything can go there, and "` +`" means one or more spaces.

```
\bthis\b.*\bdocument\b.*\bproperty\b
\bdocument\b.*\bproperty\b.*\bhis\b +\bbritannic\b
\bproperty\b.*\bbritannic\b +\bmajesty\b
\bdocument\b.*\bproperty\b.*\bmajesty\b
\bthis\b +\bdocument\b.*\bgovernment\b
\bproperty\b +\bof\b.*\bgovernment\b
```

We compute the match score by counting the patterns that match, and dividing by the total number of patterns n to normalize the result

to the interval $[0, 1]$, and if the normalized score surpasses our threshold, we consider it a match. In this case a threshold of $1/n$ (triggered when the score is greater but not equal to it) seemed to work fine.

The notice we use as marker was sometimes repeated in the first pages of a document, so we also consider the size of the candidate text: if it is shorter than a certain limit (60 lines in this case), we assume it is not a document but a single page and we concatenate it to the next.

From 178 files we separated 2395, which works out to an average of around 16 documents per file but there was a high variance with some files only yielding one or two documents, and others up to 50.

The results we mention in this paper correspond to a small subset of the collection for practical purposes: it is possible to load the whole collection, but processing it takes a relatively long time: 22 seconds vs. 91 minutes. This subset is the first 66 documents as plain text files with a total of 1.6 MBytes and 258061 "words" as measured by the `wc` utility.

	Collection subset		
	Small	Complete	Factor
Size	1.16 MB	131.61 MB	113
Number of documents	66	2395	36
Number of "words"	258,061	29,407,687	114
Time	22 s	90:53 s	248

Table 1. Processing time with the Stanford NER engine in a computer with CPU Intel Core2Duo T9600 (2.8 GHz) and 4 GB of RAM.

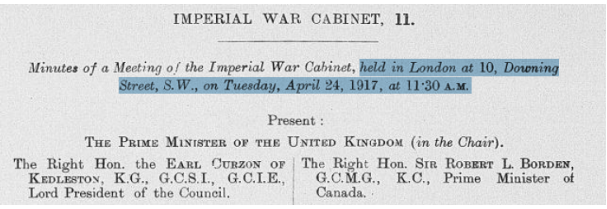
3 METADATA EXTRACTION

We need at least the date of the document to sort the collection chronologically. The metadata annotations in the source PDF files reflects their scanning date, not the date of the original document, so we extract that information from the OCR text as described in the report[3].

First we needed to correct some of the OCR errors. The main problem affecting the dates is spaces inserted between single characters (digits or letters) due to kerning (glyph separation). We correct for that with regular expression substitutions in five steps:

1. Remove spaces between single characters.
2. Split words with capital letters on them.
3. Separate numbers from words.
4. Remove spaces between abbreviations (identified by ending with a dot ".") and the dot.
5. Separate abbreviations from the next word.

We then look for the dates, take the first one as the one of the document, and normalize it.


IMPERIAL WAR CABINET, 11.

Minutes of a Meeting of the Imperial War Cabinet, held in London at 10, Downing Street, S.W., on Tuesday, April 24, 1917, at 11·30 A.M.

Present :

THE PRIME MINISTER OF THE UNITED KINGDOM (*in the Chair*).

The Right Hon. the EARL CURZON OF KEDLESTON, K.G., G.C.S.I., G.C.I.E., Lord President of the Council.

The Right Hon. SIR ROBERT L. BORDEN, G.C.M.G., K.C., Prime Minister of Canada.

Figure 1. Document header with location and date information.

Figure 1 shows a typical document header where we have highlighted the location and date.

Dates in this collection have a variety of forms, and is further complicated by the OCR errors. For instance, these are some we found:

```
Tuesday, 7th August, 1 9 4 5 , at 5 - 0 p.m. Street,
 S.W. 1
Thursday, lst. November, 1 9 4 5 , at 1 1 a.m.
Tuesday, lst January, 1 9 4 6 , at 1 1 a.m.
December 9, 1916, at 11-30 A.M.
March 1 , 1 9 1 7 , at 1 1 - 3 0 A . M .
Tuesday, June 5, 1917, at 11*30 A.M.
Tuesday, January 1 , 1 9 1 S , at 1 1 * 3 0 A.M.
Monday, April .1, 1913, at 1130 A.M.
Monday, July 1, 1018, at 12 noun.
Octdber 1, 1913, at 1T30 A.M.
Thursday,3 , 1 9 1 9 , at 12
Tuesday, July 1 , 1 9 1 9 , at 1 1 * 3 0 A.M.
Friday, June 8, 1917, at IF..30 a.m.
Friday, August 15, 1919, at 1 1 3 0 A.M
Friday, June 8, 1917, at IF..30 a.m.
Tuesday, January 2, 1940, at 11 A . M
WEDNESDAY, 21st JUNE, 1939, at 10030 a,m
MONDAY, 24th APRIL, 1939 at 5.6 p.m
WEDNESDAY, 15th MARCH, 1939, at 11.0 a.m
WEDNESDAY, 22nd MARCH, 1959, at 10.0 a.m
```

After applying the correction steps mentioned above, we get:

```
Tuesday, 7 th August, 1945 , at 5 - 0p.m. Street,
 S.W. 1
Thursday, lst. November, 1945 , at 11a.m.
Tuesday, lst January, 1946 , at 11a.m.
December 9, 1916, at 11-30 A.M.
March 1 , 1917 , at 11 - 30A . M.
Tuesday, June 5, 1917, at 11*30 A.M.
Tuesday, January 1 , 191S , at 11 * 30A.M.
Monday, April . 1, 1913, at 1130 A.M.
Monday, July 1, 1018, at 12 noun.
Octdber 1, 1913, at 1T30 A.M.
Thursday,3 , 1919 , at 12
Tuesday, July 1 , 1919 , at 11 * 30A.M.
Friday, June 8, 1917, at IF.. 30 a.m.
Friday, August 15, 1919, at 1130A. M
Friday, June 8, 1917, at IF.. 30 a.m.
Tuesday, January 2, 1940, at 11 A. M
WEDNESDAY, 21st JUNE, 1939, at 10030 a,m
MONDAY, 24th APRIL, 1939 at 5. 6p. m
WEDNESDAY, 15th MARCH, 1939, at 11. 0a. m
WEDNESDAY, 22nd MARCH, 1959, at 10. 0a. m
```

Adapting this to other collections is simply a matter of modifying the regular expressions used.

4 EVALUATION OF NER ENGINES

Apart from our own hand-crafted gazetteer, we integrated two different external NER engines that are independent of the collection content.

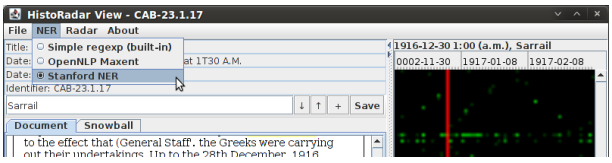


Figure 2. The Named Entity Recognition engine to use can be selected in the "NER" menu.

The initial one is the built-in "simple regexp" one we implemented with regular expressions. It has a hard-coded gazetteer that just searches for occurrences of some words like a few country names and some job titles we found in the Cabinet Papers. It would be easy

to read that list from a file instead of having it fixed in the source code, but preparing that file would still require prior knowledge about the collection contents which defeats the main purpose of our tool so we prefer to focus on the collection-independent NER engines.

The result of the Named Entity Recognition is a list of annotated segments. Other NER engines use similar data structures that we translate to our own so that all of them present a common interface to our application.

In the screenshot (Figure 2) we have selected the Stanford Natural Language Processing Group's Conditional Random Field NER engine [4]. The other one we have integrated so far is OpenNLP's Maximum Entropy classifier [1]. Details about this can be found in the NER section of the report[2].

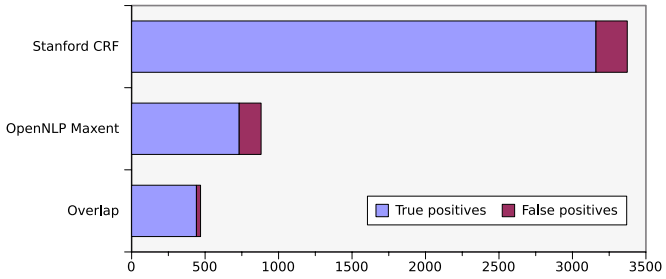


Figure 3. Number of entities found by each NER engine, and overlap between them.

In the collection subset, Stanford's CRF found 3373 entities and OpenNLP's Maxent found 881. Of those, there were 469 entities in common (overlap).

We then removed manually those that were obvious false positives and arrived at the figures shown in Figure 3. Most of those false positives were due to the OCR errors, like "Theregulationofthequantity", "Strictobservanceof" and "Bocriiioeat Is fes Property of His Bsitamiic Majesty". A few were attributable to the NER engine's limitations: "APPENDIX", "APPENDIX I. Draft Telegram", etc.

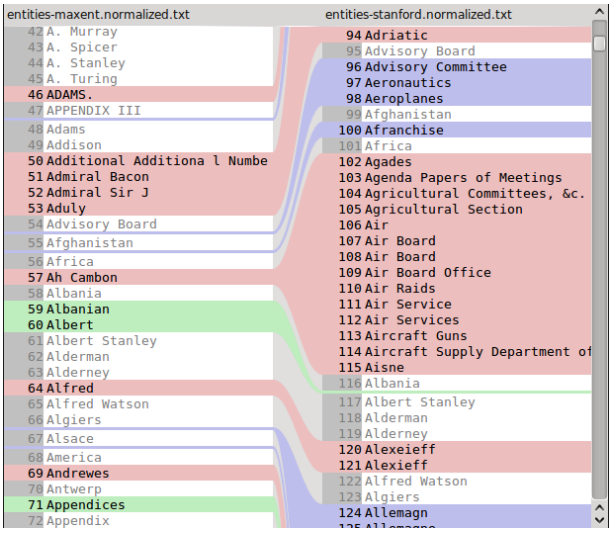


Figure 4. Comparison of Named Entities found by OpenNLP's Maxent (left) and Stanford's CRF (right).

Figure 4 shows some of the differences between the Named Entities found by each engine, displayed with a standard text diff application called "Kompare". The pink areas are substitutions, where some lines in the first file occupy the place of other lines in the second file. Green areas indicate lines from the first file that are missing in the second, and conversely blue areas are places where lines from the second file are missing in the first.

These results indicate that the NER engines we tried are more concerned with precision (fraction of entities found that are correct) than with recall (fraction of entities in the collection that were found).

For this application however the biggest issue is the false negatives (missed entities) as can be seen in Figure 5, where "War Cabinet", "French Government", "Italian Conference", "Chief of the Imperial General Staff", "the Admiralty", "the Greeks" are missed. This happens across the collection.

5 WORKFLOW AND USER INTERFACE

A document collection is built by putting the plain text version of the documents to be studied into a directory/folder. It can then be loaded with the menu option "File→Load collection", selecting that directory. All files with names ending in ".txt" will be loaded by the application.

The typical workflow would be loading the collection, loading the snowball document and working on it, then at the end of the session the snowball is saved to continue later. There is no need to keep another application open to write down notes as it can be done directly in HistoRadar.

The snowball can then be imported into a normal word processor (most current ones support importing HTML documents) to form the base of an essay.

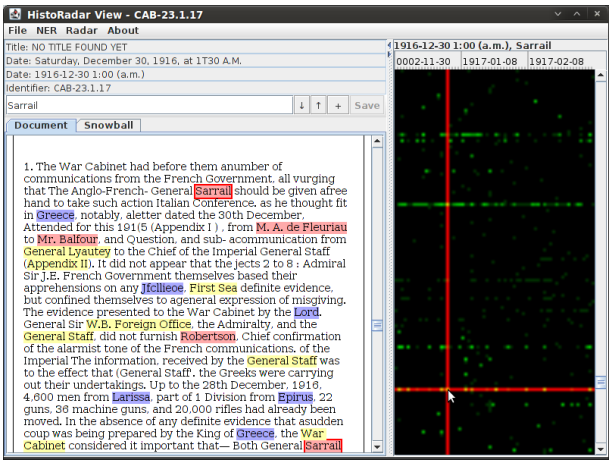


Figure 5. Appearances of "Sarrail" are the bright points in the horizontal line. Points in the vertical line represent different Named Entities in the same document.

The interface is split in two main parts: left is the document area, and right is the radar area.

The radar shows a high-level view of the whole document collection, and is the method used for navigating it. The "radar screen" represents all entities found in the documents using Named Entity Recognition. The horizontal axis represents time, with the timeline above the radar screen. The vertical axis represents the entities. Therefore, vertical lines are synchronic slices, and horizontal ones diachronic.

Clicking on the radar screen (Figure 5) loads the corresponding document, and copies the entity at the point to the search box which

selects its first appearance and scrolls the document view to show it. The arrow buttons next to the search box select the next and previous appearance respectively.

In the document area there are two documents in tabs. One is the current source document being explored, and another is the snowball where the working historian writes down annotations about his findings.

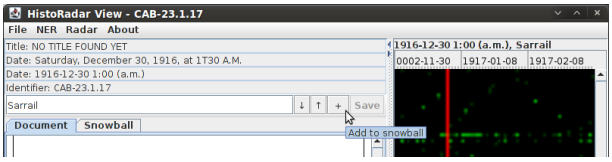


Figure 6. Clicking on the button labelled with a plus sign "+" adds an entry to the snowball for the current appearance of the selected entity, and advances to the next one.

Each entry contains some document metadata, some indications of the current location in the document (search query and result index, character position, percentage of total characters and page), a quotation around the current position for context, and an empty field for additional annotations.

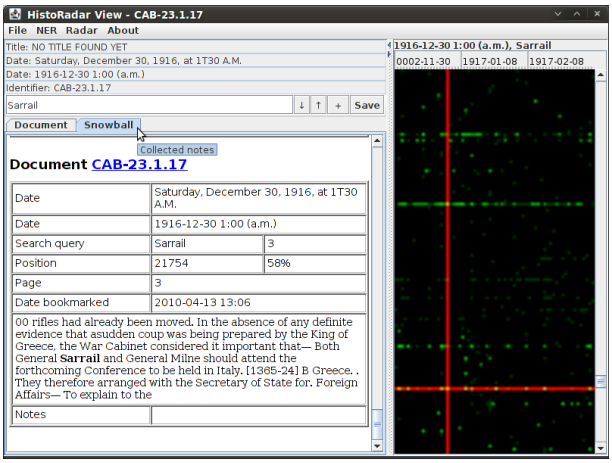


Figure 7. The snowball document with the last entry automatically added.

The amount of context copied in the quotation is all text in a 100-character radius from the entity rounded up to the next complete word. We tried first ending the quotation when reaching a new paragraph but we found that sometimes we were left with too little context for no practical advantage.

The snowball is an HTML document that can be edited directly in HistoRadar, modified by external tools, and loaded again to continue working on it.

More details about the user interface are available in the report [6].

6 EVALUATION AND CONCLUSION

Readily available Named-Entity Recognition libraries seem oriented towards great precision at the expense of recall. That is adequate for many tasks, but in this application, recall is more important than precision because the historian does not want to miss any potentially interesting event. Having some false positives is tolerable as long as they are few relative to the number of true negatives so using our program still saves a considerable amount of effort.

As we mention is Section 4 most of the errors in identifying named entities come from OCR errors. Those errors affect not only our application but others like keyword search too, so it is important to find ways to solve that problem in general. This is where a rich user interface can be of help: it could for instance compute some similarity metric among the entities (such as Levenshtein distance) and display them clustered according to it. This would allow the user to find them even in the presence of OCR errors. Search engines such as Google already suggest lexically similar keywords when they find a close alternative with high ranking. We could make use of extra information like semantic distance (used in query expansion) to position them in the radar screen plane.

The information we extract so far is just a count of mentions of each found Named Entity in the collection, for each document. We had planned to implement more refined information extraction and display the points where that information changed: for instance, if we had X-supports-Y with X being a member of the cabinet it would be interesting to look at places where it changes to, say, X-rejects-Y or X-supports-Z where Z was an alternative to Y.

This turned out to be too ambitious for now, but we keep our intention to explore what is possible in that direction.

The best results of this application are in the user interface area, where we successfully automated some of the historian's workflow. The automatic citation into the snowball document seems like a superior alternative to manual copy and paste, and the radar screen provides a quick way to start exploring historical document collections without any previous knowledge about their content.

REFERENCES

[1] Jason Baldridge, Gann Bierner, and Thomas Morton. The opennlp tools api. `http://opennlp.sourceforge.net/`, 2008.

[2] Uwe Boltz. Historadar seminar report: Named-entity recognition. `https://code.google.com/p/historadar/wiki/ReportNER`, 2010.

[3] Johannes Braunias. Historadar seminar report: Metadata extraction. `https://code.google.com/p/historadar/wiki/ReportMetadata`, 2010.

[4] Jenny Rose Finkel, Trond Grenager, and Christopher Manning, 'Incorporating non-local information into information extraction systems by gibbs sampling', in *43nd Annual Meeting of the Association for Computational Linguistics (ACL 2005)*, pp. 363–370, (2005).

[5] Free Software Foundation FSF. Gnu general public license. Software License available at `http://www.gnu.org/copyleft/gpl.html`, 1991.

[6] Alberto González Palomo. Historadar seminar report: Graphical user interface. `https://code.google.com/p/historadar/wiki/ReportUserInterface`, 2010.

[7] Alberto González Palomo, Johannes Braunias, and Uwe Boltz. Historadar seminar report. `https://code.google.com/p/historadar/wiki/Report`, 2010.