Information Retrieval
Assignment 2:
Topic Modelling using Latent Dirichlet Allocation

Daniëlle Jongstra
s0172260

Jens Leysen
s0201907

Johannes Voshol
s0200178

1. Introduction

For the course "Information Retrieval", the goal of the second assignment is to implement a simple topic modelling algorithm. The specific algorithm we will use is Latent Dirichlet Allocation (LDA). We will be implementing this system and document it's capabilities, seeking to learn more about the real-world usage of topic modelling.

The goal of section 2 is to give an overview of LDA and topic modelling in general. Here, we'll explain the probability model of LDA and talk about concepts such as Dirichlet distributions, multinomials, plateau notation and finally we give an overview of the Gibbs Sampling algorithm to approximate this probability model.

In section 3, we will write about the implementation of our topic modelling system, the pre-processing that has been applied to the data set and what frameworks and libraries have been used.

In section 4, we discuss the influence of the different number of topics, possible techniques for finding the optimal amount of topics and we compare our implementation to a library implementation.

Finally, we discuss things we wanted to improve or revisit in section 5.

2. Background

Topic models are a suite of algorithms whose aim is to discover the hidden thematic structure in large archives of documents [1]. One would expect to find related words more often together in a document than words that are not related. The theme of a document is a bag of words that can be described as a mixture of topics, where we are mostly interested in the most significant topics. Furthermore, we want to describe the most significant words that represent such a topic. The goal of topic modeling is to find such a word-topic model that corresponds with your documents in terms of semantics and effectiveness.

2.1. Topic modeling techniques

LDA is just one algorithm in a family of topic modeling techniques. We will briefly go over other popular methods in topic modeling.

Latent Semantic Analysis (LSI) uses singular value decomposition (eq. 1) on the term-frequency matrix to extract features that explain the relation between these terms and documents. A dimensionality reduction can be applied, which leaves us with the most important concepts/features called latent topics. It reduces the dimensionality of our vectors, which is often a problem in machine learning. Using these topics can improve the effectiveness of an information retrieval (IR) system [7]. However, the reduced vectors are not sparse anymore and will be large in memory size. LSI also does not solve the problem where words have multiple meanings (also called polysemy), because the meaning of a word can be represented as the weighed average of meanings in the matrix.

$$X = U\Sigma V^T \tag{1}$$

Probabilistic latent semantic analysis (pLSI) evolved from LSI and is based on a mixture decomposition derived from a latent class model [8]. To simplify, each document consists of a mixture of topics and each topic consists of a collection of words. The multinomial distribution for this model (eq. 2) corresponds directly with the singular value decomposition of the LSI method [11]. This model can be trained using the expectation-maximization algorithm (EM), which will find the best parameters to fit the model. This model is flexible, but is also prone to overfitting, since the parameters pLSA grow linearly with the number of documents. Futhermore, we do not know how to assign the probabilities to new documents entering our corpus, since adding documents require retraining of the model. pLSI is rarely used, as people will turn to LDA if they require a baseline beyond LSI [11].

$$\sum_c P(c)P(d|c)P(w|c) \tag{2}$$

Lastly, lda2vec is a deep learning topic modeling technique that extends the skip-gram model of word2vec and LDA [13]. It is a neural network that tries to predict surrounding context words.

2.2. Overview of LDA

LDA is a generative probabilistic model for collections of discrete data such as text corpora [2] and can be described

as the Bayesian version of pLSI. It allows sets of observations to be explained by unobserved groups that explain why some parts of the data are similar [3].

$$P(\boldsymbol{W}, \boldsymbol{Z}, \boldsymbol{\theta}, \boldsymbol{\phi}|\alpha, \beta) = \prod_{j=1}^{M} P(\theta_j|\alpha) \prod_{i=1}^{K} P(\phi_i|\beta) \prod_{t=1}^{N} P(Z_{j,t}|\theta_j) P(W_{j,t}|\phi_{Z_{j,t}}) \tag{3}$$

Equation 3 show the probabilistic model of LDA. It represents the probability of producing a corpus $\boldsymbol{W}$ with topic-document assignment $\boldsymbol{Z}$ for generated multinomials $\boldsymbol{\theta}$ and $\boldsymbol{\phi}$, given the Dirichlet distributions $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$.

2.3. Dirichlet distributions

A common way to visualize the relation in terms of topics and a documents, is to draw a 2-simplex (Figure 1), where each of the corners represent a topic and the density within this simplex represent the distribution of of documents for these topics. The position of a point in such a simplex represents how close it is related to the corners. In other words, how close the document is related to the topic. The density shows how likely the document can be found there. For a large number of topics K, the relation can not easily be visualized, because it would require a (K-1)-simplex. In LDA, we are interested in the position of documents in relation to the topics. However, we do not know these positions beforehand.

The probability model of LDA is based on Dirichlet distributions, a multivariate probability distribution. The K-dimensional vector $\boldsymbol{\alpha}$ represents a Dirichlet prior of how likely a document has a specific topic. The V-dimensional vector $\boldsymbol{\beta}$ represents a Dirichlet prior of how likely a topic has a specific word. Documents are often about a specific topic (or two) and never about all the topics. The Dirichlet distribution $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ will often resemble the right bottom distribution in Figure 1 $(0.2, 0.2, 0.2)$, where all the values are between 0 and 1.

LDA assumes a Dirichlet prior over the latent topics [14]. That means that LDA assumes that $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ exist, therefore they can be approximated.

2.4. Multinomials

From the Dirichlet distribution $\boldsymbol{\alpha}$, we can create the multinomial $\boldsymbol{\theta}$, where $\theta_{d,k}$ is the probability of topic k occurring in document d. In the generative process, $P(\theta_j|\alpha)$ denotes the chance of a document j generating a certain multinomial for picking topics. For example, if our corpus mostly consists of one specific topic. There is a large chance to create a multinomial favoured towards this one topic.

From the Dirichlet distribution $\boldsymbol{\beta}$, we can create the multinomial $\boldsymbol{\phi}$, where $\phi_{l,w}$ is the probability of word w occurring in topic k. In the generative process, $P(\phi|\beta)$ denotes the chance of a topic k generating a certain multinomial for picking words.

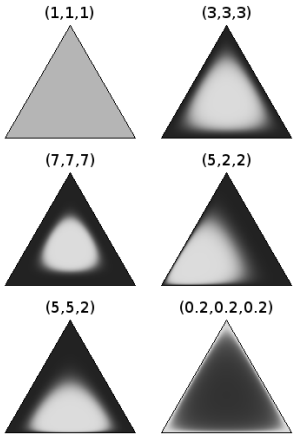

Figure 1. Example of word distributions for three topics [15]. Often a topic distribution looks like (0.2,0.2,0.2), where one topic or a combination of two topic is preferred. It will never be a distribution like (7,7,7), as this would mean almost all our documents are a perfect mix of all the topics.

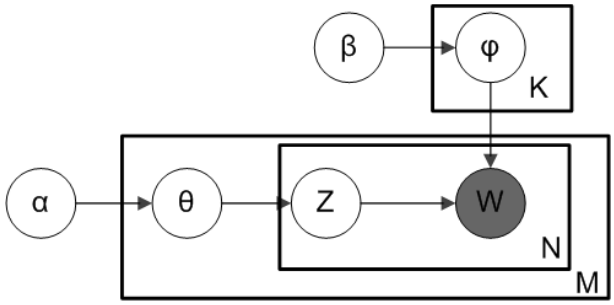

Figure 2. Plate notation for LDA with Dirichlet-distributed topic-word distributions [3].

2.5. Generative process

As we discussed the multinomials in the previous section, we now understand that in equation 3, $P(Z_{j,t}|\theta_j)$ is the probability of a topic being drawn from θ_j. Lastly, $P(W_{j,t}|\phi_{Z_{j,t}})$ is the probability of a word being drawn from $\phi_{Z_{j,t}}$ using the topic we just drew from θ_j.

The plate notation is already being used in pLSI to describe the generative process of creating a document, however this is still very simplistic, with the drawn topic for a word being the only latent variable. The generative process of LDA can also be described with the plate notation as shown in Figure 2.

We will use the plate notation to describe the generative process of creating a corpus. First, for every document a topic multinomial is generated from $\boldsymbol{\alpha}$ and from $\boldsymbol{\beta}$ a word multinomial is generated for each topic. Then for every word in our new document, we draw a topic from this topic multinomial and use this drawn topic to draw a word form the associated word multinomial.

2.6. Gibbs Sampling

Gibbs Sampling is a Markov Chain Monte Carlo algorithm [5]. Monte Carlo methods are algorithms that help obtain a desired value by performing simulations regarding probabilistic choices [10]. Markov Chains are stochastic models where the probability of a next state only depends on the current state of the model [10], as can be seen in Figure 3. The basic idea of Gibbs Sampling is that [10]: "Rather than probabilistically picking the next state all at once, you make a seperate probabilistic choice for each of the k dimensions, where each choice depends on the other k-1 dimensions." It's an algorithm that, among other uses, can be used for sampling dependent variables when it's difficult to sample from their joint distribution. We'll first discuss a simple example before delving deeper into Gibbs Sampling.

2.6.1. Simple Gibbs Sampling. We'll present the basic idea by means of the following example inspired by [4]: Given two dependent variables A and B, which can each assume only two values (a binomial experiment). We seek to draw a sample from a joint distribution P(A,B). Gibbs Sampling will use the conditional distributions: $P(A|B=0)$, $P(A|B=1)$, $P(B|A=0)$, $P(B|A=1)$ and an iterative approach as a way of approximating this joint probability distribution. Instead of sampling A from the joint distribution, we will sample A from either one of the biased distributions $P(A|B=0)$ or $P(A|B=1)$. Similarly, B will be sampled from $P(B|A=0)$, $P(B|A=1)$. These last two steps are combined into an iterative approach:

```
# Pseudo Code
# iterations (int): range, number of iterations
for i in iterations :
    Sample A_i from P(A|B_{i-1})
    Sample B_i from P(B|A_i)
```

This iterative approach is also visualised in Figure 3. Doing this for a large number of iterations, this sampling method will approximate the joint probability distribution of A and B ($P(A,B)$). Ofcourse, this algorithm generalizes to multivariate distributions.

2.6.2. Collapsed Gibbs Sampling. Given a multivariate distribution, with say three three variables A, B, C. The simple Gibbs sampler discussed above would sample from the biased distributions of $P(A|B,C)$, $P(B|A,C)$ and $P(C|A,B)$. A Collapsed Gibbs sampler would integrate out one of these variables. Effectively sampling from a marginal distribution such as $P(A|C)$ where B has been integrated out (in this example) [5].

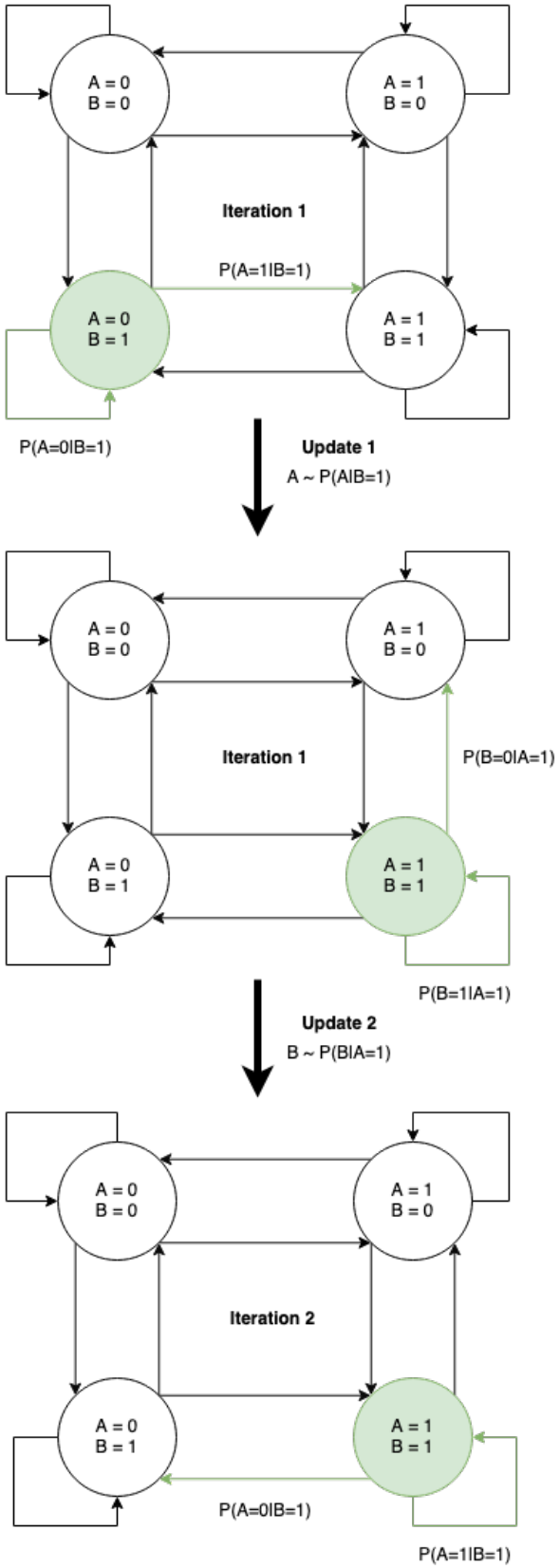

Figure 3. Markov Chain: Gibbs Sampling Example.

2.7. LDA with Collapsed Gibbs Sampling

2.7.1. Why use LDA. The following formula describes a common way of modelling the contribution of a topic to a document [12]: $P(w_i) = \sum_{j=1}^{T} P(w_i|z_i = j)P(z_i = j)$. Here, $P(w_i)$ is the probability of the ith word in a document. $P(w_i|z_i = j)$ is the probability of word w_i under the jth topic, or in different words, it models the words that are important to a specific topic. $P(z_i = j)$ is the probability of choosing a word from topic j in the current document, said differently, it's the prevalence of topics in a certain document. z_i is a latent variable indicating the topic from which the ith word was drawn. According to [12]: "Treating documents as mixtures of probabilistic topics makes it possible to formulate the problem of discovering the set of topics that are used in a collection of documents." So, given D documents, T topics and a vocabulary of W words. We seek to express $P(w|z)$ with a set of T multinomial distributions: $\boldsymbol{\phi}$. Where $P(w|z = j) = \boldsymbol{\phi}^{\boldsymbol{j}}_{\boldsymbol{w}}$. Similarly, we seek to represent $P(z)$ with a set of D multinomial distributions: $\boldsymbol{\theta}$. Where $P(z = j) = \boldsymbol{\theta}^{\boldsymbol{d}}_{\boldsymbol{j}}$. This brings us to our strategy for discovering the topics in a corpus, we want to obtain an estimate of $\boldsymbol{\phi}$ that gives us a high probability for the words that appear in the corpus. This could be done by maximizing $P(\mathbf{w}|\phi, \theta)$, to do this we could use the maximum likelihood estimate (MLE) or maximum a posteriori estimation (MAP) [10]. However, according to [12], these approaches can be slow to converge and have local maxima problems. This encouraged the development of models that make assumptions about the source of $\boldsymbol{\theta}$ [12]. LDA is one such model, it introduces a generative approach (discussed in section 2.5) and makes assumptions about the prior probability distributions of $\boldsymbol{\theta}$ and $\boldsymbol{\phi}$ by means of Dirichlet distributions. The total probability function of the LDA model can be seen in Equation 3. Solving this equation would require maximizing an extremely complex integral, this is where Gibbs Sampling comes to the rescue.

2.7.2. Why use Collapsed Gibbs Sampling. Our custom implementation relies on the mathematical model outlined in [12]. We won't go into detail on why the method outlined in [12] is a valid approximation of LDA. However, we will quickly summarize the main points. The main ideas are as follows [12]:

- We seek to estimate the posterior distribution $P(\mathbf{z}|\mathbf{w})$.
- $\boldsymbol{\theta}$ and $\boldsymbol{\phi}$ are collapsed (integrated out), allowing us to compute $P(\mathbf{w}, \mathbf{z})$.
- The distribution $P(\mathbf{z}|\mathbf{w})$ cannot be computed directly from $P(\mathbf{w}, \mathbf{z})$, this is because it involves evaluating the probability distribution over a large state space.
- To estimate $P(\mathbf{z}|\mathbf{w})$, we use Gibbs Sampling and the full conditional distribution. It's formula can be seen in Figure 4. For more information about each variable we refer to [12], a small overview is given in Table 1.

$$P(z_i = j|\mathbf{z}_{-i}, \mathbf{w}) \propto \frac{n^{(w_i)}_{-i,j} + \beta}{n^{(\cdot)}_{-i,j} + W\beta} \frac{n^{(d_i)}_{-i,j} + \alpha}{n^{(d_i)}_{-i,\cdot} + T\alpha}$$

Figure 4. Full Conditional Distribution.

TABLE 1. VARIABLES LDA AND COLLAPSED GIBBS SAMPLING

Math. Variable	Code Variable	Meaning
n^{w}_{j}	m_w_k	Number of times a word is assigned to a topic
n^{d}_{j}	n_i_k	Number of times a topic occurs in a document
$n^{(d_i)}_{-i}$	n_d	Number of words in document - 1.
$n^{(\cdot)}_{-i,j}$	n_z	Number of assignments to topic minus current.
α	alfa	Hyperparameter
β	beta	Hyperparameter
W	vocabulary_len	Size of Vocabulary
T	topics	Number of Topics

It's important to note that the variables n represent counts, where $n^{(\cdot)}_{-i}$ represents a count that does not include the current assignment of z_i. For example, $n^{(d_i)}_{-i}$ represents the total word count of a document minus the current word. These counts are the only variables needed to calculate the conditional distribution. Intuitively, the first ratio in Figure 4 expresses the probability of w_i under topic j. While the second ratio expresses the probability of topic j in document d [12]. The algorithm continues as follows:

- Variables are initialized.
- Each iteration, the sampler finds a new state for z_i. As can be seen, the only information needed for each iteration is the number of times a topic occurs in a document and the number of times a word is assigned to the topic.
- After enough iterations, the current values for z_i are recorded.

According to [12], this formula results in an easy to implement, memory efficient and competitive algorithm.

3. Implementation

The source code for this project can be found here: **Information_Retrieval_Assignment_2**. The notebooks can be run on Google Colab. Running the notebooks on the larger dataset might not work when using a free Colab account (due to runtime or RAM limitations). All notebooks do run on a Google Colab Pro account. It should be noted that we use Google Drive to store the output from the notebooks (as csv files). So if you want to run the notebooks yourselves, a Google Drive account with sufficient space is required.

3.1. Frameworks

It was chosen to make this assignment using Jupyter Notebook. We created these notebooks on Google Colab,

but they should be able to run on any local Jupyter notebook server. The notebooks can also be run as regular python files with some very limited changes to the code.

3.2. Pre-processing

For the pre-processing steps we relied on existing python libraries, mainly python's Natural Language Tool Kit (nltk). The following pre-processing steps were applied to the news dataset:

- Corpus Creation: Only the content column of the news dataset was used for creating the corpus.
- Imputation: Content that included NaN values was dropped from the dataset.
- Tokenization: The tokenization was done by applying the nltk.word_tokenize function to each document's content.
- Small Words Removal: Tokens smaller than 3 character were removed.
- Stemming and Lemmatization: We used nltk's WordNetLemmatizer and SnowballStemmer classes.
- Stopwords: We removed stopwords, for this we used nltk's stopwords library.

For the stopwords, lemmatization and stemming steps, the English language versions were used.

3.3. Custom Implementation

The custom implementation has some custom utility functions for further processing the tokens, creating a Bag-of-words representation of the data using Python dictionaries and a custom word encoder class. We will discuss the custom functions and class first.

3.3.1. tokens_doc_freq. This function has four parameters:

- data (series): series object holding lists of tokens.
- num_above (int): drop tokens who appear in less than num_above documents.
- num_under (float): drop tokens who appear in more than num_under * amount of documents.
- most_freq (int): return most_freq tokens.

The function returns:

- tokens_doc_freq (dict): dictionary with key = token and value = # documents token appears in.

In short, this function will create an ordered dictionary where the keys are tokens and the value is in how many documents that token appears. As can be seen, parameters can be set to only return tokens that appear in more than num_above documents and less than num_under of the total documents. The most_freq parameter can be set to only return the most_freq amount of tokens.

3.3.2. create_bow. This function has two parameters:

- data (series): series object holding lists of tokens.
- tokens (list): list of tokens to use in bag of words.

The function returns:

- documents (list): bag of words, a list of dicts.

This function will create a bag of words for usage in LDA. It expects a series object holding lists of tokens. With the tokens parameter the user specifies what tokens it wants to keep in the bag of words. You're thus supposed to first select the tokens you want to keep with the function get_freq_tokens (or your own list) and pass it to this function to create the bag-of-words.

3.3.3. WordEncoder. The LDA implementation doesn't work on string tokens, so all tokens first need to be converted to an integer representation. To do this, a custom WordEncoder class was created. This class has two functions encode and decode. They both expect a bag of words in the form of a list of dicts and will return the same list of dicts but with encoded/decoded tokens.

3.3.4. LDA. Our implementation of the LDA algorithm using collapsed Gibbs Sampling was inspired by [9] and was described in section 2.7. The class exists of four functions.

- __init__ takes the encoded documents, the vocabulary and amount of topics that we want as input. It randomly initializes all the necessary variables.
- runLDA takes the amount of iterations to run and generates better topic assignments after each iteration.
- getTopicsPerDocument returns the matrix that contains all the topics that are assigned to a document.
- getWordsPerTopic returns the matrix that contains all the words assigned to a topic.

3.4. Library Implementation

In order to evaluate the result of our LDA implementation, we thought it would be a good idea to also create a topic model using an existing library. The library implementation can be found here: **Information_Retrieval_Assignment_2/LDA-Library.ipynb** This notebook makes use of the gensim.corpora.Dictionary class:

- dictionary.filter_extremes(): for filtering out tokens in the dictionary by their frequency.
- dictionary.doc2bow(): for converting a document into the bag-of-words (BoW) format.

The LDA is done using the gensim.models.LdaMulticore method.

4. Evaluation

We evaluate both the library and the custom implementation on different aspects. The generated files might change by running the LDA implementations again, as the implementations use randomness to assign topics. Except for the evaluation of the memory usage, all the evaluations are made using the small news data set.

Important Note: the evaluation was done on the result of pre-processed data that didn't have any stopword removal. This has since been fixed in the code but not in the csv files or following evaluation sections. Please note that any issues mentioning stopwords (in the topics for example) should now have been resolved.

4.1. Execution Time

Running both the library and the custom implementation with a small data set (NewsArticles, [16]), there is already a significant difference in the execution time of the two. We only take the execution of the LDA itself into account, and not the execution time for creating the bag of words, pre-processing etc. The custom implementation is timed in two parts. The first part is initializing the values, and the second is optimizing these values. The initializing only takes 2 seconds using this data. However, optimizing the topic assignments takes a lot longer with 4 minutes and 12 seconds. These time measurements are for the LDA running with 10 iterations. If the library version runs with the same amount of iterations, it only takes a total of 1 minute and 33 seconds.

4.2. Memory Usage

We did not implement a memory profiler but Google Colab does give estimates of the total RAM usage of each notebook. Running both the library and custom implementation on the large data set (news_dataset.csv), it was found that the custom implementation uses a maximum of 15GB of RAM (rough estimate). The library implementation uses a maximum of 17,8GB of RAM (also a rough estimate).

4.3. Topics Custom

The ten most important words for every topic are saved into an csv-file. For the custom implementation with ten iterations, this is in the file analysis_topics10.csv. In this file, there are already a few topics that seem to consist of words that form a good description. In topic 0, words like russia, russian, foreign and moscow appear, which are clearly connected to each other. Another interesting topic is topic 9, which contains words as donald, trump, white, house and elect. When we keep in mind that the used articles are from between December 2016 and March 2017, it is likely that this topic contains articles about when Donald Trump was president-elect and all the things he was doing to prepare for his presidency. Other topics that make sense are topic 1, containing multiple words about time, topic 7, containing multiple social media, topic 8, containing words about violence, and a few other topics. There are also some topics that do not seem very descriptive of a certain kind of article. For example, topic 18 contains a lot of words that do not add information (like, take, first, last, thing, could).

The topic distribution changes a quite a bit when we go from ten to fifty iterations. The topics for fifty iterations is in analysis_topics50.csv. Here we can see that topic 6 contains the social media like topic 7 did in ten iterations. However, now it is also clear that some European countries, in particular Germany, have something to do with it. This might be around the time that the privacy risks of different social media came to light. Topic 1 in the fifty iterations version seems to be about history and culture. This topic does not seem to appear in the topics of ten iterations. It might be that topics that only contain fewer documents need more iterations to be discovered by LDA. There still are topics about violence and time in the 50 iteration topics, even though the words describing the topics are not exactly the same. A thing to keep in mind is that even though the topics seem to be more clearer when using fifty iterations, there still are some topics that do not immediately make sense. For example topic 7 with words like "change", "much", "like".

There is a notebook analysing the similarity between the topics using ten and fifty iterations (LDA_Analysis.ipynb). Here we can see that the topic in the ten iterations has between 2 and 5 words in common with the topic in the most alike fifty iteration topic, when using only the top ten words. Interesting to notice is that both topic 11, 13 and 18 are most similar to topic 9 (fifty iterations). They have respectively 3, 5 and 4 words in common. This adds to 12 words in common in the top 10. We can conclude from this that the three topics have some of the same words in the top 10 that best describe them. Topic 11 is quite vague, the only words that are descriptive are president, people and court. Topic 13 is also rather vague, but seems to be about how people spend their time. Topic 18 is also vague, but again contains people and time. Topic 9 of fifty iterations seems to have taken all three topic together. It seems more descriptive with the words learn, think, share, like, know and people. All that can be said for sure is that the topic is about people and most likely what they are doing.

4.4. Topics Library

The topics for the library version are generated for two and ten iterations, because the topics generated by the library seemed to be more descriptive then the custom implementation. The top-10 words describing the topics are in analysis_topcs_lib2.csv and analysis_topcs_lib10.csv. The same jupyter notebook that contained the analysis of custom topics (LDA_Analysis.ipynb) also contains the analysis of the library topics. In this analysis, every topic we gotten from using ten iterations is matched to the topic that is most similar that was generated using only two iterations. Very interesting is topic 0, which matches topic 0 using two iterations most closely, but only has one word in common.

This would mean that topic 0 in ten iterations roughly is a newly generated topic. Both topic 16 and 17 (ten iterations) are most similar to topic 16 (two iterations). Topic 16 (two iterations) is about China and Russia, and most likely the development of these countries and the investments made. Topic 16 and 17 (ten iterations) are both still about Russia, China however is in neither of the topics. One topic is about Russian sport, both the world cup and the Olympics. This might be about the discovery of Russian athletes that used doping and the consequences for athletes from Russia. The other topic is about Russia, foreign ministers and Syria. This might be about Russia taking part in the war in Syria. Neither of these topics seem to be closely related to the original topic 16. The topics among the ten iteration topics talking about China can be found using the analysing jupyter notebook. In this we see that topic 3, 7, 12 and 13 contain china in the top-10 words. Both 3 and 7 are about the chinese economy. Topic 12 is about Chinese immigrants and topic 13 about both Korea's and the nuclear program of most likely north-Korea. China cannot immediately be placed in this context. Taking this in consideration, topic 3 and 7 seem to most closely resemble the Chinese part of topic 16 of the two iteration topics.

4.5. Library vs. Custom Topics

For using the same amount of iterations, the library implementation appears to have less topics that are unclear than the custom implementation. None of the topics in the library topics seem to be very randomly picked together, while at least four topics in the custom implementation seem to be quite strange. To get good topics, it is clear that the custom implementation needs more iterations than the library implementation does. In the jupyter notebook, we can see that comparing the topics of the two-iteration library implementation to both the ten and fifty-iteration custom implementation gives quite an evenly distributed amount of words that are in common between the topic that is most alike. It is also clear however, that the same topic is often detected as the most alike topic. This might mean that in the library topics with two iteration, there are words that appear in a lot of topics. That would not be a good situation, because then some topics that have less articles about it might not have been detected. It could also mean that we have too many topics for the data set we are using. When we inspect the results of analysing the ten iteration library version, it seems like the topics are more alike the custom topics generated using fifty iterations then the ones using ten iterations. The amount of words in common are mostly 4 or 5, while for ten iterations it is mostly 2 or 3.

4.6. Representative Documents

The 100 most representative documents can be found in the four csv-files starting with topic_document_rank. There is a file for both the library implementation and the custom implementation, combined with the amount of iterations previously discussed. Discussing the differences between the type of implementation and the amount of iterations is in this case not useful, as we have already seen that the topics differ greatly. There are also files that inspect for the first ten documents with which percentages it represents topics. These files start with analysis_topic_per_doc. An interesting thing to notice for the library version is that the percentages do not add up to 1 (the percentages are given as a number between 0 and 1). This is easily seen if there is only one number. In these numbers, up to 0.016 is lost. It is most likely that the library version does not return topics that are represented by the document for a very small percentage. This might also explain why the custom implementation has more topics per document it represents, as the custom does not have any type of filter. However, if we left out every number below 0.02 (this seems to be the cut-off point in the library), every document would still have more topics it represents. The library also has for each document a topic that is represented with at least 0.42 for both iteration values, and on average around 0.7. The amount of iterations is in this case more important in the custom implementation. The most-represented topic for a document is only around 0.25-0.30 for ten iterations. When using fifty iterations, these amounts rise to a minimum of 0.27 and a maximum of around 0.80. On average it is around 0.40-0.50. From the numbers of the library implementation, we can see that a document hardly being very representative for a topic and not being divided across many topics, is not a good way of measuring how representative the generated topics are for the documents. This is because the topics are less divided after two iterations than after ten iterations, but we have already seen that these topics were not good classifications, while the topics were much better after ten iterations.

5. Future Work

5.1. Optimization

We already concluded that our model model compared to the one of gensim falls short in terms of performance. The gensim model is very well optimized in terms of performance and well tested by its developers and its users. In the recent years processing on multiple cores has become more standard. It is therefore only natural that gensim also provides a highly optimized parallelized sampler.

5.1.1. Parallelism. Our model uses a custom implementation of Gibbs Sampling that runs on a single thread. The gensim model on the other hand, uses parallelism and uses the Expected Maximization (EM) algorithm. The reason that Gibbs Sampling is not used in this gensim model is that it is not suited for parallelism. The naive way of implementing Gibbs Sampling in parallel would be to generate a completely new state from the previous state instead of updating the values (topic assignments) one by one. This should not work, because it breaks the assumption of sequentiality in the Gibbs sampler. An asynchronous version of Gibbs Sampling has been proposed in a recent study [17], where parallelism is achieved by simply ignoring

these sequential requirements, but it appears to be situational as they conclude with a set of heuristics to describe settings where the algorithm can be used effectively. However, in a different study [18], it is concluded that the end result of parallel Gibbs Sampling is not distinguishable from the sequential version and that this is due to the inexact nature of the sampling method.

5.1.2. Extended stopword removal. Stop words are removed before the corpus documents are converted into a bag of words. This is a great way to reduce the vocabulary and speed up the process while still maintaining the relevancy, as stop words do not represent the contents of a document. Sometimes we would like to retrain the entire LDA model with different parameters (e.g. a different amount of topics). It could prove useful to temporarily store words that do not seem relevant for any topics, which would decrease our vocabulary even further for the next run. This might, however, require human judgement of the words. Although this idea sounds plausible, more research is needed to draw any conclusions about this.

References

[1] Blei, David M. "Probabilistic topic models." *Communications of the ACM 55.4* (2012): 77-84.

[2] Blei, David M., Andrew Y. Ng, and Michael I. Jordan. "Latent dirichlet allocation." *Journal of machine Learning research* 3.Jan (2003): 993-1022.

[3] Wikipedia. "Latent Dirichlet allocation." https://en.wikipedia.org/wiki/Latent_Dirichlet_allocation, accessed 9-12-2020.

[4] Lambert, Ben "An introduction to Gibbs Sampling" https://www.youtube.com/watch?v=ER3DDBFzH2g&t=739s, accessed 11-12-2020.

[5] Wikipedia. "Latent Dirichlet allocation." https://en.wikipedia.org/wiki/Gibbs_sampling, accessed 11-12-2020.

[6] Anaya, Leticia H. "Comparing Latent Dirichlet Allocation and Latent Semantic Analysis as Classifiers." ProQuest LLC. 789 East Eisenhower Parkway, PO Box 1346, Ann Arbor, MI 48106, 2011.

[7] P. Ciaccia. "Latent Semantic Indexing." http://www-db.deis.unibo.it/courses/SI-M/slides/03.2.LSI.pdf, accessed 1-12-2020.

[8] Hofmann, Thomas. "Probabilistic latent semantic analysis." *arXiv preprint arXiv:1301.6705* (2013).

[9] Sterbak, Tobias. "Latent Dirichlet allocation from scratch" https://www.depends-on-the-definition.com/lda-from-scratch/, accessed 11-12-2020.

[10] Resnik, Philip. Hardisty, Eric. "Gibbs Sampling for the uninitiated", 2010. http://users.umiacs.umd.edu/~resnik/pubs/LAMP-TR-153.pdf, accessed 11-12-2020.

[11] J. Xu. "Topic Modeling with LSA, PLSA, LDA & lda2Vec." https://medium.com/nanonets/topic-modeling-with-lsa-psla-lda-and-lda2vec-555ff65b0b05, accessed 1-12-2020.

[12] Griffiths, Thomas L. Steyvers, Mark "Finding scientific topics." 2004. https://www.researchgate.net/publication/5948829_Finding_Scientific_Topics, accessed 11-12-2020.

[13] Moody, Christopher E. "Mixing dirichlet topic models and word embeddings to make lda2vec." *arXiv preprint arXiv:1605.02019* (2016).

[14] J. Turian. Quora. (2012) https://www.quora.com/Whats-the-difference-between-Latent-Semantic-Indexing-LSI-and-Latent-Dirichlet-Allocation-LDA, accessed 9-12-2020.

[15] Aditya. "ML for dummies." (2018) http://ml4dummies.blogspot.com/2018/01/the-dirichlet-distribution-is-conjugate.html, accessed 9-12-2020.

[16] Harvard Dataverse "News Articles" (2017) https://dataverse.harvard.edu/dataset.xhtml?persistentId=doi:10.7910/DVN/GMFCTR, accessed 11-12-2020.

[17] Terenin, Alexander, Daniel Simpson, and David Draper. "Asynchronous Gibbs Sampling." *International Conference on Artificial Intelligence and Statistics.* PMLR, 2020.

[18] Kyrölä, Aapo. "10-702 Project Report: Parallel LDA, Truth or Dare?."