Master Thesis Recommendation Tool

Magnus Rushfeldt **Martin Stiles**
M.Sc. Computer science, year 4
magnurr@stud.ntnu.no martsst@stud.ntnu.no

Abstract

Students in the fourth year of Computer Science are faced with a messy and non-user-friendly interface for selecting a master thesis topic, which results in the process being inefficient in an already stressful time for many students. The aim for this project is to provide students with a tool to receive recommendations for thesis topics based on interests and efficiently gather information about the topics. We thus present a narrative-driven recommendation system, along with a simple summarization tool. We show how we used TF-IDF vectorization and cosine similarity to create a narrative-driven recommendation system, which generates thesis topic recommendations based on the users' interests. We also show how we provide a summarization tool, generating keywords and a summary for the thesis topics by finding the words with the highest TF-IDF weights and sentences with the highest sum of weights. We used data from a survey to measure the average F-score of the recommendation tool to be 0.56. We did not have a way of measuring the performance of the summarization tool, but by manual inspection we can conclude that it performs worse than we initially hoped.

1 Introduction

The process of selecting a master thesis is time consuming and stressful for many students. In the span of a month, students have to curate a ranked list of five choices of Master thesis topics to submit to the Department of Computer Science (IDI). To do this, the students have to use the department's website listing available master thesis topics[1]. This website will be referred to as the "Master thesis website", while the thesis topics will be referred to as "theses" from here on. The Master thesis website presents seemingly over one thousand possible theses (of which many are duplicate theses listed in several places). Every student has to read, understand and inevitably prioritize enough of these theses to be able to curate their top five choices. To make matters worse, there is no search function on the website, thesis descriptions are often tedious and hard to read, and theses can only be filtered according to the corresponding study specialization and supervisor of each thesis.

Our goal with this project is to give students aid in their pursue of a master thesis by providing them with a recommendation tool, consisting of a recommender and a text summarizer. The goal of these tools are to provide students with recommendations for thesis topics based on listed interests, as well as to give shorter, more concise information about each thesis.

The classical approach to recommendation systems uses content-based filtering, collaborative filtering, or a combination of both Karlgren (1990). However, due to no pre-existing user data, we opted for a narrative-driven approach, as. described in Lika et al. (2014). We use Term Frequency-Inverse Document Frequency (TF-IDF) vectorization, as originally used for document retrieval in Jones (1972), in combination with with a vector similarity measurement. By taking a user's interests as input, we can compute which thesis topics that are closest, and thus most relevant, to the interests. This is essentially an information retrieval (IR) system, but in the context of how we want to use it, we will refer to it as a recommendation tool. At first, Named Entity Recognition (NER) was perceived as a good approach to

[1] https://www.idi.ntnu.no/education/fordypningsprosjekt.php

extracting features for similarity measuring, due to its use in Karatay and Karagoz (2015). However, due to difficulties in recognizing meaningful entities, we decided to switch to TF-IDF feature extraction.

Meanwhile, the summarization tool is based on a combination of keyword and sentence summarization. This approach was used in Opidi (2019). The purpose of using a combination of the two, is that a user can quickly gauge whether a thesis is relevant or not based on the keywords, as well as get a better understanding of the task by reading the summarization. Both of these rely on TF-IDF vectorization, with slightly different approaches than used in the recommendation tool.

A significant challenge with creating the system, was to get relatively uniform data based on unstructured textual data from the thesis description. While the general HTML thesis containers of the Master thesis website followed the same pattern, the title and description of theses seemed to follow many different more-or-less structured patterns. In addition, the language used in the thesis descriptions and titles was not specified, so there was no simple way of checking whether a thesis was written in English or Norwegian, or sometimes even a combination of both. All these data quality challenges meant that significant effort has been spent on extracting and pre-processing the data to make the final thesis data format as uniform as possible.

Throughout this report we will present the logic behind the recommendation and summarization methods implemented as well as our results. We will cover the entire process from data extraction to model evaluation based on user data gathered from a survey. The report is structured as following: In section 2 the methodologies and tools used in the recommendation tool is presented, while section 3 discusses other approaches to recommendation and summarization. Then, section 4 delves into how the Python-based system is designed and works, before section 5 discusses the labeled thesis data set used and how the performance of the recommendation tool was evaluated. Finally, section 6 analyses the result of the recommendation tool run with different parameters, and section 7 discusses the utility of the recommendation tool and the contributions we have made.

2 Background

In this section we will provide the background information required to be able to grasp the entirety of the project. This includes technologies used in the code as well as concepts related to the model.

2.1 Selenium and Beautiful Soup (BS4)

Selenium and BS4 are two Python libraries commonly used in combination to extract data from websites. Selenium provides a webdriver that allows you to emulate a browser and retrieve HTML code. BS4 provides tools to easily extract data from HTML files, using what is known as a “soup” construct.

2.2 NLTK

NLTK is a popular python library providing quality-of-life methods for text manipulation. In particular, NLTK provides methods to tokenize words, tokenize sentences, remove stop words and stem word. Tokenization splits paragraphs, sentences and phrases into single-sentence or single-word tokens to isolate each component of a larger text structure (i.e. word tokenization of “thesis recommendation” yields “thesis", "recommendation”). Stop words are frequent words in a language that carry little semantic value (e.g. “the” and “over” in English), and these are often removed to increase the ratio of words with more semantic meaning in a text (mostly nouns). Meanwhile, stemming is the action of removing suffixes from words so that two words with different suffixes, but identical word stems, can be identified as the same base form of the word (i.e. stemming of "recommendation" and "recommender" yields the same word stem “recommend”).

2.3 TextBlob

TextBlob is a multi-purpose python library that in particular provides a method for language detection. The language detection method has limits however, as it sends https requests to TextBlob’s API. If a user sends too many requests in a short time period, their IP address will get blacklisted by their server.

2.4 TF-IDF vectorization

TF-IDF is a combination of the terms TF and IDF, meaning "term frequency" and "inverse document frequency" respectively. TF-IDF is often used as a measure for the importance of a term in a given document, where frequent occurrences in the document is rewarded and frequent occurrences in other documents is punished. The IDF part is important to punish common terms across the documents, such as "AI" or "machine learning" in a corpora about machine learning, as we do not want them to be labeled as important terms for the document. The mathematical combination of TF and IDF is described in the following equation:

$$tf\_idf(t, d, D) = tf(t, d) * idf(t, D) = log(1 + f(t, d)) * log(\frac{N}{f(t, D)})$$

Here $f(t, d)$ is the frequency of the term t in document d, $f(t, D)$ is the combined frequency of t in all documents in collection D, and N is the total number of documents in D.

2.5 Cosine similarity

Cosine similarity is a way to measure the distance between two vectors. One can utilize this method to find similarity between textual objects by representing the texts as a vectors, e.g. bag-of-words, and then assume that the cosine similarity between the vectors corresponds to the similarity between the textual objects. Cosine similarity is defined by the following equation:

$$cosine\_similarity(A, B) = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2}\sqrt{\sum_{i=1}^{n} B_i^2}}$$

Here A and B are the two input vectors, A_i and B_i are the components of A and B respectively and n is the number of elements in each vector.

2.6 scikit-learn

scikit-learn is a popular python library for data analysis. In particular, scikit-learn provides simple and efficient ways of performing TF-IDF vectorization and computing cosine similarity.

2.7 Precision, Recall and F-score

Precision, recall and F-score are three common measurements for evaluation of information retrieval (IR) systems. All three measurements has a maximum value of 1 and a minimum value of 0, where closer to 1 is good. Given a query, a set of documents labeled as relevant or irrelevant for the query, and the returned documents from the IR system: precision describes how many of the returned documents are relevant and recall describes how many of the relevant documents are actually returned. This is illustrated in Figure 1. F-score is simply the harmonic mean of precision and recall, described by the following equation:

$$F_1 = 2 * \frac{precision * recall}{precision + recall}$$

3 Related Work

In this project we were facing the cold start problem, as described in Lika et al. (2014). In short, the cold start problem is a common problem in recommendation system design where the developers has no user history or labeled data to support the recommendations of the system. In such cases, Lika et al. proposes a model where widely known classification algorithms in combination with similarity techniques are used. In our case, we did not want to store user data and instead base recommendations solely on user input.

Thus we wanted to have a nearest-neighbor approach by finding similarity between the user input and the thesis descriptions. Such an approach is described in Deng (2019), where he mentions the possibility of using nearest-neighbor-based methods such as cosine similarity. He also points out that this approach,

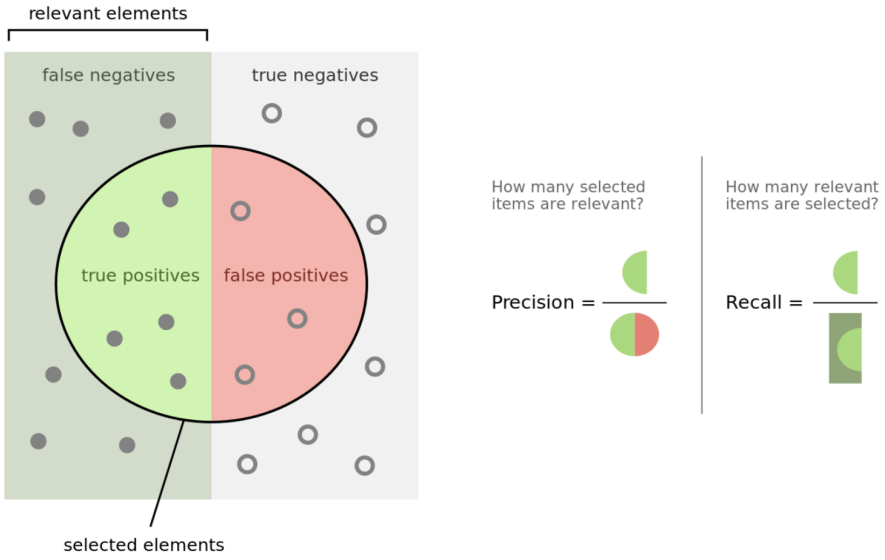

Figure 1: Illustration of precision and recall, adapted from (Wikipedia, Walber 2014)

in combination with other approaches, is adapted by companies like Amazon (2003), YouTube (2010) and Linkedin (2014).

The nearest-neighbor approach is widely discussed in Bogers and Koolen (2017), although they refer to it as narrative-driven recommendation. Bogers and Koolen suggests that recommendation systems should not be based only on users' past transactions, but also a narrative description of their current interests. The part of this approach regarding the use of users' interests to generate recommendations aligns with our plans for the recommender tool.

4 Architecture

In this section we will present the architecture used in the recommendation tool. The full pipeline, from the scraping of the Master thesis website to the thesis summaries and recommendations, can be seen in Figure 4 of Appendix A. Meanwhile, Figure 2 shows the architecture of the Python program, and the files that are written and read by the different parts of the program. This Python program constitutes the full software of the recommendation tool. Each part of the pipeline and architecture will be explained in the following subsections.

4.1 Data set and data retrieval

In order to have data for the system to work on, the Master thesis website must first be scraped and converted to a JSON object format using Selenium and BS4 respectively. This takes place in `scraper.py`, as seen in Figure 2. Figure 4 shows that the scraping yields the raw "Thesis HTML elements". Using BS4, a BeautifulSoup soup object of the HTML is created, the "Thesis BS4 soup", from which all the interesting HTML contents are extracted. These contents are then formatted as a JSON object containing all the theses formatted as thesis objects, as seen in the "THESIS OBJECT" in Figure 4. This JSON object is then written to the file `raw_data.json` by `scraper.py`, which is shown in Figure 2. As seen in Figure 4, each thesis object contained in `raw_data.json` contains eight data fields. The details of each of these fields can be found in Table 1.

In addition to making `raw_data.json`, we also feed this file into `make_data_dict.py` which writes a new version of the `raw_data.json` file, called `data_dict.json`. Whereas `raw_data.json` is one large object containing all the thesis objects, each with all of the fields listed in Table 1, `data_dict.json` creates a dictionary of the thesis key-value pairs. The key of each thesis in this dictionary is the thesis ID and the value contains only the thesis and description fields, the latter of which is sentence-tokenized. This dictionary will become useful later, both for the summarization and

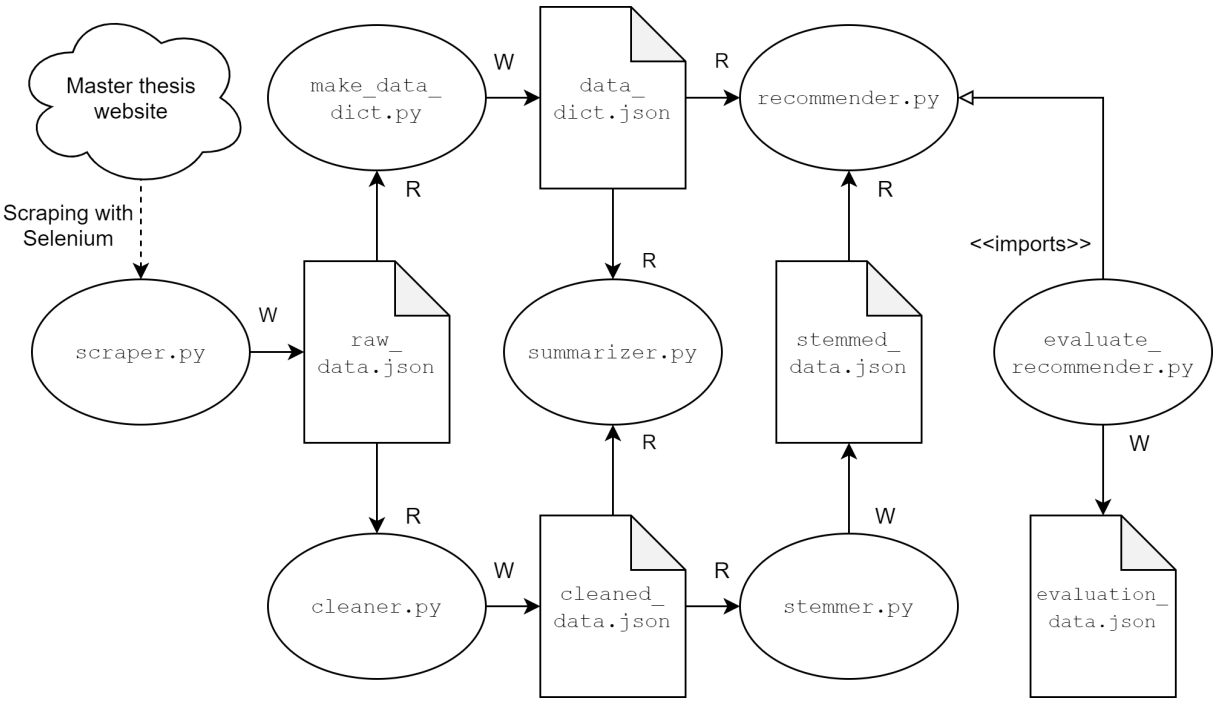

Figure 2: Diagram showing the Python code architecture of the recommendation tool. The direction of arrows show the logical flow of data and process control in the system. Legend:
W = the Python program writes this JSON file, R = the JSON file is read by this Python program.

recommendation tool, to easily find the original text contents of a thesis given a thesis ID.

The scraper retrieved a total of 1285 theses (at the time of scraping). However, many of these theses are duplicates of each other, as they are associated with several specializations and thus listed several times, once for each corresponding specialization. However, since each thesis has the same ID every time its listed, we are able filter out all duplicates and end up with only 375 unique theses (at the time of scraping). Out of these, 348 are written in English, while 27 are written partially or fully in Norwegian.

We planned to use the fields supervisor, specialization, assigned status and number of students as filtering options in a potential graphical interface where the user could browse theses and their summaries. However, we chose not to focus on this as it was outside the topic of the project and course. The fields are still kept for posterity, if we decide to implement an application for the system in the future.

4.2 Data cleaning

After the unique thesis objects have been collected, the thesis and description data must be cleaned, tokenized and stemmed before they can be used to generate TF-IDF vectors in the recommendation and summarization tool. In the pipeline of Figure 4, the "Cleaned titles and descriptions" of all thesis objects are produced after cleaning, which yields the "Tokens" of both text fields after tokenization. These two steps are done in `cleaner.py` in Figure 2, and the tokens produced by tokenization are written to the file `cleaned_data.json`. The cleaning step involves removing some special encoding characters, splitting the descriptions into sentences and removing non-letter characters. This removes most of the clutter that appears in some descriptions, and results in a more uniform description format. Then, the tokenization step splits the sentences of the title and description of a thesis into tokens, lower-cases each token and removes all stop words from the lists of tokens. As there are theses in both English and Norwegian, and stop words are different from language to language, stop word removal is done in both English and Norwegian on each thesis. The final cleaned data in `cleaned_data.json` thus contains letter-based tokens in lower case, where most tokens carry proper semantic meaning.

After cleaning the data, the data is both converted to TF-IDF vectors and fed into the summarization tool, as will be discussed in subsection 4.3, and fed into the stemmer. The stemming is performed by

Field name	Type	Possible values	Description
Thesis ID	Integer	1..9999	A unique number identifying the thesis. Derived from the query parameter of the thesis URL.
Assigned status	String	{“Tildelt”, “Valgbart”}	Whether the thesis have been assigned to a student/group of students or not.
Number of students	String	{“En student”, “Gruppe / En eller flere studenter”}	Whether the thesis is meant to be completed by one student alone or a group of students.
Supervisor	String	Any string	The name of the supervisor that is supervising this thesis.
URL	String	`https://www.idi.ntnu.no/education/oppgaveforslag.php?oid={ThesisID}`	The unique URL to the thesis page on the Master thesis website.
Title	String	Any string	The name of the thesis. Will be used for TF-IDF vectorization in the recommendation and summarization tools.
Description	String	Any string	A shorter or longer text describing the problem, background, methods and/or project outline of the thesis. Will be used for TF-IDF vectorization in the recommendation and summarization tools, and for extracting a sentence summary in the summarization tool.
Specialization	String	Any of the specializations listed on the Master thesis website	The name of one of the study specialization that the thesis is listed under. NOTE: some theses are listed under several specializations, even though only one is included in the thesis object.

Table 1: All the fields contained in a thesis object stored in `raw_data.json`.

`stemmer.py` as seen in Figure 2, which yields `stemmed_data.json`. This is seen as "Stemmed tokens" in Figure 4. The stemmed tokens are used to improve the cosine similarity matching in the recommendation tool, which will be discussed in subsection 4.4. Because stemming is a language-dependent operation, due to different suffixes in different languages, the stemming can only be performed properly after the language of the given text to be stemmed is determined. This is where TextBlob is utilized to detect the language of the given thesis. This detection returns a language tag which is added to each thesis object in `stemmed_data.json`. However, due to the TextBlob API blocking high rates of request, our code adds sleep timer to slow the rate of requests down to one per second. This allows the language detection to be run on each of the 375 thesis descriptions, at the cost of a runtime penalty (at least 375 seconds of runtime to produce `stemmed_data.json`). Once the tokens have been stemmed, they are ready to be used in the recommendation pipeline.

4.3 Summarization

The unstemmed tokens from `cleaned_data.json`, along with the full title and sentences of the description of each thesis in `data_dict.json`, are used to provide the data for the summarization

tool. The summarization tool will create a summary of each thesis by returning the five highest-weighted keywords and the two highest-ranked sentences in each of them. This can be seen in Figure 2, were both files are sent to `summarizer.py`, as well as the "Summarization" part of Figure 4. For the summarization process, the tokens in `cleaned_data.json` are first converted to vectors using TF-IDF vectorization. This means each sentence, or list of tokens, in each thesis (either the title or a sentence in the description) is represented as a vector. The terms that are frequent in the given sentence, but not very frequent in other sentences of the thesis, have the highest value. This summarizer works both for English and Norwegian theses at the same time, because it weighs the terms of each thesis independently of other theses.

By finding the two TF-IDF vectors with the highest weighted sum in each thesis description, the summarizer ideally finds the two sentences in the description with the most important terms of the thesis. The summarizer does this by extracting the indices of those vectors in the thesis and mapping them directly to the indices of the original sentences of the thesis description in `data_dict.json`, as both structures keep the same internal order of sentences. This can be seen as the output "Sentences" of the "TF-IDF vectors (not stemmed)" and the "Description" field in "Summarization". The sentences are returned in the same order as they originally appeared in in the thesis description, to avoid messing up any chronological order of what is mentioned in the extracted sentences.

The summarizer then finds the most important keywords of the thesis by extracting the five terms in each thesis with the highest TF-IDF weight, and returning them as a list of keywords. This is seen as the output "Keywords" of "TF-IDF vectors (not stemmed)" in Figure 4. Together, the keywords and sentences attempts to provide the user with some of the most important content of the thesis.

4.4 Recommendation

The stemmed tokens are used in combination with the original thesis objects and interests terms from the user, to do recommendation of theses. This can be seen in "Recommendation" of Figure 4, and is performed by `recommender.py` with the use of `stemmed_data.json` and `data_dict.json` in Figure 2. The recommender will try to find the n theses (n is a parameter that will be tuned in subsection 5.1) that best match the user-specified interests, and return them as recommendations, ordered according to which thesis best match the user interests.

In a similar fashion as the summarizer, the recommender also vectorizes the stemmed tokens with TF-IDF. However, as proper matching of interest terms and thesis terms is highly dependent upon language (and whether or not terms are the same in both languages), the vectorization has to be performed exclusively with theses in one language at a time, either the Norwegian or English. Therefore, a language parameter must be passed to the recommender, which then uses the language tags generated by the stemmer to keep only the theses in the specified language, while filtering out the rest.

After generating TF-IDF vectors, the recommender then accepts user interests specified as a list of terms, which is also vectorized. Examples on how such a query is formatted can be seen in Table 2. Following this, the TF-IDF vectors of both the thesis tokens and user interests are compared for cosine similarity. This can be seen in "Cosine similarity calculation" in Figure 4. When the similarity between each thesis vector and the query vector has been calculated, the weighted sum of all vectors in each thesis is computed to provide each thesis with a total similarity score. The IDs of the n theses with the highest weighted sums are then extracted. To find the original thesis information for each thesis, the thesis IDs are used to index the thesis objects in `data_dict.json`, and extract only those theses to return to the user. The user then receives a list of the n thesis objects that should be most relevant to what the user wanted, ordered according to match on the query terms. This is seen as the output "Recommended theses" in Figure 4.

4.5 Evaluation

In order to evaluate whether the recommendation tool actually recommends relevant theses, an additional step is added in the pipeline to perform evaluation. The evaluation is performed by `evaluate_recommender.py` in Figure 2, which imports the code of `recommender.py` to perform the actual

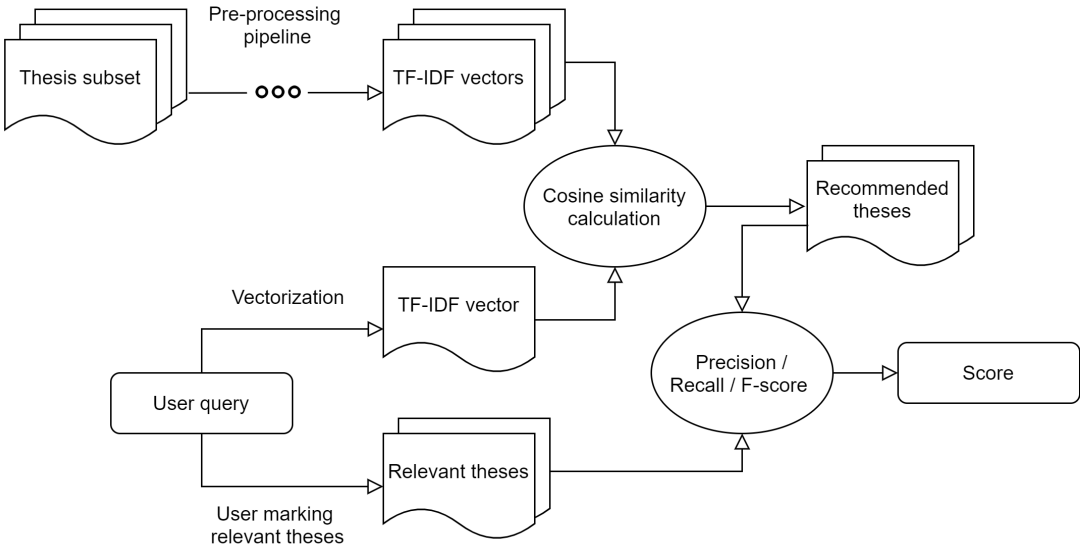

Figure 3: Diagram showing the recommendation tool evaluation pipeline. This figure is based on the part of the pipeline seen in the "Recommendation" box in Figure 4, with the added step of calculating the precision, recall and F-score of each result.

evaluation and produces the results in `evaluation_data.json`. Figure 3 shows the recommendation part of pipeline with the added evaluation step. Precision, recall and F-score is used for evaluating the performance. To be able to calculate these measures for each query, all theses that are included in the similarity calculation must be marked as either relevant or not relevant to the given query by the user. After selecting relevant theses, and producing the recommended thesis in the recommendation pipeline, the evaluation is performed. By comparing the number of relevant theses found or not found among the recommended theses, as well as determining how many of the theses recommended that are actually relevant, the precision, recall and F-score of each query with given recommender parameters can be calculated. This is seen as the "Precision / Recall / F-score" calculation and the output "Score" in Figure 3. section 5 details how the evaluation was carried out for this project.

5 Experiments and Results

To evaluate the performance of the recommendation tool we wanted to use precision, recall and F-score, as described in subsection 2.7 and subsection 4.5. However, to compute these evaluation scores one needs labeled testing data, which we originally had none of. We thus orchestrated a survey for our friends that also studies computer science, where we asked each participant to send us their interests regarding the theses, along with the theses they would expect to be recommended out of a subset of theses. The subset consisted of forty theses: ten randomly chosen theses from each of the four main specializations. This was done to simplify the labeling process, as it would otherwise require too much time. We received six answers from the survey and they represent typical user inputs and the theses expected to be recommended for each input. This resulted in us having labeled evaluation data that wasn't as biased as it would have been if we were to label it by ourselves.

Due to the time consuming process of manually labeling data, we decided to focus on evaluation of the recommendation tool, as we claim that it is the most important part of our project. Consequently, we have no valid way of measuring the performance of the summarization tool. We will however present some examples of summaries and keywords generated.

5.1 Experimental Setup

The code for this experiment can be found in the GitHub repository of this project[2]. The labeled data set contains a total of six user queries for the recommendation tool. Each query can be seen in Table 2. The data set is needed to run the evaluation process and can be found in `/src/recommender/evaluation_data.json`. This means that after cloning the repository, the data set is ready for evaluation, and it is only necessary to run the evaluation code in `/src/recommender/evaluate_recommender.py`. Inside `evaluate_recommender.py` there is a variable n which describes the maximum number of theses the recommender will return. n is the only parameter that can be tweaked, and we can observe how it affects the evaluation scores.

Query 1	"AI computer vision deep learning transformers autonomous vehicles"
Query 2	"frontend games agile web strategy"
Query 3	"user-centered design agile development security testing software architecture"
Query 4	"algorithms gpgpu graphics low-level non-hardware"
Query 5	"music art nature history fashion"
Query 6	"database cloud big data OLAP mining"

Table 2: Queries gathered from the survey and used in the evaluation

5.1.1 Recommendation results

For the sake of tidiness we have decided to include the evaluation scores for only query 1, 3, 4 and 5, as those scores are the ones we found most interesting. Query 4 and 5 are gives the worst scores out of all while query 3 gives the best. We also show the average of the evaluation score of all six queries. The results are presented in Table 3.

	Query 1	Query 3	Query 4	Query 5	**Average**
n = 5	P = 0.80 R = 0.29 F = 0.42	P = 1.0 R = 0.45 F = 0.63	P = 0.4 R = 0.2 F = 0.27	P = 0.4 R = 0.5 F = 0.44	P = 0.73 R = 0.39 F = 0.49
n = 10	P = 0.8 R = 0.57 F = 0.67	P = 0.9 R = 0.82 F = 0.86	P = 0.6 R = 0.6 F = 0.6	P = 0.2 R = 0.5 F = 0.29	P = 0.63 R = 0.63 F = 0.62
n = 15	P = 0.67 R = 0.71 F = 0.69	P = 0.67 R = 0.91 F = 0.77	P = 0.4 R = 0.6 F = 0.48	P = 0.13 R = 0.5 F = 0.21	P = 0.48 R = 0.71 F = 0.56

Table 3: Precision, recall and F-score for query 1, 3, 4 and 5, as well as the total average. This is shown for three different values of *n*

5.1.2 Summarization results

As mentioned earlier, the performance of the summarization tool has not been evaluated in the same way as the recommendation tool as we simply have no data to evaluate it on. In Table 4 we present a good and a bad summary to provide examples of both cases. The associated keywords are considered poor in both cases.

6 Evaluation and Discussion

As mentioned in subsubsection 5.1.1 the best performing query is query 3, query 1 performs in the middle and the two worst performing queries are query 4 and 5. This makes sense if we look at the words in the three queries. Query 3 consists of a wide range of commonly used terms in Computer Science Master theses and every term are related to each other to some degree. Query 3 thus represents a quite average

[2]https://github.com/martinstiles/TDT4310-Master-Thesis-Recommendation-Tool

Title	Sentence summary	Keywords
AR Game to Motivation Socialisation and Physical Activity	This project is part of a larger project with the goal of commercialising a concept. The project will involve a study of existing theory, game concepts and technology, design and development of a game concept (both front-end and back-end) and an evaluation of the concent involving real users.	['project', 'developed', 'unity', 'game', 'end']
A RISC V Oberon compiler backend	Using a self-designed programming language, Oberon, and his own RISC processor implemented on a tiny FPGA, Wirth was able to create a system including a graphical user interface, networking, and a compiler running on a 25 MHz CPU in 1 MB of RAM. inf.ethz.ch/personal/wirth/CompilerConstruction/ inf.ethz.ch/personal/wirth/CompilerConstruction/ [4] David Patterson, Andrew Waterman, The RISC-V Reader: An Open Architecture Atlas, 2017, ISBN 9780999249116. riscvbook.com/ riscvbook.com/ [5] Peter de Watcher, Oberon RISC Emulator:.	['pp', 'construction', 'backend', 'system', 'systems']

Table 4: Two examples of output from the summarization tool. The sentence summary of the first example is considered good and the second example could be considered less so. Both keyword summaries are relatively poor.

query with respect to the terms it contains, and should therefore provide somewhat of a benchmark for how the system performs on most queries. It has perfect precision for $n = 5$, and higher F-score than all the other queries for each n. Note, however, that this particular combination of terms and returned theses could be lucky, and would not might be as good for other similar queries. Query 1 is similar to query 3 in that it includes common terms, and especially so because the term "AI" appears in large number of the subset theses. This means also that many theses were labeled as relevant, which explains why the recall is lower than for query 3. On the other hand, query 4 contains oddly specific and rather uncommon words and thus yields quite poor evaluation scores, especially for a low value of n. In such cases, implementing some sort of thesaurus could be helpful, considering TF-IDF only accounts for exact term matches. Expanding the queries with a thesaurus is a common method to boost the performance of IR systems, as stated in Imran and Sharan (2009), and is a suggestion for future implementations of the recommendation tool. Query 5 is, in contrast to query 3, a hard query for the recommender tool as it consists of terms that are quite unrelated to each other and that hardly appear in any of the theses. Thus, it makes sense that the precision is low for query 5. If we look at the average scores, we can observe that precision is quite high for low values of n, but to maximize the F-score it is seems to be better to use higher values of n.

We did not have any evaluation data for the summarization tool, but by inspecting the output from the code we can safely say that the tool is not performing optimally. The examples provided in Table 4 shows that the tool can perform well in some cases, but also very poorly in other cases. However, it must be mentioned that in most cases the generated summaries and keywords are not that great. This can be due to multiple reasons, many of which will be discussed in detail later in this section.

When we evaluate the recommendation tool we fit and transform the TF-IDF vectorizer on the subset of theses specified in the evaluation data. Thus, recommendations are somewhat different from what they would have been if we were to use the entire data set. This means that the evaluation scores are not 100% representative for the entire recommendation tool. We believe that this is the right way to do it for our situation. By using the entire data set we would get recommendations that were not labeled, and thus could not possibly be relevant. This would have obscured the evaluation scores. The best way to evaluate the tool would of course be to have labeled data for the entire data set, but this was deemed unachievable due to time constraints.

A problem we faced was the fact that the data was difficult to work with. The HTML layout of the different project descriptions varied heavily, and thus the retrieved data set from the scraper had many

flaws despite our efforts to generalise the extraction process. For instance, some descriptions contained HTML lists, sometimes with dot and sometimes without, which made separation into sentences very hard. This wasn't a big problem for the recommendation tool, as word tokenization worked regardless of format. It was however a massive problem for the summarization tool, as it returned the sentences with the highest weighted TF-IDF sum. But if the sentences was poorly structured in the first place, the output from the summarization tool could not possibly be any better. An example of how poorly extracted data leads to poor summaries can be seen in the second example of Table 4.

The sometimes poor keyword results could possibly have been improved a lot if we used a stricter stop word policy. NLTK stop words for English is simply a list of the most common words in English. However, there are still many words left in the text that have little semantic meaning and provide little meaningful discrimination between texts. To remove more unnecessary words, Chowdhury and McCabe suggests tagging Part-of-Speech (POS). POS-tagging finds word classes such as nouns, verbs and conjunctions and tags the words with the right class. In Chowdhury and McCabe (1998), it is hypothesised that only nouns, verbs and adverbs alone carry most of the semantic meaning in a text, and thus all other word classes can be removed while still retaining what the text is about. By performing POS-tagging and keeping only nouns, verbs and adverbs, we could possibly have avoided most of the poor keywords extracted in the summaries.

Another problem we faced was the fact that some theses are in English and some theses are in Norwegian. This means that we have to detect the language of the theses to perform operations like stemming. Theses being in different languages also means one cannot get recommendations for theses in Norwegian if the user input is in English. We solved this by simply adding an additional input parameter in the recommender that specifies the language of the theses you want to get recommended. There are even some occurrences of theses where the title is in Norwegian and the description is in English. This is the reason we detect language only based on the thesis description, because it is the largest body of text in a thesis and is thus has the highest impact on the similarity computation.

Initially, we wanted to use spaCy[3], a popular open source NLP tool, for NER. The entities could be used to improve the recommendation tool by increasing the weights of the tokens regarded as entities. This would be especially useful for the keyword generation, as we could return only the important entities instead of important tokens. One obvious problem we have with the keyword generation is that many general words has not been cleaned away, e.g. "get" and "end", often gets high TF-IDF score within a thesis, even though they are not good keywords. But after some experimentation with the spaCy NER, we observed that the spaCy NER yielded poor results. This was most likely due to the fact that the thesis descriptions were too short for it to properly train on, and thus were unable to extract the different entities in the text. We then decided to simply use TF-IDF on all the tokens. In future iterations of the project we would like to implement some sort of NER, as we still think it would increase the performance of the system, especially the keyword generation.

In future iterations we would have handled the language detection better. The detection worked very well, but it forced the code to run much slower than it otherwise would have. This is due to the fact that TextBlob would ban our IP address from their server if we sent too many requests in a short period of time, so a sleep timer had to be implemented. This meant that we could not run the cleaner code very often, because detecting the language of every master thesis took about 6 minutes. This in turn made us less agile, as it was hard to make changes without making a mess in the pipeline. One thing we could have done is to make a separate file only for the mapping between thesis ID and the detected language, so we would not have to run the language detection every time.

7 Conclusions and Future Work

In this report we have presented how we have extracted and cleaned data from the Master thesis website. We have shown how we used TF-IDF vectorization and a similarity measurement like cosine similarity to retrieve textual documents related to a set of terms. We thus have a tool that can be looked upon as a narrative-driven recommendation system. Additionally we have shown how we used TF-IDF vec-

[3] https://spacy.io/

torization to compute the weighted sum of terms in sentences, and then use the weights to extract the most important sentences in a text. This forms the basis of a simple summarization method. During the summarization process we also used the TF-IDF weights to extract the most important words of a text, which forms the basis of a simple keyword generation method.

We orchestrated a survey to obtain labeled data so that we could evaluate the performance of the recommendation tool. As observed in the evaluation scores of query 3 in Table 3, and to some extent query 1, the recommendation tool performs well if the input words are common and somewhat match with the words used in the theses description. But the evaluation scores of query 4 and 5, also presented in Table 3, shows that the recommendation tool struggles with cases where the input words are uncommon or represents too many different subjects. The average score of the F-scores is computed to 0.56, which we think is fine. For future implementations we have suggested the use of a thesaurus to expand the queries with related words, as this will most likely improve the similarity measures, especially for queries that consists of uncommon words.

The summarization and keyword generation methods became a supplement to the recommendation tool, and we did not have valid ways of measuring the performance of the methods. We have no valid way of measuring the performance of the tool, but after manual inspection we can conclude that it performs worse that we hoped. Two examples of summaries and keywords generated are presented in Table 4 and they show that the tool can perform well. However, summaries heavily relies on the quality of the raw data extracted from the website, and thus it performs quite poorly in most cases. Better structured data would improve the generated summaries by a large margin, and we have suggested that the keyword generation could be vastly improved by implementing some form of named entity recognition.

References

Toine Bogers and Marijn Koolen. Defining and supporting narrative-driven recommendation. RecSys '17, page 238–242, New York, NY, USA, 2017. Association for Computing Machinery. ISBN 9781450346528. doi: 10.1145/3109859.3109893. URL https://doi.org/10.1145/3109859.3109893.

Abdur Chowdhury and M. McCabe. Improving information retrieval systems using part of speech tagging. 1998.

Houtao Deng. Recommender systems in practice. https://towardsdatascience.com/recommender-systems-in-practice-cef9033bb23a, 02 2019. Accessed: 2021-05-15.

Hazra Imran and Aditi Sharan. Thesaurus and query expansion. *International Journal of Computer Science Information Technology*, 1, 11 2009.

Karen Spärck Jones. A statistical interpretation of term specificity and its application in retrieval. *Journal of Documentation*, 28:11–21, 1972.

D. Karatay and Pinar Karagoz. User interest modeling in twitter with named entity recognition. *CEUR Workshop Proceedings*, 1395:17–20, 01 2015.

Jussi Karlgren. An algebra for recommendations: Using reader data as a basis for measuring document proximity. *SYSLAB technical reports*, 179, 1990.

Blerina Lika, Kostas Kolomvatsos, and Stathes Hadjiefthymiades. Facing the cold start problem in recommender systems. *Expert Systems with Applications*, 41(4, Part 2):2065–2073, 2014. ISSN 0957-4174. doi: https://doi.org/10.1016/j.eswa.2013.09.005. URL https://www.sciencedirect.com/science/article/pii/S0957417413007240.

Alfrick Opidi. A gentle introduction to text summarization in machine learning. https://blog.floydhub.com/gentle-introduction-to-text-summarization-in-machine-learning/, 04 2019. Accessed: 2021-05-15.

Appendix A

A.1 Model architecture/pipeline

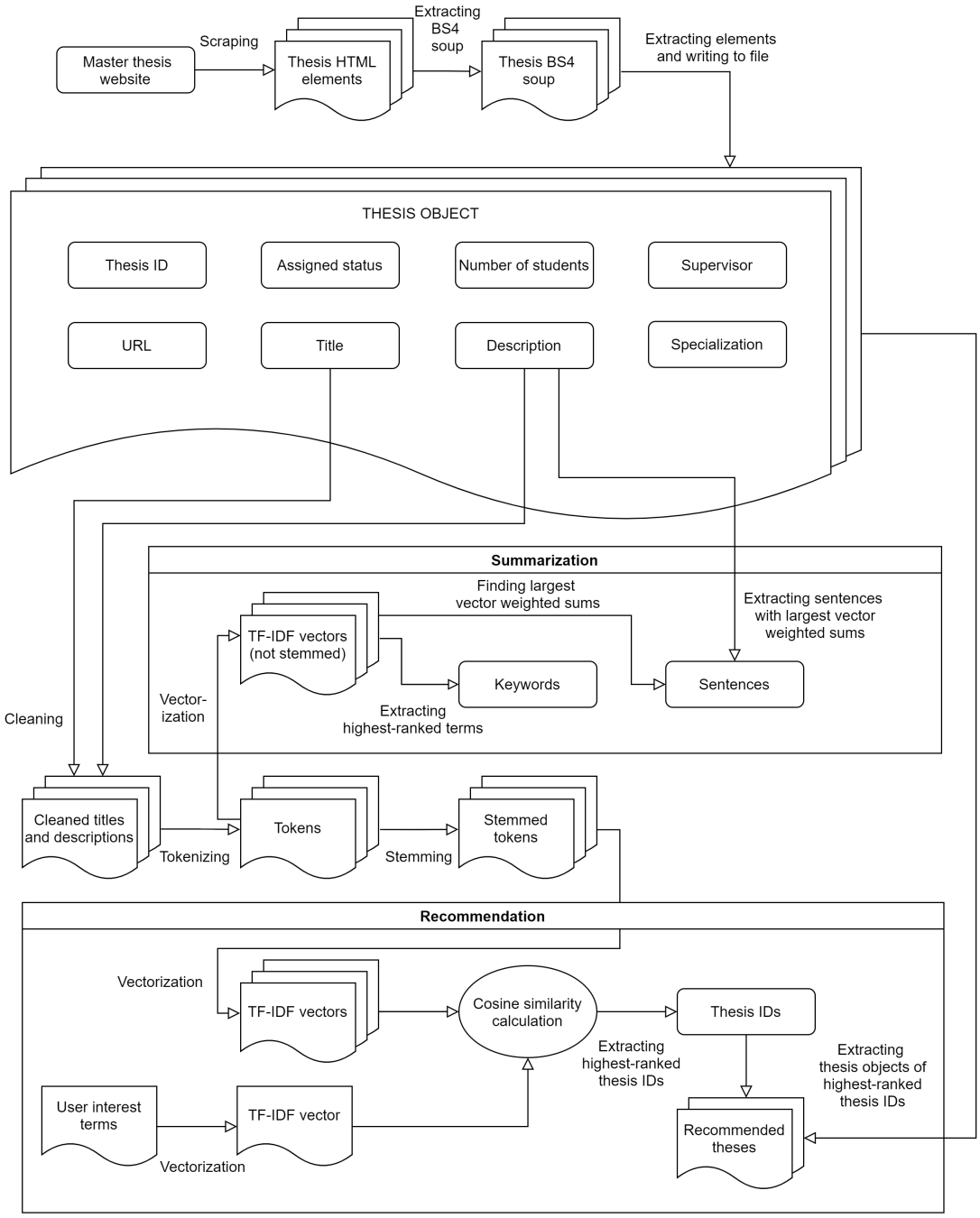

Figure 4: Diagram showing the pipeline (model architecture)