Universität Leipzig
Institut für Informatik
Digital Humanities

Final Write Up

Freie wissenschaftliche Arbeit
im Rahmen des Fachs "'Digital Philology"'
an der Fakultät für Mathematik und Informatik der Universität Leipzig

Conclusion on the project work of digital philology

Betreuender Hochschullehrer:	Prof. Dr. Crane
Betreuender Assistent:	Matt Munson
Verfasser:	Alexander Richter, Dan Häberlein, Nathanael Phillip, Peggy Lucke
	MatNr. 2517643
	M.Sc. 2. Semester
Eingereicht am:	September 30, 2014

Table of contents

1 Introduction

In this essay the project work 'Open Legislature' will be discussed, which was created during the subject 'Digital Philology' in the sommer term 2014. The goal of this essay is to present our results in an understandable fashion, reflecting the way we produced our results. In course of this work all our created (software) artifacts and analysed results regarding politicians of the german Bundestag should be presented. We begin with a short introduction of our posed questions and the corpus which was used trying to answer them. Afterwards we talk about simple misunderstandings of the problem space. In the next chapter we going to talk about our methodology, software artifacts and achived results. Finally, we conclude this work with a critical reflection of your own research process, namely what we done right and wrong during the last term.

1.1 Questions

At first we wanted to know: Can it be detected, if some politicians use a ghost writer? Or is it possible, that even politicians from different parties have the same ghost writer? Those questions are based on the fact that in your digital world it seems kind of easy to copy the work of others or event assign someone else to do your work. For reference, this article [1] of a german online newspaper summarises current plagiarism cases of many german politicians. Even Wladimir Putin, the current president of the russian federation, has been suspected cheating in his dissertation [2]. This has been in 2006. Another interessting story can be told of an software developer from the USA, who 'outsourced' all of his personal work to a company in china [3]. In return, he transfered the half of his salary to the chinese coorperation. He was free to do whatever he wants in the office, spending the most of his day to watch cat videos or surfing on social networks. On top of that, before anyone noticed, he has been honored as 'one of the best employes of his department'.

Those examples should illustrate that copying is all around us. It is kind of unfair, because many people work hard and on their own to reach their goals. We should have some tools, to be able to detect plagisms.

A whole other dimension of the original issue is the political side. In your modern parliamentary democracies, we need more or less to trust the elected politicians. Who has time to really follow their debates? If we can't or not want to change the political system, the majority has at least the right of information and transparency. And computer science can help us to achieve this.

As everybody knows, the current presented questions were not easily to anwser. We also did not find any data about ghost writers for german politicians (which should

be self evident), only the speeches they gave in the paralaments. Therefore, we have abandoned this really interessting questioning. As we discuss later in the conclusion chapter, we contradicted the scientific approach to answer questions: We tried to find questions considering our data. For example we compared speeches of different politicans to find similarities in there speeches, which can be interessting, but without an concreate set of questions it may not be expedient. So we redefined our main question to be: 'What are typical used words for a specific politician? Which politicians use similar words?'. The corpus, luckily, was capable to answer many interessting questions. But theoretically speaking, we had a basic problem in our methology: Finding questions for a dataset.

1.2 Corpus

During the first couple of weeks in our digital humanities course, we were asked to search for data which can be used for linguistic analysis. We should have collected more data regarding the original questioning. But we did not find any obvious hints for ghostwriting in the german parlament (Bundestag). As already mentioned, we only found the official records of the german parlament. For every session of the parlament, all stenographic reports are freely available [4]. All 'protocols' of the Bundestag since the founding of the german federal republic (1949) are extractable in the pdf format. In total, there are about 4000 PDF documents for all sessions and election periods. They make up a dataset of nearly 10 GB. For our analysis, we used this dataset exclusivly.

1.3 Understanding the problem

As computer scientists we were not used to linguistic questioning. We learned during the course a bit to late, what should be asked and what is not answerable easily. That is, most likely one of the reasons we come up with typical data and statistical results (like for decision support systems in an enterprise). Another point is that computer scientists produce software and tools sometimes way to early. They should concentrate on their goals, achieving results (so in this case the answers of our questions) in the most simple fashion. I think we did use to complicated approaches, which did not show any results so quickly, frustating the whole team. For example, we had a document pipeline to extract all Bundestag protocols from the internet. Instead of reducing the dataset by selecting only protocols of one period we invested time to make the process parallel. Of course this was helpful, but did not produce any answers. If we would have been already finished for a small subset (maybe the current election period), we could have implemented this feature. But before that it was just a waste of time.

2 Our Achievements

In this chapter we will talk about our results and discribe how anyone could reproduce them. The main goal is to express how did we do it and what have we done. At first in the chapter, we will show the timeline of the last term and which was done when, giving also an overview over the produced artifacts. Than we go over every artifact in detail, reflecting our work critically.

2.1 Artifact Overview

As results in any form, we created the following during the last term:

- An Java application called 'document pipeline' to extract and process all Bundestag protocols of the past and the future. This application have been designed to extract the text from the protocols using an optical character regocnition library. Afterwards, we using it to create structured data from each protocol.
- Using the data created by the our document pipeline, we were able to build an easy website providing adhoc analysis of the data. We call this module 'Data visualization'. We also provide our later results of the SLDA Classification on this website.
- In our intention to group speaches by topics we used the SLDA classification algorithm. We had really problems to use it on such an overwhelming amount of data.
- We created a meta database using a graph database (neo4J). This database concludes information about the composition of the Bundestag of every election period, the detailed numbers how many mandats each parties got, which party form the gouvernment coalition and so on.

In the last term during our digital humanties course we had to master a great amount of workload. For every group member, there should have been about 8 hours per week spending on the group project. We did our best, but we think the whole project was a bit to heavy for a 10 ECTS course (although this time is ensured by 10 ECTS course). Furthermore, we should have had a clear finishing date (not vaguely by the end of the term). In the following figure 2.1, we want to show what we done and how much time we spent for it. Notice that we misestimated quite often how long we need for a feature, which took many time from our project. For each artifact we will talk about the misestimation in detail in their associated chapters.

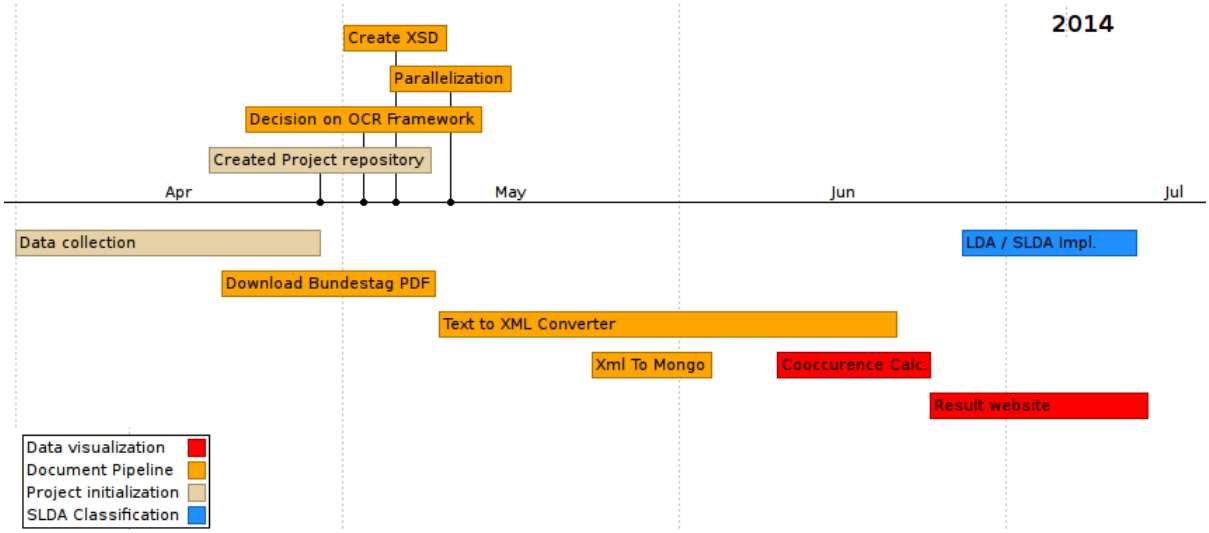

Figure 2.1: Timeline of the Open Legislature project during summer term 2014

Roughly speaking we spent more than the half of the term on implenting the document pipeline, followed in terms of effort by our result visualization module and the LDA / SDLA calculation. Alone judging by the time taken, it was most complicated to transfer the unstructured PDF protocols into a more mechine readable format. We will talk about this process in the next part of this essay.

2.2 Document Pipeline

From the perspective of our project github repository, we now going to explain the content of the openLegislature folder. This is shown in figure 2.2:

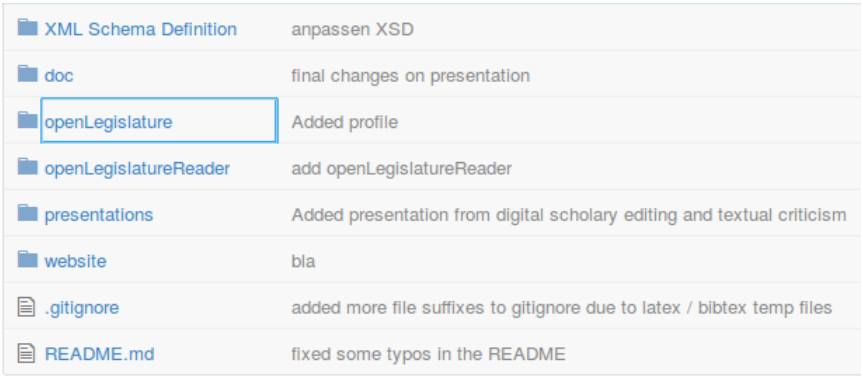

Figure 2.2: Document pipeline folder in the project github repository.

As already in chapter 1.3 and 2.1 introduced, we created an Java application which converts the unstructured Bundestag protocols in pdf format into structured data. This data will be stored into a database called 'Mongo DB' (a schemaless document oriented database with a powerfull query language with similar functionalities to SQL).

This Java project is organized with maven. Maven is a simple tool to create (compile, resolve third party libraries, execute) Java applications. Concluding these points, we can state the prerequistes of our document pipeline if you want to build and run it:

- Java JDK 7 or later
- Maven 3
- MongoDB (running instance on default port 27017)

Than just clone the repository on, and issue the following command in the 'openLegislature' folder (the folder shown in graphic 2.2) to build and run all of our Java applications (the document pipeline AND the SLDA transformation):

```
mvn compile exec:java \
-Dexec.mainClass=org.openlegislature.App
```

Listing 2.1: Maven command to build and execute the document pipeline

This command can take some time to execute, because it will download, process and insert all Bundestag protocols available. There are some command line parameters which can be added to the maven call using '-Dopenlegislature.XXX', enabling fine grainded settings, e.g. the reduction of loaded protocols (for example to select on a different election period range). For more information please consult the appendix **??**.

The document pipeline module works for over 90% of the protocols. For the others, our problem module, the text to xml parser, does not generate valid xml. As illustrated in figure 2.1, this part of the document pipeline took the most time. We misestimated the effort of the text to xml parser. For example, during the different election periods, the structure of the protocols changed. This knowledge had to be gained manually reading the protocols, transfering it into the source code of the text to xml parser. An idea for comparable future work is to invest much more time into the analysis of the differnt types of text (pdf) files and categorizing them. But still it would not be easily possible to produce 100 %, because exception do occur. If there is not much time and when focusing on results, it is usally also a good idea to reduce the dataset.
The text to xml parser should also be created with an API (like lxml in python or the xerces DOM implementation in Java) to ensure valid xml documents. It was created with simple strings, which caused us large maintenance overhead.
Another issue from the organizing perspective was the design of our process. We first thought, that xml will serve all our problems and structured the data enough. We could have used XPath or XQuery as powerful query languages. As we moved more to

the data visualization part, it turned out that a database handeling the content of our protocols could be a better choice regarding more convenient queries. So we transfered our data from files to Mongo DB. This was just a step for convenience. We could have worked with XPath as well. This took not so long, but we lost again some time there. To finish this part, our results of the shown software should be emphasized: We have a program, that can download all of the Bundestag protocols, convert those pdfs to text, convert again the text to xml and finally insert this xml into Mongo DB. We have now structured data which can be analysed much easier.

2.3 Data visualization

This module was called data visualization, although we also analyse parts of our crated dataset (we calculate cooccurences of words in different speeches, the log likelihood for them, etc). For our github repository, the artifacts for this module are located as shown in figure 2.3.

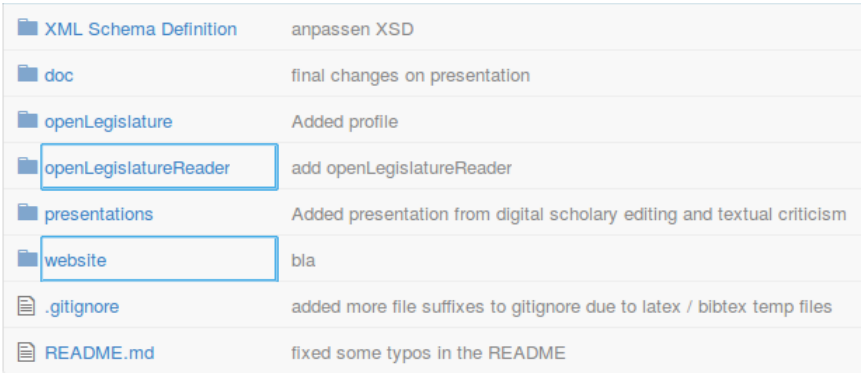

Figure 2.3: Analasys and visualization modul folder in the project github repository.

In the 'openLegislatureReader' folder we created an Java application which read our Mongo DB database, calculating for all speeches cooccurence matrices. For simplicity, we wrote this data back into Mongo DB. Because this calculation also took very long, we provide the data in the json format for importing, sparing anyone the long calculation time. There are:

- ollspeeches.json
- partystats.json
- plenarprotokolle.json
- speakerstats.json

containing our results for the visualization website. We would have been provided those files using github, but they are to large to upload. Instead, we put them into a cloud service repository called 'Copy'. In the moment because of their size, those files are not open to the public. Of course they are retrievable from us.
When the data have been imported successfully, you finally need to install our result webpage. The orignal idea was to put the page avaiable for the public, but unfortunatly we didn't have a own webserver and domain with the capabilities to hold our data. Therefore, it must be installed by your own on your machine. In total, for running the data visualization module you need:

- Webserver (e.g. apache2)
- PHP (The used language for the server side)
- The Mongo DB driver for PHP

For example on Linux / Unix, you can retrieve those dependencies using your systems package manager. Afterwards, copy the 'website' folder from figure 2.3 of our github repository into the root folder of your webserver (e.g. /var/www for apache2). When all is well installed, Mongo DB is running and the data is properly imported and your web server runs too you can access our result webpage using the URL given by listing 2.3.

```
localhost/website/index.php
```

Listing 2.2: URL for the browser of the data visualization module

You should see a webpage comparable to this, illustrated in figure 2.4:

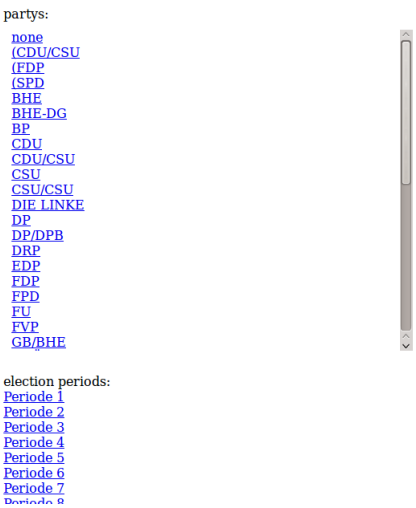

Figure 2.4: Data visualization website

We assume that the user interface is really minimal, but you should get a good first idea what the data looks like. Using the webpage, we can infer many metrics about the data for speakers, parties and election periods. One interessting diagram type shows the top word of a politican. In figure 2.5, we show an example for Dr. Sarah Wagenknecht (member of the party Die Linke):

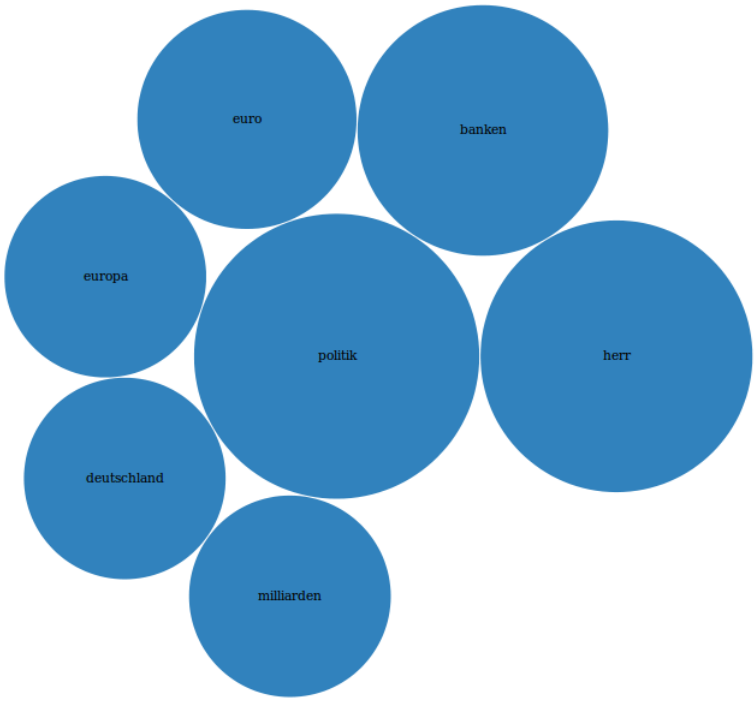

Figure 2.5: Top used words by Dr. Sarah Wagenknecht (without stopwords)

We deleted any stopwords using a stopword list to elimate typical german words without a deeper meening. As you can see, Dr. Sarah Wagenknecht talks in her speeches much about euro politics. Such insights can be performed for every speeker of the german Bundestag. There is one drawback: As shown in figure 2.1, we spent our time sometimes really carelessly. Therfore we did not have much time to clean up our dataset. Let's assume that some speaker like Dr. Sarah Wagenknecht gives multiple speeches in different election periods. Sometimes the protocols gave her name (our distingulish attribute for a speaker) in the full form, sometimes they used an abbrebivated form, e.g. Dr. S. Wagenknecht, or Dr. S Wagenknecht. With those three different spelling styles we would have to clean up our dataset and consolidate all data to one speaker. This is called data cleaning. It usally takes much time so we decided to leave it out.

2.4 SLDA Classification

For our classifiaction purposes we use SLDA (Supervised Latent Dirichlet Allocation). It is an extension of the Latent Dirichlet Allocation. The bases for both of these Algorithm is the Topic Model.
The Topic Model is a statistical Model in which a document (text, image, etc.) is described as a collection of topics. A topic is likewise a collection of words. Therefor a document is assembled mostly of words which belong to the topics of the document. The more significant a topic is the more of its words is in the document.
In documents about dogs words like dog and bone are more commonly like in documents about cat or mouse. Words like and or is are equally common in both documents. Since most documents have more than one topic it is expected that in a text which is 90
The LDA is a generative probability model. Here a document is represented as a mixture of different topics (latent topics). The number of topics is set at the beginning. Each word of a document belongs to one or more topics. The topics represent the relationship between the documents. Each document is characterized as a portion of the topics. This is the standard assumption of the bag of words model. Thus the words are exchangeable.
Within the generative process a document is seen as a random mixture overall topics and each topic is seen a distribution over the words. A document i in the corpus D is as follows defined:

Choose $\theta_i \sim Dir(\alpha)$, $Dir(\alpha)$ Dirichlet-prior αChoose $\phi_k \sim Dir(\beta)$, $k \in 1, \ldots, K$

1. a) Choose topic $z_{i,j} \sim Multinomial(\theta_i)$
   b) Choose word $w_{i,j} \sim Multinomial(\phi_{z_{i,j}})$

As mentioned above the SLDA is a extension of the LDA, so it can be used for supervised learning. For each document a response variable is added. It is not import what is represented by the response variable, the starts of a rating or the topic of a document. The documents will modeled together with the response variable to find the latent topics to performed better predictions. The generative Process is defined as followed:

1. Draw topic proportions $\theta|\alpha \sim Dir(\alpha)$.
2. For each word
   a) Draw topic assignment $z_n|\theta \sim Mult(\theta)$.
   b) Draw word $w_n|z_n, \beta_{1:K} \sim Mult(\beta_{z_n})$

3. Draw response variable $y|z_{1:N}, \eta, \sigma^2 \sim N(\eta^T \overline{z}, \sigma^2)$ für $y \in R$

We choose the SLDA for our classification purposes because of the different results we can get. The first is the original result of the LDA, a number of words for each topic. The second is the reason for the SLDA, so we can perform predictions on new documents.

Results and data

Because of the amount of data we had to do modify our original approach for the data generation. In our first approach we tried to generate a single input file for the complete election period. In case of the 17th election period this sums up to about 51000 speeches with 250000 words. After the generation of the data file we performed the SLDA classification on it, as topics for the speeches we used the speakers. The result of this is we had the most important words for each speake in the election period. For the second approach we generated a data file for each protocol and performed the SLDA analysis on each of them. Then merged the result for the complete election period an performed the analysis again. The second approach is not as in-depth as the first but gets faster results.

3 Conclusion

In this last chapter, we will reflect again our results. We give a brief overview how well we our adjusted questioning answered. Afterwards some critical thoughts about our methology. We will conclude with what could be done in the future, if we want to enhance the project any further.
We answered our questions only partly. We were not able to say which speech is identical to another one, because the dataset was to big. Regarding the original questioning, we have also to wrap up that they were to inconcrete. Instead of asking 'Which speeches are similar to others?' we should have asked something like 'How differ speeches by Angela Merkel to Peer Steinbrück? What do they tell us, what promises they make? Are they contradict theirselfs ragarding their election promises?'. I think with those different questions our project would be a much greater success. What we did right is the creation of a (structured) dataset. We performed simple cooccurence calculations and word counting on it, but we really need to advance our methology. As an asset, we did the similarity match using the SLDA algorthm. It performed not to well in terms of speed, but we were able to use it for the 18th election period. It told us,

Bibliography

[1] RP Online. Politiker und plagiats-affären, 2014.

[2] FAZ. Dr. putins arbeit ist ein plagiat, 2006.

[3] Die Zeit. Entwickler lagert eigenen job heimlich nach china aus, 2013. refrence http://www.zeit.de/digital/internet/2013-01/outsourcing-china-verizon, visited at 21.09.2014.

[4] Bundestag. Bundestag plenarprotokoll link, 2014. reference http://dip21.bundestag.de/dip21/btp/{period}/{period}{session}.pdf, replace period with election period (e.g. 01 for the first) and session with a three digit number of the session(001 for the first session).