TECHNISCHE HOCHSCHULE NÜRNBERG
GEORG SIMON OHM

Fakultät Informatik

Topic Segmentation and Visualization for Video Lectures

Bachelorarbeit im Studiengang Informatik

vorgelegt von

Patrick Sabau

Matrikelnummer 3206099

Erstgutachter:	Prof. Dr. Korbinian Riedhammer
Zweitgutachter:	Prof. Dr. Jens Albrecht

Kurzdarstellung

Die vorliegende Bachelorarbeit beschreibt den Prozess, Themen (im Folgenden "Topics" genannt) aus Vorlesungen zu extrahieren, diese auf Zeitperioden anzuwenden sowie das Ergebnis in einer WebApp zu visualisieren. Bei den besagten Vorlesungen handelt es sich um zwei verschiedene Fächer, "Interventional Medical Image Processing" und "Pattern Analysis" , beide gelesen von Prof. Dr.-Ing. Joachim Hornegger. Nachdem die Transkripte dieser Vorlesungen vorverarbeitet sind, wird jedes Wort zu einem sogenannten Token umgewandelt. Diese Token werden zu Vektoren verarbeitet, welche Dokumente und die darin enthaltenen Worte symbolisieren. Algorithmen des Topic Modelings, unter anderem Latent Semantic Analysis, extrahieren aus diesen Vektoren die Topics. Durch Transformation von Dokumenten kann anschließend eine Verteilung dieser Themen errechnet werden. Dieser Prozess wird auf unterschiedliche Zeitblöcke der Vorlesung angewendet. Die dadurch entstehende Topicverteilung einer Vorlesung wird in einer WebApp mittels des JavaScript Frameworks ChartJs visualisiert. Zusätzlich wird dieser Ansatz mit einem anderen Algorithmus verglichen, dem sogenannten "TopicTiling". Um die Methoden zu evaluieren, werden zum eine die Schlüsselwörter mit von Menschen erstellten Themenlisten verglichen. Zusätzlich werden die Techniken an Biologie Vorlesungen getestet. Im Ausblick wird die Integration in das Media-Sharing Portal der Hochschule, detailliert beschrieben.

Abstract

The following bachelor thesis describes the process of extracting Topics from lectures, applying periods to them, and afterward visualizing the result on a website. Subjects of the lectures are "Interventional Medical Image Processing" and "Pattern Analysis", both held by Prof. Dr.-Ing. Joachim Hornegger. First, the transcripts are preprocessed and then each word gets converted into a token. Combining these into multiple vectors results in a matrix that represents the documents and their including words. Algorithms for Topic Modelling, like Latent Semantic Analysis, calculate the topics of this matrix. New documents can be transformed into this topic space, returning the topic distribution of it. This process is now applied to different timestamps of the lecture. Finally, the distribution is visualized on a web app, using the JavaScript library ChartJs. Additionally, this method is compared against another Algorithm called "TopicTiling". To evaluate the techniques, the keywords will be compared against a list of topics, which was created by humans. Additionally, the algorithms will be applied to biology courses for testing. The outlook describes the integration of this project into the media-sharing platform.

Contents

1. Introduction **1**

2. Data **3**

3. Method **5**
3.1. Data Preparation 5
3.1.1. Filtering and Normalization 5
3.1.2. Vectorization 8
3.2. Latent Semantic Analysis 10
3.2.1. Singular Value Decomposition 10
3.2.2. Method of Latent Semantic Analysis 11
3.2.3. Interpretation of LSA 11
3.2.4. LSA-Implementation using gensim 14
3.3. Non-negative Matrix Factorization 14
3.3.1. Method of NMF 15
3.3.2. NMF-Implementation using gensim 17
3.4. Topic Coherence 17
3.4.1. Coherence Pipeline 18
3.4.2. Topic Coherence implementation using gensim 20
3.5. Segmentation 22
3.6. Visualization 23
3.7. TopicTiling 24
3.7.1. Latent Dirichlet Allocation 25
3.7.2. Method of TopicTiling 26
3.7.3. Implementation 27

4. Software architecture **29**
4.1. Adding lectures 29
4.2. Getting the data 32
4.3. Deployment 33

5. Experiments **35**
5.1. Comparision on "Pattern Analysis 06" 35
5.2. Biology courses 36

5.3. Detect border of chained lectures . . . 37

6. Outlook . . . **41**

A. Calculating SVD . . . **43**

List of Figures . . . **47**

List of Tables . . . **49**

List of Listings . . . **51**

Bibliography . . . **53**

Glossary . . . **57**

Chapter 1.

Introduction

Online lectures are gaining more and more popularity in recent years, not only caused by COVID-19. Some of the most prestigious universities in the world, like the Massachusetts Institute of Technology, provide a wide variety of courses online, ranging from Biology to Physics [MIT]. The Technische Hochschule Georg Simon Ohm added Microsoft Teams and Zoom into their portfolio in 2020. On the one hand, this is an adaption to the Coronavirus, but also, on the other hand, a step into the future. Online learning allows every student to learn at their own pace, at their preferred location, and at their most favorable time. To supply the best education for students, universities try to expand their own online learning platforms' functionality. These additional features include 24/7 availability, subtitles, adaptive video speed, and other content like articles. Another method of improving this is a Topic Segmentation approach described in detail in this bachelor thesis.

Topic Segmentation is a method for splitting documents into topically coherent sections. Since a lecture for the whole semester can reach a total duration of over 30 hours(per subject), Topic Segmentation can make searching for information much faster and easier. This way, students have more time which they can efficiently use to learn.[Reyn 98, S. 3]

After the introduction, this thesis will explain the methods used in this project. It starts with exploring the data (like most other data-related projects do). The used lectures are the so-called LMELectures. [Ried 13]
Preprocessing the data is the next step. It consists of extracting the texts and metadata out of the individual files and sorting them. Afterward, not-needed parts for the models are filtered. These are, for example, stopwords, words like "the", which do not carry much meaning. Each remaining word now represents a token. These will be transformed into vectors using the Bag-of-words model. The result is a vector per document.
Next, these vectors are used as input for Topic Modeling, multiple methods to extract topics, and suitable text keywords. Two different algorithms are evaluated for this: Latent Semantic Analysis and Non-Negative Matrix Factorization. Both of them require an important parameter: the number of topics. Evaluation of the best number with the coherence score follows after this. It is calculated through a four-step pipeline consisting of Segmentation,

Probability Estimation, Confirmation Measure, and Aggregation.
From here on, the self-developed method differs from most other Topic Segmentation algorithms. Instead of getting time segments, as a result, the lectures are split in advance, e.g., into 90-second parts. The Topic Models are applied to each of these parts and thus getting topic distribution in this period
To make this useable for students, embedding the results in an HTML page with ChartJs, is required. It consists of a radar chart, where each data point represents the distribution of a specific topic.
In addition to the self-developed method, an algorithm "TopicTiling" is applied to the vectors as an alternative. TopicTiling is a topic segmentation method based on "TextTiling". The used Implementation of the algorithm requires a topic model, created by the Latent Dirichlet Allocation "GibbsLDA++".

To check if the method is useful, testing and evaluation are needed. First, the results are compared with the lecture "Pattern Analysis 06". For this specific lecture, labeled data about the main topics exist, which were created by students. These can be checked against the extracted keywords from the topics.
Second, the same techniques are applied to other data, containing four biology courses. Examination of the usefulness of the results follows.
The third and last experiment inspects if the "TopicTiling" algorithm would split multiple lectures that are chained together.

Most of the analysis is created in a jupyter notebook environment. It provides much functionality for developing, but it can not run as a production system. Therefore the pipeline is wrapped into a python app, which uses Flask to be reachable over a REST API. Other differences to the development environment are picked up in a separate chapter.

The outlook describes, who the system can be integrated into the media-sharing system of the TH Nuernberg. Additionally, it gives some ideas, how the system can be improved further and thus becoming more useful.

Chapter 2.

Data

The data which is used for this project was the LMELectures corpus. It consists of two courses, "Interventional Medical Image Processing" (in the following also referred to as "IMIP") and "Pattern Analysis" (referred to as "PA"). Both were read by Prof. Dr.-Ing. Joachim Hornegger, in summer 2019 at the "Friedrich Alexander Universität" (FAU) of Erlangen-Nuremberg. Today, Prof. Dr.-Ing. Hornegger is the president of the FAU[Der]. The courses took place in the RRZE e-studio. The audio was recorded with a close-talking microphone, while HD cameras took the video. There is also a version of the videos, where the slides are directly added to them.
Additionally, the corpus does contain not only the video but also the transcripts of it. These are managed in one file per lecture, where each line represents a time period reaching from roughly two to twenty seconds. Listing 2.1 shows some lines of the IMIP-01 lecture, where the first part represents the metadata and the second part the spoken text.

```
20090427-Hornegger-IMIP01_0001480_0003230.wav so welcome to the
20090427-Hornegger-IMIP01_0004360_0025300.wav first lecture on...
20090427-Hornegger-IMIP01_0025960_0035550.wav the summer sem...
```

Listing 2.1: example of transcript lines, LMELectures

As the transcriptions contain what was said and not what was meant to be said, it has some particular guidelines for different things. Prof. Hornegger is a native german speaker, so if he accidentally uses a german word, it is annotated with a " deu". For example, "ja~deu" means a german word was used instead of "yes".
Typical hesitations and also acoustic events are prefixed with the symbol "#". This contains e.g "#ahm" (hesitations) "#laugh" (event). These were not all the rules by which the transcripts are created, but the most important ones. [Ried 13]

The methods of this project are also applied to four unpublished biology courses, read by Prof Dr. Irmtraud Horst. She is a professor at the "Technische Hochschule Nuernberg Georg Simon Ohm" in the department of Applied Chemistry. The four courses are called

"B14", "B19", "BVT" and "Mikrobiologie", each with 10 to 13 lectures, sometimes split into multiple parts. These are all biology courses, some of them held for chemistry students. The transcripts are in german, the original language the lectures were read, and differ from the LMELecture transcripts. Only the time were a part was finished is provided, and thus we use the end time of the last part as the start of the current line. For the example lines in listing 2.2, we would set the first part from 0s to 45s, and the period for the second part from 45s to 74s.

```
A: herzlich willkommen zur ... #00:00:45-2#
A: die Erben Loesung ... #00:01:14-3#
```

Listing 2.2: example of transcript lines, biology lectures

Chapter 3.

Method

The following chapter is going to explain the technical methods used for the project. It first describes how the techniques work and afterwords how to apply them to real data. The code can be found on GitHub.[Saba 21]

3.1. Data Preparation

"**Garbage in, garbage out**" is such a famous phrase in computer science that it has its own Wikipedia page [Garb]. What it means is that if the input data is flawed, the output will also be. Accordingly, data preparation is one of the most important steps in data-related projects.

3.1.1. Filtering and Normalization

The goal of this step is to reduce the text to only the most essential parts. An example sentence is:

"The small dog likes the very long walks"

In the beginning, the sentence needs to be split into **tokens**. For this sentence, it seems quite trivial because it could just be split at each whitespace.

["The","small","dog","likes","the","very","long","walks"]

But in general, this task isn't always as trivial as in this sentence. Assume a sentence, like: "He doesn't like the U.S.A.". Splitting at whitespaces and punctuation, would result in the following token list:

["He","doesn","t","like","the","U","S","A"]

"U.S.A." should be one token, but how about "doesn't"? Either it could be seen as one token or split into two tokens, "does" and "not". As seen here, tokenizing is not trivial and tokenizer can get very complex. It would be possible to extend it with regex and other rules, but a much simpler way would be to use an external library. **spaCy** provides an accurate, flexible and fast tokenizer , which will be used here. [Lane 19, S. 43-47]

Listing 3.1 show the conrresponding code for it.

```
>>> nlp = spacy.load('en_core_web_trf')
>>> spacy_doc = nlp("The small dog likes the very long walks")
>>> print([token for token in spacy_doc])
[The, small, dog, likes, the, very, long, walks]
```

Listing 3.1: spaCy tokenizer

The example reveals another problem. First, the most common word in the sentence is "the", which does not have much meaning. Second, "The" and "the" are two different tokens. To deal with the first problem, filtering some words is the solution. These can be the so-called "**stopwords**", who often belong to the most common words in a language, but do not carry much information.[OEC] Additional examples include "a", "and", "of". [Lane 19, S. 51-52]

```
>>> nlp = spacy.load('en_core_web_trf')
>>> print(nlp.Defaults.stop_words)
{"'d", "'ll", "'m", "'re", "'s", "'ve", 'a', 'about', 'above',...}
```

Listing 3.2: spaCy stopwords

Also, it is recommendable just to keep the words of some word classes. For Topic Modeling, nouns have probably the highest value. spaCy provides a **Part-of-speech** tagger for this takes. It refers to a pre-trained and statistical model, which can predict the most likely label in a sentence.[Ling]. The model used is the "en_core_web_trf" from spaCy, which provides the most accuracy. A full list of labels can be found on the Github page of spaCy.[glos] Listing 3.3 shows the POS-tagger in action.

```
>>> nlp = spacy.load('en_core_web_trf')
>>> spacy_doc = nlp("The small dog likes the very long walks")
>>> print([token.pos_ for token in spacy_doc])
['DET', 'ADJ', 'NOUN', 'VERB', 'DET', 'ADV', 'ADJ', 'NOUN']
```

Listing 3.3: spaCy POS-tagger

The second problem requires some form of **normalization**. Converting all words to the lower case would solve the issue in the upper sentence (if the word "the" would not be filtered out with other stopwords). Verbs conjugation like "walk", "walks", or "walked" and plural nouns like "tree" and "trees" should also be sum up to the same token. It can make sense to keep every separate token for other NLP tasks but not for topic modeling. It is more important to read out a general overview of the text instead of getting each time form in a sentence correct. Two methods for achiving this are **stemming** and **lemmatization**. Stemming is a rule-based approach. It cuts off the ending of words, according to the implemented rules. This can be as simple as removing the trailing "s" at the end of words to having multiple complex rules. This project is using lemmatization. Lemmatization takes stemming further and does not only use rules, but it instead detects the association between words. It also uses lookups with the additional POS-tag, which can result extracting the lemma "good" out of the word "better".[Lane 19, S. 57-62] Listing 3.4 demonstrates the spaCy lemmatizer: "likes" got transformed to "like" and "walks" to "walk".

```
>>> nlp = spacy.load('en_core_web_trf')
>>> spacy_doc = nlp("The small dog likes the very long walks")
>>> print([token.lemma_ for token in spacy_doc])
['the', 'small', 'dog', 'like', 'the', 'very', 'long', 'walk']
```

Listing 3.4: spaCy lemmatizer

Combining all the steps results in the final tokenizer. Listing 3.5 shows an abstract of it. Supplementary, the tokenizer uses other function:

- Line 4: The text isn't directly passed to spaCy. Before, the method filter_special_tokens() is called, which removes the acoustic events prefixed with "#"
- Lines 5-6: Tokens get lemmatized if they belong to the given POS-tags. Other tokens get removed. Keeping more than just the nouns makes sense here because the data is limited, and otherwise, it would get relatively small.
- Lines 7-8: Stopwords get removed
- Line 9: The function combine_tokens() is called. It is responsible for combining some tokens, which were split before. An example would be to combine "x." and "ray".
- Lines 10-11: tokens of extreme sizes get filtered

```
 1  import spacy
 2  nlp = spacy.load('en_core_web_trf')
 3  def tokenize(text):
 4  spacy_doc = nlp(filter_special_tokens(text))
 5  tokens = [token.lemma_ for token in spacy_doc
 6      if token.pos_ in ['NOUN', 'PROPN', 'VERB']]
 7  tokens = [token for token in tokens
 8      if token.lower() not in nlp_spacy.Defaults.stop_words]
 9  tokens = combine_tokens(tokens)
10  tokens = [token for token in tokens if len(token)>2
11                        and len(token)<25]
12  return tokens
```

Listing 3.5: tokenizer

The example sentence gets transformed into ['small', 'dog', 'like', 'long', 'walk']. Listing 3.6 shows the result of other sentences.

```
>>> tokenize("The small dog likes very long walks")
['small', 'dog', 'like', 'long', 'walk']
>>> tokenize("first lecture on interventional image processing")
['lecture', 'interventional', 'image', 'processing']
>>> tokenize("today i'm going to talk about magnetic navigation")
['talk', 'magnetic', 'navigation']
```

Listing 3.6: example tokenizer

3.1.2. Vectorization

At this point, the data is read from the files, preprocessed, and saved in a pandas dataframe. The next step is to vectorize the tokens. This is achieved through counting the number each tokens appears in a document. For the example sentence, this means:

{'small':1, 'dog':1 , 'like':1, 'long':1, 'walk':1}

This representation is called **Bag-of-Words** (BoW). The documents with their BoW representation are combined into a matrix, the so-called **term-document matrix**. Each column represents a document, and each row how often a token is included in the document. Table 3.1 shows such a matrix, created out of 4 documents. Doc1 was the example sentence, doc4

	doc1	doc2	doc3	doc4
small	1	0	1	0
dog	1	1	2	1
like	1	1	2	2
long	1	0	1	1
walk	1	0	1	1
play	0	1	1	0
ball	0	1	1	0
cat	0	0	0	1
home	0	0	0	1

Table 3.1.: example of a term-document matrix

could be a sentence like: "Dogs like to go for long walks, while cats like to stay at home". The collection of all documents (here: doc1 to doc4) is called **corpus**. Each document and, therefore, each column is a vector. Their dimension is identical to the number of unique tokens in the corpus.[Jura 20]

The Bag-of-Words model uses the pure number of appearances of each token. For some algorithms, this makes sense as input. But this can also result in suboptimal weighting for the importance of a word. One way to avoid this, is to use **tf-idf** vectors. These consists of two components, the local **term-frequency** and the global **inverse document frequency**.

The first component, term-frequency, describes the number of appearances of each unique word t in document d. These would be the values of Table 3.1. To avoid distortion in long documents, the document result can be normalized or calculated differently than the raw count. The python library used for calculating these vectors, **gensim**, normalizes just the whole result as default and not the term-frequency[TF I].

$$tf(t,d) = count(t,d) \tag{3.1}$$

The second component, inverse document frequency, is a measure to determine if a word is rare in the corpus or if it appears pretty common. It is calculated by dividing the total number of documents N by the number of documents containing word t $D : t \in D$. Afterwards, the result is logarithmized.

$$idf(t) = log\frac{N}{count(D : t \in D)} \tag{3.2}$$

Accordingly, the tf-idf weight for a word t in document d is:[Baez 11]

$$tfidf(t,d) = tf(t,d) * idf(t) \tag{3.3}$$

Implementation of the above methods is achieved by using the library gensim. The benefit of using it for this task is that gensim is also used afterward for topic modeling. This makes using the tf-idf vectors work flawless with other models.

```
1 >>> from gensim import corpora
2 >>> from gensim.models import TfidfModel
3 >>>
4 >>> dct = corpora.Dictionary()
5 >>> dct.add_documents(speech_df['Text_tokens'].tolist())
6 >>> corpus = [dct.doc2bow(text) for text in
7 >>>                  dataframe['Text_tokens'].tolist()]
8 >>> tfidf_vectorizer = TfidfModel(corpus)
9 >>> corpus_tfidf = tfidf_vectorizer[corpus]
```

Listing 3.7: gensim tfidf-vector

Listing 3.7 shows how the implementation is done:

- Lines 4-5: Create a gensim dictionary. It saves all words of a corpus with a unique id. The tokens from the documents are added to it (which are stored in the speech_df DataFrame)
- Lines 6-7: The term-document matrix is created. Instead of being mapped to the tokens directly (like in table 3.1), they are mapped to their corresponding id from the dictionary

- Line 8: The tf-idf-vectorizer gets fit to the data
- Line 9: The vectorizer can now be used to transform the corpus(BoW) into tf-idf format

3.2. Latent Semantic Analysis

Latent Semantic Analysis (sometimes referred to **Latent Semantic Indexing**) is an algorithm for Topic Modeling. These are statistical techniques to find **topics** from a large number of documents. These can provide a quick overview of the content of the documents. LSA is based on the so-called singular value decomposition, a method that will be described in detail.[Albr 20]

3.2.1. Singular Value Decomposition

Singular value decomposition (SVD) is a method from linear algebra, which represents a matrix as the mathematical product of 3 special matrices.

$$A = USV^{\mathrm{T}} \tag{3.4}$$

If A is a $t \times d$ matrix, than:

- U is a unitary matrix of size $t \times t$. Unitary means, that $U^{\mathrm{T}}U = I$. The columns of it, are the orthonormal eigenvectors of AA^{T}.
- S is a diagonal matrix of size $t \times d$. It contains the square roots of eigenvalues from U or V un descending order.
- V^{T} is the transpose of a unitary matrix of size $d \times d$. The rows of V are the orthonormal eigenvectors of $A^{\mathrm{T}}A$.

Appendix A contains a detailed calculation of an example matrix

$$A = \begin{bmatrix} 4 & 1 \\ 1 & 3 \\ 2 & 1 \end{bmatrix}$$

which results in the following SVD:

$$USV^{\mathrm{T}} = \begin{bmatrix} 0.77 & 0.49 & (-0.41) \\ 0.47 & (-0.87) & (-0.16) \\ 0.44 & 0.06 & 0.90 \end{bmatrix} \begin{bmatrix} 5.13 & 0 \\ 0 & 2.39 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} 0.86 & 0.51 \\ 0.51 & -0.86 \end{bmatrix} \approx \begin{bmatrix} 4 & 1 \\ 1 & 3 \\ 2 & 1 \end{bmatrix} = A$$

3.2.2. Method of Latent Semantic Analysis

The first step of the algorithm is to apply SVD to the term-document matrix (size $t \times d$, which gives the three matrices U, S, V^T. To get a specific number of topics k (which is selected), only the k square roots of the eigenvalues are kept in S. This reduces S to a matrix of size $k \times k$. The corresponding columns of U and the rows of V_T need to be removed according to the size of the eigenvalue. This means the rows get removed from the bottom to the top, and the rows from right to left. Otherwise, the matrix multiplication is not possible.[Bake 05]

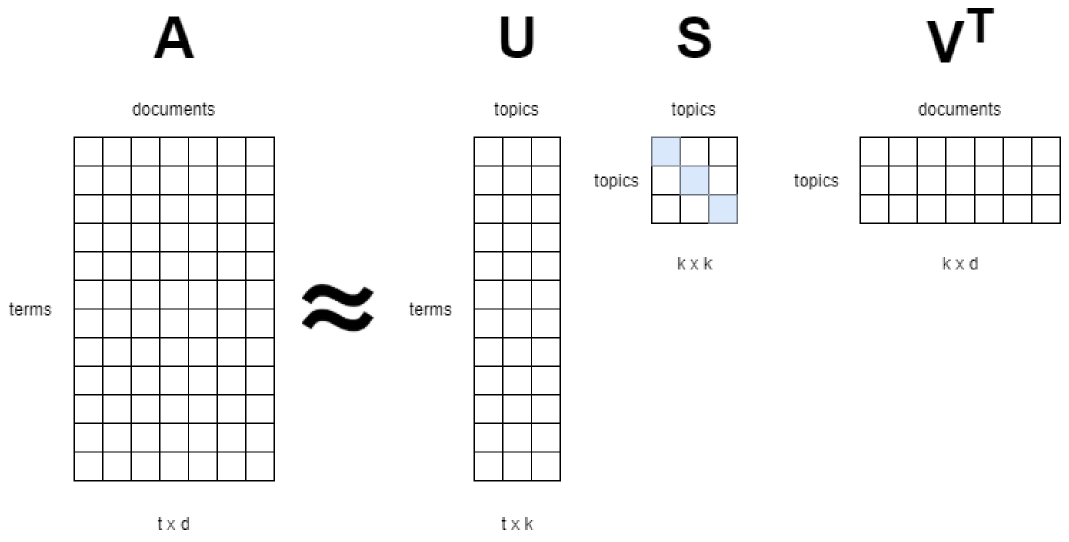


Figure 3.1.: Visual Reprensentation of LSA

Figure 3.1 (created after [Deer 90]) illustrates the LSA visually. The number of topics k is set to 3, which reduces S to size 3×3. Since A has 12 rows (= dictionary), U is a 12×3 matrix. V_T is a 3×12 matrix, which matches the number of documents from A. Multiplying U, S and V^T only approximates the original matrix A.

3.2.3. Interpretation of LSA

LSA returns three matrices, but what do these mean? Figure 3.1 also describes what the columns and rows mean for each matrix.

- A is the original term-document matrix. Each column is a document, with the corresponding tokens inside it(eventually weighed with tf-idf). Each token has its own row in the matrix.

- The columns of U are the different topics. The rows are equal to the rows of A, even with the same index. But in this case, the values do not represent how often a token appears inside a document but the affiliation of a term to a topic. Higher values mean if the term appears in a document, it is rather assigned to this specific topic. Negative values mean the exact opposite.

- S has the square roots of the eigenvalues, and thus the importance of each topic, in the diagonal. Reducing the number of topics k results in removing the least important topic and thus the last row/column.

- V^{T} holds the documents and their distribution of topics.

	doc1	doc2	doc3	doc4	doc5	doc6
cat	2	4	1	6	0	0
dog	2	0	4	1	0	0
bird	2	1	3	2	0	0
car	0	0	0	0	4	2
engine	0	0	0	0	6	1

Table 3.2.: term-document matrix used as example for LSA

Table 3.2 is an example of a term-document matrix for a calculation of LSA. The interpretation of the real data (LMELectures) follows the same structure but is much more complicated since the number of documents and terms is much larger. The table shows that there are two topics: animals(cat, dog , bird) and another about cars(car, engine). Therefore k=2 was chosen for this example.This returns the matrices USV^{T} :

$$USV^{\mathrm{T}} = \begin{bmatrix} 0.82 & 0 \\ 0.36 & 0 \\ 0.44 & 0 \\ 0 & 0.59 \\ 0 & 0.81 \end{bmatrix} \begin{bmatrix} 8.69 & 0 \\ 0 & 7.47 \end{bmatrix} \begin{bmatrix} 0.37 & 0.43 & 0.41 & 0.71 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0.96 & 0.27 \end{bmatrix}$$

First, analyzing U: It gives the affiliation of the terms to a topic. Topic 1 (first column) has an affiliation ranging from 0.36 to 0.82 for the animal words (cat, bird, dog), while the car words (car, engine) are both 0. Otherwise, Topic 2 is the exact opposite; the car words have an affiliation of 0.59 to 0.81, while the animals are all 0. In most real-world examples, this isn't so clear cause much words overlap. There could, for example, be a document containing words from both topics, which would make them overlap.
Second, S has only two eigenvalues left. These are the highest eigenvalues left, meaning that these two topics have the highest dominance in the corpus. This can be verified because each document contains either of the topics, not both.
Third, looking at the topic distribution of the documents contained in V^{T}. Document 1 (column 1) has a distribution of 0.37 to Topic 1 and 0 to Topic 2. Since each animal word

has an appearance of two times in the document, while car words have 0 appearances, this is right. It would make sense to normalize these values for some applications, mapping the highest Topic appearance to 1 and the lowest to 0. The rest is mapped to the corresponding number between 0 and 1.
One feature of such a model should be, to apply it to a new document. This can be for comparison of documents, search queries, or get the topic distribution. To get it, the new document is treated as a matrix A_{q} of size $m \times 1$, which is like a single column of the original term-document matrix. A new version of V, called V_{q}, needs to be calculated. This is accomplished with:

$$V_{\mathrm{q}} = {A_{\mathrm{q}}}^{\mathrm{T}} U S^{\text{-}1} \tag{3.5}$$

where U and S are taken over from the calculated LSA model.[Deer 90]. If the new document would be

$$A_{\mathrm{q}} = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 0 \\ 0 \end{bmatrix}$$

for the previous example, it would be assigned to the animal topics, because all words appearing in that document, are assigned to Topic 1. Using the formula can verify this.

$$V_{\mathrm{q}} = {D_{\mathrm{q}}}^{\mathrm{T}} U S^{\text{-}1} = \begin{bmatrix} 1 & 1 & 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} 0.82 & 0 \\ 0.36 & 0 \\ 0.44 & 0 \\ 0 & 0.59 \\ 0 & 0.81 \end{bmatrix} \begin{bmatrix} 0.11 & 0 \\ 0 & 0.134 \end{bmatrix} =$$
$$= \begin{bmatrix} 1.62 & 0 \end{bmatrix} \begin{bmatrix} 0.11 & 0 \\ 0 & 0.134 \end{bmatrix} = \begin{bmatrix} 0.18 & 0 \end{bmatrix}$$

The transposed matrix ${V_{\mathrm{q}}}^{\mathrm{T}} = \begin{bmatrix} 0.18 \\ 0 \end{bmatrix}$ confirms that the document is assigned to topic 1.
Besides topic modeling, LSA can also provide good results for comparisons, but they aren't used in this project. A shortlist of what is possible is still provided: [Deer 90]
– between two terms
– between two documents
– between a term and a document

3.2.4. LSA-Implementation using gensim

As mentioned, gensim is used for the topic modeling of this project. The LSA model (called LsiModel) provides several algorithms to adapt to every scenario, e.g., it supports running on a cluster instead of a single machine[Late]. Gensim only stores the U and S matrices. V is not held directly because it can exceed the memory of a system (if a huge dataset is used). But it can be recreated by applying the original corpus to the model.
Listing 3.8 shows how a new model is created. The LsiModel class takes three arguments, the first is the term-document matrix of the corpus weighted with tf-idf. The second argument id2word=dct, hands over the dictionary. It is used to map the word ids back to the original token, thus making the words human readable. The third argument is the number of topics. Here they are set to 5. Later, CoherenceModels are evaluateing the best number of topics.

```
>>> from gensim.models import LsiModel
>>> model = LsiModel(tfidf_vectorizer[corpus], id2word=dct,
>>>                  num_topics=5)
>>> model.show_topics()
[(0, 0.207*"derivative" + 0.192*"differential" + 0.182*"rank"...),
(1, '-0.231'*"functional" + 0.194*"euler" + 0.151*"prime"...),
...]
```

Listing 3.8: LsiModel gensim

Calling the function model.show_topics() lists the topics and the most important words for defining the specific topic. As seen, the keywords can also have negative values. The following method, Non-negative matrix factorization, avoids this. Listing 3.9 shows how a new document is transformed into the vector space. Each topic distribution can be printed by looping through the resulting vector.

```
>>> vector = model[tfidf_vectorizer[new_document]]
>>> for topic_dist in vector:
>>>     print(topic_dist)
(0, 0.23345)
(1, 0.07343)
...
```

Listing 3.9: Apply new document to LsiModel

3.3. Non-negative Matrix Factorization

Non-negative Matrix Factorization (short **nmf**) is another method for Topic Modeling. The goal is to approximate the term-document matrix A by two smaller matrices U and V

with the constraint that every matrix consists of only positive numbers. This condition is proper for A, as a word can't have a negative number of appearances in a document. Listing 3.2 visualizes NMF. It also shows other characteristics of NMF:

- U is a matrix of size $t \times k$, where t is the count of rows from A (terms) and k is a set value for the number of topics.
- V is a matrix of size $k \times d$, where d is the number of columns from A (documents).

For NMF to be helpful, k should be smaller than the dimensions of A. Otherwise, the interpretation of the resulting matrices U and V would be more complicated than that of A.

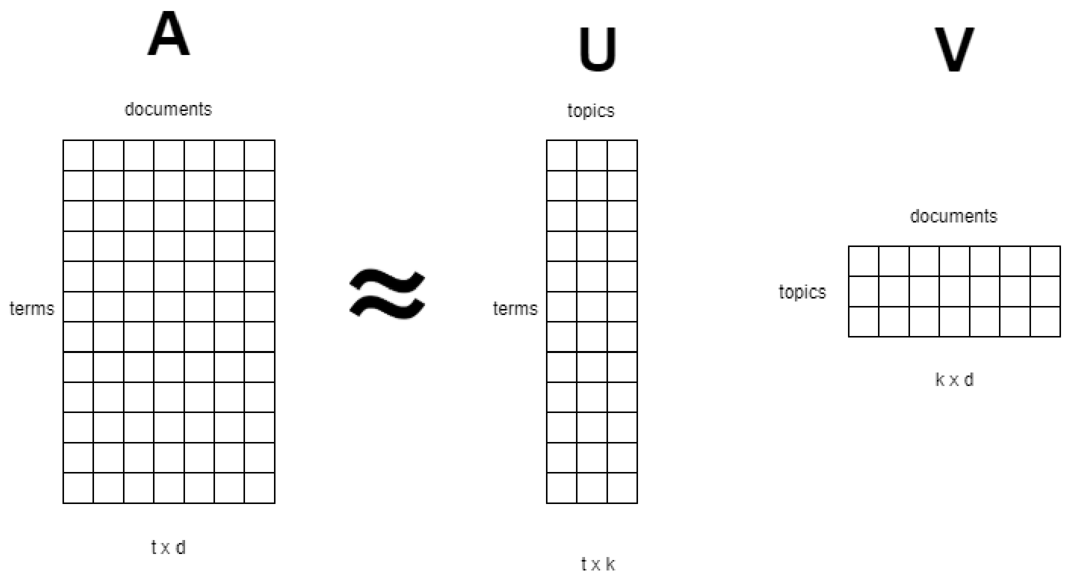


Figure 3.2.: Visual Reprensentation of NMF

Besides the application for text data, NMF can also be applied in other fields, like face recognition or spectral data analysis.[Berr 07]

3.3.1. Method of NMF

The objective of the method is to find two matricies, U and V that approximate A. Therefore a way to evaluate the approximation is needed, somekind of distance measure. The square of the euklidean distance between A and the product of U and V can be used:[Lee 01]

$$||A - UV||^2 = \sum_{ij} (A_{\text{ij}} - (UV)_{\text{ij}})^2 \tag{3.6}$$

Other distance measures can also be used. In each case, the goal is to minimize this function, which means the approximation for A is as accurate as possible. The algorithms can be divided into three main groups:[Berr 07]
– multiplicative update algorithms
– gradient descent algorithms
– alternating least squares (ALS)
Algorithms can also be a mixture of multiple groups, like the one used in gensim. It is a mixture of the first and second group[Non].

One of the most known methods for NMF created by Daniel Lee and Sebastian Seung. It us based on the following multiplicative update rules, which results in a lower or equal result of the cost function:[Lee 01]

$$V_{\mathrm{kd}} \leftarrow V_{\mathrm{kd}} \frac{(U^{\mathrm{T}}A)_{\mathrm{kd}}}{(U^{\mathrm{T}}UV)_{\mathrm{kd}} + 10^{-9}} \tag{3.7}$$

$$U_{\mathrm{tk}} \leftarrow U_{\mathrm{tk}} * \frac{(AV^{\mathrm{T}})_{\mathrm{tk}}}{(UVV^{\mathrm{T}})_{\mathrm{tk}} + 10^{-9}} \tag{3.8}$$

The indices $_{\mathrm{kd}}$ and $_{\mathrm{tk}}$ reference the coordination of the values. It means that each value of the matrix is updated individually. In the fractal, 10^{-9} is added to avoid division by zero. The pseudo code for the NMF of a given matrix A (term-document matrix, size $t \times d$) and a provided number of topics k would look like: [Berr 07]:

Algorithm 1 NMF using multiplicative updates

```
1: Initialize $U$ as a $t \times k$ matrix with random numbers $>= 0$
2: Initialize $V$ as a $k \times d$ matrix with random numbers $>= 0$
3: for $iteration = 1, 2, \ldots, maxIters$ do
4:     Update Matrix $V$ with equation (3.7)
5:     Update Matrix $U$ with equation (3.8)
6: end for
```

Each column of A is a vector a_{i} that has an equal column in V, v_{i}.

$$a_{\mathrm{i}} \approx U \times v_{\mathrm{i}} \tag{3.9}$$

This comes in handy for applying new documents to the model. The new term-document vector is equal to a_{i} and U is already calculated. Therefore, v_{i} is calculated for this document, and thus the corresponding topic distribution.

3.3.2. NMF-Implementation using gensim

As mentioned, gensim uses a mixture of multiplicative update and gradient descent to calculate NMF. It is an online algorithm created by Renbo Zhao and Vincent Tan[Zhao 17, Non]. Online means that the input corpus is split into smaller batches, each containing, e.g., only ten documents instead of all documents. The model is updated iterative, meaning for every batch, the matrix U gets improved. The matrix V is only calculated temporarily for each batch and afterward removed from memory.

The gensim NMF API is similar to the LsiModel, as seen in listing 3.10. The topic keywords only have positive values, like it was forced with the NMF method.

```
>>> from gensim.models import nmf
>>> model = nmf.Nmf(tfidf_vectorizer[corpus], id2word=dct,
>>>                     num_topics=5)
>>> model.show_topics()
[(0, 0.023*"quaternion" + 0.017*"column" + 0.017*"rank" ...
(1, '0.016*"pattern" + 0.014*"marker" + 0.013*"orthographic"...),
...]
```

Listing 3.10: NMF gensim

Listing 3.11 shows how to get the topic distribution of a new document. This differs from the LsiModel since only values greater than 0 are returned.

```
>>> vector = model.get_document_topics(vectorizer[new_document])
>>> for topic_dist in vector:
>>>     print(topic_dist)
(2, 0.16345)
(6, 0.47643)
...
```

Listing 3.11: Apply new document to NMF-Model

3.4. Topic Coherence

Most topic modeling algorithms are unsupervised methods, meaning they will automatically extract topics from an extensive collection of documents. The occurring problem is that these models have no guarantee that the topics make sense. Consequently, a measure to evaluate and compare the models is needed. One method to gain such metrics is **Topic Coherence**. It tries to provide a rating as close as possible to that of a human.

3.4.1. Coherence Pipeline

Topic Coherence is a method, consisting of four parts piped together:
– Segmentation
– Probability Calculation
– Confirmation Measure
– Aggregation

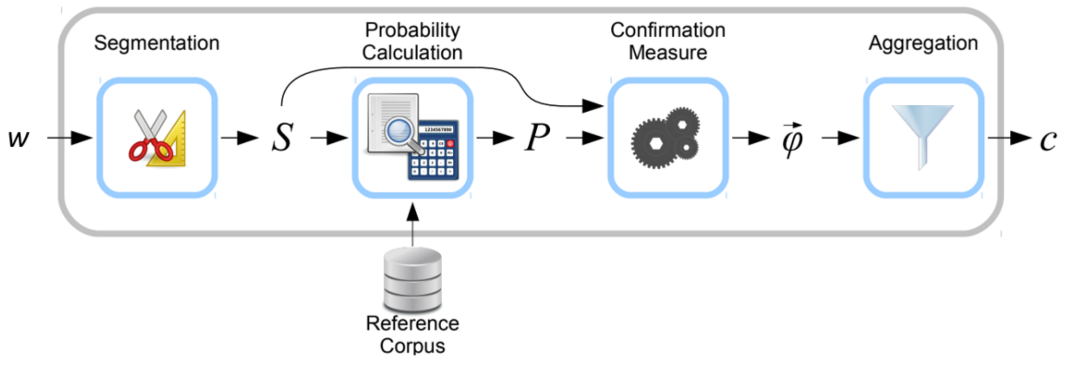


Figure 3.3.: Coherence Pipeline [Roed 15]

The first step, Segmentation, splits the word set W into multiple parts. For evaluating topics, the most important topic keywords are weighted more.

Each part contains two subsets (W^1| W^2), where the size is determined by the chosen segmentation:

- S^{one}_{one} Segmentation maps each word w_i to another single word w_j

$$S^{one}_{one} = \left\{(W^1 \mid W^2) \mid W^1 = w_i; W^2 = w_j; w_i, w_j \in W; i \neq j\right\} \tag{3.10}$$

- S^{one}_{all} Segmentation maps each word $W^1 = w_i$ to all other words $W \backslash \{w_i\}$

$$S^{one}_{all} = \left\{(W^1 \mid W^2) \mid W^1 = w_i; w_i \in W; W^2 = W \backslash \{w_i\}\right\} \tag{3.11}$$

- S^{one}_{set} Segmentation maps each word $W^1 = w_i$ to all words W

$$S^{one}_{set} = \left\{(W^1 \mid W^2) \mid W^1 = w_i; w_i \in W; W^2 = W\right\} \tag{3.12}$$

The next step, Probability Calculation, enumerates how likely it is that a word appears in a text. Also, two methods exist for this step in the pipeline.
Boolean document divides the number of documents in which the word (or all words of a word set) appears by the total number of documents. Different variants like boolean sentence or boolean paragraph can be used, where instead of a document, only sentences or paragraphs are counted. The method is called boolean, as it only considers if a word appears and not how often.
Boolean sliding window applies the same method, but instead of using whole documents, it operates with a sliding window over a document. It slides token by token over a text with a fixed length, where the window of each step is treated as a separate document.

The penultimate step, Confirmation Measure, calculates how much the word (or word set) W^2 supports the occurrence of W^1 and thus their coherence. There are two ways for doing this, either direct or indirect. Some examples for the direct measures would be the difference-measure (equation 3.13), the ratio-measure (equation 3.14), the log-ratio measure (equation 3.15) and the normalized log-ration measure (equation 3.16).

$$m_{\mathrm{d}}(W^1, W^2) = P(W^1 \mid W^2) - P(W^1) \tag{3.13}$$

$$m_{\mathrm{d}}(W^1, W^2) = \frac{P(W^1, W^2)}{P(W^1) \times (W^2)} \tag{3.14}$$

$$m_{\mathrm{lr}}(W^1, W^2) = \log \frac{P(W^1, W^2) + \epsilon}{P(W^1) \times (W^2)} \tag{3.15}$$

$$m_{\mathrm{nlr}}(W^1, W^2) = \frac{m_{\mathrm{lr}}(W^1, W^2)}{-\log(P(W^1, W^2) + \epsilon)} \tag{3.16}$$

The epsilon ϵ is added to avoid taking the log of 0. There are a lot of other measures. The indirect approach can be useful if W^1 and W^2 support each other while not often appearing together. This can be the case if they have multiple words with whom they both co-occur. For this measure,W^1 and W^2 are transformed to vectors with the size of the whole word set W. To get these vectors, a direct measure (like normalized log-ration measure) is applied to $W^{1,2}$ and all the single words of W. The vectorization can also be non-linearly distorted.

$$\overrightarrow{w}_{\mathrm{m},\,\gamma}\,(W^1) = \left\{ \sum_{w_{\mathrm{i}} \in W^1} m(w_{\mathrm{i}}, w_{\mathrm{j}})^{\gamma} \right\}_{j=1,\ldots,|W|} \tag{3.17}$$

The resulting vectors can be compared e.g throug cosine similarity or jaccard similarity. Accordingly, for an indirect confirmation measure, a direct measure like m_{nlr} and a similarity method like cosine similarity and optional a value for γ need to be chosen.

$$\widetilde{m}_{sim(m,\gamma)}(W^1, W^2) = s_{sim}(\overrightarrow{w}_m, \gamma(W^1), \overrightarrow{w}_m, \gamma(W^2)) \tag{3.18}$$

The last step, Aggregation, takes all the calculated confirmation and aggregates them. This can be done through a wide variety of methods, ranging from basic arithmetic mean σ_a, over the median σ_m to the harmonic mean σ_h. Most methods use the arithmetic mean.

There are different measures, which combine the pipeline with different methods, like C_{UMass} or C_{NPMI}. The one with the highest correlation to human ratings is C_V. It consists of S_{set}^{one} Segmentation, the boolean sliding window, an indirect confirmation measure using cosine similarity and the normalized log-ratio measure, as well as an aggregation through arithmetic mean.σ_a [Roed 15]. It can be written as:

$$C_V = (P_{sw(x)}, S_{set}^{one}, \tilde{m}_{cos(nlr,1)}, \sigma_a) \tag{3.19}$$

3.4.2. Topic Coherence implementation using gensim

Gensim implements the coherence pipeline with the ability to customize every step. Predefined measures like C_V are also directly supported. Listing 3.12 shows, how a to calculate the coherence of an existing model.

```
>>> from gensim.models import CoherenceModel
>>> coherence = CoherenceModel(topics= topics, dictionary=dct,
>>>                texts = lecture_df['tokens'].tolist(),
>>>                coherence='c_v')
>>> print(coherence.get_coherence())
0.506447
```

Listing 3.12: CoherenceModel gensim

Coherence can be used to compare different models. In this way, it's possible to compare models with a different number of topics and decide which number to choose based on the results. Accordingly, the model creation and coherence evaluation are wrapped into the same function and only the best model is kept.

```
def evaluate_model(dataframe,min_topics=5, max_topics = 20,
                   measure='c_v', model_type= 'lsi'):
    model_list=[]
    coherence_values = []
    #create the corpus for the model
    corpus,tfidf_vect = prep_for_model(dataframe)

    for topics_num in range(min_topic,max_topic + 1):
        #Create the the models with increasing number of Topics\
        if model_type == "nmf":
            model = nmf.Nmf(tfidf_vect[corpus], id2word=dataset,
                num_topics=topics_num)
```

```
13         else:
14             model = LsiModel(tfidf_vect[corpus], id2word=dataset,
15                 num_topics=topics_num)
16         model_list.append(model)
17
18         topics= [[word for word, prob in topic] for topicid, topic
19                     in model.show_topics(formatted=False)]
20         #Create the CoherenceModel and evaluate its score
21         coherence_model = CoherenceModel(topics=topics, dictionary=dct,
22                 texts=dataframe['tokens'].tolist(),coherence=measure,
23                 window_size=30)
24         coherence_values.append(coherence_model.get_coherence())
25
26     # Show graph
27     x = range(min_topic_num, max_topic_num + 1)
28     plt.plot(x, coherence_values)
29     plt.xlabel("Number of Topics")
30     plt.ylabel("Coherence score")
31     plt.legend(("c"), loc='best')
32     plt.show()
33     return model_list, coherence_values, corpus
```

Listing 3.13: evaluation with CoherenceModel

Listing 3.13 shows how this is accomplished. Models are created with an increasing number of topics(lines 8-16) depending of the min and max number of topics and the model type. Afterward, the topics are extracted(lines 18-19). These are plugged into the coherence pipeline(lines 21-24). The number of topics and the corresponding coherence score are then visualized using matplotlib. Figure 3.4 shows such a plot. An ideal number of topics would be 10, since it has a local maximum there and it's the highest value of the plot.

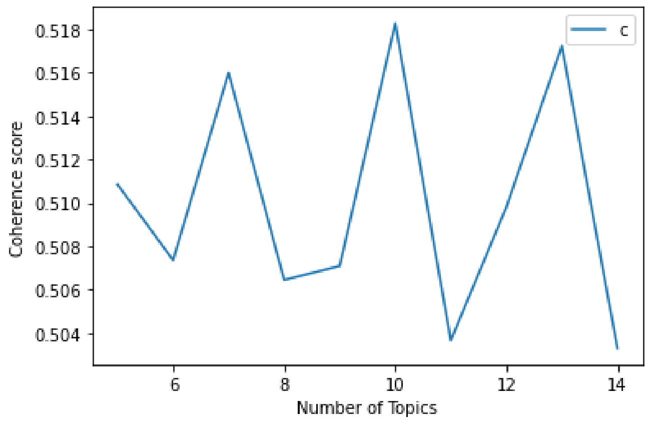


Figure 3.4.: Coherence Scores

3.5. Segmentation

Since the lecture transcripts are already split into small parts of 2-20 seconds, it's reasonably simple to segment them into smaller pieces. Therefore, creating a function, which takes the segments_size in seconds as an argument and returns a new dataframe where each part is such a segment. The size can vary, depending on the length of the parts in the transcript, because only parts that as a whole part of the segment will be used. An overlap can also be provided, which makes the segments overlay for this time period. E.g., if the segment size is 90s and the overlap is 30s, then every 60s, a new segment will start, creating the following timestamps:
−0s to 90s
−60s to 150s
−120s to 270s ...

Figure 3.5 shows how the current pipeline looks after integrating the segmentation.

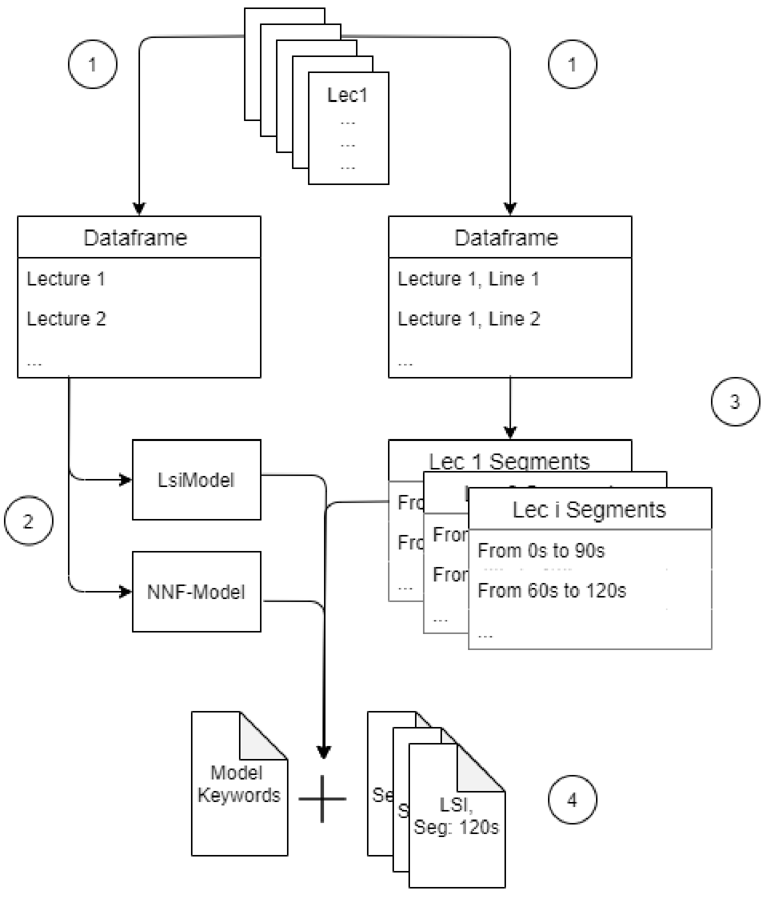


Figure 3.5.: Segmentation pipeline

The pipeline contains multiple steps:

- 1.) The transcripts get read from file, preprocessed, tokenized, and then stored into two separate dataframes. The first keeps the lectures as a whole for creating the models. The second dataframe keeps every line of the transcript, which is needed for the segmentation

- 2.) For each lecture, a LsiModel and an NMF-Model are created. As training data, all lectures of the same course are used, except the one itself. The number of topics is evaluated with the coherence pipeline (number of topics ranging from 5 to 15), which chooses the model with the highest coherence score.

- 3.) Each lecture is segmented, with a segment_size that extends from 60s to 120s and an overlap varying between 0s and 30s.

- 4.) The segments are applied to the model and thus resulting in the topic distribution of this time period. For each model, a keyword file is created, and for each segmentation, two files (one for the LsiModel and one for NMF) are created. The topic distribution for the LsiModel is normalized because otherwise, they can range from -1 to 1. The files are afterward used for the visualization.

3.6. Visualization

To make the topic distribution helpful for students, they need to be visualized and thus being tangible. Therefore, a simple WebApp, consisting of pure HTML, CSS, and JavaScript, is created. The landing page shows a simple banner , where a lecture can be selected.

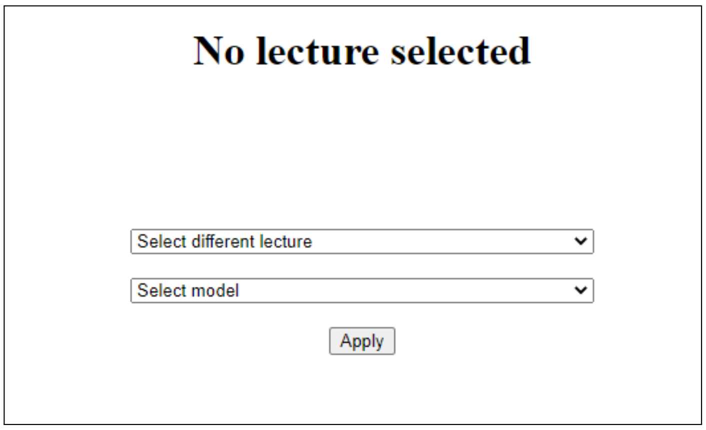
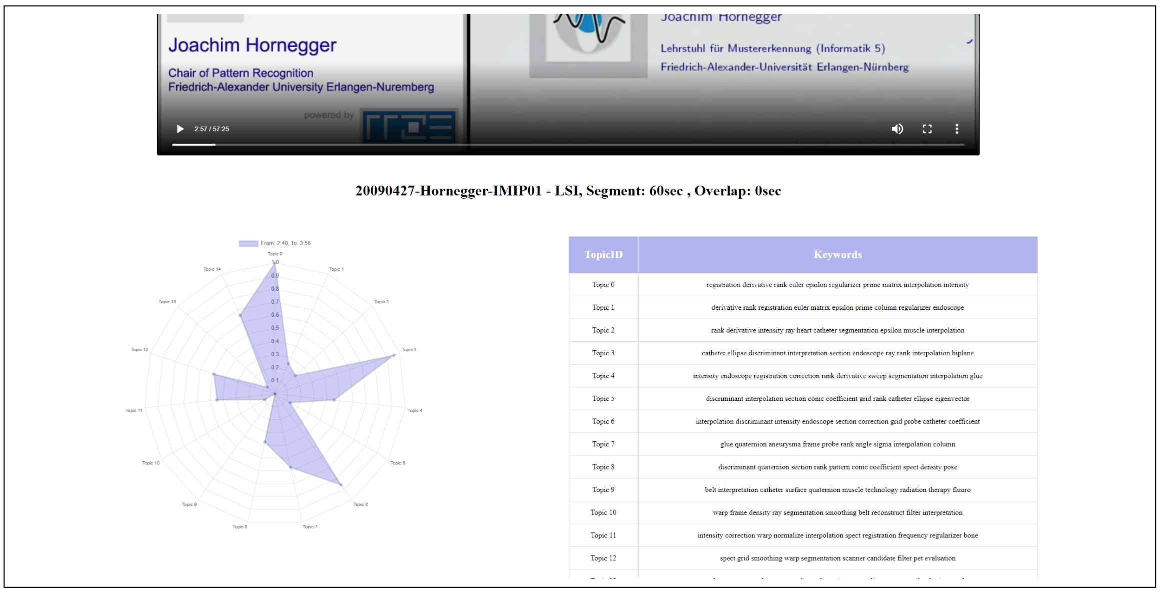


Figure 3.6.: Landing Page

After selecting the lecture and the model, the page re-loads with a video of the lecture. Afterward, the app will check which model and thus which keywords should be loaded. The keywords will be stored in a table on the right side of the page. Next, the file with the topic distributions is loaded. It's a mixture of the lecture name, the model type, and the segment and overlap times.



20090427-Hornegger-IMIP01 - LSI, Segment: 60sec , Overlap: 0sec

TopicID	Keywords
Topic 0	registration derivative rank euler epsilon regularizer prime matrix interpolation intensity
Topic 1	derivative rank registration euler matrix epsilon prime column regularizer endoscope
Topic 2	rank derivative intensity ray heart catheter segmentation epsilon muscle interpolation
Topic 3	catheter ellipse discriminant interpretation section endoscope ray rank interpolation biplane
Topic 4	intensity endoscope registration correction rank derivative sweep segmentation interpolation glue
Topic 5	discriminant interpolation section conic coefficient grid rank catheter ellipse eigenvector
Topic 6	interpolation discriminant intensity endoscope section correction grid probe catheter coefficient
Topic 7	glue quaternion aneurysma frame probe rank angle sigma interpolation column
Topic 8	discriminant quaternion section rank pattern conic coefficient spect density pose
Topic 9	belt interpretation catheter surface quaternion muscle technology radiation therapy fluoro
Topic 10	warp frame density ray segmentation smoothing belt reconstruct filter interpretation
Topic 11	intensity correction warp normalize interpolation spect registration frequency regularizer bone
Topic 12	spect grid smoothing warp segmentation scanner candidate filter pet evaluation

Figure 3.7.: Section of the WebApp

Every 1.5 seconds, a method checks the current position of the video. This position is used to get the current topic distribution. This can be from 0 to 2 different time segments. These segments will be visualized in a radar chart, created with ChartJs[char]. It is placed on the left side of the page. The number of variables (topics) is adapted to these of the model. At the bottom of the page, another lecture or a different model can be selected.
Figure 3.8 shows two examples of these radar charts. The first one has 15 different topics and shows two segments, 5:00-6:26 and 5:59-7:25. This means it has the segment size 90s and an overlap of 30s. It is reasonable that the topic distributions spike for some of the same topics because both segments share roughly a third with the other. Also, this is a LsiModel, where the values are normalized and thus ranging from 0 to 1. The second chart has only five different topics and just shows one segment.

3.7. TopicTiling

Compared to the fixed time period segmentation, **TopicTiling** is an algorithm that tries to find spots for segmenting the documents. The possible segmentation points are set when a subtopic shift occurs. Accordingly, the result should be multiple segments where each one

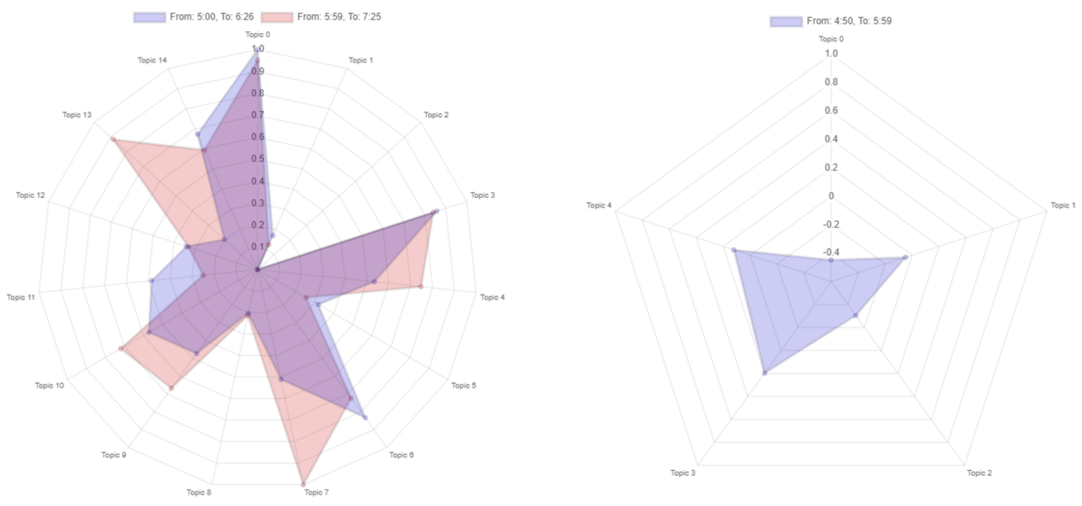


Figure 3.8.: Radar Charts

can be assigned to one topic. Since gensim doesn't implement this method, this project uses the Java code from the creators of the technique [Ried 12]. As a prerequisite, it needs a topic model with a fixed data format, from JGibbsLDA(or GibbsLDA++), which is a Latent Dirichlet Allocation implementation. Therefore LDA will be described shortly.

3.7.1. Latent Dirichlet Allocation

Latent Dirichlet Allocation is another technique for topic modeling. It was first introduced in 2003[Blei 03]. Since then, it became the most popular method for topic modeling. LDA presupposes that each document is a mixture of different topics, like topics are mixed from words. It tries to keep the number of topics in a document and the number of words in a topic low. This is accomplished with the help of the **Dirichlet prior** distribution.[Albr 20] The pseudo-code for LDA with the chosen number of topics k would look like[Darl 01]:

Algorithm 2 Latent Dirichlet Allocation

```
1: for $k_i = 1, 2 \dots k$ do
2:     $\phi^{(k_i)} \sim Dirichlet(\beta)$
3: end for
4: for each document $d_i$ in Corpus $A$ do
5:     $\theta_d \sim Dirichlet(\alpha)$
6:     for Each word $w_i \in d_i$ do
7:         Choose a topic $K_{wi} \sim Multinomial(\theta_{d_i})$
8:         Choose a word $w_i \sim Multinomial(\phi^{(K_{wi})})$
9:     end for
10: end for
```

α and β are hyperparamter for tuning the model. What the algorithm basically does, is it assigns a random topic to each word in each document. Then it iterates through each document and calculates θ_d, the topic distribution for this document (according to the word-topic assignment). Afterward, for each word w in the document, the probability of being assigned to the topic $\theta_{1\ldots k}$ is calculated. This results in a new distribution. Repeating this process eventually converges into a stable status.

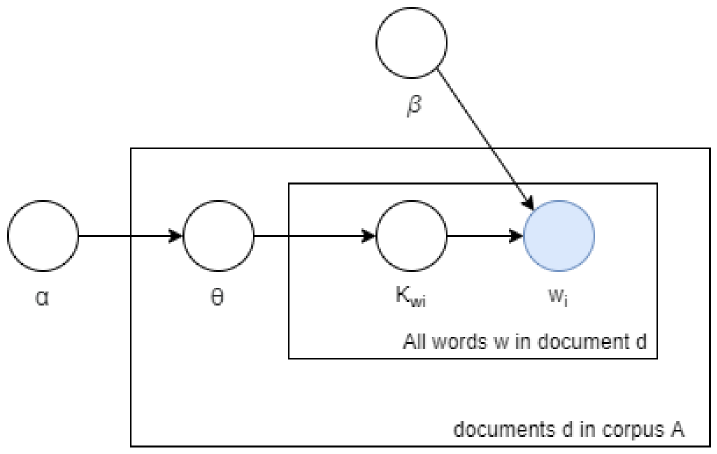


Figure 3.9.: Graphical representation of LDA [Blei 03]

Figure 3.9 shows a graphical representation of LDA. All words in a document w_i are controlled by their topic assignment K_{wi}. The whole document is influenced by the topic distribution of the entire document θ. The "plate" for each word in a document w_i is marked blue because it's the only visible feature; the rest is latent.

3.7.2. Method of TopicTiling

TopicTiling is a method for segmenting text into topically similar parts. The method has some similarities to the in 1997 presented technique TextTiling[Hear 97]. One of the main improvements of TopicTiling is the use of a pre-trained topic model. It could be any method, e.g., LSI or NMF, but in this case and the original paper, LDA is used. Each word of the document is assigned to a topic through the provided topic model. This reduces the computation because the word-space is transformed into the topic space.

Figure 3.10 visualizes the method. TopicTiling uses sentences as the smallest part. Between each sentence, there is a position p. For each position p_i a coherence score C_p is calculated. The number of sentences used for the computation is determined by the window size s. It specifies the number of sentences before and after position p_i, that make up a block each. A vector of size k, the number of topics, can represent each block. The values of the vectors

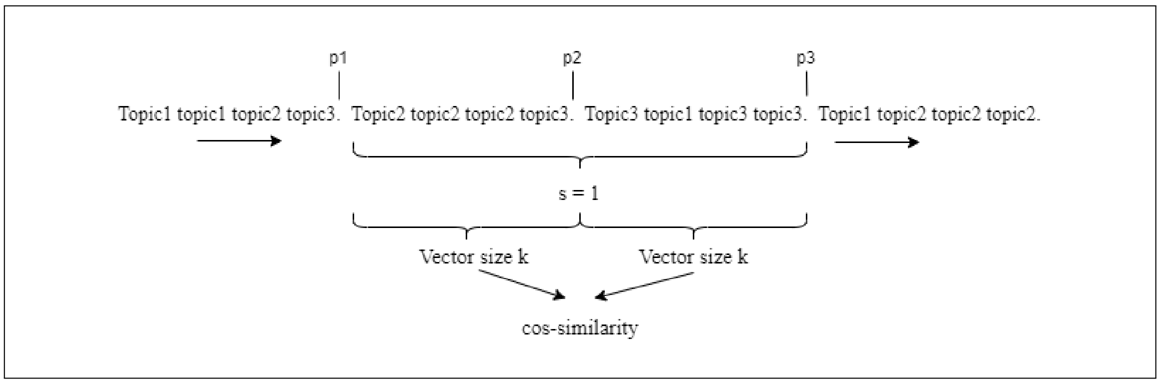

p1
p2
p3
Topic1 topic1 topic2 topic3. Topic2 topic2 topic2 topic3. Topic3 topic1 topic3 topic3. Topic1 topic2 topic2 topic2.
s = 1
Vector size k
Vector size k
cos-similarity

Figure 3.10.: Visualization of TopicTiling

are the appearances of a topic in the according block. Coherence in this case, isn't the whole pipeline like the one described in chapter 3.4, rather just the cosine similarity.

$$C_p = cos(\psi) = \frac{\overrightarrow{b_1} \cdot \overrightarrow{b_2}}{||\overrightarrow{b_1}|| \cdot ||\overrightarrow{b_2}||} \tag{3.20}$$

Coherence values close to 0 means that the blocks have a small relatedness, while values close to 1 represent a strong connectivity. The values can't be negative, since all values of the input vectors are positiv. For each minimum of the coherence values, a depth score d_p is calculated.

$$d_p = \frac{1}{2}(hl(p) - C_p + hr(p) - C_p) \tag{3.21}$$

The depth score measures deepness of a minimum. It does this with the functions $hl(p)$ and $hr(p)$. These iterate to the left and to the right, until the coherence score decreases. The highest values are then returned. If the number of segments is n, then the n-1 highest depth scores are used to locate the segmentation positions.[Ried 12]

3.7.3. Implementation

Using TopicTiling is a three step process: data preparation, topic modeling and applying the method itself. For data preparation, the same techniques from chapter 3.1 are used inside the jupyter notebook. As training data for LDA, we split the data into two separate files, one for the IMIP and one for the PA lectures. Each contains the tokens of all lectures of the course. The files which are used for the TopicTiling, need punctuation. Therefore, we add a full stop (".") after every line of the original transcript files.
The TopicTiling implementation from the creator of the method, M. Riedl, is a Java implementation that requires LDA with either JGibbsLDA or GibbsLDA++. We use the latter, since the C++ binary provides a better performance than the other. Both GibbsLDA++ and the TopicTiling ".jar"-file are wrapped into a bash script, to make the execution reproducible, shown in listing 3.14.

```
1  for folder in ./gibbslda_training_documents/*; do
2      GibbsLDA++-0.2/src/lda -est  -niters 2000 -ntopics 10 \\
3                      -twords 10 -dfile $(echo "$folder")/training_data.txt;
4  done
5
6  cd topictiling_v1.0;
7  for folder in ../topictiling_documents/(IMIP|PA)/*; do
8      sh topictiling.sh  -fp 'tokens.txt' -fd $(echo "$folder") \\
9      -tmd ../gibbslda_training_documents/(IMIP|PA) -tmn "model-final" \\
10     -out $(echo "$folder")/topictiling.txt;
11 done
```

Listing 3.14: bash script for LDA and TopicTiling

Lines 1-4 create the LDA models for the IMIP and the PA courses. Since GibbsLDA is used as a means to an end, the number of topics is just set to 10. The coherence scores for the other models (NMF and LSI) were quite around this number. Lines 6-11 execute TopicTiling on the given documents in the given folder. There are all lectures individually and also the data for the experiment in chapter 5.3.

Chapter 4.

Software architecture

Creating the models inside a jupyter notebook is acceptable for development, but it is not sustainable in a production environment. Each time a new lecture would be added, the functions need to be triggered manually. To avoid this, the functionality is provided through a REST API. The app uses **Flask**, a framework for creating RESTful applications in python [Flas]. As a database, **MongoDB** was chosen. It is a document-based NoSQL database, where documents are stored in BSON format. BSON is an addition to JSON but still works flawlessly with dicts in python.

4.1. Adding lectures

Figure 4.1 demonstrates the workflow of adding a lecture. It's based on 4 steps:

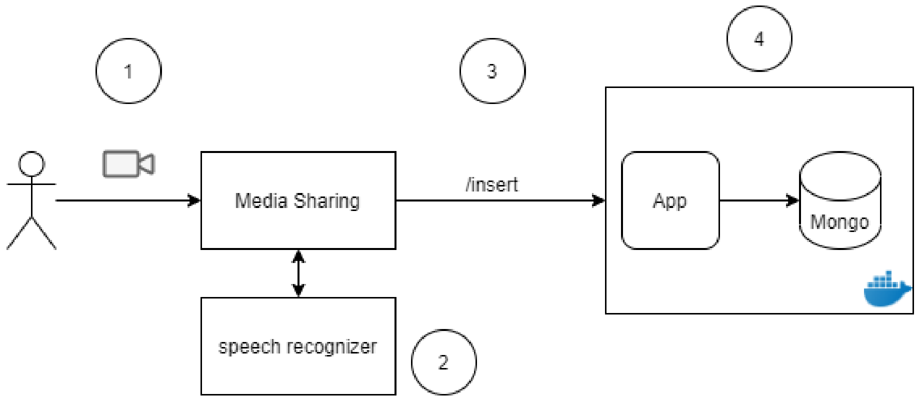


Figure 4.1.: Insert a lecture

- Step 1.) The user (probably the lecturer) uploads the lecture to the media sharing platform. This is all he has to do.
- Step 2.) The media sharing platform creates the transcripts of the lecture

- Step 3.) Through a POST request to the python app, the lecture will be added to the database
- Step 4.) If not configured otherwise, adding a new lecture will create a re-creation for the topic models for this course

The endpoint which needs to be triggerd is "POST /insert". It has 5 parameters, "lecture", "lecture_key", "texts", "lang" and "update". "Lecture" is simply the name of a specific lecture. For the data, it could be e.g., "20090427-Hornegger-IMIP01". The "lecture_key" has the function to link all lectures of the same course, e.g., "IMIP". If multiple courses are added to the platform, it will make sense to have a naming convention for the key, like "<lecturer>_<course>_<semester>". The language can be changed with the "lang" parameter. The default is value is 'en', which does not need to be explicitly specified. An alternative would be 'ger' which changes the tokenizer to a german language model. The update parameter controls if a new model should be created or not. This can be useful if a batch of lectures will be added, and only for the last lecture should the model be created or updated. Otherwise, it would trigger a model creation every lecture, which is time and energy-consuming, and useless because the model would get overwritten by the next model. "Texts" is the parameter that actually holds the text data of the lecture. It is a list containing the parts. Each of them is itself a JSON object with 3 parameters "time_from", "time_to" and "text".

```
 1  >>> import requests
 2  >>> f = open("LMELectures/trl/20090427-Hornegger-IMIP01.trl" ,r)
 3  >>> parts = []
 4  >>> for row in f:
 5  >>>     #extract time period and text
 6  >>>     [...]
 7  >>>     parts.append({'time_from': time_from, 'time_to': time_to,
 8  >>>                   'text': row})
 9  >>> data = {'lecture': '20090427-Hornegger-IMIP01',
10  >>>         'lecture_key': 'IMIP',
11  >>>         'texts': parts,
12  >>>         'update': True}
13  >>> resp = requests.post(url= 'http://localhost:5000/insert',json=data)
14  >>> print(resp.text)
15  ' Lecture was added '
```

Listing 4.1: POST /insert

Listing 4.1 shows an example request. It is created with python, since in the jupyter notebook, functions to read the files were already implemented. Lines 2-8 extracts the text parts and their corresponding timestamps. Lines 9-12 prepare the data for the request. "Update" is the only optional parameter, so it could also be removed. Line 13 makes the actual request. In this example, the app is running on localhost:5000. If the request was successful, the response would be a simple text.

The first thing the app does once receiving the request is to check if the lecture already exists. This would cause an error as a response. If not, each part of the lecture will be tokenized individually and then also put together as a whole token list. Afterward, it will be saved in the database. Listing 4.2 shows how a lecture looks in the database. MongoDB automatically sets the "_id" field, which is used as the primary key of this object.

```
{
    "_id" : ObjectId("60730660c4bf7e786a774d45"),
    "lecture" : "20090427-Hornegger-IMIP01",
    "lecture_key" : "IMIP",
    "text" : "first lecture on ...",
    "tokens" : ["lecture", "image", "processing", ...],
    "parts" : [{
        "text" : " first lecture on ...",
        "tokens" : [ "lecture", "image", ... ],
        "time_from" : 4.36,
        "time_to" : 25.3
        },{
        "text" : " the summer semester...",
        "tokens" : [ "summer", "semester", ... ],
        "time_from" : 25.96,
        "time_to" : 35.55
        } ,...]
}
```

Listing 4.2: lecture in database

If the update parameter wasn't false, a new model for the lectures of the provided "lecture_key" is created. As training data, all lectures with the same key are extracted out of the database. Each token list represents one document. The models are created the same way as in the development part, by trying different numbers of topics and choosing the highest coherence score. The app sets training data = production data, so only one NMF and one LsiModel are needed for each lecture_key. This makes it also possible to have topics for a lecture_key, where only one lecture was uploaded. The used corpus and the keywords are stored in the database. Since gensim provides the function to save models to a file, the app stores them in this way. Otherwise, a way to serialize them would be needed.

4.2. Getting the data

Figure 4.2 demonstrates how the web page can load the topic distribution data. This process also contains four steps:

- Step 1.) The user loads the web page of the media sharing portal. This includes the video of the lecture
- Step 2.) A GET request to the endpoint "/lectures" can verify if the lecture exists in the app. This step is optional.
- Step 3.) For a chosen model, the keywords can be requested through a GET request to "/keywords".
- Step 4.) The topic distribution for each segment can be requested via the "/topicdist" endpoint. This can be executed multiple times if e.g., the segment size is changed by the user.

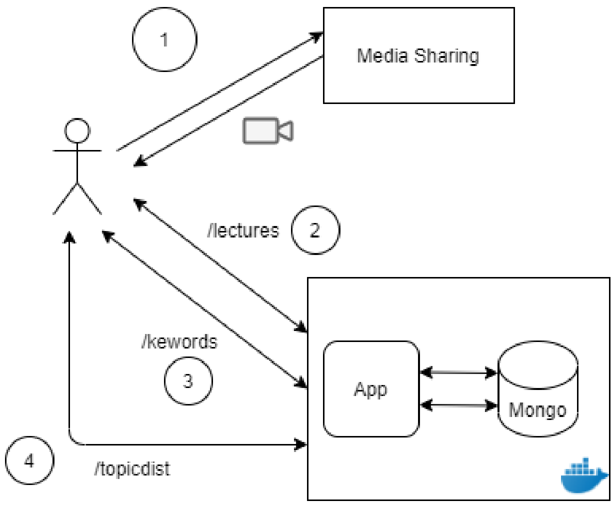


Figure 4.2.: Getting the topic distribution

```
>>> base_url = 'http://localhost:5000
>>>              /keywords?lecture_key={0}&model_type={1}'
>>> resp = request.get(base_url.format('IMIP', 'nmf')
>>> print(resp.text)
{"result": {
    "0": "epsilon intensity histogram ... ",
    "1": "ellipse catheter discriminant ... ",
    ...
}
```

Listing 4.3: GET /keywords

The "/keyword" endpoint accepts two parameters, "lecture_key" and "model_type". As already mentioned, only a single NMF and a single LsiModel were trained for each lecture_key. Either "nmf" or "lsi" is accepted as the model parameter. The "/topicsdist" endpoint accepts four parameters, "lecture", "model", "seg_sec" and "overlap". Since the text of a specific lecture should be segmented and applied to a model, it would not be apparent if the key would not be provided. Another benefit of using a REST API instead of static files, is that the "segm_sec" isn't fixed. This means it can be choosen as wanted. Same is valid for the overlap.

```
>>> base_url = 'http://localhost:5000/topicdist
>>>             ?lecture={0}&model={1}&seg_sec={2}&overlap={3}'
>>> res = request.get(base_url.format('20090721-Hornegger-IMIP23',
>>>                                   'nmf', 90,30)
>>> print(res.text)
{"result": [
    {
      "time_from": 4.55,
      "time_to": 93.24,
      "distributions:{
          "0": 0.2953377215798021,
          "1": 0.06905712887174008,
          "2": 0.6356051495484578,...}
    },...]}
```

Listing 4.4: GET /topicdist

4.3. Deployment

The app and the database can be deployed as **Docker** containers. Docker makes it possible to run the same app on every platform, similar to a VM. But instead of a complete operating system, containers use separate namespaces on the current operating system. This means that each app can run sealed off but can still share some of the functions of the underlying system. Arising overhead is thus minimized.[Dock] One prerequisite for such a container is a Dockerfile. It describes how the container is built. Usually, a pre-existing Dockerfile of an operating system or an application is used as a base. All the code files are copied into the container, and additionally, some other steps are executed, like downloading dependencies. The last step is a command, which starts the application and thereby the application. Listing 4.5 shows a simplified version of the Dockerfile for the app.

- Line 1 defines a python container as the base, which means python and additional tools like pip are already installed.

- Line 2 copies the code into the container.
- Lines 5-6 downloads the dependencies.
- Lines 8-9 starts the python application

```
1 FROM python:3.9.2-buster
2
3 COPY ./ /app
4 WORKDIR /app
5 RUN pip3 install -r requirements.txt
6 RUN python3 -m spacy download en_core_web_trf
7
8 ENTRYPOINT ["python"]
9 CMD ["/app/topic_seg.py"]
```

Listing 4.5: Dockerfile

MongoDB offers a docker image that can be used directly. These two images can be combined to a docker-compose file, configuring how the containers are started. Listing 4.6 shows the config file. Each container can have different configs, like exposed ports (lines 11-12), or folders from the operating system that should be mounted into the container to make data persistent (lines 15-16).

```
 1 version: '3'
 2 services:
 3   mongodb:
 4     image: mongo
 5     container_name: mongodb
 6
 7   pymongo:
 8     build: ./python_module
 9     hostname: pymongo
10     container_name: pymongo
11     ports:
12       - 5000:5000
13     depends_on:
14       - mongodb
15     volumes:
16       - /data/saved_models:/app/saved_models
```

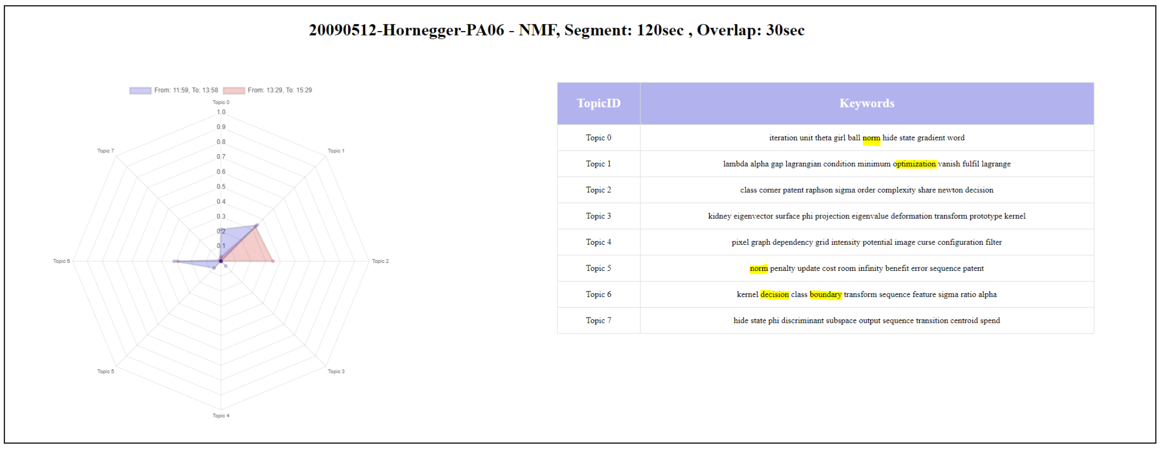
Listing 4.6: docker-compose.yml

Chapter 5.

Experiments

5.1. Comparision on "Pattern Analysis 06"

For the lecture "Pattern Analysis 06", six students manually noted what the topics of the lecture were, rated from 1 (main topics) to 6 (mentioned but in the focus). Some of the topics overlap, while some are unique. This shows that even humans sometimes have difficulties of mapping topics to text/speech. Nevertheless, the goal is to evaluate if the models can extract similar topics as humans would do. For this evaluation, the WebApp with the visualization is used. The models are trained on all "Pattern Analysis" lectures, except number 06.



20090512-Hornegger-PA06 - NMF, Segment: 120sec , Overlap: 30sec

TopicID	Keywords
Topic 0	iteration unit theta girl ball norm hide state gradient word
Topic 1	lambda alpha gap lagrangian condition minimum optimization vanish fulfil lagrange
Topic 2	class corner patent raphson sigma order complexity share newton decision
Topic 3	kidney eigenvector surface phi projection eigenvalue deformation transform prototype kernel
Topic 4	pixel graph dependency grid intensity potential image curse configuration filter
Topic 5	norm penalty update cost room infinity benefit error sequence patent
Topic 6	kernel decision class boundary transform sequence feature sigma ratio alpha
Topic 7	hide state phi discriminant subspace output sequence transition centroid spend

Figure 5.1.: NMF Model of PA06

One topic that three students rated with a 1 is "decision boundary". Looking at the topics of the NMF model, topic 6 has both keywords among the most impactful. Throughout the whole lecture, this topic has quite often a distribution between 0.3 and 0.5 (figure 5.1). Another topic that has a high appearance in the students' notes was "norm" (5 students mentioned it). This keyword can be found in 3 different topics and can be found inside the segments. "decision", "boundary", "norm", and other topic keywords mentioned by the students can be found in all models. However, there are also topics that the students rated high, that cannot be found inside the topic keyword. An example would be "linear

regression".These words may influence the topics but are not inside the ten most impactful words per topic. This is the case for all models, the NMF and LSA model trained inside the jupyter notebook without PA06, and the model created with the flask app, where the lecture was among the training data. Some keywords appear, while others do not.
Conclusion: The models can provide a good overview of the lectures but are far from perfect. Many important topics are found, even if the model is trained on the whole course where the focus of topics can differ. Some topics can go under.

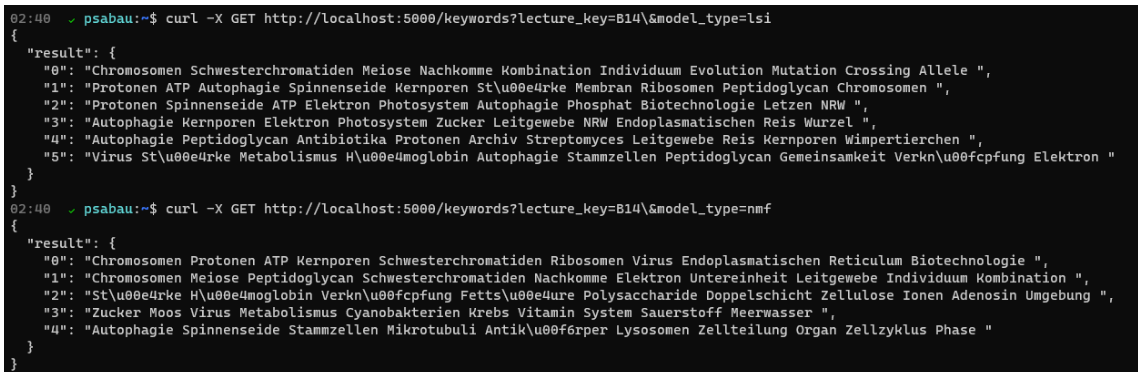
5.2. Biology courses

The two main differences between the LMELectures and the four biology courses are the different data format and the German language. To deal with the first difference, the method for extracting metadata changes a bit. We search with a regular expression for the time stamp at the end of the line, grab the hours, minutes, and seconds and convert them to only seconds.

```
for i in range(len(files)):
    f = open(files[i],"r", encoding='UTF-8')
    parts_lec = []
    time_from_old = 0
    for row in f:
        regex_time = r"(.*) #(\d*):(\d*):(\d*)-(.*)"
        parts = re.search(regex_time,text_row)
        time_to= (int(parts.groups()[1])*60*60)+
                 (int(parts.groups()[2])*60)+
                 int(parts.groups()[3])
        text = parts.groups()[0]
        parts_lec.append({'time_from': time_from_old,
                          'time_to': time_to, 'text': text})
        time_from_old = time_to
    f.close()
    data = {'lecture':lecture,'lecture_key':lecture_key,
            'lang':'ger','texts':parts_lec}
    x = requests.post(url= 'http://localhost:5000/insert',json= data)
```

Listing 5.1: Adding biology lectures

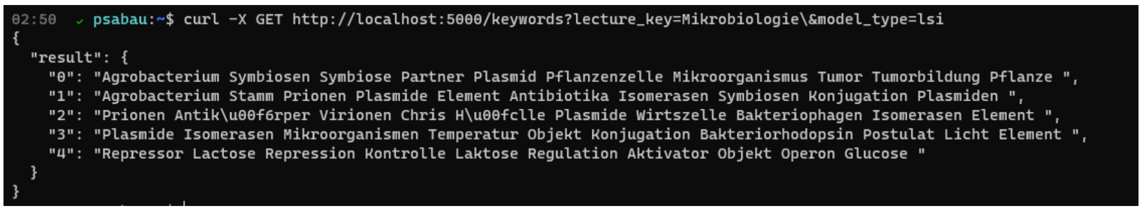
The second difference, the german language, is also not problematic. We adjusted the preprocessing in the flask app by adding a german language model to spaCy. The "/insert" endpoint supports the parameter "lang=ger", which changes the tokenization to the german variant. Other techniques, like the model creation and coherence pipeline, remain the same.



```
02:40  ✓ psabau:~$ curl -X GET http://localhost:5000/keywords?lecture_key=B14\&model_type=lsi
{
  "result": {
    "0": "Chromosomen Schwesterchromatiden Meiose Nachkomme Kombination Individuum Evolution Mutation Crossing Allele ",
    "1": "Protonen ATP Autophagie Spinnenseide Kernporen St\u00e4rke Membran Ribosomen Peptidoglycan Chromosomen ",
    "2": "Protonen Spinnenseide ATP Elektron Photosystem Autophagie Phosphat Biotechnologie Letzen NRW ",
    "3": "Autophagie Kernporen Elektron Photosystem Zucker Leitgewebe NRW Endoplasmatischen Reis Wurzel ",
    "4": "Autophagie Peptidoglycan Antibiotika Protonen Archiv Streptomyces Leitgewebe Reis Kernporen Wimpertierchen ",
    "5": "Virus St\u00e4rke Metabolismus H\u00e4moglobin Autophagie Stammzellen Peptidoglycan Gemeinsamkeit Verkn\u00fcpfung Elektron "
  }
}
02:40  ✓ psabau:~$ curl -X GET http://localhost:5000/keywords?lecture_key=B14\&model_type=nmf
{
  "result": {
    "0": "Chromosomen Protonen ATP Kernporen Schwesterchromatiden Ribosomen Virus Endoplasmatischen Reticulum Biotechnologie ",
    "1": "Chromosomen Meiose Peptidoglycan Schwesterchromatiden Nachkomme Elektron Untereinheit Leitgewebe Individuum Kombination ",
    "2": "St\u00e4rke H\u00e4moglobin Verkn\u00fcpfung Fetts\u00e4ure Polysaccharide Doppelschicht Zellulose Ionen Adenosin Umgebung ",
    "3": "Zucker Moos Virus Metabolismus Cyanobakterien Krebs Vitamin System Sauerstoff Meerwasser ",
    "4": "Autophagie Spinnenseide Stammzellen Mikrotubuli Antik\u00f6rper Lysosomen Zellteilung Organ Zellzyklus Phase "
  }
}
```

Figure 5.2.: B14 keywords

Though a simple request, we can see the different keywords for the courses. Figure 5.2 shows the keywords for the "B14" course. The course is mainly about biochemical processes inside the cells. Therefore, keywords like "Chromosomen" (eng: chromosomes) seem reasonable. In general, the keywords seem to match the original courses.



```
02:50  ✓ psabau:~$ curl -X GET http://localhost:5000/keywords?lecture_key=Mikrobiologie\&model_type=lsi
{
  "result": {
    "0": "Agrobacterium Symbiosen Symbiose Partner Plasmid Pflanzenzelle Mikroorganismus Tumor Tumorbildung Pflanze ",
    "1": "Agrobacterium Stamm Prionen Plasmide Element Antibiotika Isomerasen Symbiosen Konjugation Plasmiden ",
    "2": "Prionen Antik\u00f6rper Virionen Chris H\u00fclle Plasmide Wirtszelle Bakteriophagen Isomerasen Element ",
    "3": "Plasmide Isomerasen Mikroorganismen Temperatur Objekt Konjugation Bakteriorhodopsin Postulat Licht Element ",
    "4": "Repressor Lactose Repression Kontrolle Laktose Regulation Aktivator Objekt Operon Glucose "
  }
}
```

Figure 5.3.: Mikrobiologie keywords

Another example is shown in figure 5.3. The course "Mikrobiologie" (eng: microbiology) discusses subjects like viruses and other unicellular life forms. Keywords like "Agrobacterium", "Plasmid" or "Pflanzenzelle" (eng: plant cell) fit quite well. But the keywords show another problem. Since German is a much more complicated language than English, many methods do not work as well in german. Topic 0 has the keywords "Symbiosen" and "Symbiose", a singular and plural form of the same noun. Lemmatization should have fixed the issue, but it did not. The model still provides value because other keywords approximate the topics of the lecture adequately.

5.3. Detect border of chained lectures

The last experiment appraises if TopicTiling would detect the border of lectures. Each course has 18 lectures, so for this experiment, four lectures (roughly 29 %) in the middle of the semester are selected. Therefore, the IMIP lectures 10, 12, 13, and 14 as well as the PA lectures 08, 09, 13, and 14 are chained together. This is accomplished by attaching the individual files (created in chapter 3.7.3) to each other. Running the TopicTiling algorithm, returns a xml-file for each course. It is structered as follows:

```
<documents>
  <document>
    <documentName>tokens.txt</documentName>
    <segments>
      <segment>
        <depthScore>0.0</depthScore>
        <text>
          welcome tuesday ...
        </text>
      </segment>
      <segment>
        ...
      </segment>
      ...
    </segments>
  </document>
</documents>
```

Listing 5.2: TopicTiling output

As listing 5.2 shows, no timestamps are included. Therefore, the depthscores are not plotted with their time, but rather the number of segments. By looking up the sentences when a new lecture starts, the segment number where the lectures are chained together can be found out. Figures 5.4 and 5.5 demonstrate the result. The red vertical lines are the lecture borders. The

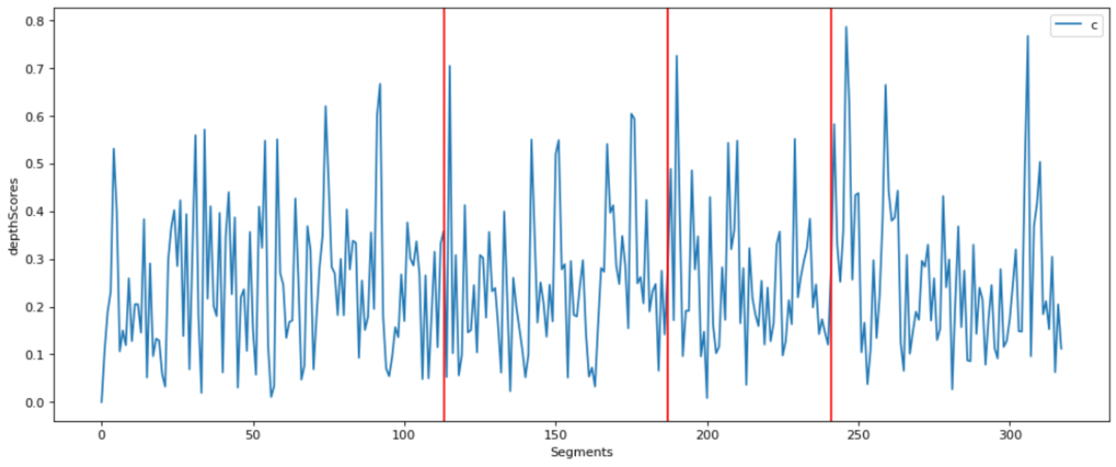


Figure 5.4.: IMIP TopicTiling

IMIP lecture borders are near the highest depthscores of the chained document. Therefore, if the number of final segments would be 5, each segment would be one lecture, except that the last lecture would be split into 2 segments. The PA lecture borders are also near some high spikes, but these aren't the highest of the documents. This means, that inside a lecture, a topic-shift is more clear than at the end of the lectures.

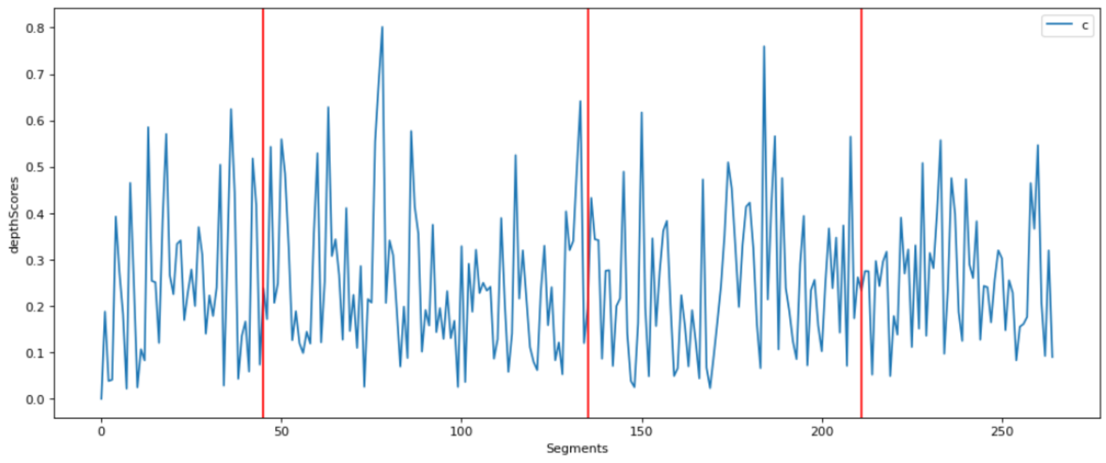


Figure 5.5.: PA TopicTiling

Chapter 6.

Outlook

As demonstrated with the experiments (chapter 5), the developed models can fit the lectures quite well and thus can bring an advantage for learning. However, to ensure that it actually provides a benefit, students would need access to the system. Chapter 4 already described how the flask app, that creates the models, could be integrated into the media-sharing portal. Nevertheless, some additional steps would be necessary. The lecture upload in the media sharing portal would need to trigger the "/insert" endpoint of this app. Also, the frontend needs to be adapted. Currently, a simple HTML file is used for visualizing the topic distribution. This should be converted into a Vue.js component to be fully integrated into the media-sharing portal.
Afterward, feedback should be gathered. This could be realized with an online questionnaire. It would point out if it is really used or if it is only decoration on the site. If the system gathers attention and is liked, then development on it should advance. The preprocessing could be more adapted for the german language with a better lemmatizer. Another possibility is to detect critical technical terms and weigh them more. These improved preparation steps would make the model perform better.
The model itself could be improved by collecting more data. This would be automatically the case if the media-sharing platform would become a standard for the whole school, and thus lecturers would upload their lectures by default. The user should then be able to choose between different options like shown in figure 6.1:

- Option 1: Only the specific lecture itself. This could improve the keyword quality for the lecture itself, but it could also result in a quite weak model since the data is very limited.
- Option 2: All lectures of the same course. Basically, this is the approach that this project took. The results can give a good overview of the content of lectures
- Option 3: Lectures of multiple, different courses. This approach can demonstrate a lecture as a combination of different subjects. A lecture then can be a mixture of physics, mathematics, economics and computer science, etc.

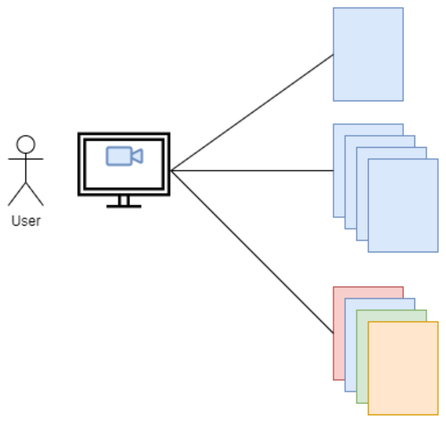


Figure 6.1.: Selection of different models

Topic Segmentation techniques like TopicTiling or even other algorithms should also be integrated into the platform. These techniques can provide a better search if a student wants to learn just about a specific topic. Therefore, the resulting segments should be annotated with a heading that clearly describes the main topic.

Even if humanity beats COVID-19 and students go back to classroom teaching, the benefits of online learning should still be used. Each lecture could be recorded and uploaded, making it simple for each student to repeat subjects or study lectures where they were prevented. This promotes equality for students that learn slower, deepen the substance for everyone and thus, increases education level.

Appendix A.

Calculating SVD

To demonstrate the calculation of the SVD, a 3×2 matrix is used. It follows the example Kirk Baker's paper, one of multiple methods to calculate the SVD. [Bake 05]. The math for higher dimension matrices is the same, but wouldn't fit well as an example. Starting with the matrix

$$A = \begin{bmatrix} 4 & 1 \\ 1 & 3 \\ 2 & 1 \end{bmatrix}$$

Transposing A gives

$$A^{\mathrm{T}} = \begin{bmatrix} 4 & 1 & 2 \\ 1 & 3 & 1 \end{bmatrix}$$

That results in

$$AA^{\mathrm{T}} = \begin{bmatrix} 4 & 1 \\ 1 & 3 \\ 2 & 1 \end{bmatrix} \begin{bmatrix} 4 & 1 & 2 \\ 1 & 3 & 1 \end{bmatrix} = \begin{bmatrix} 17 & 7 & 9 \\ 7 & 10 & 5 \\ 9 & 5 & 5 \end{bmatrix}$$

The next step is to find the eigenvalues of AA^{T} by

$$\begin{bmatrix} 17 & 7 & 9 \\ 7 & 10 & 5 \\ 9 & 5 & 5 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \lambda \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}$$

where λ is one of the eigenvalues and $\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}$ is one of the eigenvectors. This can be solved by setting

$$\begin{vmatrix} (17-\lambda) & 7 & 9 \\ 7 & (10-\lambda) & 5 \\ 9 & 5 & (5-\lambda) \end{vmatrix} = 0$$

This returns the values $\lambda_1 \approx 26.3, \lambda_2 \approx 5.7, \lambda_3 = 0$. Inserting these eigenvalues in the original equations gives the eigenvectors. These are

$$v_1 \approx \begin{bmatrix} 1.77 \\ 1.07 \\ 1 \end{bmatrix}, v_2 \approx \begin{bmatrix} 7.66 \\ -13.64 \\ 1 \end{bmatrix}, v_3 = \begin{bmatrix} -5 \\ -2 \\ 11 \end{bmatrix}$$

After combining the eigenvectors to a matrix, while sorting the eigenvectors in descending order, it gives

$$\begin{bmatrix} 1.77 & 7.66 & -5 \\ 1.07 & (-13.64) & -2 \\ 1 & 1 & 11 \end{bmatrix}$$

Each of these vectors needs to be normalized so that U is an orthogonal matrix. The example below shows the normalization of the vector $\vec{v_3}$

$$\vec{u_3} = \frac{\vec{v_3}}{||\vec{v_3}||} = \frac{[-5,-2,11]}{\sqrt{25+4+121}} = \begin{bmatrix} (-\frac{1}{\sqrt{6}}) \\ (-\frac{\sqrt{\frac{2}{3}}}{5}) \\ \frac{11}{5\sqrt{6}} \end{bmatrix}$$

Doing the same for $\vec{u_1}$ and $\vec{u_2}$ gives [1]:

$$U = \begin{bmatrix} 0.77 & 0.49 & (-0.41) \\ 0.47 & (-0.87) & (-0.16) \\ 0.44 & 0.06 & 0.90 \end{bmatrix}$$

To calculate V, the same steps are applied to $\mathrm{A^TA}$ instead of $\mathrm{AA^T}$:

[1]Note: The values are rounded, so the matrix doesn't look overfilled

$$A^{\mathrm{T}}A = \begin{bmatrix} 4 & 1 & 2 \\ 1 & 3 & 1 \end{bmatrix} \begin{bmatrix} 4 & 1 \\ 1 & 3 \\ 2 & 1 \end{bmatrix} = \begin{bmatrix} 21 & 9 \\ 9 & 11 \end{bmatrix}$$

Solving the following equation to get the eigenvalues:

$$\begin{vmatrix} (21-\lambda) & 9 \\ 9 & (11-\lambda) \end{vmatrix} = 0$$

The resulting eigenvalues $\lambda_1 \approx 26.3$ and $\lambda_2 \approx 5.7$ can be inserted into the succeeding equation:

$$\begin{bmatrix} 21 & 9 \\ 9 & 11 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \lambda \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$

The eigenvectors of these are $v_1 \approx \begin{bmatrix} 1.70 \\ 1 \end{bmatrix}$ and $v_2 \approx \begin{bmatrix} -0.59 \\ 1 \end{bmatrix}$, which can be normalized and converted to the matrix V. Since the transposed matrix is needed, the result is

$$V^{\mathrm{T}} = \begin{bmatrix} 0.86 & 0.51 \\ -0.51 & 0.86 \end{bmatrix}$$

To get the matrix S, take the square roots from the non-zero eigenvalues. These fill up the diagonal of S in descending order. The rest of the matrix size is filled up with 0s. In this example the S is of size 3×2. The eigenvalues are $\lambda_1 \approx 26.3$ and $\lambda_2 \approx 5.7$.

$$S = \begin{bmatrix} \sqrt{26.3} & 0 \\ 0 & \sqrt{5.7} \\ 0 & 0 \end{bmatrix} \approx \begin{bmatrix} 5.13 & 0 \\ 0 & 2.39 \\ 0 & 0 \end{bmatrix}$$

Multiply the three matrices would not result in the original matrix A. This is caused by a sign error, which sometimes needs to be inverted. This is accomplished for either for U or V, which can lead to multiple results. At least one result always exists. [Mark 19]. For this example, v_2 of V is multiplied by -1, thus inverting the signs. Therefore

$$V^{\mathrm{T}}{}_{\text{new}} = \begin{bmatrix} 0.86 & 0.51 \\ 0.51 & -0.86 \end{bmatrix}$$

The check if the result is correct, the three matrices are multiplied again.[2]

$$USV^{\mathrm{T}} = \begin{bmatrix} 0.77 & 0.49 & (-0.41) \\ 0.47 & (-0.87) & (-0.16) \\ 0.44 & 0.06 & 0.90 \end{bmatrix} \begin{bmatrix} 5.13 & 0 \\ 0 & 2.39 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} 0.86 & 0.51 \\ 0.51 & -0.86 \end{bmatrix} \approx$$

$$\approx \begin{bmatrix} 3.95 & 1.17 \\ 2.41 & -2.07 \\ 2.26 & 0.14 \end{bmatrix} \begin{bmatrix} 0.86 & 0.51 \\ 0.51 & -0.86 \end{bmatrix} \approx \begin{bmatrix} 4 & 1 \\ 1 & 3 \\ 2 & 1 \end{bmatrix} = A$$

[2]Note: Since the values are rounded, they do not exactly match. But it is close enough to verify the result

List of Figures

3.1. Visual Reprensentation of LSA 11
3.2. Visual Reprensentation of NMF 15
3.3. Coherence Pipeline [Roed 15] 18
3.4. Coherence Scores 21
3.5. Segmentation pipeline 22
3.6. Landing Page 23
3.7. Section of the WebApp 24
3.8. Radar Charts 25
3.9. Graphical representation of LDA [Blei 03] 26
3.10. Visualization of TopicTiling 27

4.1. Insert a lecture 29
4.2. Getting the topic distribution 32

5.1. NMF Model of PA06 35
5.2. B14 keywords 37
5.3. Mikrobiologie keywords 37
5.4. IMIP TopicTiling 38
5.5. PA TopicTiling 39

6.1. Selection of different models 42

List of Tables

3.1. example of a term-document matrix 8
3.2. term-document matrix used as example for LSA 12

List of Listings

2.1. example of transcript lines, LMELectures 3
2.2. example of transcript lines, biology lectures 4

3.1. spaCy tokenizer 6
3.2. spaCy stopwords 6
3.3. spaCy POS-tagger 6
3.4. spaCy lemmatizer 7
3.5. tokenizer 7
3.6. example tokenizer 8
3.7. gensim tfidf-vector 9
3.8. LsiModel gensim 14
3.9. Apply new document to LsiModel 14
3.10. NMF gensim 17
3.11. Apply new document to NMF-Model 17
3.12. CoherenceModel gensim 20
3.13. evaluation with CoherenceModel 20
3.14. bash script for LDA and TopicTiling 28

4.1. POST /insert 30
4.2. lecture in database 31
4.3. GET /keywords 32
4.4. GET /topicdist 33
4.5. Dockerfile 34
4.6. docker-compose.yml 34

5.1. Adding biology lectures 36
5.2. TopicTiling output 38

Bibliography

[Albr 20] J. Albrecht, S. Ramachandran, and C. Winkler. *Blueprints for Text Analytics Using Python*. O'Reilly Media, Inc., 2020.

[Baez 11] R. Baeza-Yates. *Modern Information Retrival*. Pearson, 2. ed. Ed., 2011.

[Bake 05] K. Baker. "Singular Value Decomposition Tutorial". http://site.iugaza.edu.ps/ahdrouss/files/2010/03/Singular_Value_Decomposition_Tutorial.pdf, 2005. Accessed: 2021-04-03.

[Berr 07] M. W. Berry, M. Browne, A. N. Langville, V. P. Pauca, and R. J. Plemmons. "Algorithms and applications for approximate nonnegative matrix factorization". *Computational Statistics & Data Analysis*, Vol. 52, No. 1, pp. 155–173, 2007.

[Blei 03] D. Blei, A. Ng, and M. Jordan. "Latent Dirichlet Allocation". In: *Journal of Machine Learning Research*, pp. 993–1022, 2003.

[char] "chartJS". https://www.chartjs.org/. Accessed: 2021-04-11.

[Darl 01] W. M. Darling. *A Theoretical and Practical Implementation Tutorial on Topic Modeling and Gibbs Sampling*. University of Guelph, 2001.

[Deer 90] S. Deerwester, S. Dumais, G. Furnas, T. Landauer, and R. Harshman. "Indexing by Latent Semantic Analysis". In: *Journal of the American society for information science*, 1990.

[Der] "Der Präsident der FAU". https://www.fau.de/fau/leitung-und-gremien/leitung/prof-dr-joachim-hornegger-praesident/. Accessed: 2021-03-31.

[Dock] "Docker Internals". http://docker-saigon.github.io/post/Docker-Internals/. Accessed: 2021-04-11.

[Flas] "Flask". https://flask.palletsprojects.com/en/1.1.x/. Accessed: 2021-04-11.

[Garb] "Garbage in, garbage out". https://en.wikipedia.org/wiki/Garbage_in,_garbage_out. Accessed: 2021-03-31.

[glos] "glossary.py". https://github.com/explosion/spaCy/blob/master/spacy/glossary.py. Accessed: 2021-04-01.

[Hear 97] M. A. Hearst. “TextTiling: Segmenting Text into Multi-Paragraph Subtopic Passages”. *Comput. Linguist.*, Vol. 23, No. 1, p. 33–64, 1997.

[Jura 20] D. Jurafsky and J. H. Martin. “Speech and Language Processing”. http://web.stanford.edu/~jurafsky/slp3/, 2020. 3rd ed. draft, December 30, 2020.

[Lane 19] H. Lane, C. Howard, and H. M. Hapke. *Natural Language Processing in Action*. Manning Publications Co., 2019.

[Late] “Latent Semantic Indexing”. https://radimrehurek.com/gensim/models/lsimodel.html. Accessed: 2021-04-05.

[Lee 01] D. Lee and S. Seung. “Algorithms for Non-negative Matrix Factorization”. In: *Advances in Neural Information Processing Systems*, MIT Press, 2001.

[Ling] “Linguistic Features”. https://spacy.io/usage/linguistic-features#pos-tagging. Accessed: 2021-04-01.

[Mark 19] K. Markert and J. Hitschler. “Singular Value Decomposition(SVD)/Singulärwertzerlegung”. https://www.cl.uni-heidelberg.de/courses/ss19/emb/material/svd.pdf, 2019. Accessed: 2021-04-04.

[MIT] “MIT Open Learning Library”. https://openlearning.mit.edu/courses-programs/open-learning-library. Accessed: 2021-03-31.

[Non] “Non-Negative Matrix factorization”. https://radimrehurek.com/gensim/models/nmf.html. Accessed: 2021-04-08.

[OEC] “OEC: Facts about the language”. https://web.archive.org/web/20111226085859/http://oxforddictionaries.com/words/the-oec-facts-about-the-language. Accessed: 2021-04-01.

[Reyn 98] J. C. Reynar. *Topic Segmentation: Algorithms And Applications*. University of Pennsylvania, 1998.

[Ried 12] M. Riedl and C. Biemann. “TopicTiling: A Text Segmentation Algorithm based on LDA”. In: *Proceedings of the Student Research Workshop of the 50th Meeting of the Association for Computational Linguistics*, pp. 37–42, Jeju, Republic of Korea, 2012.

[Ried 13] K. Riedhammer, M. Gropp, T. Bocklet, F.Hönig, E. Nöth, and S. Steidl. “LMELectures: A Multi-Media Corpus of Academic Spoken English”. In: *First Workshop on Speech, Language and Audio in Multimedia (SLAM)*, 2013.

[Roed 15] M. Roeder, A.Both, and A. Hinneburg. "Exploring the Space of Topic Coherence Measures". In: *Proceedings of the Eighth ACM International Conference on Web Search and Data Mining WSDM*, p. 399–408, Association for Computing Machinery, 2015.

[Saba 21] P. Sabau. "Topic Segmentation". `https://github.com/LightningMxQueen/TopicSegmentation`, 2021.

[TF I] "TF-IDF model". `https://radimrehurek.com/gensim/models/tfidfmodel.html`. Accessed: 2021-04-02.

[Zhao 17] R. Zhao and V. Y. F. Tan. "Online Nonnegative Matrix Factorization With Outliers". *IEEE Transactions on Signal Processing*, Vol. 65, No. 3, p. 555–570, Feb 2017.

Glossary

corpus Collection of all documents used.

dictionary List of unique tokens in a corpus.

Docker Container engine.

document Collection of words. This can be a sentence or a whole lecture.

Flask Python framework for creating RESTful applications.

Gensim Python library for Topic Modeling.

spaCy Python library for different NLP tasks.

topic Co-occuring words can form a topic.