Improving readability and searchability of documents provided by the dutch government under the WOB

Author: Justin Bon
Information Studies - Data Science
University of Amsterdam
The Netherlands
justin.bon@student.uva.nl

Supervisor: Maarten Marx
University of Amsterdam
The Netherlands
M.J.Marx@uva.nl

KEYWORDS

Named Entity Recognition, Meta Data Extraction, Co-occurrence Network

GitHub

https://github.com/JustinBon/thesis/

1 INTRODUCTION

The goal of this thesis is to create a tool that will greatly improve the machine-readability of Freedom of Information act (Wob) documents. This was done with off-the-shelf tools so the only data that will be used in the project are the documents themselves for testing. The documents that were used in creating this tool came from 119 separate Wob request about the governments handling of the coronavirus pandemic (in a Wob request a civilian request documents from the government about a topic and the government giving those documents they deem relevant). Because the content of the data is not important to the project, just that it comes from a Wob request, any other set of Wob documents of sufficient size could also have been used.

This thesis is part of a bigger project done in cooperation with other parties that consists of three main parts. First is the text extraction from the documents. Second is splitting the documents. These two parts will be done by other parties. The third part, and the focus of this thesis, is the knowledge extraction. This was done using python [10], and the python packages pandas [11] and spaCy [5]. Two main methods were used for knowledge extraction: Named entity recognition (NER) and pattern matching.

2 EXPERIMENTAL SETUP

2.1 Data

The data used for this research comes from a Wob request and consist of files about the dutch governments handling of the Coronavirus pandemic. The data consists of documents from 119 Wob request about the coronavirus pandemic. The 119 requests combined consist of a total of 367 documents. The contents of these documents are varied but generally fall into one of two categories: decisions and appendices. The decision documents are the decisions of the relevant government ministries about whether to release the requested documents. The decision also states which documents will fall within the bounds of the request, reasoning and motivation for why some documents don't fall within the bound of the request, and motivation for limited censorship in the released documents. The censorship is done for the sake of privacy, so names, email addresses, and personal views are subject to censoring. There is one decision document per Wob request. The appendices are the actual documents that the government released. Usually this is just one or a couple of PDF files that consist of multiple smaller documents. These smaller documents can again be categorized:

- Information presentations of different ministries or companies
- Official government documents
- Reports
- lists of emails or other messages send or received by government officials or other smaller documents like memo's

The appendices are a compilation of some or all of these types of documents. Besides the decision and appendix, every Wob request also comes with an inventory list. An inventory list states for every document if it will be made public, partly public or not public at all. These list will not be used for this thesis.

The files have a wide range in sizes. Most of the files though are less than 1 MB. The average file size is 21 MB with a standard deviation of 103 MB. The standard deviation is so high because there are some very large outliers. The largest file for example is 1241 MB. There are also 28 files that have a size of 0. These files are corrupted and cannot be opened. All documents combined are 7.8 GB.

All of the data starts out in the form of PDF files. As stated above, these PDF's can contain a lot of other, smaller files. In a perfect world, all of these smaller files would have the original documents so that the text can be easily extracted. However, this is not the case for most of the PDF's. A lot of the documents are scans of paper documents or screenshots of digital ones. When this happens, text cannot be extracted in the normal way and the documents are not machine readable. Some of the PDF's contain a combination of text that can be extracted and text that can not be extracted. To test how much of the text is readable, every PDF documents was put through the PyPDF2 PDF file reader to extract all possible text. Then, the number of pages, text, and characters was calculated, the results of which are can be seen in figure 1.

Table 1 shows the state in which the data starts out. However, this is not the state the data was in when it was used for this thesis. As mentioned above, this thesis is part of a bigger project and other parties did the work on extracting this text. After that is done, the data is clean text on which to preform knowledge extraction.

2.1.1 Data preparation. This thesis has two main goals: To extract metadata from the documents and to extract named entities and create a co-occurrence network. Before this could have been done however, two data processing tasks needed to happen. First the text of the documents needed to be put in machine readable format, and

	File size	nPages	nWords	nChars
Count	367	367	367	367
Mean	21.8	88.76	2272.04	18126.71
std	103.11	320.73	6804.01	51009.42
Min	0	0	0	0
25%	0.18	3	0	0
50%	2.39	12	17	568
75%	12.23	72	958	9754.5
Max	1241.34	4910	62174	394633

Table 1: Description of raw data

	nPages	nWords	nChars
count	367	367	367
mean	95.556	26355.095	153460.921
std	320.654	88428.242	528658.209
min	1	131	615
25%	5	2075	12933
50%	17	5758	33914
75%	80	21753.5	117794
max	4910	1200227	7213469

Table 2: Description of cleaned data

second, the documents needed to be split into separate files. All of the data preparation and handling was done using Python [10].

As stated above, a lot of the PDF's are not machine readable. Usually this is the case because these are scans of physical documents or screenshots of digital documents like emails. These need to be converted to plain text. This was done by using Optical Character Recognition (OCR). An state-of-the-art OCR engine named Tesseract [8] was used for the process of text extraction, specifically the Python wrapper of Tesseract: pytesseract. Tesseract doesn't allow normal PDF files to be processed so the files first need to be converted to images. The python library PDF2image can do this automatically. Some of the documents are machine readable so for these cases the text can be extracted using the python library pyPDF2.

Once the text is extracted from the PDF files, it needed to be split into separate documents. As stated in the data section, most of the files are compilations of several other documents. For example, some appendix files only contain hundreds of emails. If any meaningful knowledge extraction needs to take place, these larger files needed to be split into separate documents. Once the documents were split, the data could be used.

2.1.2 Cleaned data. The OCR and the document splitting were done by other parties within the project, but they are included here anyway for the sake of clarity. This research worked with the results of these processes. These results were delivered in a CVS file format, which can be imported to a data frame with the python module pandas [7]. Every row in the data frame is one page in from a document. It contains the full name of the document the page is of, the page number, and the full text of the page. With this data frame a similar analysis was preformed as done on the raw PDF data. The results can be seen in table 2. When compared to table 1, it shows that there is a significant increase in all metrics (with the exception of file size as the cleaned data is not contained in separate files). It also shows that there are no files with that do not contain zero machine readable text.

2.2 NER

With the data machine readable, Named entity recognition is performed. This was done using spaCy [5]. SpaCy is a very powerful NLP engine for python. It allows the user to do a number of important steps in the NLP pipeline automatically. When text is fed into a trained NLP processing pipeline it will first split the text into tokens after which it will assign part-of-speech tags to said tokens. Then a parser assigns dependency labels to the tokens and last the named entity recognizer detects and labels named entities. There is also a lemmatizer and a text categorizer but these are not necessary for this research. SpaCy also allows for customization in the form of updating the model with new training data of adding new components to the pipeline like a pattern matcher. See the pattern matching section for more about the pattern matcher. SpaCy has language packs for 18 different languages but for this application only the dutch pack was used. The dutch language pack has a small, middle and large model. The small model is more efficient and the large model is more accurate. For the purpose of this thesis the large model was used as a shorter run time is not as important as an accurate model. The base spacy recognizes thirteen different categories for named entities:

- Numbers
- Dates
- Event
- Locations
- Languages
- Articles of law
- Money
- Organizations
- Percentages
- Persons
- Products
- Time
- Work of art
- Government agencies

Some key entities however aren't extracted by spaCy. This is why the updating the existing model of with new training data is an important feature. This allows the user to create a new category and input a piece of text to spaCy and tell it what words belong to the new category. The model will then relearn itself and update it into a new one. This new model can then be used on other pieces of text to recognize occurrences of the new category. This method was used for the recognition of the following new categories: dutch ministries, TBD . To do this, a selection of random pages was made. Every page was checked for instances of category that was being labeled. This was done for 500 pages for custom category. All of the found instances are then used to train a new model for every custom category. These models can then be combined with the base spaCy model to add the new categories to the NER pipeline. The results of running the NER pipeline were a list of tuples. Every tuple contained the match itself, the document it was found in, and the category to which the match belongs. The evaluation section details how the performance of the base model and all new custom models were measured.

add all cat
egories thi
method wa
used for

```
[
    {"LOWER" : {"IN" : days}, "OP" : "?"},
    {"IS_DIGIT": True},
    {"LOWER" : {"IN" : months}},
    {"IS_DIGIT": True, "OP" : "?"}
]
```

Listing 1: Pattern example

2.3 Pattern matching

As mentioned above, pattern matching can be added to the spaCy pipeline. With this feature some standard format meta data can be extracted together with the NER. For example, dates and times always have a easily recognizable pattern to them. If the document is an email the sender, receiver and subject are usually preceded by a keyword that indicates that. Pattern matching with spaCy works with tokens. A match to a pattern consists of a list of tokens that match specific constrictions. Listing 1 shows an example of such a pattern that matches was used to find all occurrences of a specific date format.

The pattern in Listing 1 matches a specific format of dates. The first line matches checks if a lowercase token is in a predefined list that contains all days. The second part of the line makes it optional. The second line checks if the current token is a number. The third line does the same as the first line but with a list of months and it is not optional. The last line checks if the token is a number. It is also optional. This pattern would match string a string like "maandag 11 april 2022". "maandag" is in the list of days, 11 is a number, "april" is in the list of months, and 2022 is a number. Because the first and last lines are optional variations like "11 april 2022", "maandag 11 april", and "11 april" would also match the pattern. Besides the spaCy pattern matcher, regular expressions were also used when necessary. This was the case when a pattern needed to be found within a single token. An example of this is when trying to extract dates and some dates are written in a format similar to "14-04-2022". This would be a single token for spaCy and cannot be found with its pattern matcher.

2.4 Co-occurrence network

The co-occurrence network was made with the results of the NER. The network consists of a number of nodes and edges. The nodes, in this case, are the found named entities. If these entities occur in the same document they also share an edge. The edges can contain some data: a number that indicates the amount of times the two named entities it connects have co-occurred in a document and; a list of documents in which the two entities occur together. Once the network is visualized that first number dictates the thickness of the edge. Creating and analysing this network was done with the python library NetworkX [4].

The visualization is made as a python web app using the Flask library [3]. This allows for a quick solution to make a web application. On the web side of the application, JavaScript was used to visualize the network. The JavaScript library eCharts can be used for the purpose of visualizing a network. All of the network analysis and the data processing were still done with NetworkX and python respectively. The reason for doing it this way instead of something like a distributable, command line program or python notebook is that making it a website makes it a lot more accessible to anyone who might want to make use of it such as journalists and interested civilians.

```
[
    {"LOWER" : {"IN" : days}, "OP" : "?"},
    {"IS_DIGIT": True},
    {"LOWER" : {"IN" : months}},
    {"IS_PUNCT" : True, "OP" : "?", "TEXT":'.'},
    {"IS_DIGIT": True, "OP" : "?"}
]
```

Listing 2: Dates pattern matcher

3 EVALUATION

set numbe
of matches
found

As of writing, three parts of the knowledge extraction can be evaluated. These are: the dates extractor, the ministries extractor, and the off-the-shelf NER model from spaCy. The performance of these extractors were measured in F_1 scores. The F_1 score is the harmonic mean between the precision and the recall. This blog post [1] lays out four different ways to calculate F_1 scores that were first introduced in SemEval'13. These are: strict (both category and span of the found match need to be correct), exact (only the exact span of a found match needs to be correct), partial (there only needs to be overlap in found match), and type(category of the match needs to be correct). For every extractor one of the four methods was used to evaluate it. Comparing to the ground truth matches can be correct, incorrect, partial, missing, or spurious. Because there is no ground truth available, comparing was done manually. This was done by continuously selecting random pages and running the extractors on them until enough matches were found to get a representative score.

3.1 Dates

To extract dates, a combination of spaCy's pattern matching and regular expressions was used. Listing 2 shows the spaCy pattern that was used for one specific date format. "Thursday 14 april 2022" is an example of that pattern. The name of the day and the year are optional. Both the english and the dutch versions of the names of the months and days are included in the pattern, as well as the abbreviations of the dutch days and months.

For date formats that do would not be captured by the spaCy matcher, the following regular expressions were used.

```
[0-3]{0,1}[0-9]\/[0-1]{0,1}[0-9]
[0-3]{0,1}[0-9]\/[0-1]{0,1}[0-9]\/[0-9]{2,4}
```

These regular expressions match with the following date formats: dd/mm and dd/mm/yyyy. Two variants were also made that have a dash as separator between the days months and years. Both days and months can also be just one number and the year can be written as two numbers.

The regular expressions in combination with the spaCy matcher was evaluated in the method described at the start of the evaluation section. It has a precision of 0.92, a recall of 0.891 and an f_1 of 0.905.

3.2 Ministries

It is easy to find correct matches for ministries when using gazetteers. The precision of this test is 0.882. Most all of the matches that were found are actually ministries. However the recall of the matcher is very bad at 0.377. Combined with the precision, that gives an F_1 score of 0.528. The matcher doesn't even find half of all the ministries that were in the text. The reason for this is clear. It has to do with lists of ministries and abbreviations.

Within the texts, a lot of lists of ministries are used. For example: "de ministries van VWSS, JenV, en BZK". In this case none of these will abbreviations of ministries will be matched. The matcher looks for "ministerie van name of ministerie". VWSS will not be matched because ministries is plural in stead of singular and JenV and BZK will not be matched because these are not preceded by "ministerie van ". There are three solutions for this, one being to not look for the preceding "ministerie van ", however, if this is done, all of those abbreviations will be matched outside of context. The abbreviation of the ministry of defence for example, is "def" so the matcher will find all occurrences of the three letters "def". With this modification the recall will actually increase to 0.458 but the precision will decrease to 0.122. The F_1 score of this is 0.193 so this is not a solution. The second solution only pertains to the abbreviations and it is to least find the VWSS in the example by also looking at the plural of "ministerie" but that decreased the precision more than it increased the recall. The last solution is to update the spaCy NER model with new manually labeled data. This was done as described in the NER section. To calculate the F_1 score for this the partial method was used. This ignores categorization and can have partially good matches. This choice was made because the category doesn't matter if there is just one and matches like "ministerie van economische zaken" instead of "ministerie van economische zaken en klimaat" would be partially correct as it is still known which ministry it is. This solution gave a recall of 0.791, a precision of 0.96, and an F_1 score of 0.867. This means that the last solution was used to extract ministries from text as it has the best performance.

3.3 spaCy

Before evaluating the NER model from spaCy, it can be helpful to look at other studies in NER to set a baseline. Lample et al. (2016) compiled performance scores of a lot of different studies and showed that, although the English NER models score very high with an F_1 scores around the 0.90 mark, the Dutch models do a bit worse [6]. The average F_1 score for the eight dutch models they compared is 0.776 with one of their own models scoring the lowest at 0.699 and the highest model being at 0.828 [2]. This gives a baseline of an F_1 score between 0.75 and 0.80 to compare the spaCy model to. The spaCy NER model was tested with a dutch NER test set [9] instead of manually like the other extractors. When using the exact method (boundary of match needs to be correct but not the category) for calculating the F_1 score the model scored very well. It performed with a recall of 0.87, and a precision of 0.963 resulting in an F_1 score of 0.914. When using the strict method (both boundary and category need to match) the performance lowers a bit to a recall of 0.724, a precision of 0.791, and an F_1 score of 0.756. This is still in the bounds of the baseline between 0.75 and 0.80. These are still good scores for the model, especially the first one. But it needs to be kept in mind that the performance of a NER model can depend on the sort of text. It could be that the spaCy model was trained on documents that have a very high resemblance to this test set. The performance on the WOB document can be lower than these results suggest.

REFERENCES

[1] Batista, D. S. Named-entity evaluation metrics based on entity-level, May 2018.

[2] Gillick, D., Brunk, C., Vinyals, O., and Subramanya, A. Multilingual language processing from bytes. *arXiv preprint arXiv:1512.00103* (2015).

[3] Grinberg, M. *Flask web development: developing web applications with python.* " O'Reilly Media, Inc.", 2018.

[4] Hagberg, A., Swart, P., and S Chult, D. Exploring network structure, dynamics, and function using networkx. Tech. rep., Los Alamos National Lab.(LANL), Los Alamos, NM (United States), 2008.

[5] Honnibal, M., and Montani, I. spaCy 2: Natural language understanding with Bloom embeddings, convolutional neural networks and incremental parsing. To appear, 2017.

[6] Lample, G., Ballesteros, M., Subramanian, S., Kawakami, K., and Dyer, C. Neural architectures for named entity recognition. *arXiv preprint arXiv:1603.01360* (2016).

[7] pandas development team, T. pandas-dev/pandas: Pandas, feb 2020.

[8] Smith, R. An overview of the tesseract ocr engine. In *Ninth international conference on document analysis and recognition (ICDAR 2007)* (2007), vol. 2, IEEE, pp. 629–633.

[9] Tedeschi, S., Maiorca, V., Campolungo, N., Cecconi, F., and Navigli, R. WikiNEuRal: Combined neural and knowledge-based silver data creation for multilingual NER. In *Findings of the Association for Computational Linguistics: EMNLP 2021* (Punta Cana, Dominican Republic, Nov. 2021), Association for Computational Linguistics, pp. 2521–2533.

[10] Van Rossum, G., and Drake Jr, F. L. *Python reference manual.* Centrum voor Wiskunde en Informatica Amsterdam, 1995.

[11] Wes McKinney. Data Structures for Statistical Computing in Python. In *Proceedings of the 9th Python in Science Conference* (2010), Stéfan van der Walt and Jarrod Millman, Eds., pp. 56 – 61.