UNIVERSITY OF TARTU
Institute of Computer Science
Computer Science Curriculum

Ragnar Vent

A Framework for Analysing Topics in University Courses

Master's Thesis (30 ECTS)

Supervisor: Siim Karus, PhD

Tartu 2017

Raamistik ülikooli õppeainete teemade analüüsimiseks

Lühikokkuvõte: Erinevates teaduskondades õpetatavate ainete maht võib olla väga suur ning tihtipeale on raske saada ülevaadet ülikooli kursustel õpetatavatest teemadest ja nende kattuvusest. Käesoleva töö eesmärk on luua automaatne ja progressiivne viis, kuidas avastada kursusel õpetatavaid teemasid, tuginedes eelkõige avalikult saadaval olevatele kursuse materjalidele. Seda tüüpi analüüs pakub palju eeliseid, alustades üldisest ülevaatest kursusel õpetatavatest teemadest või näiteks avastamaks aineid, mis toetavad ühe projekti tegemist mitme kursuse raames. Selle jaoks peaksime leidma samal semestril õpetatavaid ained, mis katavad samasid teemasid ja nõuavad projekti tegemist.

Pakutud raamistik on vastutav erinevate kursustega seotud tekstiliste materjalide kogumises ja failidest eraldamises. Seejärel puhastatakse kogutud andmed ja teisendatakse sobivasse formaati, mille peal teostatakse tekstianalüüsi, täpsemalt teemamodelleerimist. Teemade ja kursusevaheliste seoste leidmiseks rakendatakse LDA modelleerimismeetodit. Viimase sammuna kasutatakse käesoleva raamistiku jaoks loodud visualiseerimiskomponenti, et presenteerida teemamodelleerimise tulemusi.

Võtmesõnad: tekstianalüüs, teemamodelleerimine

CERCS: P170 (Arvutiteadus, arvanalüüs, süsteemid, kontroll)

A Framework for Analysing Topics in University Courses

Abstract: The vast amount of courses being taught throughout different faculties of a university makes it difficult to get a general overview of the topics being taught and their overlaps from a singular point of view. The goal of this work is to provide an automated and progressive means to discover such topics, based on the publicly available course materials. The benefits for this kind of analysis range from simply gaining an insight and an overview of the topics covered by different courses, to more specific and goal-oriented purposes. For example, we can discover which courses can support cross-course projects by finding courses that cover the same topic, are taught in the same semester and require the completion of a course project.

The proposed framework is responsible for gathering and extracting course related raw textual content from various heterogeneous sources. The collected data is then cleansed and transformed to a suitable format for further textual analysis, more specifically topic modelling. LDA modelling method is used as a main tool for resolving topics and discovering relations between courses and individual course materials. As a final step of the established mechanism, the analysis results are passed to a predefined visualization component, specifically designed for the framework at hand.

Keywords: text analysis, topic modelling

CERCS: P170 (Computer science, numerical analysis, systems, control)

Contents

1 Introduction

The vast amount of courses being taught throughout different faculties of a university makes it difficult to get a general overview of the topics being taught and their overlaps from a singular point of view. The goal of this work is to investigate and tackle that issue by establishing a framework for analysing university courses, specifically, which courses are related and what are the overlaps in the topics and material being taught. There can be several benefits for this kind of analysis, starting by gaining an insight and an overview of the topics covered by different courses, finding out which courses can support cross-course tasks and projects, or for example validating the order in which courses are taught in certain curriculum.

The aim is to provide a process, which upon triggering, will extract and integrate course related textual data from various available sources, applying necessary data cleaning operations, analyse the data and finally visualize the results. The process should be fully automated, so that when new materials become available during next semester, the existing results can be updated and progressively analysed. The data sources made available by different institutes of the University of Tartu will be used as a novel example around which the entire framework will be built.

The first section focuses on the overall structure of the framework and introduces ELT approach as a cornerstone for building a solid base of data in the correct format, ready for further analysis. The following chapters provide a more detailed overview of the individual steps in the entire process. Firstly, we describe how the raw data (HTML content, files etc.) is gathered from various sources, and how in turn the textual content is extracted from the raw data. The next chapter focuses on cleaning and conforming the extracted textual content, this includes finding frequent co-occurring words and extracting acronyms. The cleansed data is then used as a foundation for performing text analysis, more precisely - topic modelling. A known topic modelling algorithm called Latent Dirichlet Allocation is used to resolve a model in which each topic consists of words (i.e a topic-word distribution) and each document is composed of a fixed number of resolved topics (i.e document-topic distribution). Different levels of granularities are explored, both at course level, where we seek courses under the same topic, and at material level, where we look for related materials. The final chapter gives an overview of an example process run and shows how the results are presented and visualized.

The entire mechanism, starting from data scraping to analysis, was implemented using *Python 2.7* along with multitude of complementary libraries.

1.1 Background

The work at hand was motivated by the easily accessible and public course materials made available by certain institutes in the University of Tartu. Combined with the need to gain a general overview of the numerous courses being taught and their overlaps, the available sources provided the means for further investigation and perhaps even the means for a solution for the problem at hand.

The present work was started as a course project in Business Intelligence Seminar. In the scope of the first version, the materials of a single academic semester were scraped and processed (basic tokenization and stop word removal). A topic modelling algorithm was then applied to the cleansed data and an initial visualization component built to present the results. All in all, the initial version served as a raw proof on concept and showed a lot of promise with the hope that it could be developed to a fully functional framework.

As a consequence, the initial version was completely revamped to guarantee a more solid base upon which new features would be developed and existing functionality improved. The visualization was then further developed to accommodate the new features and to patch existing shortcomings and defects, albeit the core visualization framework remained roughly the same. The present work aims to reflect the latest state of the project, i.e after the initial version was effectively revamped and developed into a feasible framework.

1.2 Related Work

The primary goal in this work is to provide a framework for discovering topic overlaps between university courses. This is achieved by applying topic modelling, a widely investigated and researched area in natural language processing. While numerous approaches exist, one of the most known methods is Latent Dirichlet Allocation (LDA), initially proposed in [12]. The LDA modelling approach has since been enhanced with different ways of achieving inference and adopted as a tool for a wide range of analysis purposes.

The authors of [21] applied LDA on hundreds of thousands of job postings with the intention of discovering and grouping abuse jobs (e.g spamming, fake account creation). The topic modelling concept provided an automatic means of finding related jobs and giving an insight to the grouped jobs and their relations. Despite the fact that some of the topics lacked the high level of granularity compared to the results of other supervised methods, the approach in general showed a lot of promise. The authors emphasized two significant strengths: automization of a manual process and the ability to cluster unlabeled data, i.e the necessary requirements for building the main analysis part for the framework suggested in present work.

An approach based on LDA was used in [24] for tag recommendation. The idea was to overcome the so-called *cold start problem*, i.e when some resources have tags only from a few users, reducing their overall usefulness. Basically, other tags that fall under the same topic are used for recommendation, thereby reducing the wide sparsity of tags and allowing more specific tags to be recommended. The suggested LDA based method displayed promising results in terms of precision and recall, compared to association rule based approach.

A cross-language event detection system based on LDA model was proposed in [16]. The goal was to find related documents over different languages without using any translation system. Subsequently, the authors extended the LDA method in order to compare two different topic models (one for each language). The resulting approach was then successfully applied to various news stories. Clustering documents across different languages is of a particular relevance in the scope of present work, since the gathered and used course data comes mainly in two different languages - Estonian and English.

2 Methods

The purpose of this section is to give an overview of the overall process structure and methods used throughout present work. As a first step, we describe both ETL and ELT approach in general and why ELT was chosen as the foundation for further analysis and visualization. We then give a more detailed examination of the actual ELT process that was designed and how it fits with the overall workflow, prior to giving a full overview of all the steps it contains in following sections.

2.1 ETL and ELT Approach

The process of extracting and integrating raw data from heterogeneous sources, then cleansing and transforming it before loading the final result to a data warehouse, is referred to as *Extract, Transform, Load (ETL)* [28]. The entire flow can be run periodically (e.g weekly or monthly basis, depending on the business needs) and with each run, we only update the data warehouse with new data in relation to last update cycle.

The first step in ETL process involves gathering data from various composite sources and integrating them into a single entity, bound by uniform format. Two seemingly identical extraction processes can differ significantly in terms of speed (i.e how long it takes to extract the data) and the size of the collected data, all depending on the sources. The actual relevance of the data or how much of it can be used is not always immediately evident and may be determined during later phases. The goal is to prudently extract enough raw data to suit business needs and provide a base for effective analysis.

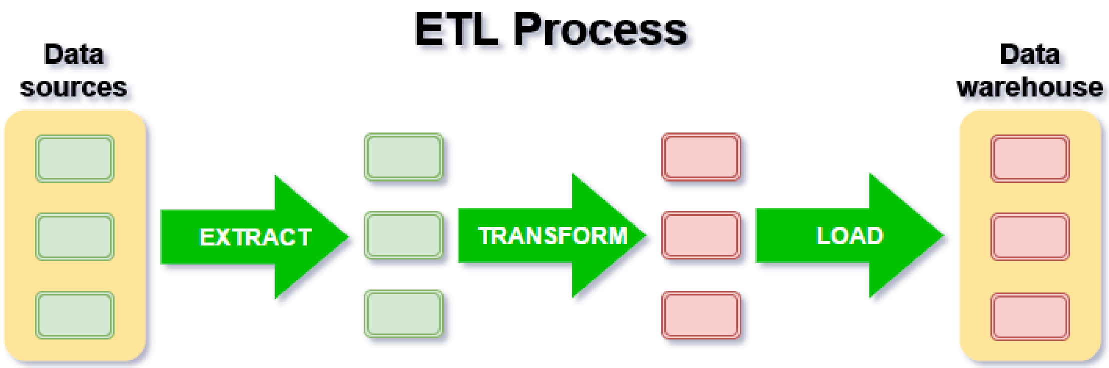


Figure 1. General overview of ETL process flow.

The second step focuses on transforming the extracted data by applying different functions depending on the end-user needs and preparing it to be loaded to data warehouse. The transformation phase usually includes data cleaning, in which faulty and inconsistent records are removed or otherwise handled (normalization, type conversions, data imputation etc.).

As a final step we load the transformed data to the target data warehouse and make it available for further analysis and visualization by end users. Numerous rules, functions and analytics, with respect to the business requirements, can then be applied to the cleansed data. The loading process is usually done periodically and in an incremental fashion [28]. The initial bulk loading takes place only the first time the data warehouse is constructed [28]. As with previous steps, the final data loading process design depends highly on the actual business requirements.

Extract, Load, Transform (ELT), however, is an alternative version of ETL, in which the data is first loaded to the target data warehouse and only then are the transformations applied. The ELT approach should be preferred over classical pipeline based ETL approach when dealing with large datasets and the target system has ample amount of resources to perform any transformations on the data at hand. Basically, the aim is to reduce data load time and to take advantage of the target systems capabilities. Thus the decision to opt for either ETL or ELT approach should be carefully weighted by taking into account both the target system capabilities and the nature of the data sources. The general ELT approach was used in the scope of present work as a cornerstone for designing an effective mechanism for analysing relations between courses, starting with the collection of relevant textual data to the eventual visualization and presentation of the relations themselves.

2.2 Framework and ELT Process Design

Designing an effective ELT process along with a suitable database model is a demanding and time-consuming task. The goal is to deliver accurate information without overcomplicating the entire flow. The process can be conceptualized using a wide range of different flow charts, models and diagrams, e.g entity relationship diagrams for storing source data, star schema for final destination database. The ELT process designed for providing a base for course relations analysis, however, is relatively simple, since the raw source data is not gathered from databases, but via other means (mainly web scraping), i.e no schema mappings are required. The overall flow applied throughout present work can be seen in Figure 2.

In essence, two phases are sequentially executed as a part of the ELT process. As a first step we gather textual data, including files, for each course from different sources

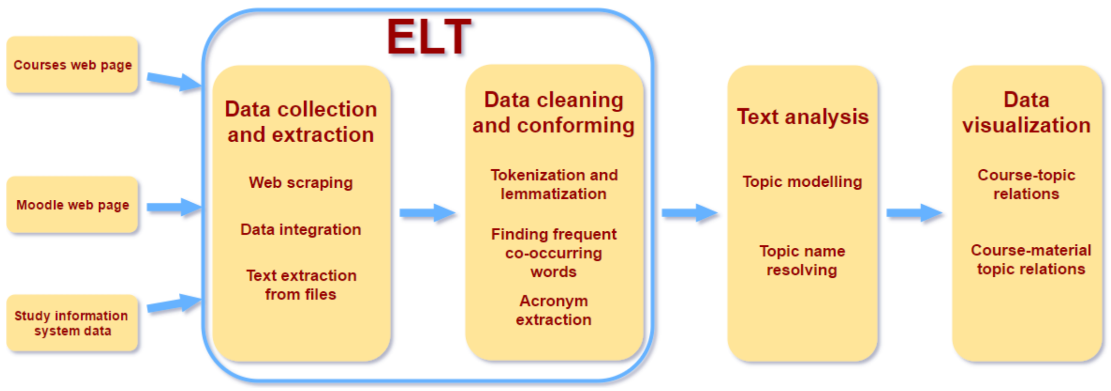

Courses web page
Moodle web page
Study information system data
ELT
Data collection and extraction
Web scraping
Data integration
Text extraction from files
Data cleaning and conforming
Tokenization and lemmatization
Finding frequent co-occurring words
Acronym extraction
Text analysis
Topic modelling
Topic name resolving
Data visualization
Course-topic relations
Course-material topic relations

Figure 2. General framework of the entire workflow including different steps in ELT process.

and integrate them in order to compose a single storage of raw data, ready for further processing. The content of the downloaded files is then extracted and loaded into the target data warehouse. In the next step we aim to the clean the textual content and transform it to a word dictionary in which we know how many times each word appeared at different levels (e.g in all the course material, or in a single course material). Further steps are taken to improve the quality of the subsequent analysis (e.g resolving frequent co-occurring words).

The result of the final step is stored in the same database file that is used throughout the entire ELT process. No separate data staging area is used, since the final size of the database file turned out to be relatively small (<100MB) and with the exception of the raw textual content, all other course material and course related attributes (e.g material source URL) were deemed necessary and usable in the visualization part. Thus no additional transformations or schema changes are made for proper data loading purposes. The constructed dictionaries stored in the database file are then used to perform topic modelling with the intention of finding out which of the courses fall under the same topic and to what extent. Once the analysis results are persisted, the database file is simply passed along in its entirety and the visualization part is built on top of it.

3 Data Collection and Extraction

The collection and integration of relevant data from multiple sources is one of the first and most influential steps of the entire ELT process. The aim of this chapter is to define and describe all the data sources that were included as a part of the data collection phase and the challenges it posed. For example, while the bulk of the material was scraped from various web pages, additional sources such as Study Information System (SIS) data were also included and integrated to an extent. In addition, text extraction from various file types and its subsequent erroneous nature is explored.

3.1 Web Scraping

3.1.1 Courses Web Pages

University of Tartu Courses web pages were considered as a primary source of data, since they publicly provide an initial list of all the available courses taught during different semesters in a specific institute, including an ample amount of textual data to start with. Courses web page itself contains various information related to the on-goings of most, if not all of the courses taught during a semester, this can include lecture materials, worksheets for practice sessions, course syllabus etc. Essentially providing an excellent base for web scraping. As a result, two web pages were included in the analysis - the Courses web pages of the Institute of Computer Science [6] and the Institute of Mathematics and Statistics [7].

The goal of web scraping is to gather as much textual data as possible with regard to the topics taught within a given course. This includes material in the form a file (e.g *PDF*, *PPTX*) or even the HTML content data from the web page itself, since it usually includes the general description of the course and other similar metadata. In addition, gathering data with regard to some course implies going through all of them one by one, therefore giving us the means to compile a healthy list of the courses taught throughout different semesters.

The general flow of the scraping of Courses web pages can been seen in Figure 3. We use Courses web page that lists all available semesters as a starting point. From there, we loop through each semester and follow only semesters that are whitelisted (i.e the target semesters). For each semester, we get a list of courses taught during that semester. During the processing of each course, we first extract the course related information (e.g course code, name) and then process each course subpage (e.g Lectures, Homeworks, Links). Whilst processing a page, we first extract all relevant textual content in HTML. Should any hyperlinks appear during the scraping process, we first check if it is a downloadable material that passes our extension list check (see Section 3.1.3).

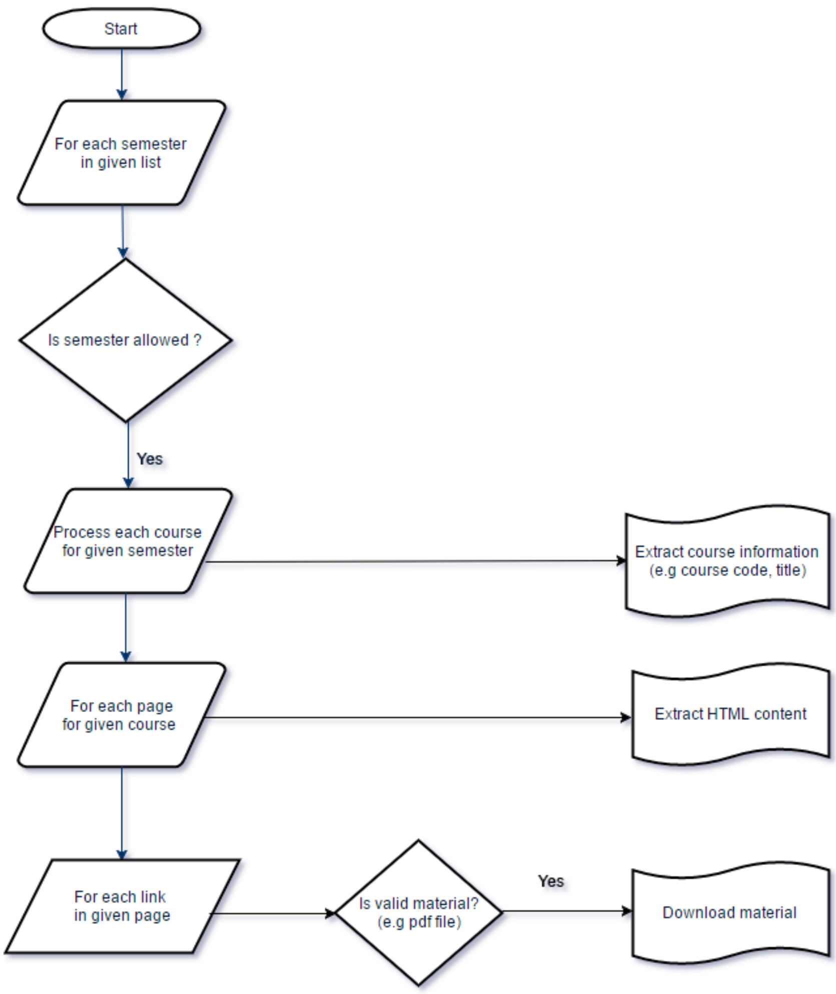

Start
For each semester in given list
Is semester allowed ?
Yes
Process each course for given semester
Extract course information (e.g course code, title)
For each page for given course
Extract HTML content
For each link in given page
Is valid material? (e.g pdf file)
Yes
Download material

Figure 3. Flowchart of Courses data scraping process.

Courses web page also allows to choose between Estonian and English language, effectively changing the course name and in some cases the course description to the specified language. Since English based text is effectively simpler to manage during later phases (e.g text cleaning), it was chosen as the primary language whilst web scraping.

As an additional note, at least two years (i.e four semesters) worth of data should be scraped in order to get a full overview of all the courses taught, since not all of them are taught every semester or every year, but over year.

3.1.2 Moodle Web Page

Even though a course given on particular semester might be listed in Tartu University Courses web page, it doesn't necessarily mean that all the material will be present there. Some courses are centered around or the very least have some lecture related material or for example homework submission possibilities present in Moodle environment. In order to incorporate as much data as possible into the analysis, the official Moodle web page of the University of Tartu [8] was included as a part of the web scraping process.

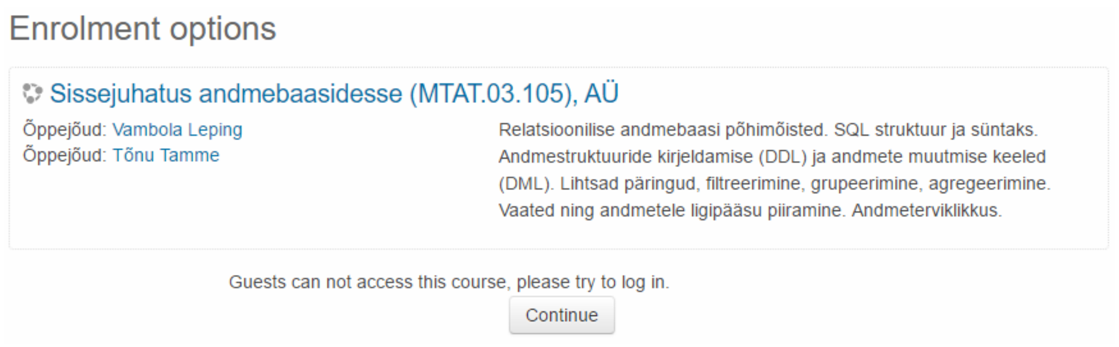
It is important to note that while there is a uniform way of determining the particular year and semester for any course in Courses web page during web scraping, it is not as simple with Moodle environment. Since there is a single entry for each course in Moodle environment, the material present in the entry could be from the on-going semester or during the last (not necessarily previous) semester the course was taught. As a result, Moodle environment could not be considered as a primary source of data, but rather a complementary data source. For the sake of properly integrating Moodle material, we first have to determine all the courses that are taught during different semesters. This will be achieved during the scraping of Courses web page.

Algorithm 1: Algorithm for determining current academic semester.

```
Function determine_semester:
    $year \leftarrow$ current year
    $semester \leftarrow$ 'fall'
    if February $\leq$ current month $\leq$ August then
        $semester \leftarrow$ 'spring'
    if current month == January then
        $year = year - 1$
    return $year, semester$
```

As a next step, we make the assumption that the courses listed in Moodle environment reflect the state of the currently on-going semester. Basically, we attempt to match the courses in Moodle with the courses scraped from Courses web site for currently on-going semester. If web scraping process encounters a course that is not listed among the courses taught in the on-going semester, it is discarded.

In order to determine the on-going semester and year, we use a simple logic that is based on current time and the official start and end dates of spring and fall semesters. The year for the on-going semester will not be determined solely based on the current year, but also on the current month, as January is still part of the fall semester that began in previous year. The semester will be determined as 'spring' if the current month falls between February (the official start of spring semester) and August (end of summer break).



Enrolment options

Sissejuhatus andmebaasidesse (MTAT.03.105), AÜ

Õppejõud: Vambola Leping
Õppejõud: Tõnu Tamme

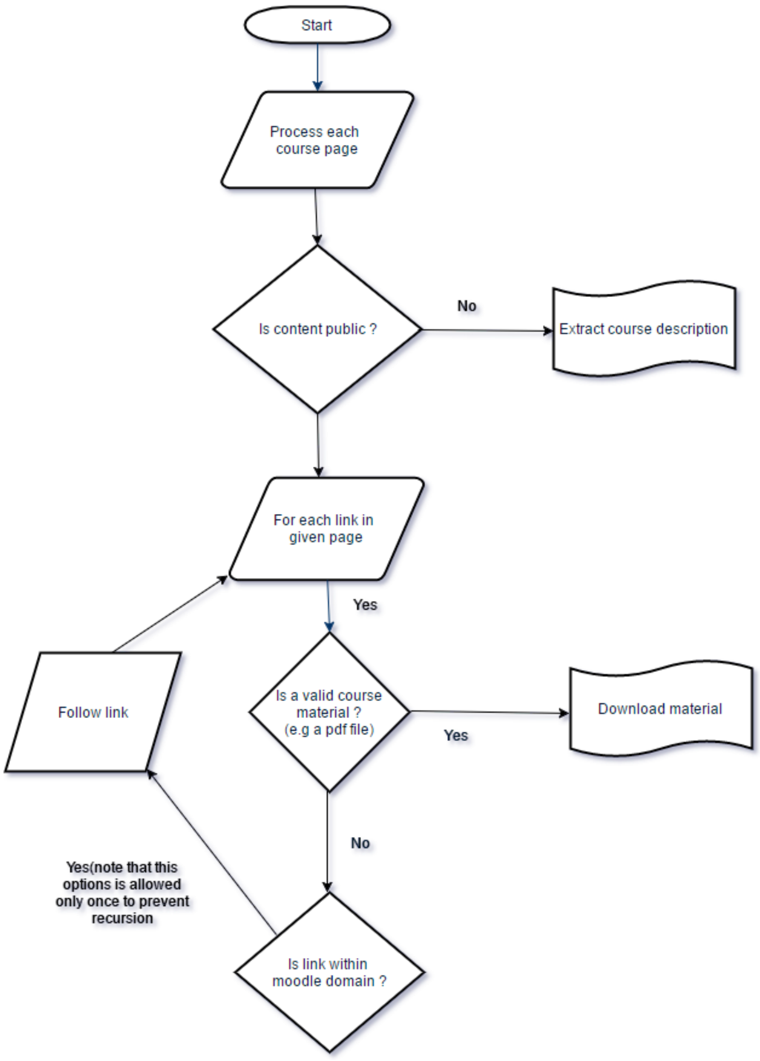
Relatsioonilise andmebaasi põhimõisted. SQL struktuur ja süntaks. Andmestruktuuride kirjeldamise (DDL) ja andmete muutmise keeled (DML). Lihtsad päringud, filtreerimine, grupeerimine, agregeerimine. Vaated ning andmetele ligipääsu piiramine. Andmeterviklikkus.

Guests can not access this course, please try to log in.

Figure 4. An example of a Moodle web page for a single course in case the access is restricted. The general course description is still scraped if it is present.

Regarding web scraping part, a different approach from the scraping of Courses web page has to be considered, as the formatting and layout can be unique to the course listed in Moodle web page. Furthermore, the course entries themselves might not be publicly available and require a user who has registered to the course to be logged in, putting limits to the amount of data we can scrape from Moodle. However, in some cases, even when a course has restricted access, the general description of the course is still presented and eligble for scraping (see Figure 4).

The extraction flow of a course listed in Moodle environment can be seen in Figure 5. Basically, we process each course present for given course category (e.g Institute of Computer Science). However, each course page in Moodle can have an individual design, making the scraping process more difficult. As a result, we scrape HTML textual content (course description) only when the course is not publicly accessible, since it is always presented in a uniform format. For each link in given course page, we first check if it refers to a valid downloadable material. If not, we will follow the link only if we stay within Moodle domain. Note that this option is allowed only once to prevent potential recursion.



Start
Process each course page
Is content public ?
No
Extract course description
For each link in given page
Yes
Is a valid course material ? (e.g a pdf file)
Yes
Download material
Follow link
No
Yes(note that this options is allowed only once to prevent recursion
Is link within moodle domain ?

Figure 5. Flowchart of Moodle data scraping process.

3.1.3 Common File Extensions

Although textual content extracted directly from HTML documents is significant, bulk of the materials still comes in the form of a document file format (e.g *DOCX*). Since extracting data from a file can be laborious depending on the file format itself, finding out which ones are the most used and worth scraping is paramount.

For the purpose of finding out all the possible extensions and their counts, we can perform a web scraping test run, in which we gather all the hyperlink references that point to downloadable file without actually retrieving it. After collecting all the links in our path during scraping, we simply attempt to extract the extensions from the links. The results of an example web scraping run, in which data was gathered for 2015 Fall, 2016 Spring, 2016 Fall and 2017 Spring semesters, can be seen in Table 1.

File extension	Count
pdf	2836
zip	256
pptx	119
tex	87
m	82
php	63
gz	38
htm	33
hs	29
ppt	28
aspx	26
doc	19
bmp	18
pka	17
xls	15
jar	13
docx	13
exe	13
txt	11

File extension	Count
dmg	9
jpg	8
wav	8
p	8
long	5
py	5
odt	4
syb	4
wma	4
v	3
xml	3
png	3
gif	2
json	2
ogv	2
xlsx	2
ml	2
ggb	2
full	2

Table 1. Various extensions that were scraped from Courses and Moodle sources along with their counts.

The goal was to combine a list of all the file extensions that were suitable for extracting textual content. Clearly the most dominant document file format is *PDF* (Portable Document Format), as seen in Table 1. Since *.zip* files don't directly contain any textual data, they were excluded from the final list of extensions to scrape. However, there's a case to be made, in which we extract the contents of *zip* file and scavenge for any files that may contain relevant textual content. As *PPTX* (Microsoft Open XML PowerPoint Presentation file), *DOCX* (Microsoft Word Open XML Format Document file) and *TEX* (LaTeX Source Document) files were also fairly abundant and can be mined for text with relative ease, they were included into the final list.

Whilst extracting content from extensions related to various programming languages (e.g *.m*, *.php*) is not difficult, the content extracted from them does not provide much value and as such are not worth scraping. Consequently, the final list of all the extensions included as a part of the main scraping process is as follows: *.pdf*, *.pptx*, *.tex*, *.docx*.

3.2 Text Extraction

Once a web scraping run over Courses and Moodle web page has been completed and all the relevant files with previously established extensions have been downloaded, our next goal aims to extract the actual textual data from the raw sources and files.

As a first step, we have to take into account that the same material might be present in two different formats (e.g lecture slides are provided in both *PDF* and *PPTX* format). Therefore, before extracting any content from files, we should remove those duplicates in order to reduce any bias during the analysis phase. As a result, a simple extension based filter was applied, in which we assume that the filename without the extension remains the same and only the extensions vary. In case a duplicate is discovered, all references to it will be removed, so it would not be included in the actual extraction process. The filter is applied only in the scope of a single course, i.e the files that were marked as duplicate, have to come from the sources of the same course during web scraping.

From all the various document formats that were allowed to be downloaded during web scraping phase, *PDF* format is the most difficult one to handle, as text extraction can be quite erroneous. While *PDF* document serves well as a final presentation format, which holds the same look as the original document, observing its structure or modifying it can be challenging [14]. The root cause is that *PDF* documents are generally untagged and don't hold the necessary high-level logical structure information [14]. Roughly put, a document consists of page objects, which could be anything from graphic objects to few characters part of a word [14]. The challenge lies in finding and composing the logical structure in order to extract meaningful content [14]. Possible text extraction errors range from being completely unable to extract any comprehensible textual data to smaller grammatical or symbol based errors. For example, the Estonian word *võimalik* can be extracted as *v∼oimalik*, which can have a noticeable impact in downstream processing. Additional concerns are raised with regard to the use of ligatures, i.e when two or more letters are appended together to form a single glyph [1]. Consequently, if any such glyphs should appear in a PDF file, they will be extracted as a whole, not as separate characters [1]. For instance, the inflected Estonian word *graafi* can contain the ligature *fi*, which will be treated as a special symbol and as a result be filtered out during data cleaning, leaving only the *graa* part.

Both *.pptx* and *.docx* are Office Open XML based file formats, basically zipped files that contain XML-like documents [4]. The main challenge lies in extracting the textual content from within the XML tags. However, once extracted, the textual content can be expected to be intact, with minimal amount of textual errors related to the extraction process. On the other hand, *.tex* files don't need to be explicitly further extracted and can be read as a plain text file. The complicated part is removing TeX syntax so that only the main textual content remains. Finally, since web scraping process extracts HTML content from web pages themselves, we also have to consider filtering out the HTML tags in that context, leaving only textual data.

3.3 SIS Data Integration

In addition to data gathered from Courses and Moodle web pages via web scraping, other complementary sources, such as course related information in Study Information System (SIS) of the University of Tartu [9], were also considered. The SIS data included in the analysis was directly extracted from the systems underlying database and came in the format of *CSV* file. Various tables and their respective columns, that could prove pertinent to the analysis and integration process, were included. The full list of included columns can be seen in Table 2.

Column	**Description**
AINEKOOD	Course code
KL_KEEL	Language (either English or Estonian)
WWW	Reference to a relevant web page (usually a link to Courses web page)
KL_OPPEAASTA	Calendar year during which time the course was taught
KIRJELDUS	The goal or general description of a single lecture given during the course
OPIEESMARK	Study goal of the course
YLDEESMARK	General goal of the course

Table 2. List of columns and their descriptions for a single entry in the *CSV* file exported from Study Information System. The data contained an entry for each lecture given during the semester for any given course (i.e the value of column *KIRJELDUS* may vary, but others can stay the same in the scope of one course).

In order to integrate SIS data with the already existing data gathered via web scraping, two vital pieces of information must first be determined in addition to course code - year and semester during which time the course was taught. Although the calendar year was given for each entry (see Figure 2), semester was not. As a result, the column *WWW* was used to determine which semester the course was taught. More specifically, if the link refers to some Courses web page, the semester will be part of the URL.

Since the data was essentially composed of duplicate entries, i.e one entry in Estonian language and a companion entry in English, only one of them was included. The latter was chosen, considering that English based text is easier to clean and incorporate during later phases.

With regard to the actual textual content that was included from each entry, all three goal related fields (*KIRJELDUS*, *OPIEESMARK*, *YLDEESMARK*) were used to compose a single entry for each course. Basically, one instance of both *OPIEESMARK*, *YLDEESMARK* and all the lecture descriptions from *KIRJELDUS* fields in the scope of a single course were aggregated to a one textual resource.

3.4 Incremental Data Updates

Progressively updating the results after the initial data warehouse construction is a vital part of the established ELT process. Capturing only the latest changes with regard to the previous extraction can have a tremendous effect on the overall speed of the data collection phase and goes well with the concept of a periodically runnable process. This can be quite useful for the framework at hand, especially when multiple semesters are targeted during data scraping and the resulting raw data is much more substantial in terms of size. The critical part in performing incremental data updates is classifying which new pieces of data to extract. This usually includes using database dates, logs etc.

Consequently, two fields were added to the table that is responsible for keeping track of the collected course material. The following fields were added with the explicit purpose of constructing a progressively updatable data collection process:

- **Collect time** - the specific date and time during which given course material was collected.
- **File size** - the size of the given course material. In essence, we obtain the meta-information (HTTP headers, assuming the method is HTTP) associated with a specific resource (URL). The material size (i.e Content-Length) is then extracted from the collected meta-information.

Both of these fields are filled during the data scraping process.

In case of an incremental update (i.e initial data warehouse has already been constructed), we first delete all existing results and records (e.g analysis phase results) from previous time the entire process was run, except information related to the collected courses and course materials. We instead nullify the collect time of each course material and then re-run the data scraping step. Once we encounter a course material during data scraping, we first check if the given material already exists locally. The course material matching is done by comparing the name and the size of the material. If we can't find the material by name, then the information is completely new and the material is eligible for downloading. Should the material already exist locally, but the local file size differs from remote file size, then some new information has been added to the document. Subsequently, the local file is deleted and the recently updated material is downloaded again.

During data scraping we always set the collect time of the course material, even if the material already exists locally. After data collection, we perform a cleaning step in which all stale data in the format of a file or database entry is deleted. This means that all courses and course materials that do not match the current target semesters are removed. Additionally, if a course material does not have a date assigned to it (recall, we nullified the collected time during the start of data scraping), we remove it. This step is vital in case a course material is related to one of the current target semesters (e.g material was collected during some previous data scraping run), however, we did not encounter it during the latest data scraping run, indicating that the resource was deleted from the remote data source.

As an additional note, the extracted HTML content is always updated locally, since all the content pages are fully processed anyway during web scraping. Complementary information not related to web scraping (in our case SIS data) is also removed, since including it is a fairly quick (the *CSV* that contains SIS data information is processed in a matter of milliseconds) and it gives the user an option to not include it during incremental updates.

The benefits of using incremental data updates are not limited to just faster data collection, but also provide the means for a potentially faster data extraction. If the textual content was already extracted from a relevant course material (e.g a *PDF* file) during some previous process run, then there is no need to perform the same step again.

3.5 Lessons Learned

Throughout testing and analysing the results of web scraping and data integration processes, several adjustments were considered and implemented in order to improve the quality of the data, before handing it to the next phase - data cleaning. The improvements concerned mainly text extraction and were intended to better the final analysis result by handling special cases, removing some of the bias caused by duplicate text etc.

HTML content extraction. One of the first incorporated changes affected text extraction from HTML content that is scraped from Courses web pages. More specifically, Courses web pages allow students to submit homeworks and other content if the course that they are taking, requires them to do so. As such, students must be logged in beforehand, if they are not, or the submission is not open for example, an error will be displayed. The error text, however, will be scraped as a part of the HTML document and since it is present throughout various courses, it will not be filtered or removed during data cleaning phase. In fact, depending on the error text and its abundance, it can have quite an impact on the result of the analysis. For example various combinations of frequently co-occurring words (see Section 4.3) can be made due to the repetitive nature of error text. Should some of those co-occurring words be linked to a course in the analysis phase, we can only conclude that the course had the option to submit content. While this itself can be informative, the benefit of it should be weighted against its possible dominance in the results of the final analysis due to numerous error text based co-occurring words. As a result, all possible content submission related errors within *<alert></alert>* tags in HTML document were included as a part of a blacklist. The following items were included in the list:

- "You must be logged in and registered to the course in order to submit solutions."
- "There are no tasks for this course."
- "Solutions to this task cannot be submitted at the moment."

In case an *alert* based tag is processed during text extraction, its textual content is first compared against all the elements in the blacklist. The tag and its content are completely removed in case of a match.

Duplicate material improvements. Additional steps were also taken to improve the effectiveness of extension based duplicate material removal. For example, new extension *.6up.pdf* was introduced to the list of allowed extensions to scrape, since in some cases the same content was provided with different layouts, but still used the *.pdf* extension. In this particular case, filenames that ended with *.6up.pdf* had six lecture slides per page, while the regular *PDF* version of it had one slide per page. Adding the new extension will guarantee that those two *PDF* versions will be handled as files with different extensions with one of them being removed due to a conflict. Regarding conflict resolution, in case a duplicate material is detected and one of the files is in *PDF* format, while the other one is not, then the non-PDF version is preferred. In all other cases, the first entry that was processed is kept. This is due to the erroneous nature of text extraction from *PDF* files.

4 Cleaning and Conforming

After collecting and integrating data from different sources, the next vital step involves cleaning and transforming it to a uniform format that is suitable for analysis phase. The chapter at hand focuses on the processing and filtering of the raw textual content gathered and extracted during data extraction phase. As a first step, we describe the tokenization of the textual content, i.e splitting the text into sentences and words. After which, we lemmatize and apply multiple filters to each token (e.g length checks, stop word check) in order to remove unwanted words that would just litter the results of the analysis. In addition, further steps are taken to improve the quality of the analysis results: finding words that frequently occur together (i.e co-occurring words) and resolving acronyms in order to make them synonymous with their definitions. The final section describes various unique cases and problems encountered along the way, both small and big in terms of impact to the final result.

4.1 Tokenization and Lemmatization

The operation of splitting natural language text into relevant and distinct pieces (i.e tokens) is referred to as tokenization [17]. It's counted among the first steps during document processing and while it is not considered a major challenge of Natural Language Processing (NLP), it can have a significant impact in downstream processing [17].

The first step in processing a single material (e.g the content of a lecture slideshow) involves splitting the text into sentences. While it might seem feasible to to detect the boundaries of a sentence based on simple orthographic conventions, it can have noticeable drawbacks when dealing with certain type of texts. For example relying solely on word capitalization is not enough, since certain word classes are capitalized even if they occur inside of a sentence [22]. Furthermore, lower case first letter doesn't definitively mean that the word is preceded by a sentence boundary (e.g in case of mathematical variables or due to some specific conventions) [22]. More sophisticated methods and techniques should be considered to determine proper sentence boundaries. One option would be to use *orthographic heuristic*, which leans on statistical evidence in order to establish if the word is preceded by sentence boundary [22]. Essentially, the heuristic takes into account whether the word occurs with lowercase or uppercase first letter in all other cases in the text, before deciding [22]. An example implementation of tokenization, which splits text into sentences by making use of *orthographic heuristic*, can be seen in *Natural Language Toolkit* (NLTK) [3]. The text-to-sentences tokenization tool provided in NLTK was used in the scope of this work to split the extracted textual content into sentences.

Given that a textual material is split into sentences, the next goal is to split each sentence into tokens. This can be accomplished again by using the sentence-to-word tokenizer provided by NLTK. The challenge here rests in properly handling various symbols and contractions. Concerning symbols, most of them (e.g **;**, **:** and **,**) are split into separate tokens, even if they are part of a word.

$$\text{don't} \rightarrow \text{do n't}$$
$$\text{they'll} \rightarrow \text{they 'll}$$

Figure 6. Example on how sentence-to-text tokenizer splits different contractions.

A special case if made for dash symbol ('-'), which is not classified as a breakpoint and handled as a separate token, unless separated by a whitespace [3]. Another vital part of tokenizing a sentence is handling different contractions (e.g let's, don't). The tokenizer provided by NLTK uses regex expressions to split contractions into separate tokens, depending on the contraction itself [3].

Even though splitting textual material into sentences, and in turn those sentences into tokens, is an important aspect of tokenization, it is equally as important to prune unwanted tokens and normalize the ones we are actually interested in. As such, multiple filters are applied throughout the entire tokenization process with the goal to reduce the amount of undesirable and irrelevant content. The following filters are applied to tokens after converting them to lowercase and trimming surrounding whitespace:

- **Length check** - tokens that are less than three characters long are removed.
- **Latin character check** - the first character of the token has to be Latin. The token is removed if it is determined as something else (e.g Cyrillic, Korean). Another option would be to remove this filter and start handling different non-Latin characters by including appropriate stop words and lemmatizers.
- **Alphabetic character check** - token is removed if it is not comprised of only alphabetical characters. In the event that the token contains even one non-alphabetical character (including numbers) or if the token itself is just some symbol, it is discarded. A an exception is made for dash symbol. If the dash appears at the end of a token, then it is aggregated with the next token to compensate for any line breaks. In case dash is present in the middle of token, we count it as valid. However, if it should appear in the beginning, we discard the entire token.
- **Stop word check** - check each token against a healthy list of stop words (see Section 4.2). Stop word check is done after lemmatizing a token.

- **Token occurrence limit check** - once all the courses and the materials they contain has been processed and tokenized, we check in how many different courses (i.e courses with different course codes) did each token appear. If the token did not occur in at least two different courses, it is discarded. The aim is to reduce the amount of erroneous words made possible by faulty text extraction, tokenization, human error etc.

The length, alphabetic character and stop word checks are done continuously during the processing of a tokenized sentence, whereas token occurrence limit check is done post-tokenization.

Since the analysis phase is based on *bag-of-words model*, we keep track and count the token frequencies throughout the entire tokenization operation. After processing a single course material, we know how many times did each token appear in it, i.e we compose a dictionary of words and their counts for each course material. Once all the material has been processed, we aggregate the token frequencies at a course level, knowing how many times did each token appear in all the materials of a course. As a last step, we aggregate the results at the corpus level.

In the interest of removing unwanted complexity and to improve the quality of downstream processing, a special consideration is put into inflected words. The purpose is to reduce an inflected word into its basic form (e.g *cars* to *car*), so that different derivatives of the same word wouldn't be counted separately. Two different well-known approaches can be considered to achieve this task - stemming and lemmatization.

Stemming attempts to normalize an inflected word by removing its inflectional suffixes, thus reducing it to its root form, i.e stem [11]. While stemming works well for more simple and straightforward cases (e.g *running* can be stemmed to *run*), it does not guarantee that the stemmed word is a dictionary word, but rather an approximation of it [11]. For example the word *calculated* can be stemmed to *calculat* [11]. Issues, such as under-stemming in which we fail to reduce two words with the same meaning into the same stem (e.g the words *throws* and *threw*) and over-stemming, where two words with different meaning are reduced to the same stem (e.g the words *new* and *news*), should be taken into account, before using stemming to normalize words [11].

Lemmatization on the other hand is a much more complex operation, which removes inflectional endings and resolves the lemma of a word by relying on a dictionary that includes a morphological analysis for each word [11]. Even though lemmatization can be considered a more costly operation than stemming in terms of speed, its benefits lay in a potentially more reliable and higher quality of word normalization. Since performance speed was not a critical aspect of the work at hand, lemmatization was chosen as primary tool for word normalization. After a token passes initial filtering (i.e length and alphabetic character checks), it is lemmatized and only then counted as a part of a single course material dictionary, assuming the lemmatized word is not a stop word.

An additional factor, namely language, has to be taken into account when performing word normalization. Seeing that the material collected for each course is mainly in English or Estonian, a different lemmatizer has to be used depending on the language. Therefore, we must first determine the natural language of the material, before executing the main tokenization and filtering logic. Several approaches have been proposed for detecting a language based on written text, for example the authors of [13] described an algorithm that computes an N-gram frequency profile of an input text and compares it with previously established profiles of various languages. Another well-known approach is to use Naive Bayes Classifier for language classification, e.g the authors of [18] proposed a Naive Bayes Classifier that is well suited to identify the language of short texts. For the purposes of present work, a language detection tool (originally based on [27]), that makes use of Naive Bayes filter, is used to classify the language of each course material. Considering that the bulk of the content is in either Estonian or English, a simple logic is applied for final classification - if the language is classified as Estonian, we use an Estonian language based lemmatizer, in all other cases we apply English language based lemmatizer.

Concerning the actual lemmatizer implementations, the WordNet lemmatizer present in [3] toolset is used to lemmatize English words and [25] is used for Estonian words. It should be noted, however, that the WordNet English lemmatizer expects *part-of-speech tag* in addition to the word to be lemmatized. Therefore, POS tagging should be explicitly performed on raw tokenized sentences (i.e before removing stop words etc.) to guarantee proper lemmatization when individual tokens are processed. On the other hand, the Estonian lemmatizer doesn't require explicit POS tagging, only the raw sentence should be provided (i.e no individual token lemmatization, all the sentences are processed as an ensemble by the lemmatizer). In fact, the more textual data is provided, the more effective the lemmatization process is, especially with regard to disambiguation. However, there are still cases where multiple lemmas can be resolved (e.g Estonian word *magasin* can have different lemmas based on the context - *magasin* or *magama*), in that event, no preference is made and the token is processed in its original form.

4.2 Stop Words

In the interest of improving the quality of analysis, words that fill only a functional role and are ambiguous in their nature, i.e *stop words*, should be filtered out as they provide no meaningful benefit to the analysis at hand. Seeing that the material consists mostly of text in either English or Estonian language, a stop word list was composed for both of them. The majority of English stop words were taken from [5], while Estonian stop words were obtained from an extensive list provided in [23].

and	any	all	are	about	around
been	begin	but	can	come	does
etc	few	ever	get	had	let

Figure 7. An example list of English stop words used to validate a token.

Each token is checked against stop words after it has been lemmatized, token itself is removed in case of a match. All the stop words are at least three characters long, since tokens with less are excluded from further processing by the length filter. A total of 678 English stop words and 560 Estonian stop words were included.

In addition, a process which composes a separate list of the names of numerous lecturers and faculty members was implemented. The reason being that the names of the teaching staff can amply occur throughout different course material and subsequently in the results of the final analysis. Considering that this information can be relatively easily looked up, thus not providing any significant value to analysis, all tokens that match with a name in the faculty members list are removed. Even though stop word filtering is done after lemmatizing a token, the check against the faculty member list should be done before, since it is possible that a part of a name (e.g surname) can be lemmatized. This can cause a possible name in text to be interpreted as something completely different and allow it to pass the check.

The list containing the names of the teaching staff is composed automatically by web scraping publicly available faculty employee contact information.

4.3 Frequent Co-occurring Words

Despite the fact that processing and counting only single words is convenient and rather inexpensive, a significant amount of context and meaning can be lost when handling words separately. For example, while words *data* and *mining* are separately relatively ambiguous, together they provide much more context as to the nature of the document. Consequently, frequent word sequence identification was introduced as part of the tokenization process in order to identify conceptually meaningful phrases and word occurrences, i.e *co-occurring words*.

The task of discovering frequent word sequences can pose different challenges and has been a subject of improvement. For example, the authors of [20] define a word sequence as frequent if it is present in at least σ number of documents and propose an algorithm for finding *maximal frequent sequences* (i.e sequences that are frequent and are not part of some other longer frequent sequences). The method also allows to set a maximum gap in between words pairs. For the purposes of present work, however, a slightly different approach was taken, in which frequent word sequence is not defined by occurrence in multiple documents, but in the material of different university courses (i.e courses that have different course codes).

Before actually turning towards counting any co-occurring words, certain pre-processing steps should be taken. As a consequence, tokenization process was involved as a pre-filtering step before finding frequent co-occurring words. More specifically, throughout the processing of a raw sentence by the tokenizer, a clean, filtered version of the same sentence is built (stop words removed, words lemmatized etc.).

Big data is a term that describes the large volume
of data - both structured and unstructured

$\downarrow$

big data term describe volume data structure unstructured

$\downarrow$

big data, *data term*, *term describe*, *describe volume*, *volume data*,
data structure, *structure unstructured*, *big data term*, ...

Figure 8. An example of a raw sentence turned into a cleaned sentence during tokenization. The cleaned sentence is then used to create a list of different N-grams, which is then reduced by various filters.

Once the tokenization process is complete and all clean sentences have been resolved for each document, another pass is made to compute N-grams of specified length. By default the N-grams range from size 2 to 4. The resulting N-grams are then aggregated at a corpus level, whilst keeping track of how many courses did each N-gram appear in. The global N-gram dictionary is then thoroughly pruned by applying following filters:

- **Corpus level occurrence limit** - the co-occurring word has to appear in at least σ number of times at the corpus level, where σ is some fixed threshold. By default σ is set to 20.
- **Course level occurrence limit** - in order to remove course specific textual idiosyncrasies, the co-occurring word is expected to be present in the material of at least δ different courses. By default δ is set to 3.

- **Duplicate word check** - if the co-occurring word is comprised of the same single word multiple times (e.g *num num*), it is removed.
- **Subsequence check** - only the highest level N-gram is kept, all subsequences are discarded (e.g *neural network* would be removed in favor of a longer frequent N-gram *recurrent neural network*)

A special case is made in case the co-occurring word is a definition of an already established acronym (see Section 4.4). In that event, the corpus and course level occurrence limits are ignored.

sender receiver	system requirements
visualization techniques	sirge paralleelne
coding errors	written java
kuluma aeg	random graph
image compression	planar graph

Figure 9. An example of list resolved frequent co-occurring words.

As a final step, each course material word dictionary is adjusted by removing the individual words that make up the newly resolved frequent co-occurring words (e.g for co-occurring word *text analysis*, individual words *text* and *analysis* are removed from course material dictionary). We then recount how many times each frequent co-occurring word appears in a course material and if it appeared at least once, we append the co-occurring word with its respective count to the dictionary. This operation is a lot cheaper than making a full pass over the material for each co-occurring word, considering the fact that if all the individual words did not appear in the dictionary, then there is no point in counting how many times did the co-occurring word appear in the course material.

4.4 Acronym Extraction

A simple observation of the initial results of the tokenization process suggested that the textual material contained a vast amount of abbreviations, more specifically acronyms. As a result, the corpus level word dictionary can include short tokens that are actually acronyms and more often than not their definitions are not always readily apparent to the reader. More importantly, when we compose the word dictionaries, we count acronyms and their definitions separately, since their use in either form is valid. In order to alleviate this concern, it might be prudent to automatically identify acronyms and find out their respective definitions. Once we have done that, we can substitute all acronyms in their short form with long form in the text, thus counting acronyms and their definitions as one.

Numerous approaches have been proposed for automatically identifying acronyms. For example the author of [15] suggest and evaluate a rule based algorithm in addition to comparing different machine learning based algorithms. Similarly, the authors of [26] propose a simple rule-based algorithm for extracting acronyms and their definitions from biomedical texts. The major advantage of rule-based algorithms is that they don't require any training data. However, detecting certain acronyms (e.g when acronym letters match with a character inside a definition word) can be problematic and require additional consideration. An automatic acronym detection approach, similar to [15] and [26] was used in the scope of present work.

As a first step we define how we categorize some tokens as acronym candidates, i.e words that could potentially be acronyms, but warrant further investigation of surrounding tokens and previously established acronym definitions. The following rules are applied during the tokenization process to determine if the token could be an acronym:

- **Length limit** - seeing that the default maximum limit for words that comprise frequent co-occurring words is 4, a similar restriction is set on the acronym character length. Recall, one of the goals is to count the acronyms and definitions as one entity, and since we can't have definitions longer than 4 words, we set a maximum character limit. In addition, tokens that have a length of less than 3 character are filtered out, so in essence, we only consider acronyms that have 3 or 4 characters.
- **Word uppercase limit** - the short form an acronym has to be uppercase, only the first or the last character may be lowercase.

None of the potential acronyms will be lemmatized, but instead processed in their raw form. In the event that a token is classified as a potential acronym, we search the surrounding tokens for a definition. We expect either the acronym short form or the long form to be in parenthesis, i.e one of the following cases:

acronym (definition),
definition (*acronym*).

The first characters of the definition words have to make up the acronym. If the acronym is 3 characters long, then the definition is expected to comprise of 3 words. After composing an acronym definition dictionary for each course material, we aggregate the definitions and build a global definition dictionary. In case a conflict should arise during aggregation (i.e same acronym short forms, but different long forms), we skip adding the definition all-together. As a final step we loop through all the course material word dictionaries and replace all potential acronyms with their definitions by first looking at the course material acronym definition dictionary and then the global dictionary.

The previously described definition finding process and rules for establishing known acronyms are fairly strict and while they do reduce the amount of false classifications, various acronyms can be missed (e.g *iOS*, since we don't consider internal characters of definition words to be part of the acronym short form). A more flexible acronym detection process, in which a wider range of abbreviations are extracted from text, should be considered as a part of future work.

4.5 Lessons Learned

Several functional improvements were considered and implemented to handle specific failure scenarios during tokenization. Furthermore, a non-functional improvement, namely better overall performance speed, was investigated.

False text language classification. Language detection is the first step when tokenizing a course material. Since we handle two different natural languages - Estonian and English - it is completely plausible that an Estonian based text might be falsely classified as some other language (not necessarily English) due to mixed language text, too short text, formulas etc. In that case, an Estonian text will be classified as an English text due to the fallback mechanism (all non-Estonian languages are classified as English). While we can't do much in terms of lemmatization in that event, we can reduce the amount of non-informative words. Consequently, both English and Estonian stop top words are combined into a single list. Although this can be an hindrance on performance speed (we have to check against a bigger list each time), the possible gain is better quality word corpus.

Post-lemmatization length check. Even though length check is one of the first filters we apply when processing a sentence token, it should also be done after lemmatization of a word. For example, if word *xss* is in uppercase form, it will be identified as an acronym, however, should it be in lowercase form for what-ever reason, it will be lemmatized to the character *x*. In order to avoid such cases (e.g single characters passing the filtering process), the token length check is also done after lemmatization.

Counting of embedded frequent co-occurring words. Recall, after finding all frequent co-occurring words, we count how many times did each of the them appear in a document. For this purpose, we build clean sentences and plainly check how many times each co-occurring word appears in them. However, using a string matching algorithm to achieve that, can have its drawbacks. For example the co-occurring word *code input* can appear inside *encode input*, as a result the counts of both co-occurring words will be incremented. This is not a concern during single word counting, since we apply tokenization and process each token sequentially, but re-tokenizing clean sentences just to count co-occurring words may seem redundant. As a solution, each sentence was padded with spaces from both ends and instead of matching co-occurring words directly, we pad them first with spaces and then count via string matching. E.g instead of searching how many times *'code input'* appears in a sentence, we instead use *' code input '* as a lookup string. This way we can still make use of the fast string matching algorithm in our implementation.

Parallelization of tokenization process. Each course material is tokenized independently, making the entire process a good candidate for parallelization. Despite the fact that multiprocessing is used to handle each course material in parallel in the scope of present work, a potential for further improvement lies in offloading the computation to GPU. For example the authors of [29] show a significant improvement over CPU-based algorithms for different text mining operations, including tokenizing text into sentences and removing stop words.

5 Text Analysis

The final step of ELT process produces a suitable foundation for performing further text analysis, more precisely - topic modelling. The first section introduces Latent Dirichlet Allocation (LDA) as a topic model and provides an overview of LDA from a more theoretical aspect. The following section then describes, how LDA is applied at a course level (i.e by using course level word frequency dictionary) in order to find out which courses fall under the same topic. Afterwards, we perform topic modelling at a different level of granularity - course material level. The final section outlines the various methods used for resolving titles for topics.

5.1 Latent Dirichlet Allocation

Topic modelling is a method in natural language processing in which the goal is to classify and summarize a collection of documents by finding a topic (i.e a group of words) or a set of topics that best describe the textual information that the documents contain. Several techniques exist for finding the topics, one of them being *Latent Dirichlet Allocation (LDA)*, first introduced in [12]. LDA is essentially a generative process that identifies a range of common topics based on given documents. The assumption is that each document is comprised of a set of limited topics and all the words in the document can be associated with one of these topics. In general, LDA aims to do the following:

- discover the predefined amount of topics;
- assign words for given topics;
- associate a subset of those topics with documents.

Subsequently, LDA can be used to find and describe similarities in the provided data by grouping different features under the same set, i.e topic.

The general LDA model for composing a single document starts by sampling a mixture of topic proportions θ, i.e a document-topic distribution for given document from Dirichlet distribution, parameterized by α (a vector of positive real numbers, which specifies the nature of Dirichlet prior on θ) [12]. Next, we choose a topic based on θ for each of the n'th position words ω_n in the document [12]. Once we know the topic, we assign an actual word based on multinomial probability $p(\omega_n|z_n, \beta)$, where β is a hyperparameter for Dirichlet prior on the topic-word distribution sampled from Dirichlet distribution [12]. The procedure is summarized in Figure 10.

The challenge lies in learning the actual topic-word and document-topic distributions, as exact inference is difficult to determine. However, various approaches that aim to approximate those distributions have been proposed. The original LDA paper [12] authors

```
For each document d:
    1. Pick $\theta_d \sim Dir(\alpha)$
    2. For each $n$ position word:
        (a) Pick a topic $z_{d,n} \sim Multinomial(\theta_d)$
        (b) Pick a word $\omega_{d,n}$ from $p(\omega_{d,n}|z_{d,n}, \beta)$, a multinomial probability, conditioned on topic $z_{d,n}$
```

Figure 10. General LDA model for generating documents [12].

made use of *variational Bayes* approximation, but other well known approaches, such as *Gibbs sampling* [19], have been put forth.

LDA based topic modelling is used as primary tool for analysis in the scope of present work. The goal is to find similarities between courses that fall under the same topic. Regarding implementation, LDA modelling tool [2], which makes use of collapsed Gibbs sampling for inference, is utilized throughout the analysis phase.

5.2 Course Topic Modelling

Considering that one of the main objectives is to find relations between different courses, it is reasonable to first apply topic modelling at a course level, i.e we use course term frequency dictionary as a unit, not the term frequencies of a singular document. As such, we use the term frequency dictionary to compose a proper *document-term matrix (DTM)*, where on one axis we have terms gathered from all the documents in our corpus, and on the other axis we have documents (in this particular case - courses). The matrix itself reflects how many times did each term appear in different courses. The DTM is then supplied to the LDA topic modelling tool for analysis.

In addition to DTM, several other parameters have to be specified, the first one being topic count. Seeing that the modelling is done in the context of university courses, we can use that to set an initial topic count. While the material of different courses can overlap, each course still has its own unique theme or goal in terms of what to teach. Therefore, in the current context, we can assume that there are as many topics as there are courses. A particular advantage of using a dynamic parameter such as course count, is that as new data is loaded, we don't have to blindly guess the topic count, thus making the automation of the entire process, from data scraping to visualization, much easier.

As a second parameter we have to provide an iteration number for the underlying algorithm. This is due to the fact that the topic modelling tool uses Gibbs sampling for inference, hence we have to establish how many iterations of Gibbs sampling to run beforehand. In order to get a better idea on how to choose the parameter, an example topic modelling run was executed on two academic years worth of data that was gathered and cleansed in previous steps. The log likelihood was recorded throughout topic modelling process.

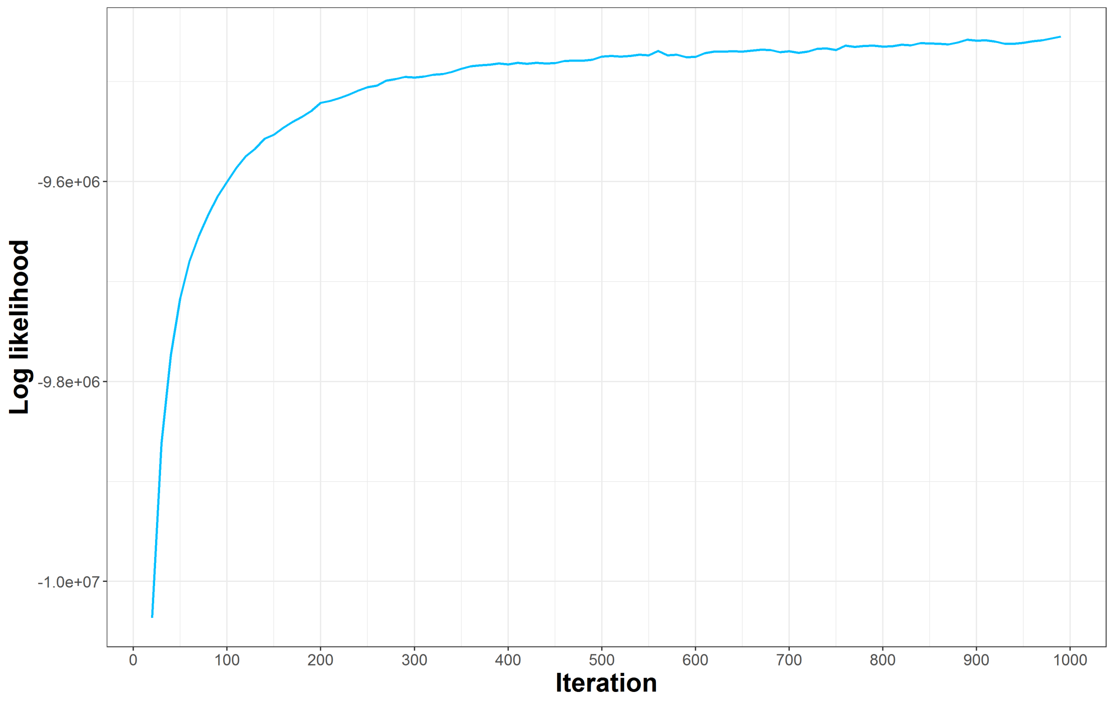


Figure 11. Log likelihood convergence based on an example topic modelling run on data gathered from 2015/2016 and 2016/2017 academic years. The first iterations of Gibbs sampling were excluded from the plot due to naturally high rate of change in log likelihood during initial iterations.

We can observe the convergence of log likelihood in Figure 11. Basically, the convergence is already achieved in the first 500 Gibbs sampling iterations, thus setting the general tone for iteration count in future topic modelling runs. As a final step during topic modelling, we record the top 5 topics (topics with highest course-topic probability) for each course and top 10 words (words with highest topic-word probability) for each topic, based on the fitted LDA model.

By default the words in each topic are ordered by their respectful topic-word probability and we only choose the top N words to describe each topic. However, by choosing the highest probability as our metric, we lean more towards common and frequent words. In the interest of providing an alternative point of view, we can apply different normalization techniques and hopefully gain a more descriptive summary for each topic. A simple normalization approach, which uses the resolved topic-word probabilities, was considered in the scope of present work. The idea is to normalize the probability value of each word in a topic-word distribution by dividing it with the average probability of given word over all topic-word distributions. Admittedly the newly normalized value won't represent a probability anymore (thus it doesn't fit our LDA model), but it is still suitable for re-ordering the words within given topic in order to gain a slightly different perspective. The objective is to find distinctive words, i.e words that uniquely separate or describe given topic with regard to all other topics. As such, we give more weight to words that have high probability within given topic compared with all other topics. After normalizing and re-ordering the words in each topic, the top 10 values are persisted for visualization purposes.

5.3 Material Topic Modelling

In addition to performing topic modelling at course level, a higher level of granularity, namely course material level, was also considered and implemented. This implies two different changes with respect to course topic modelling. First of all, instead of using course term frequency counts, we build a document-term matrix for each material in the corpus, completely disregarding the bounds of university courses. Consequently, we should be able to observe similarities between various material or for example describe which topics are handled in the scope of a single material. The second change affects the expected topic count provided before topic modelling. Since the number of different materials range in the thousands, a simple one-to-one matching of material count and topic count is not feasible. Seeing that different materials can cover the same topic, a simple formula, which takes the number of materials into account, is applied instead:

$$topic\_count = \frac{material\_count}{10}.$$

The actual fraction of the material count is a subject for further improvement and was chosen based on the overall number of materials gathered in a sample data scraping test run targeting two academic years (i.e 4 semesters).

Its should be noted that an approach in which we model topics over the material of a single course was also investigated. The goal was to perform topic modelling for each course separately and examine similarities between materials in the scope of a single course, e.g some lectures cover the same topic. However, this approach was later discarded, since the results were not that informative. Similarities in the scope of a single course are usually readily apparent due to relatively small number of materials and are already known or even intentional by the course organizer. In some cases two documents were largely comprised of the same topic when in fact they were duplicates, but due to various file naming conventions, one of the duplicate materials was not removed in the data extraction phase. All things considered, the approach to model topics over the materials of single course was deemed unnecessary, since it did not provide much value, and was subsequently removed from the entire analysis process.

5.4 Topic Title Resolving

Once the LDA topic model has been fitted and we know all the topics, we can attempt to resolve a title for each topic, e.g if the topic consists of words such as *python*, *data structure* and *class*, then the topic title would be *Programming*. Fortunately, resolving a topic title is more convenient in the current context, since the course titles themselves are a representation of what they contain. The question remains, however, which of the course titles to pick and should we alter it to make it more suitable for our purposes? In response, two different approaches were considered, the first one being more straightforward, easier to implement and understand, the second one using a little more sophisticated heuristics, thus having the potential to give better results. Both approaches make use of the document-topic distributions.

The first approach simply checks the highest document-topic probability for given topic, if the said probability falls under a certain limit, we assign *General* as topic title, otherwise we use the document name (i.e in current context - the course name) as title. The problem lies in choosing the suitable hard limit. In the interest of finding a suitable limit, an example topic modelling run over two academic years worth of gathered data was executed and the highest document-topic probabilities for each topic were used to determine topic titles. The document-topic probabilities were summarized via density plot in Figure 12.

The ultimate goal of the limit is to resolve minimum amount of *General* topics, whilst still keeping the integrity and validity of the resolved topic names. We can observe from Figure 12 that setting the hardcoded limit to as low as 25% would still mean a fairly big portion of the topics would be classified as *General*. Even then, the course would have to contain only 25% of the topic to be considered as the dominant element and the course name to be picked as topic title.

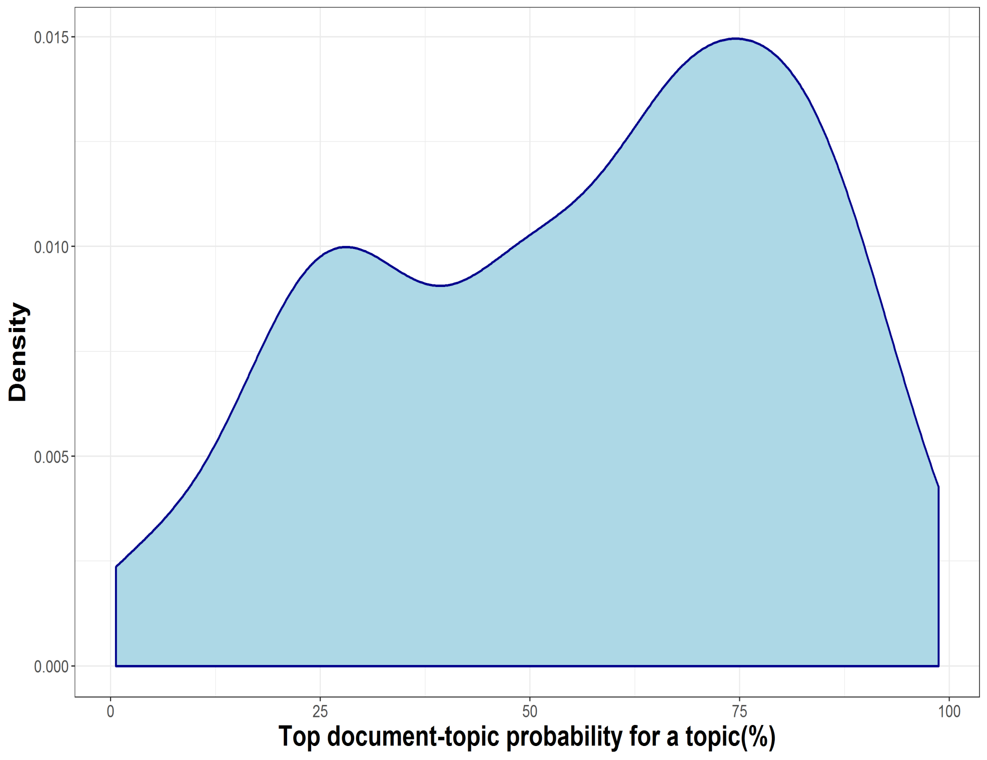


Figure 12. Density plot of the highest document-topic probability for each topic.

The second approach doesn't just focus on the top document-topic probability, but considers others as well. The goal is to compare the top document-topic probabilities for given topic with the next highest probability, if the top one is not higher by some factor we don't assign the course name as topic title.

Furthermore, instead of just comparing the top two document-topic probabilities, we aggregate the top probabilities should the course names match and then compare the aggregated value to the next highest probability for given topic. This is especially useful when more than one semester worth of data is included in the analysis. For example some courses are given every semester and if the same course is included from two semesters, then it is reasonable to assume that they would both contain the same topic, unless the course material has changed significantly. Similarly to first approach, the objective is to now find a suitable multiplier for comparison. We can observe from Figure 13 how many topics were assigned the *General* title for different multipliers. Essentially, even with multiplier in the range of $1.5 - 2.0$ the *General* topic count stayed in the reasonably low range of $4\% - 5\%$.

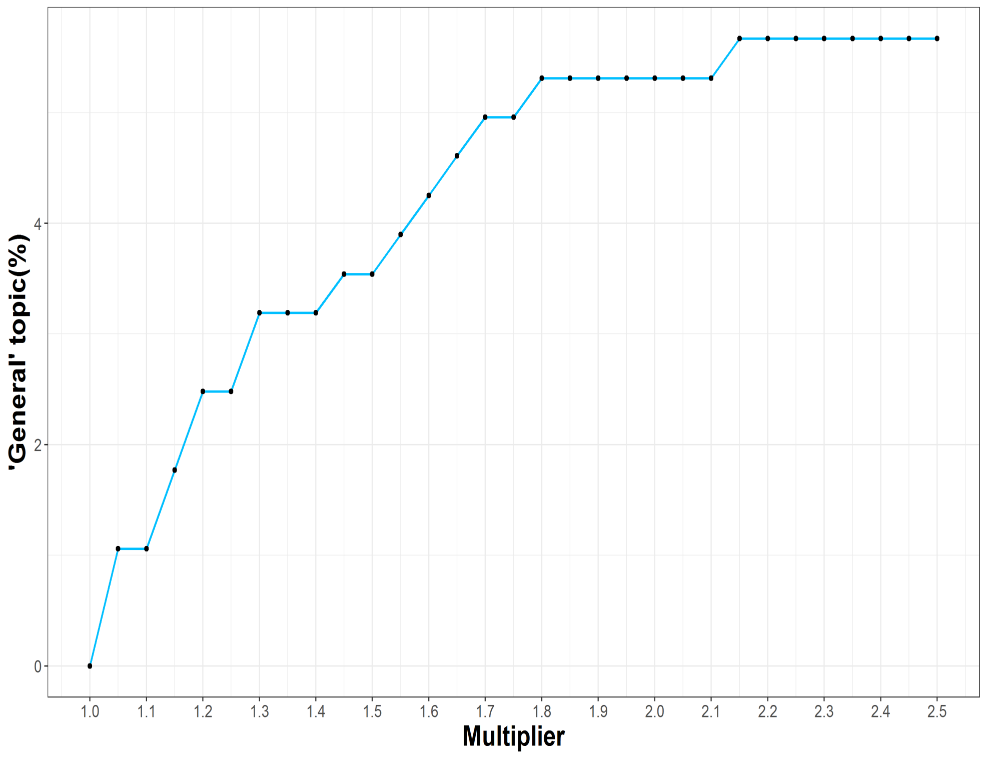


Figure 13. Resolved ’General’ topic percentage with regard to total topic count for different multipliers. After the top highest (course names have to match) document-topic probabilities are aggregated, their total sum has to be higher than the next highest probability multiplied by some factor.

However, even if the probability comparison fails, a fallback heuristic in which we look for matching words in the course names is put to use. Basically, we take the name of the course with highest document-topic probability for given topic and try to match the individual course name words with the names of other courses that contain the same topic. If such matching words are found, we aggregate them and use it as a topic title.

While both approaches were considered, the second one was chosen purely based on the significantly lower *General* topic count. A multiplier of 1.5 is used by default. Nonetheless, both approaches use course names to resolve titles and while this is suitable for most cases, the course names still require further processing. As an additional feature, if a course name was designated as a topic title, we first put the topic name through a filtering process in order to resolve a more appropriate topic name. The following steps are taken during the filtering process:

- **Blacklisted word removal** - various blacklisted words (e.g *seminar*, *introduction*) are removed from course names as they provide no value as a topic title. For example, a course named *'Introduction to Programming'* is reduced to just *'Programming'*.
- **Removal of text in parenthesis** - some course names contain some specific additional information in parenthesis. Consequently all parenthesis and the information they contain are removed.
- **Removal of course level indicators** - course names can contain Arabic or Roman numerals in order to indicate course level (e.g *Algebra II*). All such indicators are removed.
- **Removal of non-alphabetic characters** - in rare cases the course name started with non-alphabetic characters. Since they serve no purpose in topic titles, they are filtered out.

6 Results and Visualization

The final step focuses on the visualization of the results produced by previously defined process, starting from data collection, up to performing text analysis on cleansed data. We start by describing the parameters and the overall setup used to perform an example full execution of the established process, followed by a brief summary of the data that was collected and used for text analysis. Afterwards, all the results from topic modelling and otherwise are visualized in various ways. Firstly, we concentrate on presenting the course topic modelling results via heatmap and descriptive tables. Finally, we provide the means to examine each course separately, including how its documents fitted into the material topic model.

6.1 Process Setup and Results

Before running the entire process, starting from data scraping to actual visualization, certain parameters have to be set, the most important one being target semesters. Two full academic years were targeted in our example run, with the goal to extract courses that might not be given every semester or every year, but over year. Considering that the goal is to gather as much data as possible, factors related to data extraction and integration have to be taken into account. For example, in this instance, data gathered from Moodle web page is assumed to be from on-going semester. Thus the last semester we should include in our example run is the on-going semester (currently 2017 Spring semester). Consequently, the targeted semesters are as follows: 2015 Fall, 2016 Spring, 2016 Fall, 2017 Spring. Other parameters are more specific in their nature and depend on the target goals, e.g do we want to include SIS data, gather the names of the faculty members. Both previously mentioned options were enabled. The process itself is fully automated and requires only the setting of the initial parameters.

The overall results with regard to the acquired data from the example run can be seen in Table 3. As expected, the bulk of the data came from Courses web page, with 2016 Fall semester in particular having the highest number of courses and materials. While the data extracted from Moodle is mainly associated with the latest semester, there are still few exceptions in which Courses web page referred to the material in Moodle. Therefore some Moodle specific materials are associated with other semesters, currently 2016 Fall and 2016 Spring semesters. The least amount of data was included from SIS database, with only 11 composed materials. This is due to the fact that a lot of the entries could not be associated with a particular semester with the currently used semester-URL matching technique. Should there exist a column which directly associates each entry with a specific semester, the amount of data included would be significantly higher.

	2015 Fall	2016 Spring	2016 Fall	2017 Spring	Total
Courses	60	69	95	86	310
Courses materials	883	876	1304	905	3968
Moodle materials	0	3	2	103	108
SIS data materials	3	2	6	0	11
All materials	886	881	1312	1008	4087

Table 3. General descriptive statistics for the data gathered from different sources during data extraction phase.

After cleaning and normalizing all the textual content, a total of 24421 unique words were resolved at a corpus level (see Figure 14 for top words at corpus level). From those, 6325 were frequent co-occurring words. In addition, the acronym extraction feature determined 150 acronym definitions at a corpus level. Due to acronym conflicts (i.e an acronym having more than one definition), the actual amount can be even higher.

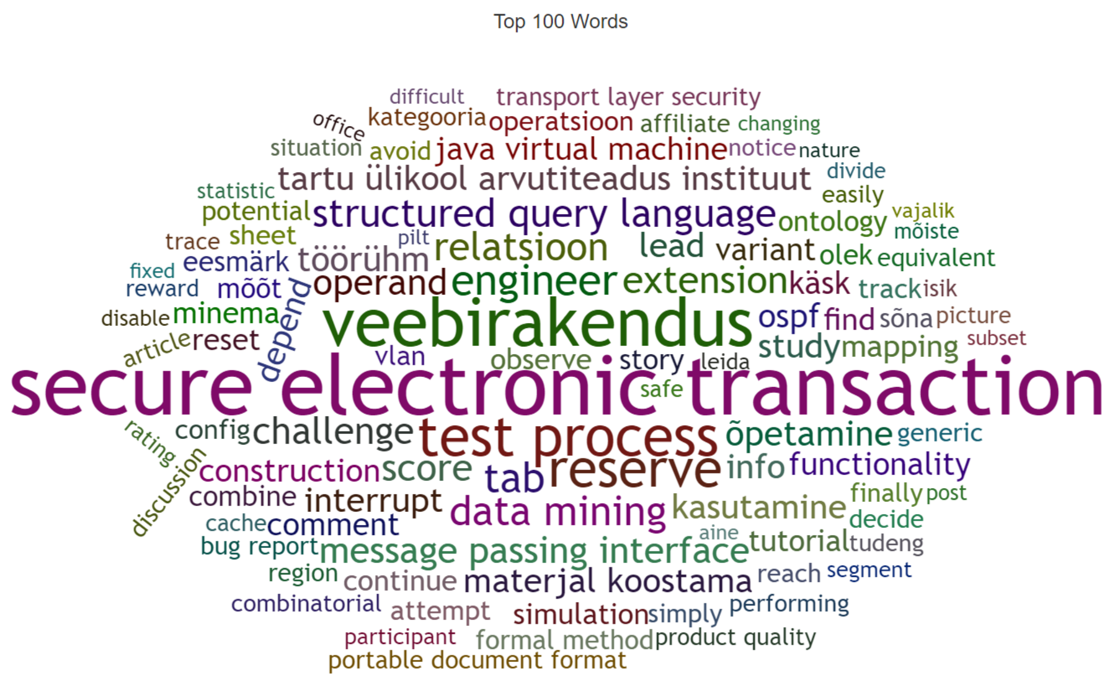


Figure 14. Word cloud based on unique words and their counts at corpus level.

.

6.2 Course-Topic Relations

Finding various relations between courses via topic modelling is the primary objective of text analysis. However, the proper visualization and presentation of topic modelling results can be challenging, especially when a fairly large amount of data is included into the analysis process, resulting in a high number of topics. Regarding the example run done over the academic years of 2015/2016 and 2016/2017, a total of 168 topics were set and resolved during the topic modelling step. Keeping in mind, that while the idea is to match the number topics with the number of courses, the number of courses for topic modelling is still counted based on unique course codes. Since some courses are given every semester, the number of unique courses codes is naturally lower than the total course count throughout the entire two academic years, thus the lower topic count.

A general overview of the course-topic distributions can be visualized via heatmap, where on one axis we have courses, the other axis has topics and the values represent the course-topic probabilities. The heatmap form of visualization can give a good overview, but should the number of courses and topics get too big, the relations can get too difficult to spot and track.

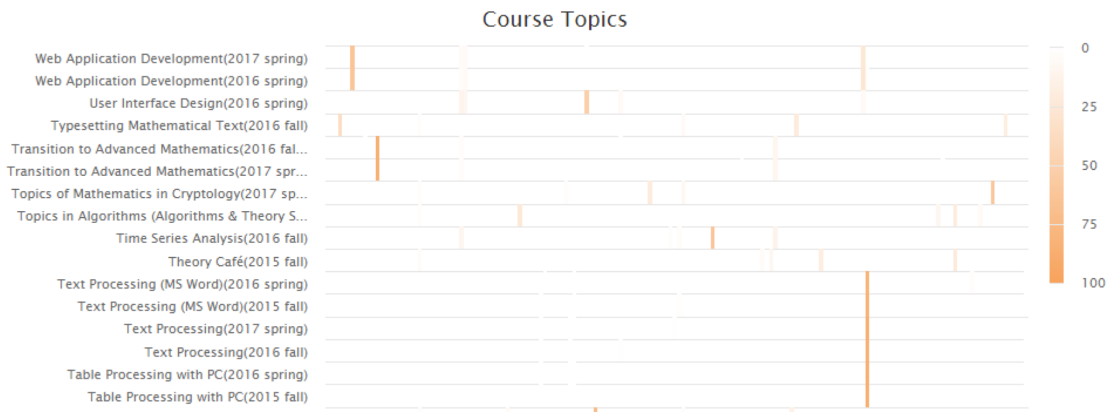
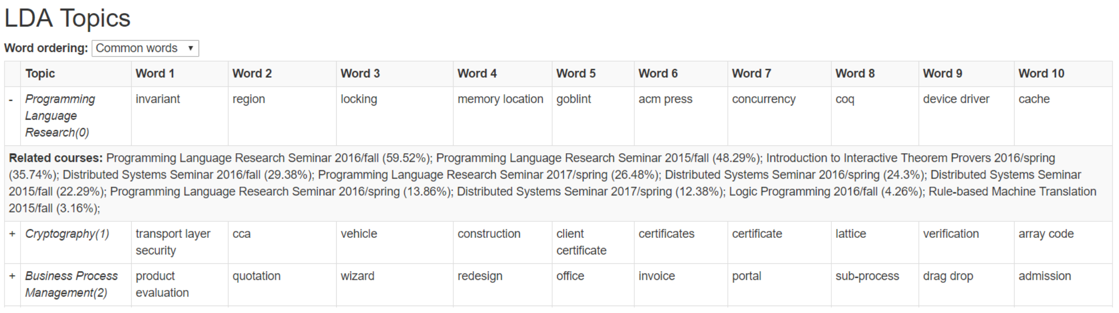


Figure 15. A sample portion of a heatmap visualization of course-topic distributions. Based on an example run done over 2015/2016 and 2016/2017 academic years. The full version of the heatmap can be observed at [10].

An alternative way of discovering relations is to look at the course-topic distributions from the topic angle, i.e which documents contain that topic. For this purpose, we can use a simple table format, where each row represents a topic and related information (e.g top words based on topic-word distribution and courses that contain that topic). Furthermore, two different word orderings for a topic can be chosen: *common words* (i.e top 10 words with highest topic-word probabilities), *distinctive words* (i.e top 10 words based on the values gained by normalizing the topic-word probabilities over all topic-word distributions).



LDA Topics

Word ordering: Common words

	Topic	Word 1	Word 2	Word 3	Word 4	Word 5	Word 6	Word 7	Word 8	Word 9	Word 10
-	*Programming Language Research(0)*	invariant	region	locking	memory location	goblint	acm press	concurrency	coq	device driver	cache
Related courses: Programming Language Research Seminar 2016/fall (59.52%); Programming Language Research Seminar 2015/fall (48.29%); Introduction to Interactive Theorem Provers 2016/spring (35.74%); Distributed Systems Seminar 2016/fall (29.38%); Programming Language Research Seminar 2017/spring (26.48%); Distributed Systems Seminar 2016/spring (24.3%); Distributed Systems Seminar 2015/fall (22.29%); Programming Language Research Seminar 2016/spring (13.86%); Distributed Systems Seminar 2017/spring (12.38%); Logic Programming 2016/fall (4.26%); Rule-based Machine Translation 2015/fall (3.16%);											
+	*Cryptography(1)*	transport layer security	cca	vehicle	construction	client certificate	certificates	certificate	lattice	verification	array code
+	*Business Process Management(2)*	product evaluation	quotation	wizard	redesign	office	invoice	portal	sub-process	drag drop	admission

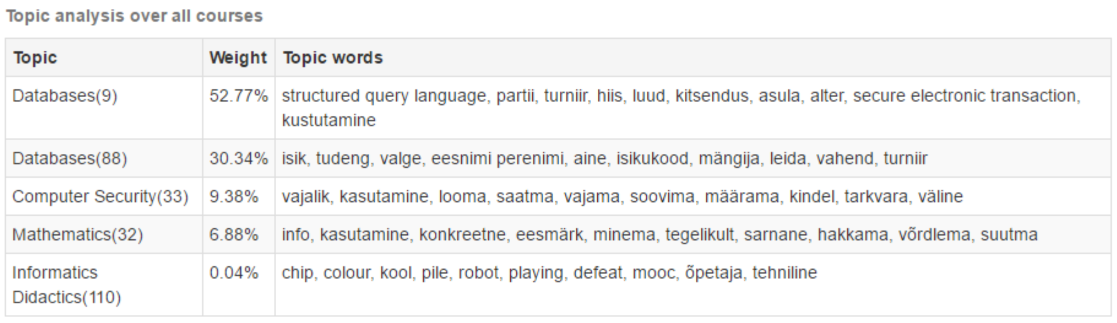
Figure 16. A sample portion of a table representing resolved topics and their details (top 10 words with highest probability in topic-word distribution and documents that contain given topic). Based on an example run done over 2015/2016 and 2016/2017 academic years. The full visualization of the table can be observed at [10].

Regarding the analysis results, numerous relations can be observed and investigated based on given example run. The most common relations are naturally between instances of the same course given in different semesters. Additionally, complementary courses usually fall under the same topic (e.g *Data Mining* and *Data Mining Seminar*). More specific relations and their causes can be further explored based on need and interest.

6.3 Course Overview

The results of the analysis can also be viewed from the perspective of a single course, including general statistics, top words via word cloud, the outcome of the topic modelling done over the material of all courses etc. For example, a complementary view (see Figure 17) of the course-topic distribution can be seen from the side of a given course, i.e which topics does it contain.

The results of the topic modelling performed at a higher level of granularity, i.e material level, can be examined from the course point of view. Similarly to course-topic modelling, the goal is to show which topics each material contains for given course and do the materials overlap with documents from other courses.


Topic analysis over all courses

Topic	Weight	Topic words
Databases(9)	52.77%	structured query language, partii, turniir, hiis, luud, kitsendus, asula, alter, secure electronic transaction, kustutamine
Databases(88)	30.34%	isik, tudeng, valge, eesnimi perenimi, aine, isikukood, mängija, leida, vahend, turniir
Computer Security(33)	9.38%	vajalik, kasutamine, looma, saatma, vajama, soovima, määrama, kindel, tarkvara, väline
Mathematics(32)	6.88%	info, kasutamine, konkreetne, eesmärk, minema, tegelikult, sarnane, hakkama, võrdlema, suutma
Informatics Didactics(110)	0.04%	chip, colour, kool, pile, robot, playing, defeat, mooc, õpetaja, tehniline

Figure 17. An example of top 5 topics along with their respective course-topic probabilities for course *Databases (2015/Fall)*, based on the example run done over 2015/2016 and 2016/2017 academic years.

An example visualization of material-topic relations can be seen in Figure 18. As a side note, even by taking 10% of material count as topic count, the number of topics will still range in the few hundreds (assuming two academic years worth of data is scraped). In order to better visualize the results, only document-topic relations with probability over 10% are persisted during analysis phase, all other marginal relations are discarded and considered in the *Other* group.

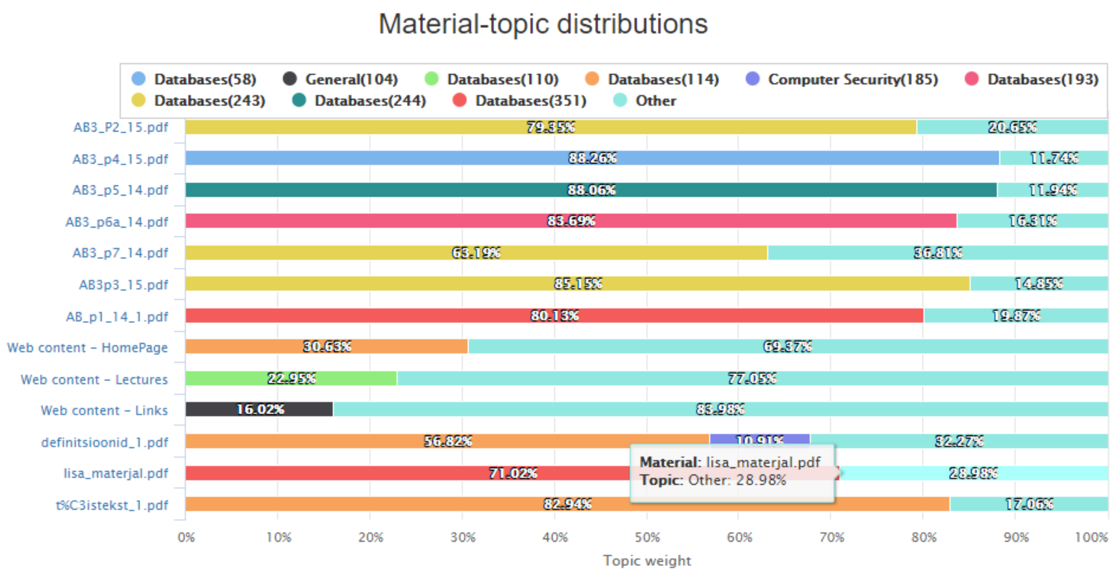
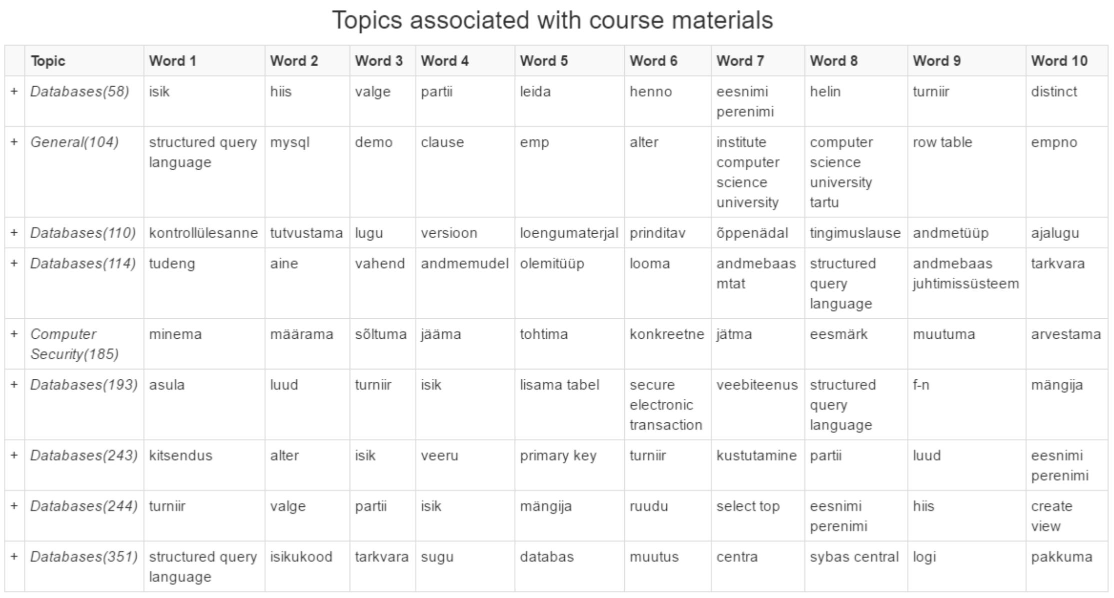


Figure 18. An example stacked bar chart showing the topics each material contains for course *Databases (2015/Fall)*.

The purpose of the material topic table (see for example Figure 19) is to examine in which topics the materials for given course were included. Furthermore, each topic row displays which materials (from all courses) contain given topic, thus providing the opportunity to spot similarities between cross-course materials. The materials of given course are marked in bold to differentiate them from other materials. The titles for course material topics are resolved in a similar matter as are titles for course topics, i.e based on document-topic probabilities and course names. The reason being that the course material names are often times not informative enough to consider them as topic titles. As such, we resolve topic titles at course material level based on the course names that given topic materials are a part of. Finally, it should be noted that we show only the course material topics relevant for given course.


Topics associated with course materials

	Topic	Word 1	Word 2	Word 3	Word 4	Word 5	Word 6	Word 7	Word 8	Word 9	Word 10
+	*Databases(58)*	isik	hiis	valge	partii	leida	henno	eesnimi perenimi	helin	turniir	distinct
+	*General(104)*	structured query language	mysql	demo	clause	emp	alter	institute computer science university	computer science university tartu	row table	empno
+	*Databases(110)*	kontrollülesanne	tutvustama	lugu	versioon	loengumaterjal	prinditav	õppenädal	tingimuslause	andmetüüp	ajalugu
+	*Databases(114)*	tudeng	aine	vahend	andmemudel	olemitüüp	looma	andmebaas mtat	structured query language	andmebaas juhtimissüsteem	tarkvara
+	*Computer Security(185)*	minema	määrama	sõltuma	jääma	tohtima	konkreetne	jätma	eesmärk	muutuma	arvestama
+	*Databases(193)*	asula	luud	turniir	isik	lisama tabel	secure electronic transaction	veebiteenus	structured query language	f-n	mängija
+	*Databases(243)*	kitsendus	alter	isik	veeru	primary key	turniir	kustutamine	partii	luud	eesnimi perenimi
+	*Databases(244)*	turniir	valge	partii	isik	mängija	ruudu	select top	eesnimi perenimi	hiis	create view
+	*Databases(351)*	structured query language	isikukood	tarkvara	sugu	databas	muutus	centra	sybas central	logi	pakkuma

Figure 19. An example table showing the topics in which the course *Databases (2015/Fall)* material were included.

7 Future work

While the framework proposed in the scope of present work shows great promise in terms of progressively evaluating and giving an overview of the courses taught and their relations, it still has a lot of room for improvement in all aspects of the entire process. Most importantly the data gathering part can be made more effective by incorporating new sources as different faculties and institutes start using Courses web page. The scraping process would be significantly more efficient when a uniform format is enforced, especially in case of Moodle environment, where each course page has its own design and structure. Moreover, a lot of the courses in Moodle environment are not publicly accessible, thus further reducing the amount of potential data to be collected.

The textual content extraction from files can be further improved by adding the support to extract data from other file types. However, considering that most of the data comes in the format of a *PDF* file, making the *PDF* content extraction more robust by handling different erroneous cases (e.g properly handling ligatures), should have a positive impact on the quality of the eventual analysis results.

The data cleaning phase has a significant impact on the analysis results, as better quality data used for analysis means better and more meaningful results. One of the more evident areas for improvement is acronym extraction, as the proposed solution is fairly strict and as a result some acronyms are missed. In addition, the cleaning phase currently focuses only on two languages - Estonian and English. The initial results, however, showed that a small, albeit noticeable portion of the content was in Russian, thus giving the opportunity to incorporate another language in to the mix by establishing an appropriate stop word list and using an appropriate lemmatizer.

Concerning text analysis, the applied LDA modelling is more inclined towards high-frequency words, i.e we consider the most common words. However, different metrics can be used to estimate the correlation between documents and words. For example, by using LDA model that is based on mutual information, we would consider more semantically important terms, instead of just the most frequent ones. Applying a different metric could have a profound effect on the quality of the resolved topics.

Finally, visualizing the results of the topic modelling in a comprehensible and a elegant way is a challenging task. Improving the existing visualization and finding new ways to represent the topics can go a long way in terms of increasing the usefulness of found results. Moreover, making the usage of the framework more user friendly is paramount. This especially entails making the use of the automized process more convenient, starting by making a user interface for setting the initial parameters and running the process, thereby disregarding the need to intimately know the underlying implementation.

8 Conclusion

The goal of this thesis was to build an automated mechanism for analysing university courses and their relations. As a first step we establish a healthy set of cleansed textual data using ELT process as a cornerstone. The designed ELT process collects raw textual data and documents from various sources made available for courses that are taught in the University of Tartu. Once the raw data has been gathered, we continue by extracting the textual content from all the downloaded files (e.g *PDF*, *TEX*, *DOCX*), thereby constructing an initial data warehouse. As a next step, a string of transformations (e.g text tokenization) and data cleaning operations are applied to the raw data. This includes removing common and un-informative words (i.e stop words) and word lemmatization in order to count the same word in its different inflected forms as one. As a result, a bag-of-words model is fitted for each course and for each individual course material - a suitable groundwork for further analysis and visualization.

After the proper construction of a data warehouse, we apply topic modelling, more specifically, attempt to resolve LDA model at two different levels of granularity - course and course material level. The idea is that similar courses or materials should be comprised of the same topics. Since each topic consists of words that describe that topic, we also get a hint as to the nature of the relation between two entities. The results of the topic modelling are then visualized via different channels (e.g heatmap, topic tables).

An example test run was performed over the academic years of 2015/2016 and 2016/2017. The results showed that the same course given in different semesters and complementary courses mainly contained the same topic, thus validating the process and opening a door for more interesting and exclusive relations to be mined and explored from the results. The established framework, based on the publicly available data for courses in University of Tartu, shows great promise and still has a lot of room for improvement in all aspects. While integrating new Courses web pages is fairly simple as they become available for other faculties and institutes, the long term perspective is to make the data scraping more adaptable and modular, so that similar analysis could be performed for different universities.

References

[1] Adobe Inside PDF blog. http://blogs.adobe.com/insidepdf/2008/07/text_content_in_pdf_files.html. Accessed: 2017.05.15.

[2] lda: Topic modeling with latent Dirichlet Allocation. https://pypi.python.org/pypi/lda. Accessed: 2017.05.15.

[3] Natural Language Toolkit. http://www.nltk.org/. Accessed: 2017.05.15.

[4] Office Open XML overview. http://www.ecma-international.org/news/TC45_current_work/OpenXML%20White%20Paper.pdf. Accessed: 2017.05.15.

[5] Ranks nl stopword lists. http://www.ranks.nl/stopwords. Accessed: 2017.05.15.

[6] University of Tartu, Institute of Computer Science courses. https://courses.cs.ut.ee. Accessed: 2017.05.15.

[7] University of Tartu, Institute of Mathematics and Statistics courses. http://courses.ms.ut.ee. Accessed: 2017.05.15.

[8] University of Tartu, Moodle environment. https://moodle.ut.ee. Accessed: 2017.05.15.

[9] University of Tartu, Study Information System. https://www.is.ut.ee/pls/ois/!tere.tulemast. Accessed: 2017.05.15.

[10] Visualization of course analysis. http://atiasa.cs.ut.ee/. Accessed: 2017.05.15.

[11] Vimala Balakrishnan and Ethel Lloyd-Yemoh. Stemming and lemmatization: A comparison of retrieval performances. In *Lecture Notes on Software Engineering vol. 2, no. 3*, pages 262–267, 2014.

[12] David M. Blei, Andrew Y. Ng, and Michael I. Jordan. Latent dirichlet allocation. *J. Mach. Learn. Res.*, 3:993–1022, March 2003.

[13] William B. Cavnar and John M. Trenkle. N-gram-based text categorization. In *In Proceedings of SDAIR-94, 3rd Annual Symposium on Document Analysis and Information Retrieval*, pages 161–175, 1994.

[14] Hui Chao and Jian Fan. *Layout and Content Extraction for PDF Documents*, pages 213–224. Springer Berlin Heidelberg, Berlin, Heidelberg, 2004.

[15] Dana Dannélls. Automatic acronym recognition. In *Proceedings of the Eleventh Conference of the European Chapter of the Association for Computational Linguistics: Posters &#38; Demonstrations*, EACL '06, pages 167–170, Stroudsburg, PA, USA, 2006. Association for Computational Linguistics.

[16] Wim De Smet and Marie-Francine Moens. Cross-language linking of news stories on the web using interlingual topic modelling. In *Proceedings of the 2Nd ACM Workshop on Social Web Search and Mining*, SWSM '09, pages 57–64, New York, NY, USA, 2009. ACM.

[17] Rebecca Dridan and Stephan Oepen. Tokenization: Returning to a long solved problem a survey, contrastive experiment, recommendations, and toolkit. In *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics: Short Papers - Volume 2*, ACL '12, pages 378–382, Stroudsburg, PA, USA, 2012. Association for Computational Linguistics.

[18] Ted Dunning. Statistical identification of language. Technical report, 1994.

[19] Thomas L Griffiths and Mark Steyvers. Finding scientific topics. *Proceedings of the National academy of Sciences*, 101(suppl 1):5228–5235, 2004.

[20] Mika Klemettinen Helena Ahonen-Myka, Oskari Heinonen and A. Inkeri Verkamo. Finding co-occurring text phrases by combining sequence and frequent set discovery. In *Proceedings of 16th International Joint Conference on Artificial Intelligence IJCAI-99 Workshop on Text Mining: Foundations, Techniques and Applications*, pages 1–9, 1999.

[21] Do-kyum Kim, Marti Motoyama, Geoffrey M. Voelker, and Lawrence K. Saul. Topic modeling of freelance job postings to monitor web service abuse. In *Proceedings of the 4th ACM Workshop on Security and Artificial Intelligence*, AISec '11, pages 11–20, New York, NY, USA, 2011. ACM.

[22] Tibor Kiss and Jan Strunk. Unsupervised multilingual sentence boundary detection. In *Computational Linguistics 32.4*, pages 485–525, 2006.

[23] Raigo Kodasmaa. Natural language processing in information retrieval, 2011. Bachelor's Thesis, University of Tartu, Estonia.

[24] Ralf Krestel, Peter Fankhauser, and Wolfgang Nejdl. Latent dirichlet allocation for tag recommendation. In *Proceedings of the Third ACM Conference on Recommender Systems*, RecSys '09, pages 61–68, New York, NY, USA, 2009. ACM.

[25] Siim Orasmaa, Timo Petmanson, Alexander Tkachenko, Sven Laur, and Heiki-Jaan Kaalep. Estnltk - nlp toolkit for estonian. In Nicoletta Calzolari (Conference

Chair), Khalid Choukri, Thierry Declerck, Marko Grobelnik, Bente Maegaard, Joseph Mariani, Asuncion Moreno, Jan Odijk, and Stelios Piperidis, editors, *Proceedings of the Tenth International Conference on Language Resources and Evaluation (LREC 2016)*, Paris, France, may 2016. European Language Resources Association (ELRA).

[26] A.S. Schwartz and M.A. Hearst. A simple algorithm for identifying abbreviation definitions in biomedical text. In *In Proceedings of Pacific Symposium on Biocomputing*, volume 4, pages 451–462, November 2003.

[27] Nakatani Shuyo. Language detection library for java. `http://code.google.com/p/language-detection/`, 2010.

[28] Panos Vassiliadis. A survey of extract-transform-load technology. In *Integrations of Data Warehousing, Data Mining and Database Technologies - Innovative Approaches.*, pages 171–199. 2011.

[29] Y. Zhang, F. Mueller, X. Cui, and T. Potok. Gpu-accelerated text mining. In *Workshop on Exploiting Parallelism using GPUs and other Hardware-Assisted Methods*, 2009.

Appendix

I Glossary

Bag-of-words model - an unordered set of a word-count pairs that represent a document or sentence.

Document-term matrix (DTM) - a matrix representation of all term occurrences in given documents, where we have documents on one axis, terms in other and each value represents how many times did a term occur in a document.

ELT (Extract, Load, Transform) process - similar to ETL process, however, all the transformations take place after loading the extracted data into the target data warehouse.

ETL (Extract, Transform, Load) process - the process of extracting data from various heterogeneous sources, performing transformations on the extracted data and finally loading the results to a target data warehouse.

Frequent co-occurring words - a sequence of words that occur often in a document(s). Classifying co-occurring words as frequent depends highly on the set definition (e.g has to appear in at least a specified number of different documents).

Inflected word - a modified version of a word, used to articulate a different tense or aspect (e.g *sleeping* is an inflected form of the word *sleep*).

Lemmatization - the process of extracting the lemma of an inflected word. E.g from word *running* we would get the root lemma *run*.

Ligature - a sequence of two or more letters that form a single glyph. For example the letters *f* and *i* are commonly appended together to form a single glyph *fi*.

N-gram - a sequence of N words in a given sentence or text.

Part-of-speech (POS) tagging - the process of resolving a part of speech (noun, verb etc.) for a particular word, based on the context (i.e surrounding words) and the definition of the word itself.

Stemming - normalization of an inflected word by removing its suffix (e.g *cars* to *car*).

Stop words - common words (e.g *this, that, and*) in a language that are usually filtered out before any further processing.

Tokenization - the process of splitting raw text into sentences and words (i.e tokens) by considering sentence boundaries, symbols, contractions etc.

Topic modelling - a method for finding a set of topics that describe given documents and associating each document with a limited subset of the found topics.

Web scraping - an automated process, which extracts data from a websites by crawling through its content and extracting text and following links, as per its configuration.

II Framework Implementation Overview and Setup

The entire implementation is split into two parts. The first part is a pure Python based implementation that covers all the phases starting from data extraction to analysis. The second part focuses on the visualization of the results, essentially an entire implementation built around the database file produced in first step. The Python implementation for the main logic behind the built framework can be obtained from following repository:

```
https://github.com/ragnarvent/courses_analysis
```

The project requires Python 2.7 and a range of different dependencies in order to run the process. All the necessary dependencies are specified in the *requirements.txt* file, located in the project root folder. A simple *Makefile* can be used to perform individual process steps, clean current results or install the project library dependencies. The following *make* commands are supported:

- **init** - uses pip to install dependencies specified in *requirements.txt* file.
- **clean-pyc** - deletes all compiled python bytecode files from project folder.
- **clean-analysis** - deletes previous results, e.g gathered faculty member names, analysis results. Rebuilds the database or creates it if missing all-together. Note that in order to fully delete all previous results (e.g raw files and the corresponding database entries), an additional parameter *fc* has to be provided with the value of `1`.
- **clean-stale** - deletes all files and database entries not related to target semesters. The target semesters can be specified via *SEMESTERS* parameter.
- **scrape-courses** - performs data scraping over Courses web page. The command requires the specification of a *SEMESTERS* parameter. The value is expected to be a comma delimited list of semesters to scrape. A single semester can be given as a concatenation of year and the first letter of the semester by season (i.e *F* for *Fall* and *S* for *Spring*). For example the following command can be used to scrape data based on 2015/Spring and 2015/Fall semesters: *make scrape-courses SEMESTERS=2015S,2015F*.
- **scrape-moodle** - scrape data from Moodle, this is a complementary command that should be run only after scraping Courses web page. As an additional condition, the target semesters for Courses web crawling had to contain the currently ongoing semester.

- **scrape-teachers** - scrape the names of the faculty members and place them in a file in JSON format. The list of names are used as a filter during data cleaning phase.
- **extract-sis** - load data from a *CSV* file that was extracted from Study Information System database. The command is complementary and requires the specification of *SEMESTERS* parameter.
- **extract** - extract textual content from downloaded course materials.
- **tokenize** - perform data cleaning, this includes tokenization, acronym extraction and finding frequently co-occurring words.
- **analyse** - perform topic modelling at a course and course material level, this includes resolving topic titles.

A shell script *analyse_courses.sh* can be used to perform all the steps in a sequential order, requiring only the specification of initial parameters.

```
usage: analyse_courses [[[-s semesters ] [-sis] [-tdir] [-t] [-m] [
    -fc]] | [-h]]

        -s, --semesters semesters : comma separated list of
            semesters to be scraped from Courses web page. E.g 2015F
            ,2016S

        -m, --moodle : scrape data from Moodle

        -t, --teachers : scrape teacher names to exclude them from
            analysis

        -sis, --studyinfosystem : extract study information system
            data from .csv file

        -tdir, --targetdirectory : directory where the resulting DB
            file will be copied

        -fc, --fullclean : remove all existing data including
            corresponding DB entries. Re-download everything instead
            of just updating the missing parts

        -h, --help : display help
```

For example the following command can be used to run the process, targeting 2015/2016 and 2016/2017 academic years:

bash ./analyse_courses.sh -s 2015F,2016S,2016F,2017S -m -sis -t -fc

The visualization part, however, requires the set up of Apache web server with a fitting PHP version. The base project for visualization part can be obtained from:

```
https://github.com/ragnarvent/courses_analysis_web
```

The project uses PHP for backend integration, HTML, Javascript (for various charts and graphs) and Bootstrap for frontend. The necessary PHP dependencies (e.g Silex framework) are specified in *composer.json* file and can be acquired via Composer dependency manager.

Concerning the integration between part one, where we perform the analysis, and part two, where we visualize the results, the only necessary action is to copy the database produced in part one to the *db* folder in the project of part two. This small step can be automized by providing the *tdir* parameter, when running *analyse_courses.sh* script. All in all, if the Apache server is already running, then the only thing needed to update the results is the successful execution of *analyise_courses.sh* script.

Additionally, it is possible to add new Courses or Moodle web pages as they become available for different faculties by adding the web scraping start points in the configuration file, i.e *utils/config.cfg*. A similar URL pattern (e.g language is explicitly specified in URL) should be used, when adding new starting points.

III Licence

Non-exclusive licence to reproduce thesis and make thesis public

I, Ragnar Vent,

1. herewith grant the University of Tartu a free permit (non-exclusive licence) to:

1.1 reproduce, for the purpose of preservation and making available to the public, including for addition to the DSpace digital archives until expiry of the term of validity of the copyright, and

1.2 make available to the public via the web environment of the University of Tartu, including via the DSpace digital archives until expiry of the term of validity of the copyright,

of my thesis
A Framework for Analysing Topics in University Courses
supervised by Siim Karus

2. I am aware of the fact that the author retains these rights.

3. I certify that granting the non-exclusive licence does not infringe the intellectual property rights or rights arising from the Personal Data Protection Act.

Tartu, 18.05.2017