Finding Files and Experts in Your Field:

How Solr Can Help

Elias Wipfli (13-123-922) [1] Julius Oeftiger (16-127-532) [2]
Brian Schweigler (16-102-071) [3]

Supervisor:

Prof. Edy Portmann

Assistant(s):

Minh Tue Nguyen Moreno Colombo Jhonny Pincay

31 December 2021

Human Smart Cities Seminar - Report

[1]elias.wipfli@students.unibe.ch, University of Bern
[2]julius.oeftiger@students.unibe.ch, University of Bern
[3]brian.schweigler@students.unibe.ch, University of Bern

Abstract

For every larger organization, and thus even the government itself, it is a challenging task to create a system, which finds relevant documents or an adequate expert on a chosen topic. This report introduces a small proof of concept, how an application can be created to provide a keyword search of uploaded documents, and finding the most relevant expert to answer questions on these topics. The application was created by using open-source libraries like Solr, NLTK, and Tika.

Keywords: Seminar report, Human-IST Research Institute, Human Smart Cities, Smart Governance, Solr,

Public Git Repository: https://github.com/Brian6330/Human_Smart_Cities.

Contents

1 Introduction

Cities are adopting Information and Communication Technologies (ICT) more and more into their services. Thus, the need to address challenges in such Human Smart Cities keeps rising. As the citizens are the main drivers of change, they must be included to best handle these challenges [8].

One such challenge that must be addressed is the dispersion of information among different systems and applications. Most importantly, this information must be found within a short time frame. If parts of the retrieved data are unclear, talking to an expert might be of importance. This leads to the follow-up issue of determining who is likely to be an expert in a specific field.

The Research Questions focused on are:

1. Given multiple text files, how can their contents be searched to return the best fitting ones?

2. How can a list of experts be extracted from the submitted text files?

3. The submitting user of a given text file, must not necessarily be the user who wrote it. How can this be considered when determining the expert list?

The remainder of this project report is structured as follows: the required background to understand certain terminology and methods used are explained in chapter 2, related works are mentioned in chapter 3, Solr is talked about in length in chapter 4, the methodology used is thoroughly explained in chapter 5, in chapter 6 the evaluation procedure is described, in chapter 7 the project results will be presented and in chapter 8 the report will be concluded while also providing a short outlook on possible extensions to the work.

2 Theory

As is customary in "hands-on" projects, the theory chapter will be rather short, due to the methodology (chapter 5) and the chapter on Solr (chapter 4) overlapping with the theory chapter.

2.1 Determining Experts

Who is an expert in a given topic? To determine this, the extracted text content is tokenized with the Natural Language Toolkit (NLTK) [7]. There are a lot of so-called "stop-words" in the English language [9], that must be filtered out, as they are not relevant to determine who is an expert in a given field. Examples of such stop-words that are filtered out: 'the', 'and', 'from'.

After this step, a list of the most frequent terms is returned. It is possible to limit the number of terms returned; currently, the limit is set to the top 50 occurrences only. To determine the score of the term, the earlier the term is within the expert's list, the more knowledgeable the expert is in regard to the chosen term. This can be expressed mathematically with index i and cut-off c:

$$\text{Score}(i) = \frac{c - i}{c}$$

To better illustrate this, we will look at an example where the $c = 50$:

$$\text{Expert } A \text{ has } \textit{snow} \text{ at index } 6 \rightarrow \text{Score} = (50 - 6)/50 = 0.88$$
$$\text{Expert } B \text{ has } \textit{snow} \text{ at index } 4 \rightarrow \text{Score} = (50 - 4)/50 = 0.92$$

Thus, Expert B would be more knowledgeable in regard to *snow* than Expert A in our scoring system.

2.2 Expert List Caveats

Despite its simplicity, the scoring method used to determine experts, coupled with our Solr-based approach, leads to a few caveats:

- Snow $\neq$ Snowing $\neq$ Snowfall $\rightarrow$ No stemming applied.
- Keywords belonging to one domain are not grouped as they should be.
  e.g. (Permafrost; Snow; Rain) $\in$ "Glaciology" or "Climatology".
- Scoring alternatives may be better suited.
- Assumption uploader = author $\rightarrow$ PDF Metadata not reliable.

While Solr has some stemming in regard to searches, most of them need pre-defined rules. Grouping by domain would also require defining what terms belong together (e.g. 'rain' and 'precipitation'). Regarding scoring, alternatives may be better suited for certain tasks, but it can always depend on the texts that are analyzed. Lastly, the assumption that the uploader is the author could be fixed with a pop-up when uploading a PDF, where one could correct or fill in the required information.

3 Related work

While there have been different approaches to dealing with the dispersion of data, we focus on two related works in the following: "Building a Hierarchical, Granular Knowledge Cube" by Alexander Denzler, Marcel Wehrle, Andreas Meier; "An expert/expert-locating system based on automatic representation of semantic structure" by Karen E. Lochbaum and Lynn A. Streeter.

3.1 Building a Hierarchical, Granular Knowledge Cube

This paper [5] proposes to build a knowledge cube with a hierarchical structure, where every lower level gets more granular and the uppermost layer is the coarsest one. A knowledge cube is composed of multiple granules on the multiple levels of the cube. These granules are composed of concepts, which have relations to other concepts on the same or different granular level. Similarly, the relations in regard to the concepts, the granules have dependencies to upper-level granules. Concepts are atomic information that are extracted from content with data-mining, which are then grouped with related concepts into a granule. The advantage of this knowledge cube is, that a domain may stretch vertically as well as horizontally, and through this, it may reveal structural regularities or irregularities. To fill the cube with knowledge, an automatic or semi-automatic approach should be chosen. The proposed cube should have three layers, where the first layer is for storing, the second for the representation of the knowledge, and the last layer is the structure of the knowledge.

3.2 An Expert/Expert-Locating System Based on Automatic Representation of Semantic Structure

This paper [10] proposes the idea of finding an expert with the use of input documents, which are then analyzed to create the corresponding expert system. The process first extracts the text of the documents and removes stop words or punctuation marks, then changes the plural words to singular and isolates compound words. By using the semantic structure analysis, which relies on singular value decomposition, the resulting EEL-system is better than standard keyword matching, because it also takes into consideration documents, which are related. Sadly, the system has a problem with negation and does not know how to deal with it.

4 Solr

Before we could start with our proof of concept, we needed an already existing framework that satisfies our demands. A good framework is Apache Solr [1]. Solr is an open-source search application built on Apache Lucene, which can handle indexing and text search on PDFs. Before making a query to Solr, it is necessary to upload the documents for indexing. Later, we can perform a search where Solr performs the following operations:

1. Querying
2. Mapping
3. Ranking

How documents are processed by Solr:

1. The searched document is converted into machine-readable form, which can be done as explained later by simply extracting all relevant information from the document.
2. The query terms have to be specified by the user, which is then used in the next operation.
3. The now machine-readable document is searched for a specified term or image.
4. A ranking is done depending on the search results.

We will later use this ranking and the already existing search feature of Solr.

4.1 Solr Setup

To start a Solr server, one could simply follow the Solr tutorial. After downloading Solr (We used version 8.10) and extracting it to e.g. the folder `Solr-Server`, the server can simply be started with the following command:

```
Solr-Server/bin/solr start -e cloud
```

This command starts the Solr server in cloud mode and the user has to enter further information like for example the port where the application is and how many nodes should be run. At the end, there is also a choice of which configuration should be used. Instead of the default configuration, the **sample_techproducts_configs** is used, because the additional request handlers of this provided configuration can then be used. What should be noted is, that the same command above can be used on a Windows computer, but if the **solr.cmd** is used instead, then there may arise the problem on how to close the started Solr server again. The solution would be to find with the command line the corresponding **pid** of the server and kill it directly because the regular Solr command to stop a server would not find this server.

Because these commands have to be made at the beginning of every start of the Solr server, and possible modification or restart, it is clear that this approach is error-prone. We asked ourselves if there is a method where we simply can start one single command, which sets the Solr server up without the need for user interaction. The technology which answers this demand is Docker (section 5.2).

4.2 Solr - Configuration

Solr is configured with a schema file. It tells Solr about the contents of documents it will be indexing. The schema file used is also **sample_techproducts_configs**, as mentioned prior, as it has a request handler defined for extracting text from documents. Text is extracted via Apache Tika, a toolkit that "detects and extracts metadata and text from [...] different file types (such as PPT, XLS, and PDF)" [4].

Solr uses its own data structure to save its content. It consists of a document containing multiple fields, each with a name and its content:

- id: A unique ID (UUID), that is created for each document upon upload.
- last_modified: The date the PDF was last modified, according to its metadata.
- content_type: What type of content the document is (e.g. 'application/pdf')
- author: The author of the PDF, according to its metadata.
- author_s: Secondary author field from the PDF, according to its metadata. Does not always exist.
- content: The by Tika extracted text, includes special symbols and line breaks (e.g. \n)
- title: The title of the PDF, according to its metadata.
- links: Links found in the PDF file, be it to external webpages or internal files and images used.
- subject: Often empty, extracted from the PDF metadata.
- _version_: An internal field used by Solr.

4.3 Solr Request handlers

"A request handler process request coming to Solr" [2]. The default request handler is 'select', which takes query parameters to search for terms. Another important one for this project is /update and /update/extract. Both can be used in conjunction with binary data to generate an HTTP POST request, allowing the pushing of new items to the Solr server. If we are dealing with a PDF or similar and want its content, only update/extract allows the extraction of the text content with Apache Tika.

4.4 PySolr

PySolr is a simple client library for Apache Solr based on the programming language Python. It allows the user to connect to an already running Solr server and query requests from the server with so-called request handlers or upload files to it. Open-source packages tend to have different extents of documentation and for this particular one, it is quite sparse. PySolr facilitated the communication between our scripts and the running Solr server.

5 Methodology

While Solr is a big part of the project, the other key players should not be neglected. The data used, docker as a technology, and the GUI shown to users, were essential parts of this proof-of-concept, which will be further explained below.

5.1 Data Source

For this project, the data used were PDFs created by "EnviDat: das Umwelt-Datenportal". It is one of the organizations that have open data on https://opendata.swiss. The PDFs used needed to be in English, have properly encoded text (so it can be extracted by Tika), and contain more than just images or diagrams.

In total, 18 PDF files were used with 6 different authors. Those with an empty author field were ignored for the expert list, but can still be perused when searching for keywords with Solr. While text analysis could have been done to determine authors from the first page of the PDF files or even from looking at the text formatting, it was decided that this would not be done for the proof-of-concept. Nonetheless, it is a valid extension and might also help to differentiate between text uploaders and text authors.

5.2 Docker

Docker [6] is a software that allows the user to deploy another software in a virtualized environment on the operating system. The deployed software is run in a so-called "container", which is like a small virtual machine with a standardized environment, where the dependencies are provided. This allows the user to run software on multiple platforms without the need to adjust the software for all possible operating systems, and instead, the user just has to make sure that Docker runs. Different from a regular virtual machine, a container does not have a high overhead and can use the underlying resources of the system better, which leads to a higher efficiency. Every deployed software is a single instance of an application, so Docker is very useful for scaling the demands of an application because to satisfy these demands, the number of instances of the application can be increased and decreased as needed.

A Docker image is a set of instructions of how the container is built and is comparable to snapshots for a virtual machine. The image contains the application code, libraries, dependencies, and other files which the software would need to run the application. Most images are stored on a so-called registry, most of them on the Docker registry. If an image is changed, then the difference from the old to the new image is stored on the registry, so it is layer-based. One could generate a custom image by setting up a Dockerfile, which is then executed, and the resulting image can be pushed to a registry. Luckily for our purpose to use Solr, there is an existing image on the Docker registry.

If we would simply use the Solr image with normal commands, then the user would have to not only start the container, but also execute a command to create a core for Solr, then another command to create the collection. This is rather troublesome because if one command is forgotten, the whole application won't work correctly. Docker Compose would be a method to run multiple containers at once, or in other words to start different containers sequentially which would then execute the needed commands. There is an example of Docker compose for Solr which we were able to use. First, it started a stand-alone Solr server and uses Zookeeper, which connects the different containers in a network together. But sadly, this example is incomplete, as we had to append the commands for creating a core and collection in an additional container. There is also a command where a directory can be selected, from which the content is uploaded to the running Solr server.

This customized Solr application can now be started, restarted, and closed pretty easily in our chosen IDE. But we had to drop this approach because the documentation was insufficient to help us how we could change the schematics of the core for uploading PDF files. These special schematics were needed, so that we could use PySolr and extract the text from an uploaded PDF file, but it is not available if Solr is started with Docker. Even though we had to discard Docker for our proof of concept, we would still advise the use of it in further human smart cities project, as explained above it removes the need to adjust software on multiple platforms. As of the writing of this paper, the documentation was slightly updated to include the Log4J vulnerability, but did not update the Docker example.

5.3 Querying / GUI

To showcase searching Solr for keywords, a simple GUI was implemented.

Ideally, multiple databases would host the different documents to simulate the dispersion of information. However, the concrete implementation of setting up and linking all those systems together lies outside the scope of this paper. Instead, the UUID can be interpreted, as mentioned in chapter 4.2, as the identifier to the database hosting the document. A scheme of how this could be envisioned is visible in the following figure 1:

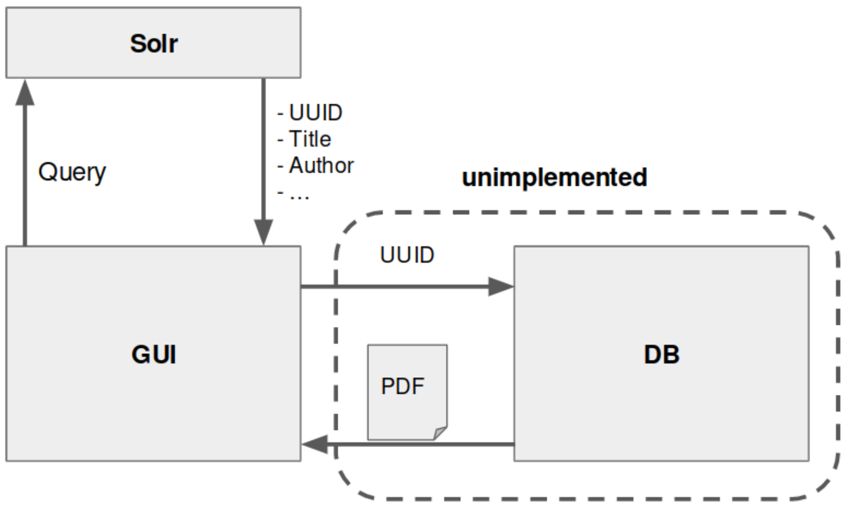


Figure 1: GUI communication scheme. The database linking is not implemented but showcases the envisioned functionality

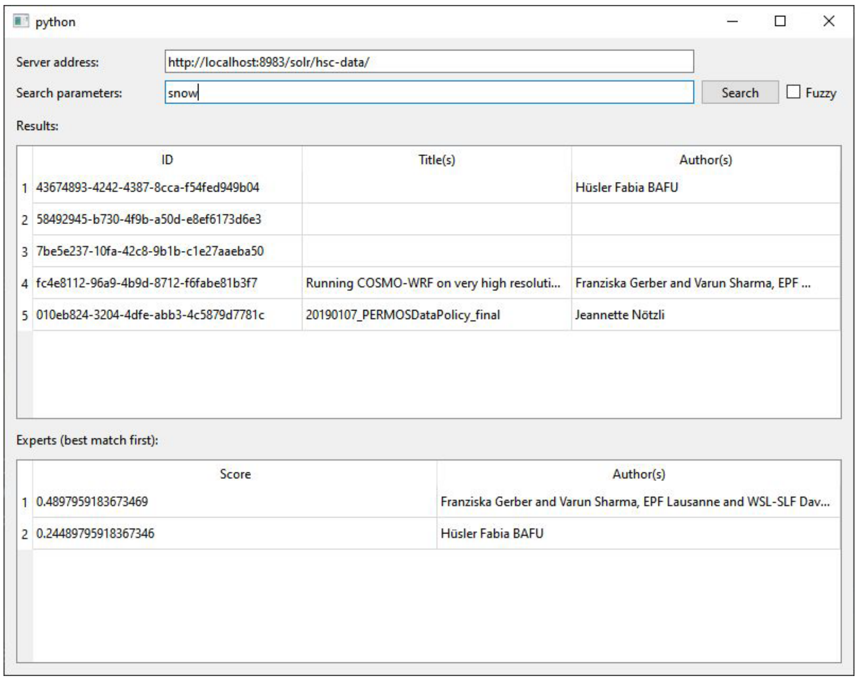
The GUI framework chosen was QT6 for Python, to enable cross-platform support. (The Implementation was tested on Linux and Windows 10 as no MacOS machines were present.)

To make use of the different collections inside Solr, a server address field is included for quick switching. This especially made running tests much easier. The search query can be expressed with Solr specific special parameters for a more controlled and fine-grained search. However, this gets fairly complex and for the proof-of-concept, it was decided to only present a simple text field and a *fuzzy* button. The *fuzzy* button allows the search to work-around human typos and enable searching for similarly typed words. E.g. searching for *znow* would still match *snow* with this toggle enabled. Additionally, the word *now* would be matched as well, leading to the drawback of having more noise in the search results.

The search results are presented as a table consisting of:

- UUID,
- Title and,
- Author(s).

As mentioned above, databases are not simulated, thus a review of retrieved documents is not possible. Additionally, a listing of the most promising experts based on our scoring in chapter 2.1 is presented.



Server address: http://localhost:8983/solr/hsc-data/

Search parameters: snow

Results:

	ID	Title(s)	Author(s)
1	43674893-4242-4387-8cca-f54fed949b04		Hüsler Fabia BAFU
2	58492945-b730-4f9b-a50d-e8ef6173d6e3		
3	7be5e237-10fa-42c8-9b1b-c1e27aaeba50		
4	fc4e8112-96a9-4b9d-8712-f6fabe81b3f7	Running COSMO-WRF on very high resoluti...	Franziska Gerber and Varun Sharma, EPF ...
5	010eb824-3204-4dfe-abb3-4c5879d7781c	20190107_PERMOSDataPolicy_final	Jeannette Nötzli

Experts (best match first):

	Score	Author(s)
1	0.4897959183673469	Franziska Gerber and Varun Sharma, EPF Lausanne and WSL-SLF Dav...
2	0.24489795918367346	Hüsler Fabia BAFU

Figure 2: Proof-of-concept GUI to query Solr for documents; searching for *snow*.

Here, it is particularly noticeable that PDFs often have missing metadata like title and author. This is a problem that should be handled during the upload of documents, where the user should be given the possibility of adding those fields.

6 Evaluation

Is the correct expert returned as the first or second recommendation? Going off of the assumption, that a user will only look at the first two experts recommended, to turn to them for further information, a correct recommendation is defined as when the first or second expert is an actual expert.

6.1 Metric

The metrics that were used are precision and recall. Let AP be the actual positives, AN the actual negatives, TP and FP true and false positives, and TN and FN true and false negatives. Then:

$$\text{Precision} = \frac{TP}{TP + FP}$$
$$\text{Recall} = \frac{TP}{TP + FN}$$

Precision can be understood as how many selected items are relevant, while recall represents how many relevant items are selected.

6.2 Test Set

The test set used is made up of 18 files, with 170 unique terms, attributed to 6 authors. This test set was created manually, by going through the papers and entering what we deem to be the relevant terms. Furthermore, it is supplemented by 5 - 200 automatically determined terms, taken from the most frequent terms for each author.

This test set has several biases:

- The 170 manually determined “test” terms have a large impact on the test set.
- Choosing randomly generated English words resulted only in true negatives and were not usable.
- No true negatives – terms either in one or both of the sets.

The impact of the test set can be minimized by increasing the ”cut-off” value for the automatically determined words, so they outnumber the manually determined keywords. If the test set looked at more PDF files, randomly generated keywords may have been a valid approach, but with the current 18 files, it is simply not sensible to use only automatically generated English words. Lastly, even if our test has no true negatives, the return data can still paint a picture of how useful the output is.

7 Result

7.1 Evaluation Result

The returned experts were compared to the test-set for cut-off/threshold values of 5, 10, 15, 20, 35, 50, 100, 200 most frequent words per author. Tests were run five times for the same configuration, with each configuration taking 100 random terms. These five runs were then averaged to receive a more stable result, with precision and recall scores falling within [0,1].

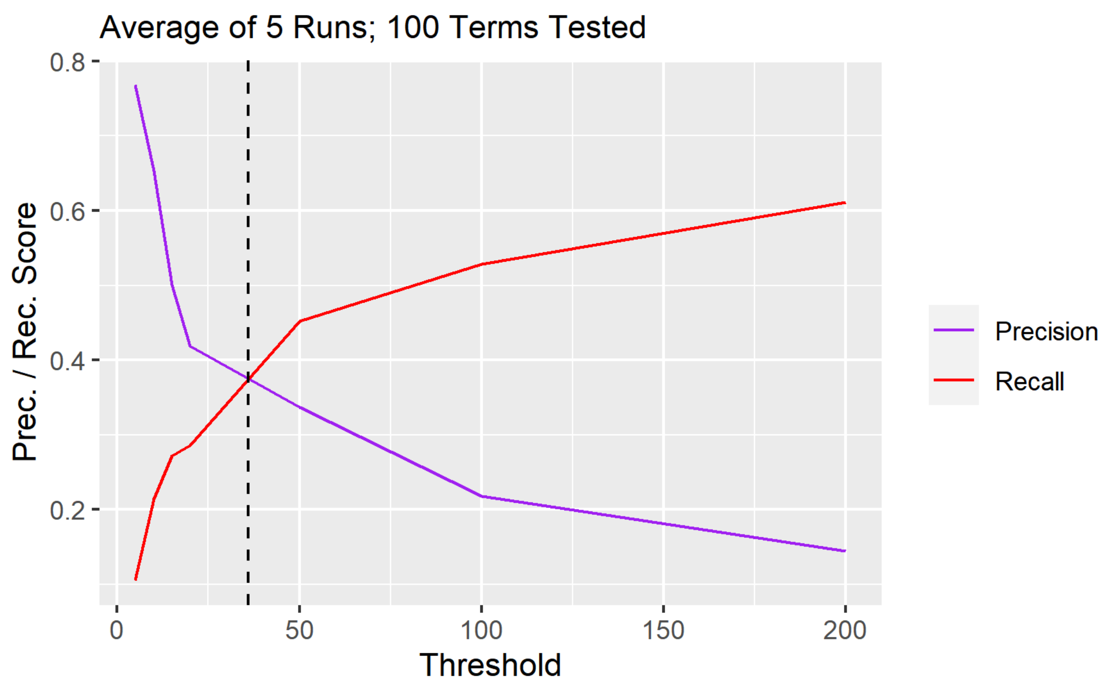


Figure 3: Precision Recall results

As can be seen in the figure 3, the optimal value for precision and recall would be 36 most frequent terms, for our data. Generally, it can be said that there is a trade-off between the quality of expert recommendation and the quantity. If an increase in precision is requested, lower threshold values are sensible to increase the quality of results. It must also be noted, with 36 terms à 6 authors, we have 216 automatically determined terms. Thus, here our manually determined keywords only make up around 44 % (170 terms), and not the majority as would be expected in a bad system for the intersection of precision and recall.

7.2 GUI

The current implementation does not take into account synonyms. This leads to the effect that the search query must contain the actual word inside the PDFs, otherwise the search results would come back empty. This can be mitigated by teaching Solr which words are linked together and mean the same. It is an endeavour that takes time to create such a pre-defined map of every topic the city government would handle and therefore is not part of this paper. Solr gives the possibility of working with analyzers, tokenizers and filters (including synonym maps) [3]. These advanced systems can lead to more sophisticated search results increasing precision and recall. However, with a wrong configuration the exact opposite can be the case as well. Therefore great care must be provided in how Solr is set up.

Due to the greatly increased complexity, these systems have not been explored. Furthermore, the current expert evaluation does not work with user typos. While Solr can work around this, the current approach does not and will always return an empty list.

8 Conclusion

The course a project takes always differs from the planning during its conception.

8.1 Research Questions

While the aim of a proof-of-concept has been carried out, our research questions must be addressed nonetheless:

- Given multiple text files, how can their contents be searched to return the best fitting ones?

  → Solr addresses this need, but a lot of fine-tuning is required to retrieve fitting results. Furthermore, grouping of keywords by a domain and respecting synonyms is also lacking.

- How can a list of experts be extracted from the submitted text files?

  → A rudimentary scoring system by using the frequency of words can be employed, but it is not useful for larger organizations.

- The submitting user of a given text file, must not necessarily be the user who wrote it. How can this be considered when determining the expert list?

  → This was covered by the assumption that the person who created the PDF and uploaded it (and is thus marked as "author" in its metadata) is the author of the text file. As an alternative, an additional GUI to add relevant metadata could be created or metadata standards would need to be enforced on the organizational level.

8.2 GUI

While the scheme of figure 1 is fairly simple, such a software architecture should not be too complex for a government system. Linking the splintered PDF databases together then gives the benefit of previewing found documents and simple retrieval, something that Solr cannot do as it only indexes search terms.

Taking into account a better expert map, this could also be visually represented with a graph for easier comprehension of who the expert is. This would certainly increase the efficiency of users finding the correct expert.

8.3 Overview

Contrary to our expectations, most open source documents, and likely also private ones, have insufficient labeled metadata. A most grievous example is that many of the PDFs do not have an author in the metadata, thus hindering the creation of the expert list (see figure 2). This shows that there is a need to extract the metadata, especially the author, from the document itself, instead of trusting the users to ensure that all metadata is provided before the upload of the document. This approach would need the help of a semantic web to extract the author because just the word count is not enough to identify the full name.

The need for a semantic web is also the solution for the second critical point of this proof of concept. As can be read in the paper [10], word count alone may not lead to finding all relevant documents, as there can be synonyms leading to additional documents. The created application only works if the searched keyword is in the extracted text of the document. Furthermore, we are unable to account for synonyms and the error-prone fuzzy search provided by Solr decreases the validity of the found documents dramatically, as it changes 1 to 2 characters in the searched word (default is 2) which can lead to unrelated results being displayed, e.g. ”rain”, ”main”, and ”gain”.

Another interesting topic would be to implement more of the already existing Solr search features, like for example the use of logical OR and AND, or the use of proximity search, where a document is searched if two keywords are in a given distance from on to another.

The main takeaway is that there are many open-source and free packages with which an expert finder system can be implemented, but most are created by a handful of programmers, leading to insufficient documentation and a huge time investment to understand the library or package. In our project this was especially obvious with the Docker technology, offering many advantages and solving problems regarding deployment and scalability, but as the image of Solr was not sufficiently documented, it had to be discarded in favor of the generic CLI method.

9 References

References

[1] Apache solr. `https://solr.apache.org/`.

[2] Apache solr reference guide: Requesthandlers and search components in solrconfig. `https://solr.apache.org/guide/8_11/requesthandlers-and-searchcomponents-in-solrconfig.html`.

[3] Apache solr reference guide: Understanding analyzers, tokenizers, and filters. `https://solr.apache.org/guide/8_11/understanding-analyzers-tokenizers-and-filters.html`.

[4] Apache tika. `https://tika.apache.org/`.

[5] Alexander Denzler, Marcel Wehrle, and Andreas Meier. Building a hierarchical, granular knowledge cube. *World Academy of Science, Engineering and Technology, International Journal of Mathematical, Computational, Physical, Electrical and Computer Engineering*, 9:334–340, 2015.

[6] Docker. Docker. `https://www.docker.com/`.

[7] Nltk : Natural language toolkit. `https://www.nltk.org/`.

[8] Alvaro Oliveira and Margarida Campolargo. From smart cities to human smart cities. In *2015 48th Hawaii International Conference on System Sciences*, pages 2336–2344. IEEE, 2015.

[9] Anand Rajaraman, Jure Leskovec, and Jeffrey D Ullman. *Mining of Massive Datasets*. Stanford University, 2014. OCLC: 870899769.

[10] L. Streeter and K. Lochbaum. An expert/expert-locating system based on automatic representation of semantic structure. In *Proceedings. The Fourth Conference on Artificial Intelligence Applications*, pages 345,346,347,348,349,350, Los Alamitos, CA, USA, mar 1988. IEEE Computer Society.