

Contents

1	Introduction and Organization of the Thesis	1
2	Natural Language Processing and Information Retrieval	3
3	Discussed Models	5
3.1	LDA Model	5
3.1.1	Variational EM Algorithm	7
3.1.2	Gibbs Sampling	8
3.1.3	Implementation	9
3.2	Artificial Neural Networks	9
3.2.1	Development	10
3.2.2	Backpropagation	13
3.2.3	Implementation	13
4	Analysis of Gutenberg Data	14
4.1	LDA applied to Textbook Chapters	16
4.2	ANN applied to Textbook Chapters	20
5	Analysis of EUROSTAT Documents	23
5.1	LDA applied to EUROSTAT Documents	25
5.2	ANN applied to EUROSTAT Documents	29
A	Appendix	32
A.1	Documentation of LDA - Gutenberg Data	35
A.2	Documentation of ANN - Gutenberg Data	59

1 Introduction and Organization of the Thesis

Even if the amount of data increases inexorably due to the permanent collection of online data, plus there are always new and better methods to derive good forecasts from big data, it is still difficult for algorithms to extract information from unstructured data. The amount of text data does not increase as quickly as the amount of structured user data. However, it should not be ignored that the written word still has an immensely high -if not the highest - information density for the human mind. Text is the primary and most accurate way to transfer and store complex information. In a nutshell: while machines and algorithms think in numbers, humans still think in words and text. A major challenge of machine learning will be to extract and link information from unstructured data such as text.

One field of computer science is already addressing the topic of text data.

This is Natural Language processing and Information Retrieval (see Chapter 2). In the past, probabilistic approaches have been considered especially promising by researchers (see e.g. [Manning, 1999]). However, the success of methods depends greatly upon the text data to which they are applied and on the desired output. It is therefore equally important to address the methods and models used, as well as to describe the use case in detail when conducting research in this area. Winter et al. for example characterized regulatory documents and guidelines via KNN [Winter, 2017]. In this thesis, I would like to follow Winter’s work thematically. I will examine related procedures and new, completely different methods and approaches to characterize regulatory documents.¹

In this work I adhere to the paradigms of Dejong’s 1979 work in the field of natural language processing. He was the first researcher in this area to move away from the story-specific approach by testing the accuracy of his program to develop a more robust model [DeJong, 1979]. I will even go a step further by optimizing my model with completely different data than that of what it will later be tested on. By doing so I try to reduce the overfitting of the model to the chosen topic and develop a universal tool for text classification. Specifically, I will train my approach for classifying text documents on a relatively large set of textbook chapters. I will optimize the models and their hyperparameters and test the optimized models with a smaller set of regulatory documents from EUROSTAT.

In **Chapter 2** I will discuss the evolution of natural language processing. The different approaches and methods will be summarized and the reader will be given a general overview of the context of this work.

Chapter 3 presents the models used and the form of application is described in detail. The two methods used do not only differ in terms of their underlying models, but also their different tasks. Different methods of fitting and the implementation are covered.

In the following sections of this paper, two examples (respectively in **Chapter 4** and **Chapter 5**) are presented. First, an introductory example with data from various textbooks is analyzed. In 4.1 an LDA model is adapted to cluster the data. In 4.2 a neural network is built up piece by piece, in order to classify new books or chapters.

Based on the models in Chapter 4, in 5.1 another LDA model is developed to logically organize a collection of 28 regulatory documents and guidelines. In a second step (section 5.2), new documents are to be classified according to these groups.

To conclude, in **Chapter 6** the results of both examples are summarized

¹When comparing the two works, however, note the difference in the topic of the documents analyzed.

and the importance of such a process of clustering and classification is outlined.

In order to give the reader a deeper insight into the examples and results and to ensure reproducibility, the **Appendix** contains four Documentation-pdfs in which the entire code and all results are presented in an edited form.

2 Natural Language Processing and Information Retrieval

The reason for the current relevance of the topic NLP is obvious. However what does the concept of natural language processing actually cover and into which research field is it to be classified? It is worth looking at how NLP has evolved over the years and where it originated in order to better understand what this area encompasses and what it does not. Since the origins of NLP lie in computer science, especially in Artificial Intelligence, NLP and AI share the same approaches. For a short review of the motivation of AI, see section 3.2.1.

“Between about 1960 and 1985, most of linguistics, psychology, artificial intelligence, and natural language processing was completely dominated by a *rationalist* approach” [Manning, 1999, p. 4]. In other words, procedures for NLP were completely static, fix-programmed solutions for processing text data. Books like [Noble, 1988] followed this approach until the late 80s. Whereby the enthusiasm gradually, just like for artificial intelligence itself, decreased and arose cyclical.

The second school of thought mentioned by Manning and Schütz is the *empiristic* approach. This is based upon the assumption, that one “can learn the structure of language by specifying a general model and then learning the values of the parameters by applying statistical modeling and machine learning methods to large amount of observed language” [Martinez, 2010, p. 253]. What finally led to the breakthrough of NLP in its current form, is what Manning defines as statistical NLP. I.e. all quantitative approaches for automatization NLP, which includes probabilistic modeling, as well as information theory, and linear algebra. This thesis will focus on methods related to this approach.

Some of this work could even advance a step further. Deep learning (see 3.2.1) was indeed already invented in 1999 - at the time of the division of the definition Mannings in its basic outlines - but still received far less attention than today. One could therefore call NLP via deep learning a subcategory of the empirical approach in its own right, and possibly even list “deep NLP” alongside Statistical NLP.

Statistical NLP always considers recurring patterns and structures in the context of certain texts and does not examine the whole language as such. This approach is no new invention by NLP researchers, but has been used by linguists even prior to computer science (see [Harris, 1951]). A collection or body of texts is called a *corpus*, which means "body" in Latin. In the course of this work we will also mention several corpora, which refers to multiple collections of texts.

In this thesis so called topic models are studied and applied. David M. Blei defines topic models "as algorithms for discovering the main themes that pervade a large and otherwise unstructured collection of documents. Topic models can organize the collection according to the discovered themes"[Blei, 2012]. This definition may therefore apply to both clustering and classification algorithms, even if many authors will refer to clustering algorithms in the context of topic models. You will find a brief introduction to clustering and classification models in the use case in Chapter 4.

Another important term in the research area of this thesis is information retrieval (IR). IR means to give access to a subset of documents of a corpus, which are relevant to a user's query. This deals with computer-aided searches for complex contents and belongs to the fields of information science, computer science and computational linguistics. According to David Grossmann et al. IR refers to a search that might cover information of any kind, e.g. text data as well as video-, image-, sound data, or even DNA sequences. The field where LDA and IR overlap is document retrieval [Grossmann, 2004]. Thus, most procedures we are discussing in this thesis might not solely fall into the topic NLP but also IR. By veering into this direction, it is intended for the reader to better understand the connection to this topic and at which intersection it is located.

In addition, there is a further differentiation of methods concerning natural language processing. One can also distinguish text interpretation with regard to the tasks one intends to accomplish with it. [Jacobs, 1993] distinguishes between three common tasks:

1. *Information Retrieval*: The task to find a subset of a corpus, which contains information concerning a user's query.
2. *Data extraction*: The task to bring texts of a corpus of a particular domain into a pre-defined key structure, suitable for use in a traditional database.
3. *Text categorization*: Separating documents of a corpus into meaningful groups.

The tasks dealt with in this paper may perhaps be best assigned to the last area. On the one hand, I specifically target the task text categorization by dividing a corpus of documents into a predefined number of groups. On the other hand, I try to assign new documents to these groups.

In Chapter 2, I will discuss the evolution of natural language processing. The different approaches and methods will be summarized and the reader will give a general overview regarding the context of this work.

Chapter 3 presents the models used and describes the form of application in detail. The two methods used here do not only differ in terms of their underlying models, but also by their different tasks.

3 Discussed Models

This section will discuss the two Topic modeling approaches which will be studied in this Thesis. The aim of both procedures is to assign one or more topics to different documents. Even if the vocabulary and the notation are similar for both approaches, the notation should be resumed at the beginning of the description of each model. The basic structural notation of the data consists of the following variables.

A collection of documents is called corpus $D = (\mathbf{w}_1, \dots, \mathbf{w}_M)$. It consists of M documents $\mathbf{w} = (w_1, \dots, w_N)$ which represents each of the N words w_i in the vocabulary. These words are vectors of length V . V refers to the length of a vocabulary which holds all the words occurring in the corpus. The vector for a specific word w_i contains all 0 except for index $j \in \{1, \dots, V\}$ which represents this very one word in the vocabulary. This specific system for storing the corpus is called **bag-of-words** format. This notation may be extended through the addition of indices for documents, but this is done neither here nor in the standard literature on topic models due to its unnecessary complexity.

3.1 LDA Model

Latent Dirichlet Allocation is a Bayesian approach and is often associated with hierarchical models [A. Gelman, 2014]. This idea is based on the representation of exchangeable random variable as mixtures of distributions as discussed by de Finetti. Given that documents \mathbf{w} and words w_i in each document - both considered as random variables in this setting - are exchangeable in such a way, a mixed model such as the LDA model is appropriate [Blei, 2003].

The following notation is used in conjunction with the LDA model. Let z_j be the topics with $j \in \{1, \dots, k\}$. In the LDA setting we assume for every

topic z_j there is a term distribution

$$\beta_j \sim Dir(\delta)$$

We further assume each document w has a distribution of topics.

$$\theta \sim Dir(\alpha)$$

Then each word w_i of \mathbf{w} is generated by the following process:

1. Choose $z_i \sim Mult(\theta)$
2. Choose $w_i \sim Mult(\beta_{z_i})$ This distribution will be referred to as $p(w_i|z_i, \beta)$

You can summarize this setup in a plate diagram as shown in figure 1. The notation above, which is also used within the diagram, coincides with the notation of [Hornik, 2011].

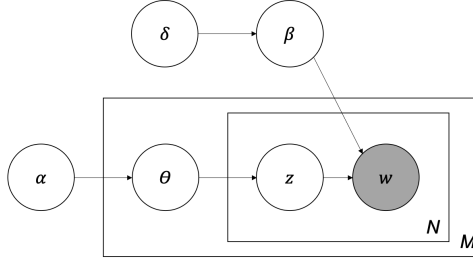


Figure 1: Well-established plate diagram for the standard LDA model extended by the parameter δ . The slightly bigger box represents the generative model of the corporis M documents. The smaller plate represents the iterative generation process of the N words of each document with the aid of the topics. See also "smoothed LDA model" in [Blei, 2003] for comparisons.

In order to estimate the model's parameters, the first step is to calculate posterior distribution, which can be done by dividing the joint distribution by the marginal distribution.

$$p(\theta, \mathbf{z} | \mathbf{w}, \alpha, \beta) = \frac{p(\theta, \mathbf{z}, \mathbf{w} | \alpha, \beta)}{p(\mathbf{w} | \alpha, \beta)} \quad (1)$$

The joint distribution numerator can be derived with the following calculation:

$$p(\theta, \mathbf{z}, \mathbf{w}|\alpha, \beta) = p(\theta|\alpha) \prod_{i=1}^N p(w_i|z_i, \beta) p(z_i|\theta) \quad (2)$$

One can obtain the marginal distribution of a document \mathbf{w} , by integrating out the parameter θ and summing up the topics z_j . Nevertheless, this expression is intractable.

$$p(\mathbf{w}|\alpha, \beta) = \int p(\theta|\alpha) \left(\prod_{i=1}^N \sum_{z_i} p(z_i|\theta) p(w_i|z_i, \beta) \right) d\theta \quad (3)$$

The literature divides the approaches to calculating posterior distribution into two main categories. [Blei, 2012] distinguishes between-sampling based algorithms and variational algorithms. [Powieser, 2012] lists a total of 6 algorithms that can be used to estimate parameters in the LDA model. This thesis will be confined to the two most cited and most used members of the two main groups. One approach is to simulate the posterior density by iteratively sampling - the so-called Gibbs Sampling method. The second approach is a deterministic method, a modified version of the well-known EM algorithm [AP Dempster, 1977]: the Variational EM algorithm (VEM algorithm) [Wainwright and Jordan, 2008]. In the following two sections the both approaches are roughly outlined to give the reader some insight into the Bayesian inference underlying the algorithms.

3.1.1 Variational EM Algorithm

In the VEM algorithm for the LDA model is a mean field approach which varies the steps E and M of the EM algorithm in a way such that this algorithm becomes solvable. Note that the main problem in calculating marginal distribution lies in deriving the conditional probability of hidden variables in the observed values ('evidence'). Variation in the EM algorithms arises mainly from approximating the directly intractable E step. Rewriting the log of the border density of \mathbf{w} as follows in (4), results in a downward estimation of marginal density given Jensen's inequality.

$$\log p(\mathbf{w}|\alpha, \beta) = \log \int \sum_{\mathbf{z}} p(\theta, \mathbf{z}, \mathbf{w}|\alpha, \beta) d\theta \quad (4)$$

$$= \log \int \sum_{\mathbf{z}} \frac{p(\theta, \mathbf{z}, \mathbf{w}|\alpha, \beta) q(\theta, \mathbf{z})}{q(\theta, \mathbf{z})} d\theta \quad (5)$$

$$\geq \int \sum_{\mathbf{z}} q(\theta, \mathbf{z}) \log p(\theta, \mathbf{z}, \mathbf{w}|\alpha, \beta) d\theta - \int \sum_{\mathbf{z}} q(\theta, \mathbf{z}) \log q(\theta, \mathbf{z}) d\theta \quad (6)$$

$$= \mathbb{E}_q[\log p(\theta, \mathbf{z}, \mathbf{w}|\alpha, \beta)] - \mathbb{E}_q[\log q(\theta, \mathbf{z})] \quad (7)$$

Here $q(\theta, \mathbf{z})$ is an arbitrary distribution which can be called the variational distribution.

$$q(\theta, \mathbf{z}) \hat{=} q(\theta, \mathbf{z}|\gamma, \phi) = q(\theta|\gamma) \prod_{i=1}^N q(z_i|\phi_i) \quad (8)$$

The right hand side $L(\gamma, \phi, \alpha, \beta) := \mathbb{E}_q[\log p(\theta, \mathbf{z}, \mathbf{w}|\alpha, \beta)] - \mathbb{E}_q[\log q(\theta, \mathbf{z})]$ is referred to as the 'lower bound'. It can be shown that $\log p(\mathbf{w}|\alpha, \beta) - L(\gamma, \phi, \alpha, \beta)$ is the Kullbak Leibler divergence (D_{KL}) of the true posterior and the variational distribution. From equations (4)-(7) follows that:

$$\log p(\mathbf{w}|\alpha, \beta) = D_{KL}(q(\theta, \mathbf{z}|\gamma, \phi) || p(\theta, \mathbf{z}, \mathbf{w}|\alpha, \beta)) + L(\gamma, \phi, \alpha, \beta) \quad (9)$$

Since the marginal distribution is fixed, we conclude that minimizing the KL-divergence is equivalent to maximizing the lower bound (see [Jordan, 1999] and [Wainwright and Jordan, 2008], for details of the derivation of the lower bound see [Blei, 2003]).

$$(\gamma^*, \phi^*) = \underset{\gamma, \phi}{\operatorname{argmin}} D_{KL}(q(\theta, \mathbf{z}|\gamma, \phi) || p(\theta, \mathbf{z}, \mathbf{w}|\alpha, \beta)) \quad (10)$$

$$= \underset{\gamma, \phi}{\operatorname{argmax}} L(\gamma, \phi, \alpha, \beta) \quad (11)$$

The EM algorithm thus is to use the variational distribution $q(\theta, \mathbf{z}|\gamma^*(\mathbf{w}), \phi^*(\mathbf{w}))$ instead the posterior distribution $p(\theta, \mathbf{z}, \mathbf{w}|\alpha, \beta)$. Now the two steps of the VEM algorithm are:

- (1) **E step** Optimize the variational parameters θ and ϕ for every document in the corpus. This can be done analytically by deriving the derivatives of the KL divergence. And set them to zero.
- (2) **M Step** Maximize the lower bound using the optimized parameter of the E step with respect to α and β .

3.1.2 Gibbs Sampling

The second method to approximate the posterior distribution is Gibbs sampling, a form of the Monte Carlo method. Instead of calculating the distributions for β and θ , the primary task is to find the posterior distribution over \mathbf{z} given the document \mathbf{w} . Gibbs sampling is also known as a Markov Chain Monte Carlo method. The name refers to the simulation process by which a chain of values is simulated whose limiting distribution desirably converges against the true distribution [M. Steyvers, 2006]. (12) shows the distribution, which is sampled from iteratively.

$$p(z_i = j | z_{-i}, w) \propto \frac{n_{-i,j}^{(l)} + \delta}{\sum_t n_{-i,j}^{(t)} + V\delta} \frac{n_{-i,j}^{(d_i)+\alpha}}{n_{-i}^{(d_i)} + k\alpha} \quad (12)$$

$z_i = j$... word-topic assignment of word i to topic j
 z_{-i} ... vector of word-topic assignments without the entry for word i
 $n_{-i,j}^{(l)}$... number of times the l th word in the vocabulary is assigned to topic j , not including the assignment for word i
 d_i ... document in the corpus which includes word i
 δ, α ... parameters of the prior distributions for β and θ

The word-topic distributions $\beta_j^{(l)}$ for the words $l = 1, \dots, V$ and topics $j = 1, \dots, k$ and topic-document distributions $\theta_j^{(d)}$ for the documents $d = 1, \dots, D$ and the topics $j = 1, \dots, k$ will be of particular interest. (13) and (14) shows the predictive distributions denoted as "estimators".

$$\hat{\beta}_j^{(l)} = \frac{n_{-i,j}^{(l)} + \delta}{\sum_t n_{-i,j}^{(t)} + V\delta} \quad (13)$$

$$\hat{\theta}_j^{(d)} = \frac{n_{-i,j}^{(d_i)+\alpha}}{n_{-i}^{(d_i)} + k\alpha} \quad (14)$$

For derivation and more details regarding the Gibbs sampling procedure see [M. Steyvers, 2006].

3.1.3 Implementation

In this thesis, the implementation of the LDA model and its estimation is mainly based on using the package `topicmodels` of Kurt Hornik. The package `topicmodels` can apply both the VEM algorithm as well as Gibbs sampling in order to fit the model. In addition, the package `tidytext` is used for text structuring and embedding. Whereby there are other packages besides this implementation of the LDA model, `topicmodels` is particularly convenient, because `tidytext` was designed by its developers to work perfectly in combination with `topicmodels` [Silge and Robinson, 2017, p. 89].

3.2 Artificial Neural Networks

Artificial neural networks (ANN) are much more versatile than the LDA model. There are not only various forms of artificial neural networks, but also a very large number of application areas. Much like machine learning procedures in general, also deep learning algorithms are divided into two broad categories: supervised learning, where a superset instance provides the algorithm with the output required to learn, and unsupervised procedures that internally train predefined models to find patterns in the input signals. In this chapter we will focus heavily on the former group of ANNs. Also, this chapter is intended to give the reader an overview of the research on neural networks as well as the background of their development.

3.2.1 Development

Research on ANNs dates back to the 1940s, when [McCulloch and Pitts, 1943] introduced the so called “M-P neuron”. Whereby this neuron had only a bi-variate input and output, Rosenblatt later extended this idea to a network of M-P neurons, which allowed to set up a simple classification algorithm [Rosenblatt, 1958]. A perceptron in its basic form (single perceptron) is a binary classifier.

Imagine input data of a simple perceptron in the form of a matrix.

$$X = \begin{bmatrix} x_{11} & \dots & x_{1k} \\ \vdots & \ddots & \vdots \\ x_{n1} & \dots & x_{nn} \end{bmatrix} = \begin{bmatrix} \mathbf{x}_1 \\ \vdots \\ \mathbf{x}_n \end{bmatrix}$$

The dependent variable thus is a vector $\mathbf{y} = y_1, \dots, y_n$, with $y_i \in \{0, 1\}$. Consider the lines of the X matrix as vectors $\mathbf{x}_1, \dots, \mathbf{x}_n$, with $\mathbf{x}_i \in \mathbb{R}^k$. The entries of each of the vectors are weighted with $\mathbf{w} = w_1, \dots, w_k$ with $w_j \in \mathbb{R}$ and aggregated in a function h e.g. a sum.

$$h(\mathbf{x}, \mathbf{w}) = \sum_{j=1}^k x_j w_j$$

Using a so called “activation function” h is mapped to the output space, which is in this case $O = \{0, 1\}$. At this point a step function serves as activation function.

$$a \circ h(\mathbf{x}, \mathbf{w}) = \begin{cases} 0 & \text{if } h(\mathbf{x}, \mathbf{w}) \leq 0 \\ 1 & \text{else} \end{cases}$$

The matrix X is passed vector by vector to the perceptron and the output is compared with the values for \mathbf{y} . During this procedure the weights are iteratively tuned by a simple updating algorithm using the pairs \mathbf{x}_i and y_i .

The algorithm of the simple perceptron is schematically shown in Figure 2. This diagram corresponds to the common representation in education [Mukherjee, 2019], although a horizontal perspective is often chosen.

If this basic perceptron is used in a clever way, designs can be developed that have many different application possibilities. For example, a sigmoid function can be used as an activation function instead of the step function. So the output layer will not project into the $\{0, 1\}$ space, but into a probability space. If you add several nodes with the sigmoid activation function rather than a single output, you basically obtain the architecture that is also called multivariate logistic regression [Bahjat, 2006]. In that case the model

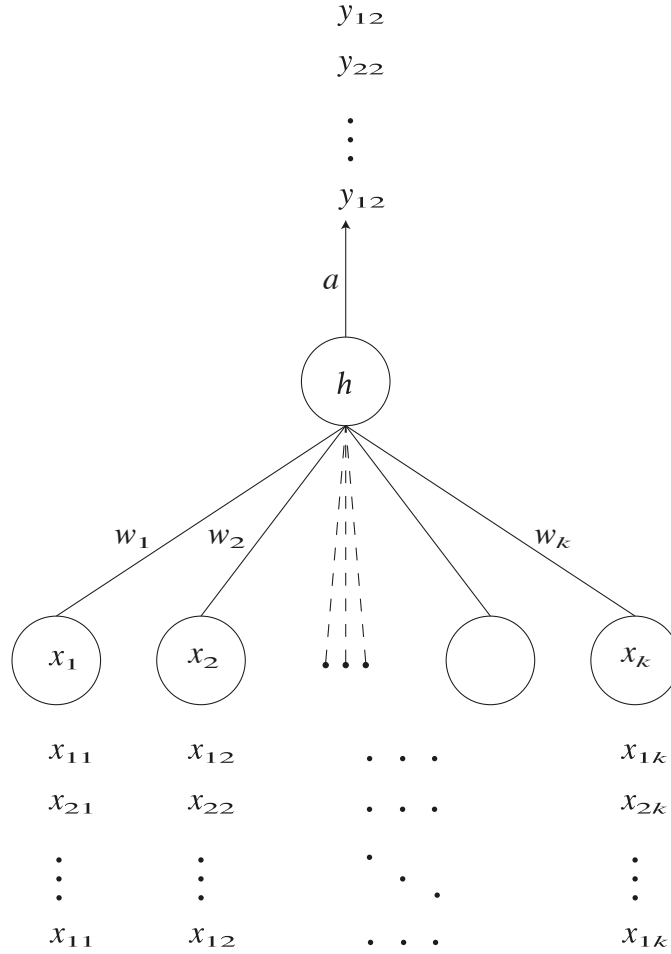


Figure 2: Schematic diagram of a simple perceptron by [Rosenblatt, 1958]

can be estimated by an individual calculation of logistic regressions for each node in the output layer. This allows to classify not only two classes, but an arbitrary number (see the network architecture of Figure 3).²

Shortly following the publication of these results, which would lay the foundation for later neural networks, ANN research had to suffer a severe setback after Minsky and Papert [Minsky and Papert, 1969] were able to prove that perceptrons cannot provide a suitable solution in certain basic scenarios. It is possible to separate two clusters by a hyperplane with the perceptron. However, if the two groups could not be separated completely, the perceptron fails. For instance, it was impossible to find a solution for data generated with an *x-or* function (also called "exclusive or" function).

²Note that in this setting the dependent variable must be one-hot encoded.

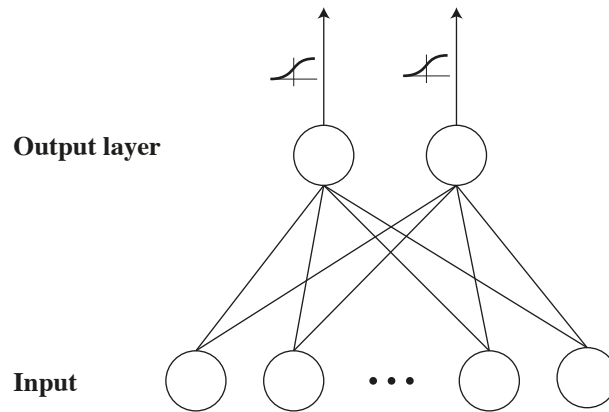


Figure 3: Schematic diagram of an adapted Rosenblatt-perceptron network

Although the researchers Minsky and Papert showed that the perceptron in this form cannot solve the “xor problem”, they argued that extending the simple perceptron to a multi layer perceptron solves this problem, if it was feasible to train this model. Instead of just one input layer and one output layer, an MLP may include an arbitrary number of hidden layers in between. Figure 4 shows the structure of such a network. However, training a model using Rosenblatt’s naive optimization algorithm of the perceptron would not have been feasible.

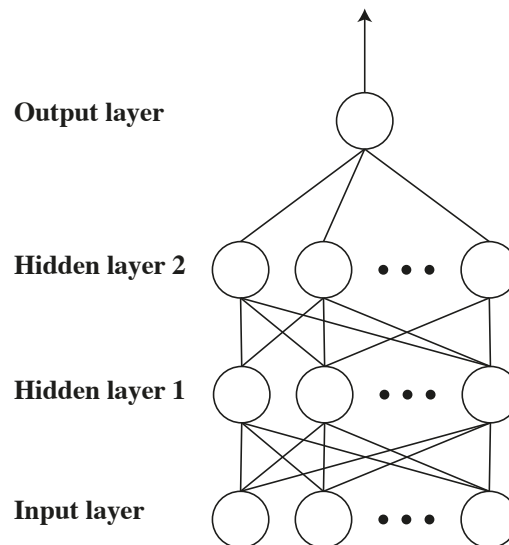


Figure 4: Schematic diagram of an multi layer perceptron (MLP)

3.2.2 Backpropagation

The procedure used to optimize the weights of a multilayer perceptron is called backpropagation. This method was first applied to neural networks by Werbos - as part of his dissertation [Werbos, 1974]- and still works the same way to this day. It is important to use a sigmoid function instead of the step function which is used in the perceptron, as this function is differentiable. Assume a random initial distribution of the weights w_1, \dots, w_d for a network with d layers, which are the starting values for the update procedure. Let τ_i be the number of units in layer i . So layer i can be denoted as function f_{w_i} and the whole network as a chain of functions.

$$\hat{y}_i = f_{w_d}(f_{w_{d-1}}(\dots(f_{w_1}(x_i)))) = f_{w_d} \circ f_{w_{d-1}} \circ \dots \circ f_{w_1}(x_i) \quad (15)$$

The goal is to update the weights of each layer in such a way as to minimise a selected loss function $L(w_1, \dots, w_d)$. A common loss function is for example quadratic loss: $L = \frac{1}{2} \sum_{i=1}^n (\hat{y}_i - y_i)^2$. The backpropagation algorithm consists of the 3 following roughly outlined steps.

1. **forward pass:** Calculation of the result y_i for all units at the set weights
2. **backward-sweep:** Each layer's derivatives are now calculated using the chain rule, step by step - starting with the output layer. For the output layer this is:

$$\frac{\partial L}{\partial w_d} = \frac{\partial L}{\partial f_d(x_i)} \frac{\partial f_d(x_i)}{\partial w_d}$$

3. **updating:** Update the weights by an arbitrary optimization algorithm. E.g. steepest descent: $\Delta w_d = \alpha \frac{\partial L}{\partial w_d}$, where α is the learning rate

Thanks to current software, the derivatives no longer have to be calculated for each node by hand, but the standard packages are capable of "symbolic differentiation" for certain network structures, which makes training neural networks comfortable [Chollet, 2018, p. 47].

Note: While Rosenblatt for the perceptron was actually inspired by the biological structure of the neurons in the brain, more complex "networks used by engineers are only loosely based upon biology" [Hecht-Nielsen, 1988].

3.2.3 Implementation

This thesis uses the `Keras` R package to create neural networks, a deep learning API (Application-Programming-Interface) to deep learning backend

engines, designed for R by Allaire in 2017. Keras is a so called “model-level” library, designed to set up complex neural net architectures using well arranged, high-level building blocks. As backend-engine the users are provided with TensorFlow, Theano and Microsoft Cognitive Toolkit. Using these backend engines computation may be processed seamlessly via CPU or GPU [Chollet, 2018].

4 Analysis of Gutenberg Data

In this first example, the chapters of individual books are classified, all of which sourced from the freely distributed Project Gutenberg. Project Gutenberg is a provider of over 60,000 free electronic books with the primary aim to “encourage the creation and distribution of eBooks”[Hart,]. The package `gutenbergr` [Robinson, 2018] preprocesses the ebooks text data and downloads it. This convenient package in combination with the free repository allows the analysis of a large number of large text documents with a secure source and a big amount of meta information. For this analysis interesting information is e.g. Gutenberg id as key variable for fast identification of books, title, author and more importantly the categorization of project Gutenberg, the so-called Gutenberg bookshelf. Table 1 lists all this information for a random sample of books downloaded from the Project Gutenberg.

Table 1: Example book corpus

gutenb. id	title	author	gutenberg bookshelf
2095	Clotel: A Tale of the Southern States	Brown, William	African American Writers
6315	The Awakening of Helena Richie	Deland, Margaret	Bestsellers, American
6971	Judaism	Abrahams, Israel	Judaism
7635	The Disowned — Volume 05	Lytton, Edward and Baron	Historical Fiction
10319	Dave Darrin’s Third Year at Annapolis	Hancock, Harrie Irving	Children’s Book Series

In order to test NLP models, two questions can be examined on the basis of these simple text documents. **Firstly (Q1), does a LDA cluster text documents similar to a human-driven classification?** In this case, this could be validated by the categorization into Gutenberg bookshelves. **And secondly (Q2), what is the best approach to reproduce such a classification using an ANN as a classification model?**

Since very large amounts of data have to be processed and analyzed in order to model the bookshelf classification for a big collection of books, a somewhat reduced approach will be used here. One breaks down a collection of books of different categories into chapters in order to cluster respectively classify these chapters as independent documents. Similarly, you could give an example, assuming N books and separate their chapters and shuffle them, is it possible to use models to reassemble the chapters into stacks that can

be assigned to individual books?

The raw text record is now transformed into the tidy format of tidytext (i.e. one word per row). Now it is possible to remove unnecessary stop-words. These words are predefined and can be modified for the appropriate use case if necessary. The words stem from 3 sources, "onix", "SMART" and "snowball", whereas the latter two are pulled from the `tm` package [Silge, 2019]. And the Onix stop words are taken from the publicly accessible site lex-tec.com. An example of stop words can be found in Table 2.

Table 2: Example of stop words from tidytext package

word	lexicon
many	SMART
mrs	onix
was	snowball
sure	onix
you're	snowball
go	onix
c	SMART
where	snowball
then	SMART

Both models that are studied in this thesis, use a bag-of-words data set as input data. This means a matrix with the documents in the M lines and the frequencies of the V used words in the entries of the columns. The dimension - i.e. the number of words in this "dictionary" V - may be reduced for two reasons. For one thing, a dimension reduction can reduce the fitting time of the model, for another thing, the diversity of the documents can be increased by skilfully reducing certain words, which occur equally frequently in all documents. Now this requires a special measure on which we decide which words to exclude. One can call this reduction of the bag-of-words dimensionality "embedding". In the course of this work 3 different embedding methods were tested.

1. The reduction by the words that occur with a low frequency.
2. No dimension reduction, i.e. use of the full dictionary of all occurring words.
3. The reduction with the aid of the measure *tf-idf*.

tf-idf is a combination of the term frequency and the inverse document frequency, defined as follows.

$$tf-idf(t, d) := tf(t, d) \times idf(t)$$

$$tf(t, d) := \frac{f_{t,d}}{\sum_{t_i=1}^V f_{t_i,d}}$$

$$idf(t) := \ln \left(\frac{M}{n_{d' \in t}} \right)$$

t	... term (word)
d	... document
$f_{t,d}$... frequency of term t in document d
M	... number of documents
$n_{d' \in t}$... number of documents containing term t

Even if “its theoretical foundations are considered less than firm by information theory experts”, *tf-idf* “has proved useful in text mining” [Silge and Robinson, 2017]. In this thesis the term frequency and the *tf-idf* are not to be used directly for the analysis of the texts but mainly for setting up the bag-of-words datasets. It shall be investigated whether the use of different embeddings has an influence on the text analysis itself.

4.1 LDA applied to Textbook Chapters

As LDA is a classification algorithm, the research question Q1 should be addressed first. I.e. to what extent does the clustering of the LDA model correspond to the mapping of chapters to books? In the first attempt only 6 books are sampled. That is, there are 6 categories, because as mentioned in 4 all these 6 books were taken from different bookshelves.

As described in Chapter 3.1, it is possible to calculate the LDA model by means of two different algorithms. These are the VEM algorithm and Gibbs sampling. In this study we examined both methods. Clearly, the algorithms may differ in the speed of the calculation. However, it does not have to be the case that both calculation methods deliver the same results.

Three fundamentally different approaches should be distinguished for embedding:

1. the entire dictionary, i.e. all words occurring in the corpus are used for the bag-of-words.
2. for the bag-of-words only words are used which occur at least 2 times in the whole corpus. This reduces the number of used words and therefore the dimension of the data by nearly 50%.
3. the bag-of-words is reduced by 50% according to *tf-idf*.

These approaches are reviewed with regard to the goodness of the fit and the speed of the computation. One intuitively may assume that the calculation

takes longer for data with higher dimensionality.

For a clustering model there are two useful methods to evaluate the fit. On the one hand it is possible to fit the model with training data and to evaluate the goodness of the fit using the test data. On the other hand, it is a comparison of the classification of the model with the categories given by the data, whereby the model does not learn from these categories. The latter evaluation method is clearly less computationally intensive, and the comparison of the entire data set instead of only the test part of the data has a positive effect on the variance. Starting this analysis the second approach is used primarily.

The findings of this study relating to the LDA model are documented in detail. You will find the documentation regarding the LDA model as well as the other models in the appendix. For the sake of clarity, a separate document with the corresponding results was created for each analyzed example and each model respectively. To ensure reproducibility, the entire code is attached to the work by means of these documentations, and will be published on GitHub as well.³

The computation of the LDA model via VEM takes a somewhat longer time in the two samples evaluated. In all cases investigated, i.e. across the two samples taken and for both VEM and Gibbs sampling, the calculation for the *tf-idf* embedding appeared to be the fastest. The calculation time for embedding according to term frequency and full embedding differs less and the calculation of the term frequency embedding is not faster in every case. While a reduction via term frequency only cuts the most rarely used words, a reduction via *tf-idf* also cuts out frequent words as long as they occur frequently in all documents. Thus, *tf-idf* reduction has more impact on the distribution within each document compared to the effect of the term frequency reduction, which only affects the tails of the marginal distribution. This supports the assumption that a higher degree of difference in the documents has a positive effect on the calculation speed. Figure 5 visualizes how the fitting time changes when reducing the bag-of-words to an arbitrary size via *tf-idf*.

With regard to the quality of the fit, it is apparent that Gibbs sampling performs much better than models fitted by VEM. The best fit using VEM still shows a higher misclassification rate than the poorest fit using Gibbs sampling in the examples examined. It is reasonable to focus on Gibbs sampling in further analyses. For this reason we will only consider Gibbs sampling in

³https://github.com/SebastianKnigge/Master_Thesis/tree/master/Documentations

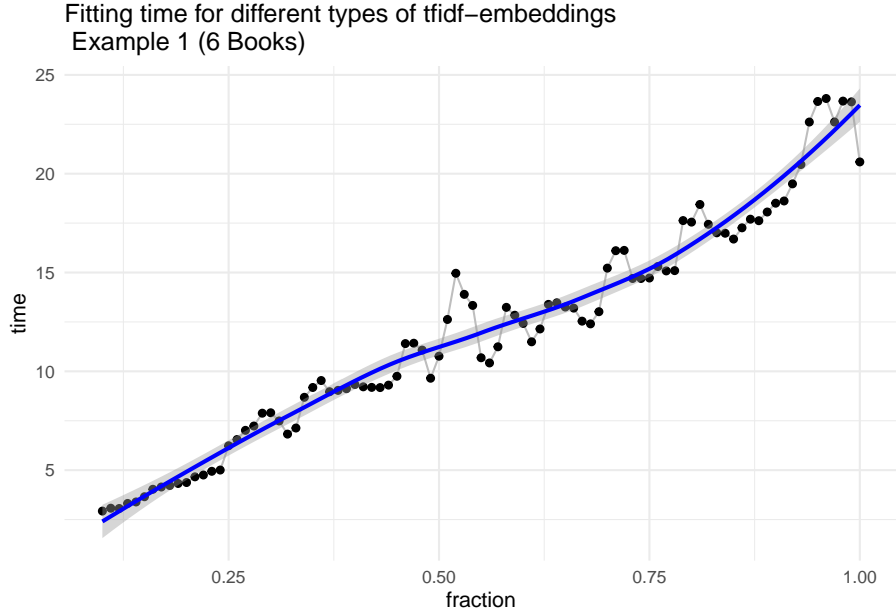


Figure 5: Fitting time of Example 1 (6 Books) depending on the reduction of the bag-of-words. A big number of fraction stands for a bigger bag-of-words, i.e. less reduction. The blue smoothed curve is a Loess kernel. The drop in fitting time for the last few values of fraction might result from parallelization reasons.

Chapter 5.1.

Surprisingly, the models with the *tf-idf* embedding do not perform significantly better than the other approaches, as one might expect that more differentiated word distributions of documents not only allow a faster calculation, but also result in a higher accuracy. In fact, frequency 2 embedding provides better accuracy results throughout the studied examples.

It may be interesting to test embedding via *tf-idf* for different values of reduction. So far, a reduction of 50% has always been considered. In this case we measure $fraction = 1 - reduction$ (i.e. what fraction of the original bag-of-word remains after *tf-idf* embedding). Figure 6 shows the misclassification rate for different values of fraction. We can obtain a general negative trend of the misclassification rate, for bigger values of fraction. This means the accuracy of the model increases with a larger bag-of-words, for the reduction via *tf-idf*. However, there are also values of fraction where the accuracy for both examples improves compared to full embedding (i.e. fraction 1).

However, this may also be overfitting of the parameter fraction, if for other samples for this value of fraction the accuracy is worse. This parameter needs to be optimized very carefully, taking into account the underlying documents and the application. In summary, *tf-idf* embedding can achieve very good results if the fraction parameter is tuned. For a generic approach, however, this method is too vulnerable to over-fit and it is recommended to choose a reduction to term frequency 2 as embedding method, if there is no further information about the data set.

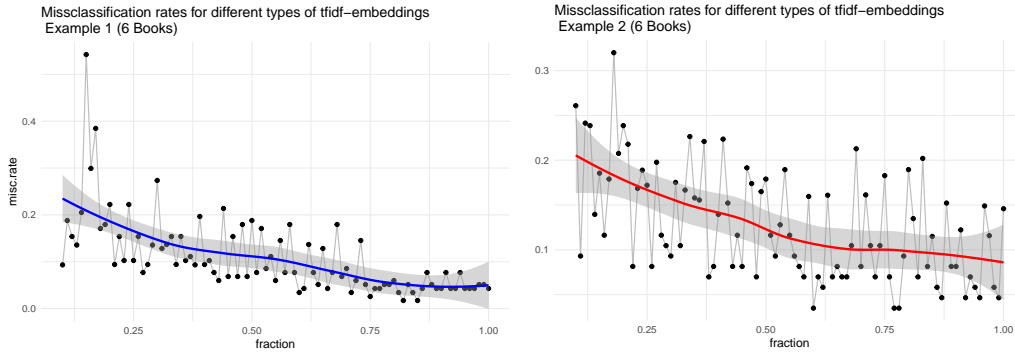


Figure 6: Misclassification rate depending on the reduction of the bag-of-words. A big number of fraction stands for a bigger bag-of-words, i.e. less reduction. The blue and red smoothed curves are a Loess kernels.

It was also analyzed how the accuracy will change when using more categories. For this purpose the number of books was increased to 10. The fit via VEM was still much worse than the fit via Gibbs sampling. The model fitted with VEM showed a misclassification rate almost twice as high as the one fitted with Gibbs sampling. For both methods the accuracy deteriorated for more topics used. This sounds intuitive since the clustering algorithm now has to distinguish between more topics. The box plot in Figure 7 illustrates the LDA algorithm in the different use cases. Here we are comparing the investigated cases. Clearly the difference in the misclassification rate between the examples with 10 and 6 categories becomes apparent. It also illustrates how different the two calculation methods - Gibbs sampling and VEM - are.

Considering the second approach for evaluation mentioned in the beginning of this section, results regarding this procedure should be discussed as well. The classical method to evaluate the model is to split the data set into a training set and a test set, in order to fit the model on the training data and to evaluate it on the test data. To calculate valid results 59 random clustered samples were used with splits of 90% training sample and 10% test sample applied. The results for those models that were only fitted to a subset of the

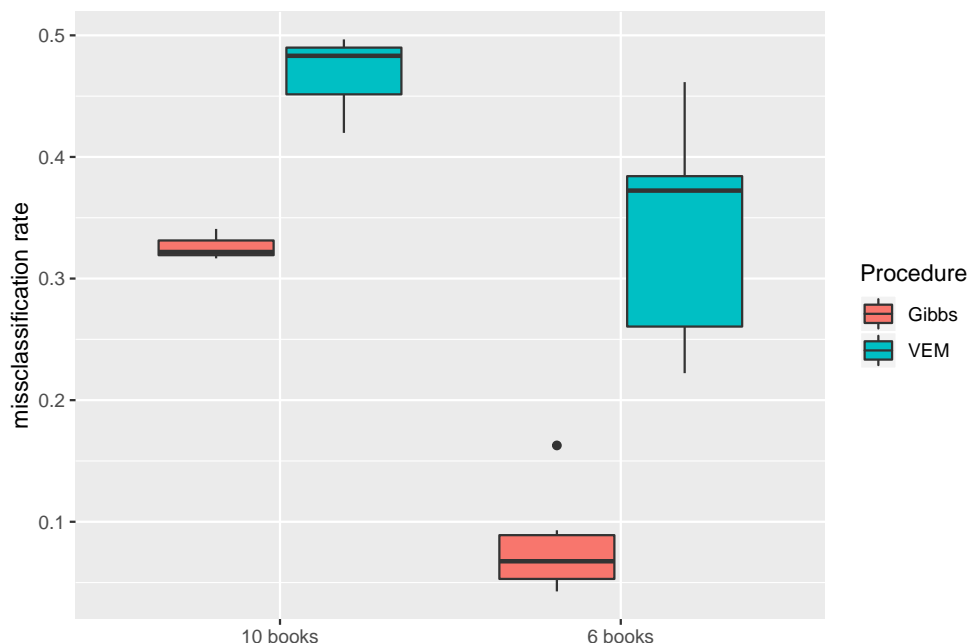


Figure 7: Boxplot of the all examples analyzed. Split for the number of categories (books) as well as for the calculation procedures (VEM algorithm and Gibbs sampling).

data are significantly worse than the comparable models that were trained and tested on the entire dataset. However, since here a clustering model is discussed that does not rely on training data, such as a classification model, it is technical correct to evaluate the model using training data.

But what do these results mean for the LDA model? Since this method aims at evaluating the fit on the basis of classifying new documents, it is basically an evaluation method for classification problems. It follows that the LDA model is a poor classification model for the use case analyzed. For this reason, I use another model to classify new documents, which is a neural network (see following section).

4.2 ANN applied to Textbook Chapters

In what context is it now possible to apply an artificial neural network to this data? Obviously neural networks are a very versatile tool, but here we focus on the reproduction of a known classification of the documents (chap-

ters) by a suitable model. With the help of such networks in the sense of a classification algorithm, research question 2 will be addressed in this section. Again, for the shake of reproducibility the entire code including outputs of the computations can be found in the documentation in the appendix.

Primarily the question shall be addressed whether a neural network is suitable as a classification algorithm for NLP with bag-of-words data. First of all, the architecture of the network used will be explained (see Figure 8 for comparison). The used network essentially consists of three layers. The

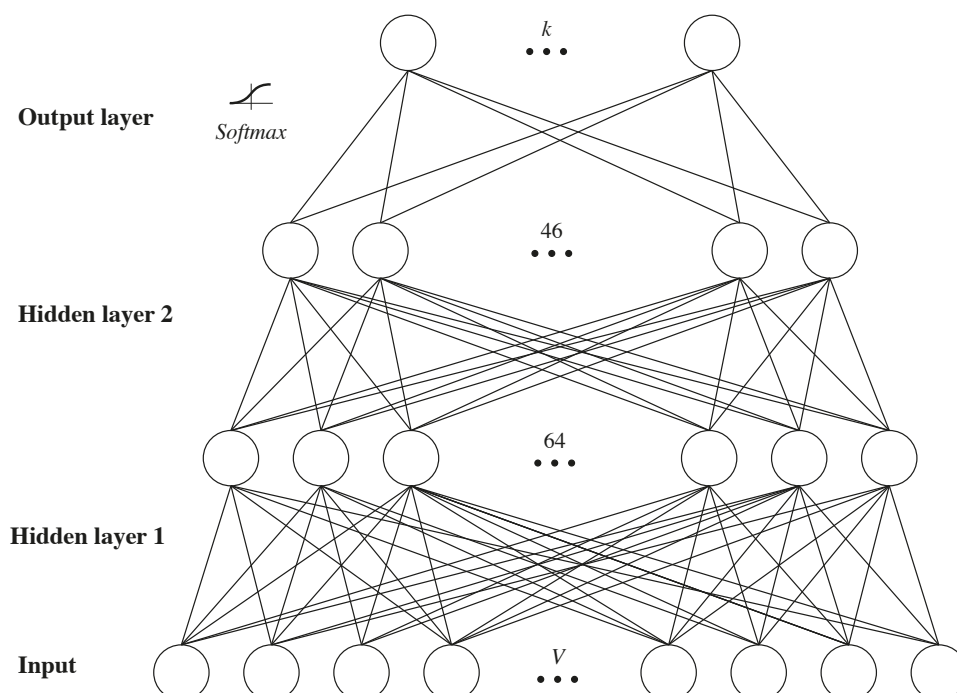


Figure 8: The network contains 3 layers. The input data is the the bag-of-words, which has dimension V . The first two layers are hidden, containing 64 and 46 neurons respectively. The last layer leads to the k classes (here the number of topics or books). The activation function is softmax.

lowest V neurons represent the dimension of the input, i.e. the dimension of the bag-of-words input vectors rather than an actual layer. The network contains two hidden layers, each containing 64 and 46 neurons. The last output layer contains k neurons, which is the number of topics - in this case books - to classify. Note that the architecture of the network was arbitrarily chosen and that there may be room for improvement. Based on experience with this data set, however, this architecture provides good results.

Whether a network is a classification system or e.g. a regression system depends almost exclusively on the activation function in the last so-called

output layer. The "softmax function", also known as the "normalized exponential function" [Bishop, 2006, p. 115] is commonly used for classification networks. The softmax function maps to a $(0, 1)$ space, where the sum of the vector in the image space equals to 1. Thus, the output corresponds to a discrete distribution over the different classes. The classification is made using the maximum probability within this distribution per instance.

$$P(y = i|\mathbf{x}) = \frac{e^{\mathbf{x}^t \mathbf{w}_i}}{\sum_{j=1}^k e^{\mathbf{x}^t \mathbf{w}_j}} \quad (16)$$

Where \mathbf{x} is the vector of the output of the second hidden layer, and \mathbf{w}_i are the weights of the i -th neuron in the output layer.

Considering the computing power of the used machine and the required computing time combined with overfitting, a training using 5 epochs and a batch size of 512 turned out to be suitable. In contrast to the LDA model, however, this procedure must be fitted using training data and evaluated using test data. For this purpose, a proportion of 10% test data, 70% training data and 20% validation data for learning has been used. Note that in a deep learning setting a validation set is essential to prevent overfitting during learning. The test set is then used to evaluate the model via repeated sampling with 59 iterations.

Table 3: Example 1 (6 books): Performance for different embeddings

embedding method	<i>tf</i> -2	full bag-of-words	<i>tfidf</i> 0.5
missc. rate	0.022	0.008	0.027
time	3.646	10.289	4.240

Table 4: Example 2 (6 books): Performance for different embeddings

embedding method	<i>tf</i> -2	full bag-of-words	<i>tfidf</i> 0.5
missc. rate	0.105	0.106	0.119
time	10.135	3.774	9.303

The results of the repeated sampling show that a large part of the classifications by the ANN exactly match the true values of the test data set. In very few samples the classification was incorrect for 10-20% of the test sample (i.e. almost perfect classification). Therefore, misclassification rates

over all samples are in the range of 1-12%. Klassification for Example 1 shows better results. The variances of each simulation amounts to 0.1-0.2%. These results in this rather small sample support the claim that classification via ANN with bag-of-words data works very well. Tables 3 and 4 show the results of the simulations for each embedding method. According to the differences in accuracy for this embedding method it is not possible to obtain a preferred method. In Example 1 the full embedding shows the lowest misclassification ratio, whereas in Example 2 *tf*-2 embedding appears to be the best method. Considering a classification model, it is possible to choose the optimal embedding procedure for each use case.

5 Analysis of EUROSTAT Documents

In this application, we would like to focus on the classification of technical, regulatory documents and guidelines. The ESS Vision 2020 ADMIN (Administrative data sources) project aims at “guarantee the quality of the output produced using administrative sources, in particular the comparability of the statistics required for European purposes”[Eurostat, 2019]. This project contains a collection of 28 official Eurostat documents, i.e. guidelines, methodological definitions, and manuals, e.g. on data access. Table 5 displays the full corpus including all documents.

This chapter will examine how the two methodologies investigated so far can be efficiently applied to the type of documents described above, in order to gain informational value. The research questions in this case are slightly modified and aim at a more specific problem solving approach in the context of the evaluated data. **Q3: To what extent can the LDA algorithm support a decision maker in clustering the ESS Vision 2020 ADMIN documents?** Question Q4 builds directly on question Q2, differs however regarding the documents examined. **Q4: How well may the classification of ESS Vision 2020 ADMIN documents be reproduced using an ANN?** In this case, we particularly focus on the setups of the two methods we already optimized for the application to the Gutenberg text data. This means that the same hyper-parameters and fitting methods are used as those tested and proven in chapter 4, because the aim of that example was to find an optimal generic architecture for problems of this kind.

Only a very small amount of preprocessing was necessary for the documents that were taken directly from the website. Solely for a few documents the table of contents had to be removed, as the same table of contents appeared several times for different documents. The same stop words were excluded

Table 5: Entire list of documents

Doc. No.	Document title
1	admin-wp1.1_analysis_legal_institutional_environment_final.pdf
2	admin-wp1.2_good_practices_final.pdf
3	admin-wp2.1_estimation_methods1.pdf
4	admin-wp2.2_estimation_methods2.pdf
5	admin-wp2.3_estimation_methods3.pdf
6	admin-wp2.4_examples.pdf
7	admin-wp2.5_alignment.pdf
8	admin-wp2.5_editing.pdf
9	admin-wp2.5_greg.pdf
10	admin-wp2.5_imputation.pdf
11	admin-wp2.5_macro_integration.pdf
12	admin-wp2.5_macro_integration.pdf
13	admin-wp2.6_good_practices.pdf
14	admin-wp2.6_guidelines.pdf
15	admin-wp3.1_quality1.pdf
16	admin-wp3.2_quality2.pdf
17	admin-wp3.3_quality.pdf
18	admin-wp3.4_quality.pdf
19	admin-wp3.5_quality_measures.pdf
20	admin-wp3_coherence.pdf
21	admin-wp3_growth_rates.pdf
22	admin-wp3_suitability1.pdf
23	admin-wp3_suitability2.pdf
24	admin-wp3_suitability3.pdf
25	admin-wp3_uncertainty.pdf
26	admin-wp5_frames.pdf
27	admin-wp5_frames_examples.pdf
28	admin-wp5_frames_recommendation.pdf

as in Chapter 4.⁴ Consideration must be given to further circumstances of the regulatory documents. Remember that there are many formulas and technical abbreviations in the documents, so each variable, each estimator, and each index is included as a single word in the bag-of-words. These terms sometimes have a big influence on the documents, because they are very specific for individual documents and occur quite often. To avoid this, all mixed words which include characters and numeric attributes, and all terms with special characters (e.g. Greek letters) are also excluded.

5.1 LDA applied to EUROSTAT Documents

A fundamental problem when clustering text documents is that for certain corpora there may not be a unique grouping of the documents, even if the number of clusters is fixed. Logical, thematic clustering always has a certain uncertainty depending on the aspects according to which it is clustered. For this reason, it is reasonable to compare the statistical clustering to a human-made grouping. This corresponds to the procedure of chapter 4.1, where the clustering of books by the LDA algorithm was compared with the man-made classification of Gutenberg bookshelves (due to the long computing time – in fact - the approach was adopted for comparing chapters with the allocation to books of different bookshelves).

*Note: The terminology used in this chapter refers to the expert assessment when mentioning **groups** and to the clusters of the LDA model when discussing **topics**.*

In this case it is reasonable to consult an expert to get a suitable initial thematic grouping. Univ.-Prof. Dr. Wilfried Grossmann from the Faculty of Computer Science at University of Vienna is an accredited expert in this field and based on his expertise he established a grouping of 7 groups for the documents described here. He proposed the grouping into the following thematic clusters:

1. Legal documents
2. Methods
3. Examples
4. Details
5. Quality
6. Tests

⁴The stop word dictionaries "onix", "SMART" and "snow- ball" are used, as they are provided by the `tm` package [Silge and Robinson, 2017].

7. Frames

The model already known from the example in chapter 4.1 with the same specifications was used. This means that 7 categories were used, and Gibbs sampling was chosen for the fit, after it turned out that the LDA model clustered better when used with the Gutenberg data. Also the embedding via minimum term frequency 2 was used, since in the course of the work for the LDA model it appeared to be a robust method in terms of accuracy. Once again, for clustering, the comparison to a man-made grouping is not trivial, because the algorithm can recognize patterns other than those assigned by humans in a certain context. Instead of evaluating the fit using a very strict measure, such as accuracy or misclassification ratio, a logical comparison of the two classifications seems to be more appropriate.

In a first step, it makes sense to find out according to which aspects the algorithm has clustered, i.e. which topics are dominant in the found clusters. I am using word clouds to illustrate the contents of the individual clusters. In this case I made use of the package `wordcloud` which automatically renders wordclouds based on a frequency distribution which is passed to the `wordcloud()` function along with the terms. Analogous to [Winter, 2017] the *tf-idf* measure is used instead of the absolute term frequency. This is advantageous as the wordclouds do not resemble each other so much when deploying *tf-idf*, especially since the documents all originate from the underlying topic statistics.

Some groups can be recognized very well in the clusters detected by the LDA. For example, the group "legal documents" corresponds exactly to one cluster. The group "Frames" can also be identified from the wordclouds as a separate topic. Group 4 is also extracted relatively well, but due to the wordcloud it may rather be viewed as topic "documents on Bayesian statistics", rather than "Details". Other groups such as "Quality" and "Tests" are mixed together and divided into two new clusters. Obviously, the model sticks in this case much more to the individual words than to the latent groups as they were assessed by the Expert.

The descriptive results of this example mainly concern embedding. Due to the number of documents (28 documents) the dimension of the bag of words is smaller. It amounts to almost 8,600 words, for the full embedding compared to 15,000 words for the analogous dictionary of the Gutenberg data. However, the pruning of the bag of words by the frequency 2 embedding is less in comparison. While the dimension of the full bag of words to the frequency 2 bag of words decreased by about 40% for the Gutenberg data, the dimension for the Eurostat data decreased by only 30%.



Figure 9: Wordclouds for the clustered topics via LDA – using $tfidf$ word proportions

Even though in the last example tf -2 embedding turned out to be the most robust approach in terms of accuracy, in this example we will investigate one more time whether the tf -idf method possibly provides better results. Specifically, I applied tf -idf embedding, then fitted and evaluated the model for different values of fraction. Fraction is the ratio of the bag-of-words dimension remaining when shrinking via tf -idf embedding. This is visualized in the plot of Figure 10. Considering this plot, fraction 0.5 is no wise choice for two reasons. The loess estimator indicates a higher misclassification rate in this area in general. And variance appears to be higher in this area as well. One can obtain a range of lower misclassification rates for values of fraction between 0.3 and 0.4. Note, that this choice of the fraction parameter might cause overfitting. Nevertheless, we will now compare the different embedding methods in the following paragraph.

Comparing the tf -idf embedding method to the frequency 2 embedding, there are several reasons for using the more robust tf -2 embedding. Because the LDA model is not repeatedly fitted in general, the fitting time is negligible. Thus, the focus is primarily on the misclassification ratio. Table 6 shows 3 different embedding methods in comparison. Term frequency 2 embedding, tf -idf embedding with fraction 0.5 and tf -idf embedding with fraction 0.35. Fitting a LDA model using tf -idf 0.35 embedding shows the

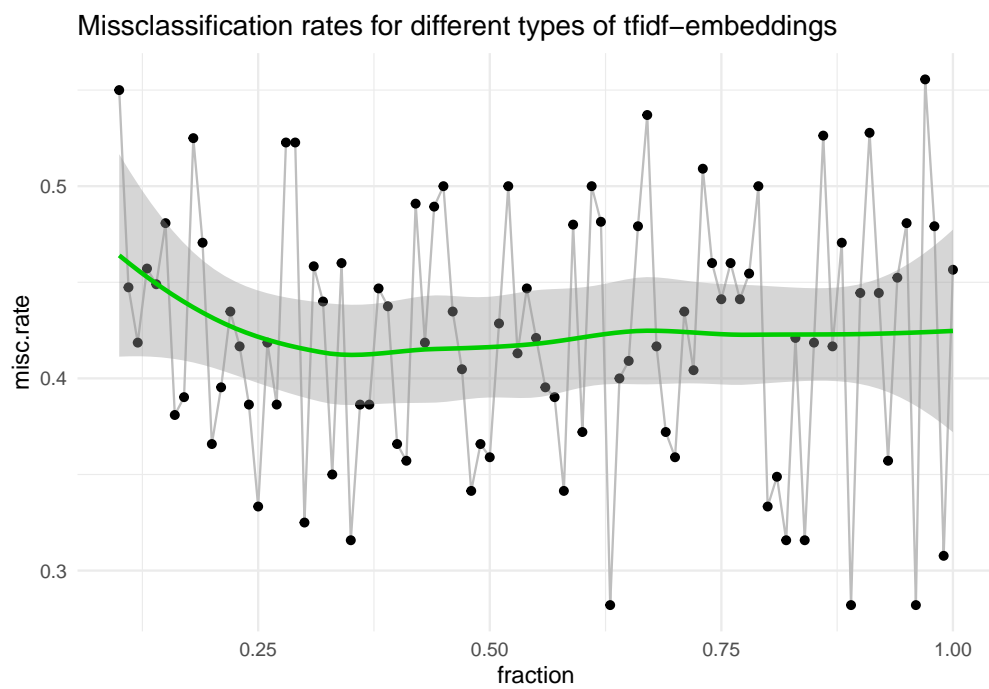


Figure 10: Fitting LDA via *tf-idf* embedding to EUROSTAT documents. Misclassification rates depending on the reduction of the bag-of-words. A big number of fraction stands for a bigger bag-of-words, i.e. less reduction. The green smoothed curve is a Loess kernel.

lowest misclassification ratio. The highest one in this comparison is observed for the fit of the *tf-idf* 0.5 embedding. In between is the accuracy of the model with *tf-2* embedding. So consistent to the Gutenberg examples *tf-2* embedding outperforms *tf-idf* embedding with fraction 0.5. Due to the lack of the results of a man-made clustering in a real life example, there is no way to optimize the parameter for every use case. In summary, *tf-2* embedding remains the better robust alternative in case of a generic problem.

Table 6: LDA via Gibbs Sampling - different embedding methods

embedding method	<i>tf-2</i>	<i>tf-idf</i> 0.50	<i>tf-idf</i> 0.35
missc. rate	0.342	0.359	0.316
time	41.605	17.702	14.494

Still, these results are worse than what experienced in the Gutenberg data examples. But note that the use case differs considerably from the first example, and that there are fewer documents analyzed. The fact that fewer documents are used for training has a strong effect on the accuracy of the model, especially if there are more groups. In comparison to the Gutenberg examples, both apply, there are fewer documents and more clusters. It needs to be emphasized that LDA is even less suitable for classification in this case, because there are already fewer documents in general.

It is advisable not to leave the entire clustering process of the corpus to LDA itself, but to find a suitable solution together with the assessment of an expert. This approach requires an expert, simply to determine how many categories are to be distinguished. Usually the user of the model coincides with the expert, but this does not have to be the case. In this case I applied the model to the data and the expert is Prof. Grossmann. He stated, that “a clustering with LDA makes sense and is useful as a reference point for a classification for experts and provides important guidance [for the user]”. I totally agree with this opinion, and recommend to use this clustering model alongside with the assessment of an expert.

5.2 ANN applied to EUROSTAT Documents

For a generic approach, to group and classify documents it already turned out, that LDA is not the best choice when it comes to the classification part. This is especially true for corpora with few documents, as in the case of EUROSTAT data. This is why I recommend using ANNs for classification. In the examples examining Gutenberg data, I outline a specific net structure, that performed very well for this type of problem. The aim of this section

is to evaluate if the net structure performs equally for this type of problem.

In this example the exact same approach is used as in section 4.2. The network contains two hidden layers, each containing 64 and 46 neurons. The output layer contains as many neurons as groups in the training data. The network was trained using 5 epochs and a batch size of 512. The models were evaluated performing 49 splits with a proportion of 10% test data, 70% training data and 20% validation data. Obviously, these hyperparameters might be improved with the aid of more computational power. But the aim is to outline that ANN are the primary choice for the classification process. This assumption is supported by the fact that the classification using ANNs provides fairly good results even for a corpus of regulatory documents.

The purpose of this paper is to find a generic approach for clustering and classifying documents, which does not preclude tuning the hyperparameters of the ANN. In contrast to clustering, there is already a categorization for the task of classifying. This gives the user the possibility to tune the model for classification (in this case an ANN) as desired. As with any classification model, it must of course be ensured that overfitting is prevented. In this paper I tried to tackle this problem by splitting the data set into training-, validation- and testing data. Due to the limited computational power, it was not possible to try many different net structures and optimize the hyperparameters. However, I recommend this procedure when applying this clustering and classifying approach.

A Appendix

References

- [A. Gelman, 2014] A. Gelman, J. Carlin, H. S. D. D. A. V. D. R. (2014). *Bayesian Data Analysis*. Chapman and Hall/CRC.
- [AP Dempster, 1977] AP Dempster, NM Laird, D. R. (1977). Maximum likelihood from incomplete data via the em algorithm. *Journal of the Royal Statistical Society*.
- [Bahjat, 2006] Bahjat, F., e. a. (2006). Multivariate logistic models. *Biometrika Trust*, 93(4):1011–1017.
- [Bishop, 2006] Bishop, C. M. (2006). *Pattern Recognition and Machine Learning*. Springer.
- [Blei, 2012] Blei, D. (2012). Probabilistic topic models. *Communications of the ACM*, 55(4):77.
- [Blei, 2003] Blei, D., N. N. J. M. (2003). Latent dirichlet allocation. *Journal of Machine Learning Research*, 3.
- [Chollet, 2018] Chollet, F., A. W. (2018). *Deep Learning with R*. Manning.
- [DeJong, 1979] DeJong, G. (1979). Prediction and substantiation: a new approach to natural language processing. *Cogn. Sci.*, 3(3):251–271.
- [Eurostat, 2019] Eurostat (2019). Ess vision 2020 admin (administrative data sources).
- [Grossmann, 2004] Grossmann, D., F. O. (2004). *Information Retrieval - Algorithms and Heuristics*. Springer, 2 edition.
- [Harris, 1951] Harris, Z. (1951). Methods in structural linguistics. *Linguistic Society of America*.
- [Hart,] Hart, M. Project gutenber.
- [Hecht-Nielsen, 1988] Hecht-Nielsen, R. (1988). Neurocomputing: picking the human brain. *IEEE Spectrum*, 25(3):36–41.
- [Hornik, 2011] Hornik, K., B. G. (2011). topicmodels: An r package for fitting topic models. *Journal of Statistical Software*, 40(13).
- [Jacobs, 1993] Jacobs, P., R. L. (1993). Innovations in text interpretation. *Artificial Intelligence*, 63:143–191.

- [Jordan, 1999] Jordan, M., e. a. (1999). An introduction to variational methods for graphical models. *Kluwer Academic Publishers - Machine Learning*, 37:183–233.
- [M. Steyvers, 2006] M. Steyvers, T. G. (2006). Probabilistic topic models. *Latent Semantic Analysis: A Road to Meaning*.
- [Manning, 1999] Manning, C., S. H. (1999). *Foundations of Natural Language Processing*. MIT press.
- [Martinez, 2010] Martinez, A. (2010). Natural language processing. *John Wiley and Sons, Inc.*, 2:253–257.
- [McCulloch and Pitts, 1943] McCulloch, W. and Pitts, W. (1943). A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*.
- [Minsky and Papert, 1969] Minsky, M. and Papert, S. (1969). *Perceptrons: An Introduction to Computational Geometry*. MIT press.
- [Mukherjee, 2019] Mukherjee, S. (2019). Lecture notes for probabilistic machine learning. Duke University.
- [Noble, 1988] Noble, H. (1988). *Natural Language Processing*. Blackwell Scientific Publications.
- [Powieser, 2012] Powieser, M. (2012). Latent dirichlet allocation in r. Master’s thesis, Vienna University of Economics and Business.
- [Robinson, 2018] Robinson, D. (2018). *Package ‘gutenbergr’*.
- [Rosenblatt, 1958] Rosenblatt, F. (1958). The perceptron, a probabilistic model for information storage and organization in the brain. *The Psychological Review*.
- [Silge and Robinson, 2017] Silge, J. and Robinson, D. (2017). *Text Mining with R: A Tidy Approach*. O’Reilly Media.
- [Silge, 2019] Silge, J. e. a. (2019). *Package ‘tidytext’*.
- [Wainwright and Jordan, 2008] Wainwright, M. J. and Jordan, M. I. (2008). Graphical models, exponential families, and variational inference. *Foundations and Trends in Machine Learning*, 1(1–2):1–305.
- [Werbos, 1974] Werbos, P. (1974). *Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences*. PhD thesis, Harvard University.
- [Winter, 2017] Winter, K., R.-M. S. G. W. F. I. M. W. (2017). Characterizing regulatory documents and guidelines based on text mining. *Lecture Notes in Computer Science*, 10573.

List of Figures

1	Well-established plate diagram for the standard LDA model extended by the parameter δ . The slightly bigger box represents the generative model of the corporis M documents. The smaller plate represents the iterative generation process of the N words of each document with the aid of the topics. See also "smoothed LDA model" in [Blei, 2003] for comparisons.	6
2	Schematic diagram of a simple perceptron by [Rosenblatt, 1958]	11
3	Schematic diagram of an adapted Rosenblatt-perceptron network	12
4	Schematic diagram of an multi layer perceptron (MLP) . . .	12
5	Fitting time of Example 1 (6 Books) depending on the reduction of the bag-of-words. A big number of fraction stands for a bigger bag-of-words, i.e. less reduction. The blue smoothed curve is a Loess kernel. The drop in fitting time for the last few values of fraction might result from parallelization reasons.	18
6	Misclassification rate depending on the reduction of the bag-of-words. A big number of fraction stands for a bigger bag-of-words, i.e. less reduction. The blue and red smoothed curves are a Loess kernels.	19
7	Boxplot of the all examples analyzed. Split for the number of categories (books) as well as for the calculation procedures (VEM algorithm and Gibbs sampling).	20
8	The network contains 3 layers. The input data is the the bag-of-words, which has dimension V . The first two layers are hidden, containing 64 and 46 neurons respectively. The last layer leads to the k classes (here the number of topics or books). The activation function is softmax.	21
9	Wordclouds for the clustered topics via LDA – using tfidf word proportions	27
10	Fitting LDA via <i>tf-idf</i> embedding to EUROSTAT documents. Misclassification rates depending on the reduction of the bag-of-words. A big number of fraction stands for a bigger bag-of-words, i.e. less reduction. The green smoothed curve is a Loess kernel.	28

List of Tables

1	Example book corpus	14
2	Example of stop words from tidytext package	15
3	Example 1 (6 books): Performance for different embeddings .	22
4	Example 2 (6 books): Performance for different embeddings .	22

5	Entire list of documents	24
6	LDA via Gibbs Sampling - different embedding methods . . .	29

Gutenberg Data via LDA - Documentation

Sebastian Knigge

6 8 2019

Contents

1	Setup	1
2	Example 1 (6 Books)	1
2.1	Get Data (Sampling)	1
2.2	Reduction of Dimensionality	5
2.3	Application of LDA Model to Full Corpus	5
2.3.1	Fitting via VEM	5
2.3.2	Comparison to Fit via Gibbs Sampling	9
2.4	Evaluate Model on Test Set	10
3	Optimal TFIDF reduction	12
4	Definition of Validation	16
5	Example 3 (10 books)	17
5.1	In Model Evaluation	21

1 Setup

Following packages were used in this script:

```
# loading packages
library(gutenbergr)
library(dplyr)
library(tidyr)
library(stringr)
library(tidytext)
library(udpipe)
library(topicmodels)
library(ggplot2)
library(parallel)
library(foreach)
```

2 Example 1 (6 Books)

2.1 Get Data (Sampling)

Sampling of the books and converting it to the *tidytext*-format is the essential step for setting up the LDA model. Following functions include the sampling procedure of the *gutenbergr* library.

```
sampling_books <- function(seed=1234, n=20){
  # sample n books from the whole library
  set.seed(seed)
```

```

gutenberg_works() %>%
  # select works with title
  dplyr::filter(!is.na(title)&!is.na(gutenberg_bookshelf)) %>%
  # set the sample sitze
  sample_n(n) %>%
  # set a special download link
  gutenberg_download(
    mirror = "http://mirrors.xmission.com/gutenberg/")
}

set_up_books <- function(n_books=4, seed=1992){
  # initial book sample
  books <- sampling_books(n=n_books, seed=seed)
  by_chapter <- books %>%
    group_by(gutenberg_id) %>%
    # split in chapters
    mutate(chapter = cumsum(str_detect(text, regex("^chapter ", ignore_case = TRUE)))) %>%
    ungroup() %>%
    # exclude books without chapters
    dplyr::filter(chapter > 0)
  return(by_chapter)
}

shorten_titles <- function(titles){
  # shorten very long book titles by setting
  # a subset of characters of the first line
  # of the title
  sub_inds <- titles %>%
    regexr(pattern="\\n|\\r")-1
  sub_inds[sub_inds<0] <- nchar(titles)[sub_inds<0]
  sub_inds <- pmin(sub_inds, 45)
  titles %>%
    substr(1,sub_inds)
}

get_titles <- function(x, n_books){
  # get the sampled gutenberg_ids
  unique_ids <- x %>%
    select(gutenberg_id) %>%
    unique() %>% unlist()
  # get the titles
  titles <- gutenberg_works() %>%
    dplyr::filter(gutenberg_id %in% unique_ids) %>%
    select(gutenberg_id, title, author) %>%
    mutate(title=shorten_titles(title))
  # get the number of gutenberg ids
  len <- nrow(titles)
  if(n_books!=len) warning(paste("--- ",n_books-len,
    " books have 0 chapters --- "))
  # the output as a list
  ret <- list(
    titles=titles,
    len=len
  )
}

```

```

)
return(ret)
}

append_by_chapter <- function(x=by_chapter, n_books, seed_index=1){
  # append the books matrix until
  # we get the desired number of books n_books
  titles <- get_titles(x, n_books)
  n <- titles$len
  while (n<n_books) {
    book2add <- sampling_books(n=1, seed=seed_index)
    by_chapter_add <- book2add %>%
      group_by(gutenberg_id) %>%
      # split in chapters
      mutate(chapter = cumsum(str_detect(text, regex("^chapter ", ignore_case = TRUE)))) %>%
      ungroup() %>%
      # exclude books without chapters
      dplyr::filter(chapter > 2)
    titles2add <- get_titles(by_chapter_add, 1)
    # adding the book to by_chapter if there are chapters in the
    # book plus it is not in the data already
    if (titles2add$len==1) if(!titles2add$titles$gutenberg_id%in%titles$titles$gutenberg_id) {
      x <- bind_rows(x, by_chapter_add)
    }
    n<-get_titles(x, n)$len
    seed_index <- seed_index+1
  }
  return(x)
}

exclude_stop_words <- function(x){
  # unite chapter and document title
  by_chapter_word <- x %>%
    unite(document, gutenberg_id, chapter) %>%
    # split into words
    unnest_tokens(word, text)
  # import tibble stop words
  data(stop_words)
  # find document-word counts
  word_counts <- by_chapter_word %>%
    # exclude stop words
    anti_join(stop_words) %>%
    # count each word by chapter
    count(document, word, sort = TRUE) %>%
    ungroup()
  return(word_counts)
}

convert_to_dtm <- function(x, minfq = 2){
  # get into a format lda can handle
  chapters_dtm <- x %>%
    select(doc_id=document, term=word, freq=n) %>%
    document_term_matrix() %>%

```

```

    # reduce by low frequencies
    dtm_remove_lowfreq(minfreq = minfq)
  return(chapters_dtm)
}

convert_to_dtm_2 <- function(x, n=n, minfq = 2, top=10000){
  # get into a format lda can handle
  chapters_dtm <- x %>%
    select(doc_id=document, term=word, freq=n) %>%
    document_term_matrix() %>%
    # reduce by low frequencies
    dtm_remove_tfidf(top=top)
  return(chapters_dtm)
}

```

Now we can use these functions to get to the initial corpus sample. In this example 6 books are chosen.

```

n_books <- 6
by_chapter <- set_up_books(n_books=n_books, seed=222)
get_titles(by_chapter, n_books)

```

```

## Warning in get_titles(by_chapter, n_books): --- 5 books have 0 chapters ---
## $titles
## # A tibble: 1 x 3
##   gutenber_id title                                author
##   <int> <chr>                                <chr>
## 1      2095 Clotelle: A Tale of the Southern States Brown, William Wells
##
## $len
## [1] 1

```

The function `set_up_books()` returns a warning that several books seem to consist of only one chapter. In order to get a corpus consisting out of 6 books, the function `append_by_chapter()` is used, which fills up the corpus to the desired number of books.

```

appended_by_chapter <- append_by_chapter(x=by_chapter, n_books = n_books)
word_counts <- exclude_stop_words(appended_by_chapter)

```

```
## Joining, by = "word"
```

In table 7 the sampled titles for the book sample with the seed 222 are displayed. It appears through the function `append_by_chapter()` one book was added, called “Clotelle: A Tale of the Southern States”.

```

titles <- get_titles(appended_by_chapter, n_books)
titles$titles %>% stargazer(summary=FALSE, font.size = "footnotesize",
                           header=FALSE, title="Book-titles", rownames=FALSE,
                           label="titles:6books")

```

We also want to check if the book categories (i.e. gutenber bookshelves) are different. See Table (2) for comparison.

```

gbids <- titles$titles$gutenber_id
categories <- gutenber_works() %>%
  filter(gutenber_id %in% gbids) %>%
  select(gutenber_id, gutenber_bookshelf)
categories %>% stargazer(summary=FALSE, font.size = "footnotesize",

```

Table 1: Book-titles

gutenberg_id	title	author
2095	Clotel: A Tale of the Southern States	Brown, William Wells
6315	The Awakening of Helena Richie	Deland, Margaret Wade Campbell
6971	Judaism	Abrahams, Israel
7635	The Disowned — Volume 05	Lytton, Edward Bulwer Lytton, Baron
10319	Dave Darrin's Third Year at Annapolis; Or, Le	Hancock, H. Irving (Harrie Irving)
21039	Boycotted, and Other Stories	Reed, Talbot Baines

```
header=FALSE, title="Book-categories", rownames=FALSE,
label="categories:6books")
```

Table 2: Book-categories

gutenberg_id	gutenberg_bookshelf
2095	African American Writers
6315	Bestsellers, American, 1895-1923
6971	Judaism
7635	Historical Fiction
10319	Children's Book Series
21039	School Stories

Obviously the corpus is very diverse. It is a good sample, in order to try to cluster the chapters of the books with the aid of LDA.

2.2 Reduction of Dimensionality

In the set up we have another parameter to adjust. The function `convert_to_dtm` takes the parameter `minfq`, which is used to reduce the “bag of words” (i.e. dimensionality). `minfq` is the minimum frequency for the bag of words dictionary. I will refer to this as “embedding”. Let us set it to 2 in this case, meaning that we include a word only if the frequency is 2 or more.

```
chapters_dtm <- convert_to_dtm(word_counts, minfq=2)
( M_f2 <- ncol(chapters_dtm) )
```

```
## [1] 8597
```

Let us compare it to the case including all words.

```
chapters_dtm_all <- convert_to_dtm(word_counts, minfq=0)
( M <- ncol(chapters_dtm_all) )
```

```
## [1] 15186
```

We also want to compare this to a reduction of the word dictionary by the TFIDF. We are using a reduction by 50% of the dimension of the original bag of words.

```
chapters_dtm_tfidf <- convert_to_dtm_2(word_counts, top=(0.1*M))
ncol(chapters_dtm_tfidf)
```

```
## [1] 1518
```


2.3 Application of LDA Model to Full Corpus

2.3.1 Fitting via VEM

We set up the LDA model for the shrunked embedding corpus via frequency=2. In a first try we are using the default “VEM-Algorithm” to fit the model.

```
tim1 <- Sys.time()
chapters_lda <- LDA(chapters_dtm,
                    k = n_books, control = list(seed = 1234))
tim2 <- Sys.time()
u_1 <- tim2-tim1
```

In comparison we will set up the LDA model for the full word embedding corpus.

```
tim1 <- Sys.time()
chapters_lda_all <- LDA(chapters_dtm_all,
                       k = n_books, control = list(seed = 1234))
tim2 <- Sys.time()
u_all <- tim2-tim1
```

The third LDA fit builds up on data from the shrunked dictionary/bag of words by TFIDF.

```
tim1 <- Sys.time()
chapters_lda_tfidf <- LDA(chapters_dtm_tfidf,
                          k = n_books, control = list(seed = 1234))
tim2 <- Sys.time()
u_tfidf <- tim2-tim1
chapters_lda
```

A LDA_VEM topic model with 6 topics.

Now we evaluate the model all in once - that is - we analyze the clustering on the entire data set.

```
ext_gamma_matrix <- function(model){
  # get gamma matrix for chapter probabilities
  chapters_gamma <- tidy(model, matrix = "gamma")
  # split joint name of book and chapter
  chapters_gamma <- chapters_gamma %>%
    separate(document, c("gutenberg_id", "chapter"), sep = "_", convert = TRUE)
  # get matrix with probabilities for each topic per chapter
  # this matrix is just information and will in this form of the function
  # not be returned
  gamma_per_chapter <- chapters_gamma %>%
    spread(topic, gamma)
  return(chapters_gamma)
}

validate_LDaclassification <- function(gamma_matrix){
  # gamma_matrix is an object of the function ext_gamma_matrix()

  #First we'd find the topic that was most associated with
  # each chapter using top_n(), which is effectively the
  # "classification" of that chapter
  chapter_classifications <- gamma_matrix %>%
    group_by(gutenberg_id, chapter) %>%
    top_n(1, gamma) %>%
```

```

ungroup()

# We can then compare each to the "consensus"
# topic for each book (the most common topic among its chapters),
# and see which were most often misidentified.
book_topics <- chapter_classifications %>%
  count(gutenberg_id, topic) %>%
  group_by(gutenberg_id) %>%
  # just keep the most frequent one
  top_n(1, n) %>%
  ungroup() %>%
  # keep title called consensus and topic
  transmute(consensus = gutenberg_id, topic)

# check the fraction of missclassification
Join <- chapter_classifications %>%
  inner_join(book_topics, by = "topic")
# mismatches
Join %>% dplyr::filter(gutenberg_id != consensus) %>%
  nrow()/nrow(Join)
}

```

Now we exclude for each of the 3 embeddings the - so called - beta matrix and compare the most likely result with the real results. Depending on how good the LDA will separate the books, this influences the goodness of the fit.

```

misc.rate_1 <- ext_gamma_matrix(chapters_lda) %>%
  validate_LDAClassification()

```

```

misc.rate_all <- ext_gamma_matrix(chapters_lda_all) %>%
  validate_LDAClassification()

```

```

misc.rate_tfidf <- ext_gamma_matrix(chapters_lda_tfidf) %>%
  validate_LDAClassification()

```

The following matrix gives an overview of the fitting time and the results of the 3 different fits.

```

performance_matrix <- data.frame(freq2.embedding=c(misc.rate_1, u_1),
  all.embedding=c(misc.rate_all, u_all),
  tfidf=c(misc.rate_tfidf, u_tfidf))
rownames(performance_matrix) <- c("missc. rate", "time")
performance_matrix %>% stargazer(summary=FALSE, header=F, title = "LDA via VEM")

```

Table 3: LDA via VEM

	freq2.embedding	all.embedding	tfidf
missc. rate	0.385	0.462	0.383
time	12.800	9.911	0.492

We run the same calculations for a different sample. This means that we set up a new random sample from the Gutenberg books by again using the sample function with another randomness factor (seed). Again, it has to be a sample that is as diverse as possible, i.e. contains books from different categories. In the following we set up again three bag of words, each using the three embedding methods mentioned above. Lastly, we perform the same evaluation method as just seen to evaluate the method for this sample as well.

```

n_books_sec <- 6
by_chapter_sec <- set_up_books(n_books=n_books_sec, seed=101)
appended_by_chapter_sec <- append_by_chapter(x=by_chapter_sec, n_books = n_books_sec)
word_counts_sec <- exclude_stop_words(appended_by_chapter_sec)

```

```

## Joining, by = "word"
titles_sec <- get_titles(appended_by_chapter_sec, n_books)
gbids_sec <- titles_sec$titles$gutenberg_id
categories_sec <- gutenbergs_works() %>%
  filter(gutenberg_id %in% gbids_sec) %>%
  select(gutenberg_id, gutenberg_bookshelf)

```

```

# embedding 1
chapters_dtm_sec <- convert_to_dtm(word_counts_sec, minfq=2)
ncol(chapters_dtm_sec)

```

[1] 8565

```

# embedding 2
chapters_dtm_all_sec <- convert_to_dtm(word_counts_sec, minfq=0)
( M2 <- ncol(chapters_dtm_all_sec) )

```

[1] 15328

```

# embedding 3
chapters_dtm_tfidf_sec <- convert_to_dtm_2(word_counts_sec, top=(M2*0.5))

```

```

# embedding 1
tim1 <- Sys.time()
chapters_lda_sec <- LDA(chapters_dtm_sec,
  k = n_books, control = list(seed = 1234))
tim2 <- Sys.time()
u_1 <- tim2-tim1

```

```

# embedding 2
tim1 <- Sys.time()
chapters_lda_all_sec <- LDA(chapters_dtm_all_sec,
  k = n_books, control = list(seed = 1234))
tim2 <- Sys.time()
u_all <- tim2-tim1

```

```

# embedding 3
tim1 <- Sys.time()
chapters_lda_tfidf_sec <- LDA(chapters_dtm_tfidf_sec,
  k = n_books, control = list(seed = 1234))
tim2 <- Sys.time()
u_tfidf <- tim2-tim1

```

```

# embedding 1
misc.rate_1_sec <- ext_gamma_matrix(chapters_lda_sec) %>%
  validate_LDAClassification()

```

```

# embedding 2
misc.rate_all_sec <- ext_gamma_matrix(chapters_lda_all_sec) %>%
  validate_LDAClassification()

```

```

# embedding 3
misc.rate_tfidf_sec <- ext_gamma_matrix(chapters_lda_tfidf_sec) %>%
  validate_LDAClassification()

# overview
performance_matrix <- data.frame(freq2.embedding=c(misc.rate_1_sec, u_1),
  all.embedding=c(misc.rate_all_sec, u_all),
  tfidf=c(misc.rate_tfidf_sec, u_tfidf))
rownames(performance_matrix) <- c("missc. rate", "time")
performance_matrix %>% stargazer(summary=FALSE, header=F, title = "LDA via VEM second sample")

```

Table 4: LDA via VEM second sample

	freq2.embedding	all.embedding	tfidf
missc. rate	0.227	0.362	0.222
time	8.244	8.191	1.664

Surprisingly, the run-time to fit the LDA model for the embedding using all words, does not take way longer than the embedding using a lower frequency. The fit of the model with the reduced bag of words via tfidf takes considerably less time.

2.3.2 Comparison to Fit via Gibbs Sampling

For comparison, we will check the results and the run-time for the fit via Gibbs Sampling.

```

tim1 <- Sys.time()
chapters_lda_Gibbs <- LDA(chapters_dtm, method="Gibbs",
  k = n_books, control = list(seed = 1234))
tim2 <- Sys.time()
u_1_Gibbs <- tim2-tim1

tim1 <- Sys.time()
chapters_lda_all_Gibbs <- LDA(chapters_dtm_all, method = "Gibbs",
  k = n_books, control = list(seed = 1234))
tim2 <- Sys.time()
u_all_Gibbs <- tim2-tim1

tim1 <- Sys.time()
chapters_lda_tfidf_Gibbs <- LDA(chapters_dtm_tfidf, method="Gibbs",
  k = n_books, control = list(seed = 1234))
tim2 <- Sys.time()
u_tfidf_Gibbs <- tim2-tim1

misc.rate_1_Gibbs <- ext_gamma_matrix(chapters_lda_Gibbs) %>%
  validate_LDAClassification()

misc.rate_all_Gibbs <- ext_gamma_matrix(chapters_lda_all_Gibbs) %>%
  validate_LDAClassification()

misc.rate_tfidf_Gibbs <- ext_gamma_matrix(chapters_lda_tfidf_Gibbs) %>%
  validate_LDAClassification()

```

```
performance_matrix <- data.frame(freq2.embedding=c(misc.rate_1_Gibbs, u_1_Gibbs),
                                all.embedding=c(misc.rate_all_Gibbs, u_all_Gibbs),
                                tfidf=c(misc.rate_tfidf_Gibbs, u_tfidf_Gibbs))
rownames(performance_matrix) <- c("missc. rate", "time")
performance_matrix %>% stargazer(summary=FALSE, header=F, title = "LDA via Gibbs sampling")
```

Table 5: LDA via Gibbs sampling

	freq2.embedding	all.embedding	tfidf
missc. rate	0.051	0.043	0.077
time	18.359	22.408	7.649

Apparently Gibbs Sampling takes a bit longer than the VEM algorithm, but its results with regards to the correct “classification” (missclassification rate) are way better.

Again we try the second sample, using a different seed. Only Gibbs Sampling is evaluated in this section.

```
tim1 <- Sys.time()
chapters_lda_Gibbs_sec <- LDA(chapters_dtm_sec, method="Gibbs",
                             k = n_books, control = list(seed = 1234))
tim2 <- Sys.time()
u_1_Gibbs <- tim2-tim1

tim1 <- Sys.time()
chapters_lda_all_Gibbs_sec <- LDA(chapters_dtm_all_sec, method = "Gibbs",
                                 k = n_books, control = list(seed = 1234))
tim2 <- Sys.time()
u_all_Gibbs <- tim2-tim1

tim1 <- Sys.time()
chapters_lda_tfidf_Gibbs_sec <- LDA(chapters_dtm_tfidf_sec, method="Gibbs",
                                   k = n_books, control = list(seed = 1234))
tim2 <- Sys.time()
u_tfidf_Gibbs <- tim2-tim1

misc.rate_1_Gibbs_sec <- ext_gamma_matrix(chapters_lda_Gibbs_sec) %>%
  validate_LDAClassification()

misc.rate_all_Gibbs_sec <- ext_gamma_matrix(chapters_lda_all_Gibbs_sec) %>%
  validate_LDAClassification()

misc.rate_tfidf_Gibbs_sec <- ext_gamma_matrix(chapters_lda_tfidf_Gibbs_sec) %>%
  validate_LDAClassification()

# performance matrix
performance_matrix_sec <- data.frame(freq2.embedding=c(misc.rate_1_Gibbs_sec, u_1_Gibbs),
                                    all.embedding=c(misc.rate_all_Gibbs_sec, u_all_Gibbs),
                                    tfidf=c(misc.rate_tfidf_Gibbs_sec, u_tfidf_Gibbs))
rownames(performance_matrix) <- c("missc. rate", "time")
performance_matrix_sec %>% stargazer(summary=FALSE, header=F, title = "Gibbs second sample")
```

Table 6: Gibbs second sample

	freq2.embedding	all.embedding	tfidf
1	0.058	0.163	0.093
2	14.866	16.745	6.965

2.4 Evaluate Model on Test Set

First we split the data randomly into training- and test-sample.

```
split_for_fit <- function(data, test_ratio=0.03, seed=1234){
  # function to set up the training
  # and the testing set from the input data which
  # is an object from the function convert_to_dtm()
  set.seed(seed)
  N <- nrow(data)
  n_test <- (N*test_ratio) %>% ceiling
  test_ind <- sample(1:N, n_test)
  train_ind <- (1:N)[-test_ind]
  ret <- list(train=data[train_ind,],
              test=data[test_ind,])
  return(ret)
}
```

The `fit_n_evaluate()` function will fit the LDA model and evaluate the goodness of fit for an object of the function `split_for_fit`. The section Validation gives insights in the procedure of how the “consensus” is set up.

```
fit_n_evaluate <- function(split, k=n_books){
  LDA_model <- LDA(split$train, method="Gibbs",
                    k = k, control = list(seed = 1234))
  # use the predict function of udpipe
  # the topic predict function already extract the most likely topics
  prediction <- predict(LDA_model, newdata=split$test) %>% .$topic
  # get "consensus" via maximum likelihood
  # first extract the gamma matrix of the model fitted on the training
  # data
  chapters_gamma <- ext_gamma_matrix(LDA_model)
  spreaded_gamma <- chapters_gamma %>% spread(topic, gamma)
  # get pdfs
  plotm <- spreaded_gamma %>%
    group_by(gutenberg_id) %>%
    # note: pdfs are unnormalized
    summarise_at(2:(titles$len+1), sum)
  topic_link <- plotm %>%
    apply(1, function(x) which.max(x[2:length(x)])) %>%
    cbind(plotm$gutenberg_id) %>%
    as.data.frame()
  # exclude the
  consensus <- split$test %>%
    rownames() %>%
    substr(1, regexpr("_", .)-1) %>%
    as.numeric() %>%
    as.data.frame() %>%
```

```

    # merge it to the topic
    merge(topic_link, by.y="V2", sort=FALSE) %>%
    select(..y)
    # missclassification rate will be returned
    sum(consensus!=prediction)/length(prediction)
}

```

We now can evaluate several fits of a model for different splits. Here is a parallelization for the individual for loops applied. In this case only embedding via TFIDF is used.

```

# setting up how many cores to be used
useable_cores <- parallel::detectCores() - 1
# registering cluster
cl <- parallel::makeCluster(useable_cores)
doParallel::registerDoParallel(cl)
n <- 59
results <- foreach(i = 1:n, .combine = 'c', .export = ls(.GlobalEnv), .packages = c("dplyr", "udpipe",

    chapters_dtm_tfidf %>%
    split_for_fit(seed=12*i) %>%
    fit_n_evaluate()

}

```

```

## Warning in e$fun(obj, substitute(ex), parent.frame(), e$data):
## already exporting variable(s): chapters_dtm_tfidf, ext_gamma_matrix,
## fit_n_evaluate, n_books, split_for_fit, titles
parallel::stopCluster(cl)

```

This is the output of the simulation over 59 splits and fits. The mean is the final result:

```

results %>% summary

##      Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
## 0.0000  0.0000  0.2500  0.3093  0.5000  1.0000

```

3 Optimal TFIDF reduction

We want to visualize what the best values for a reduction of the original bag-of-words via tfidf is. I.e. by what % should the bag-of-words be reduced for the best results (lowest misclassification ratio).

```

evaluation_for_embedding <- function(word_counts, frac=0.5) {
  # embedding 3
  chapters_dtm_tfidf <- convert_to_dtm_2(word_counts, top=(M2*frac))
  # fitting
  tim1 <- Sys.time()
  chapters_lda_tfidf_Gibbs <- LDA(chapters_dtm_tfidf, method="Gibbs",
                                k = n_books, control = list(seed = 1234))
  tim2 <- Sys.time()
  u_tfidf_Gibbs <- tim2-tim1
  # calculating missclassification rate
  misc.rate_tfidf_Gibbs <- ext_gamma_matrix(chapters_lda_tfidf_Gibbs) %>%

```

```

    validate_LDAClassification()
    # function returns a vector including the misc. ratio and the fitting time
    return(c(misc.rate_tfidf_Gibbs,
             as.numeric(u_tfidf_Gibbs)))
}

# set up the fractions we want to use
# we use 0.1, 0.2, ..., 1
fractions <- seq(0.1,1,0.01)

# Use parallelization
# registering cluster
cl <- parallel::makeCluster(useable_cores)
doParallel::registerDoParallel(cl)
embedding_performance_matrix <- foreach(i = 1:length(fractions), .combine = 'rbind', .export = ls(.GlobalEnv))

    evaluation_for_embedding(word_counts, frac=fractions[i]) %>%
    c(.,fractions[i])

}

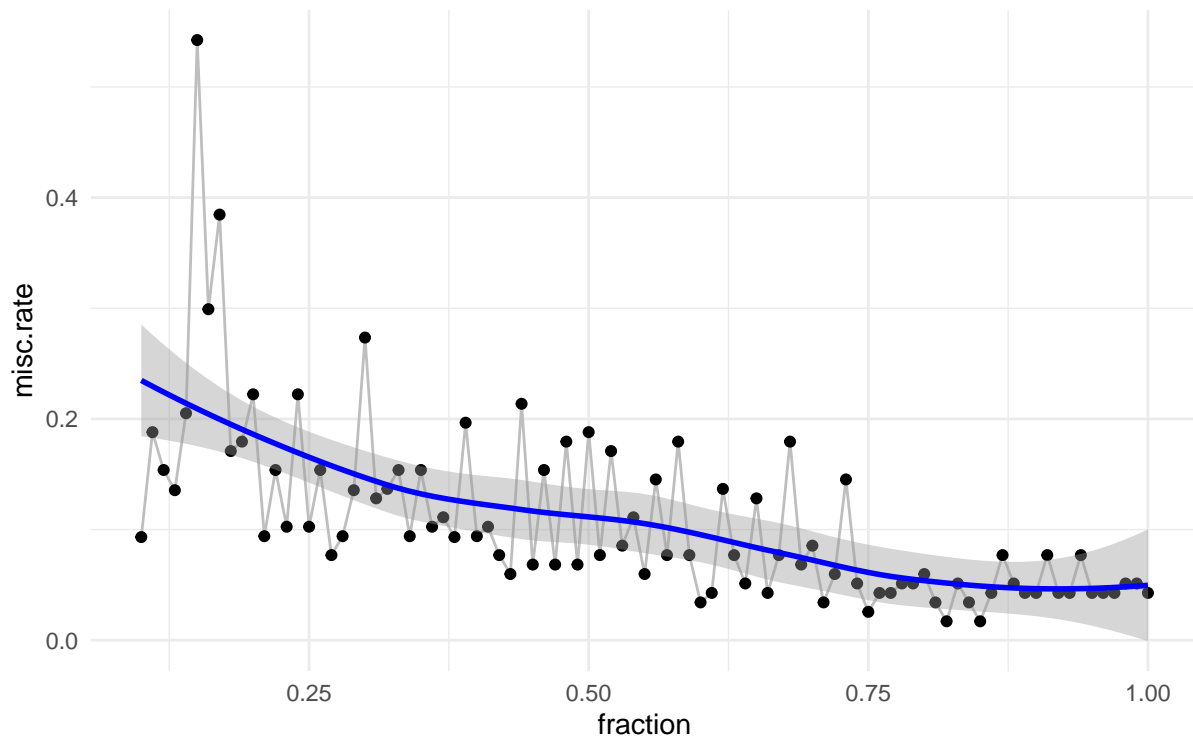
## Warning in e$fun(obj, substitute(ex), parent.frame(), e$data): already
## exporting variable(s): convert_to_dtm_2, evaluation_for_embedding,
## ext_gamma_matrix, fractions, M2, n, n_books, validate_LDAClassification,
## word_counts
parallel::stopCluster(cl)

# adjust the resulting matrix to a data frame for plotting
colnames(embedding_performance_matrix) <- c("misc.rate","time","fractions")
embedding_performance_matrix <- as.data.frame(embedding_performance_matrix)

ggplot(data=embedding_performance_matrix,
       aes(x=fractions, y=misc.rate)) +
  geom_line(col="grey") +
  geom_point() +
  geom_smooth(col="blue", method="loess", span=0.7) +
  theme_minimal() +
  ggtitle("Missclassification rates for different types of tfidf-embeddings \n Example 1 (6 Books)") +
  xlab("fraction")

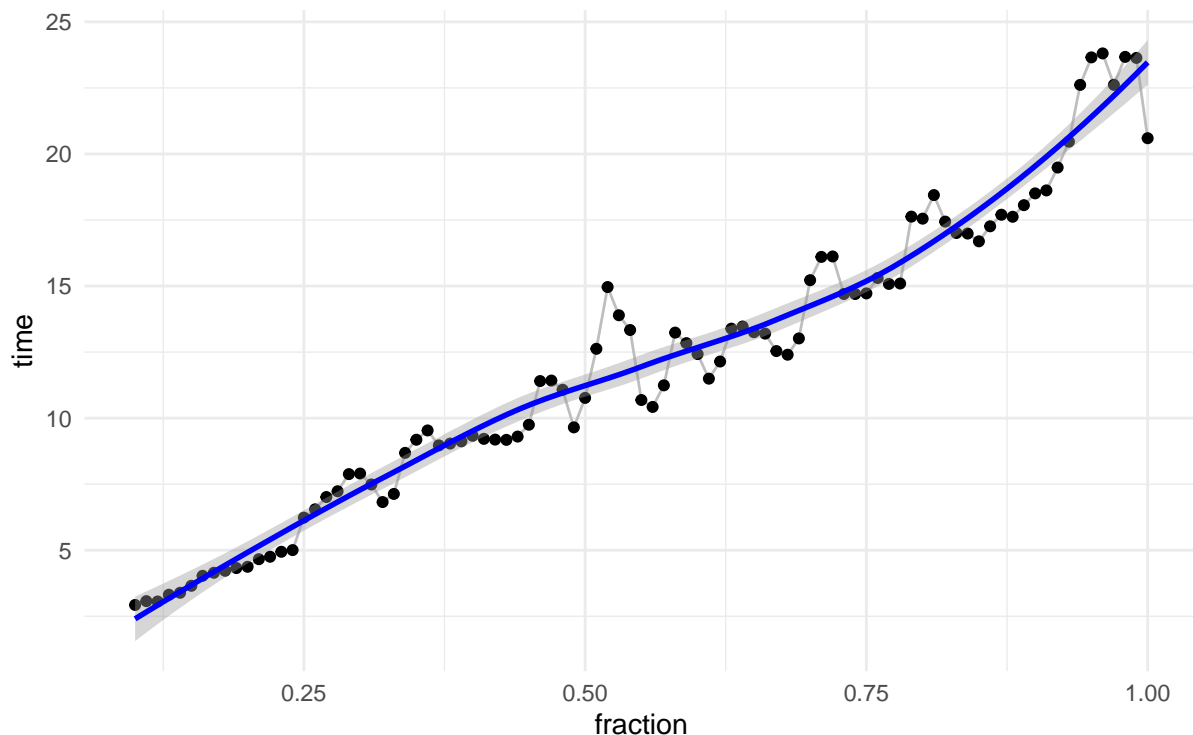
```


Missclassification rates for different types of tfidf-embeddings Example 1 (6 Books)



```
ggplot(data=embedding_performance_matrix,
  aes(x=fractions, y=time)) +
  geom_line(col="grey") +
  geom_point() +
  geom_smooth(col="blue", method="loess", span=0.7) +
  theme_minimal() +
  ggtitle("Fitting time for different types of tfidf-embeddings \n Example 1 (6 Books)") +
  xlab("fraction")
```

Fitting time for different types of tfidf-embeddings Example 1 (6 Books)



Now let us study Example 2.

```
# set up the fractions we want to use
# we use 0.1, 0.2, ..., 1
fractions <- seq(0.1,1,0.01)

# Use parallelization
# registering cluster
cl <- parallel::makeCluster(useable_cores)
doParallel::registerDoParallel(cl)
embedding_performance_matrix_sec <- foreach(i = 1:length(fractions), .combine = 'rbind', .export = ls(
  evaluation_for_embedding(word_counts_sec, frac=fractions[i]) %>%
    c(.,fractions[i])
})

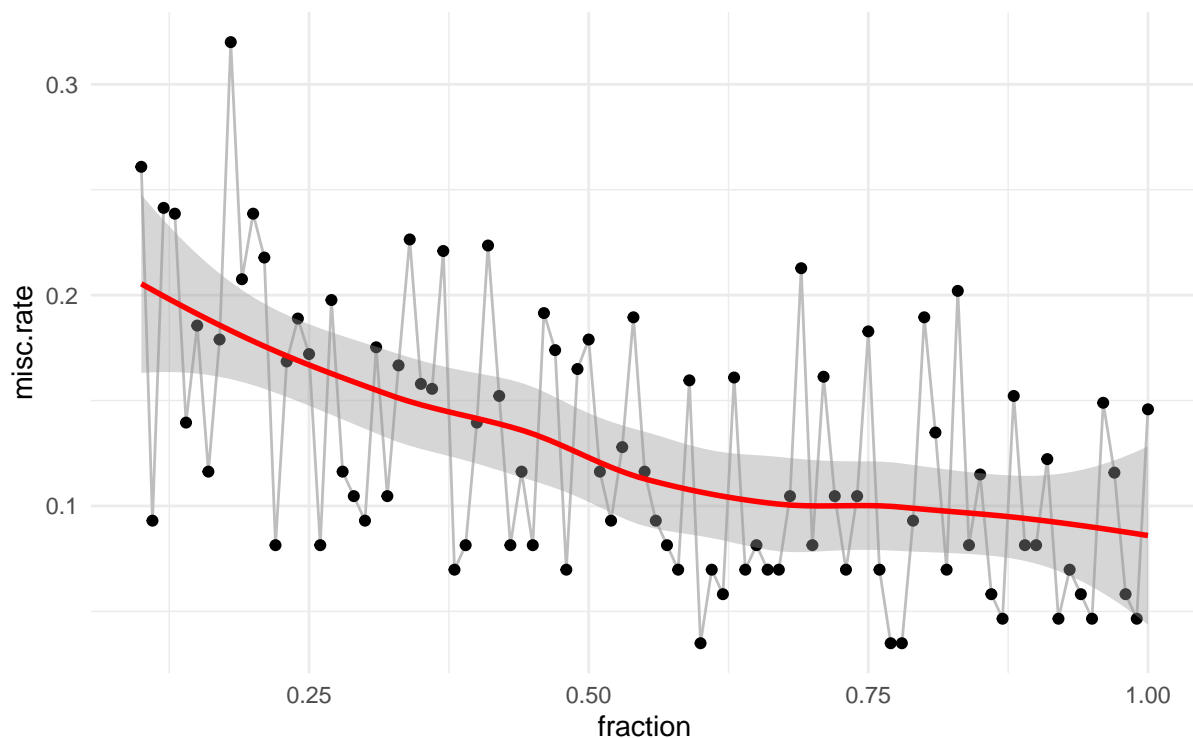
## Warning in e$fun(obj, substitute(ex), parent.frame(), e$data): already
## exporting variable(s): convert_to_dtm_2, evaluation_for_embedding,
## ext_gamma_matrix, fractions, M2, n, n_books, validate_LDClassification,
## word_counts_sec
parallel::stopCluster(cl)

# adjust the resulting matrix to a data frame for plotting
colnames(embedding_performance_matrix_sec) <- c("misc.rate","time","fractions")
```

```
embedding_performance_matrix_sec <- as.data.frame(embedding_performance_matrix_sec)
```

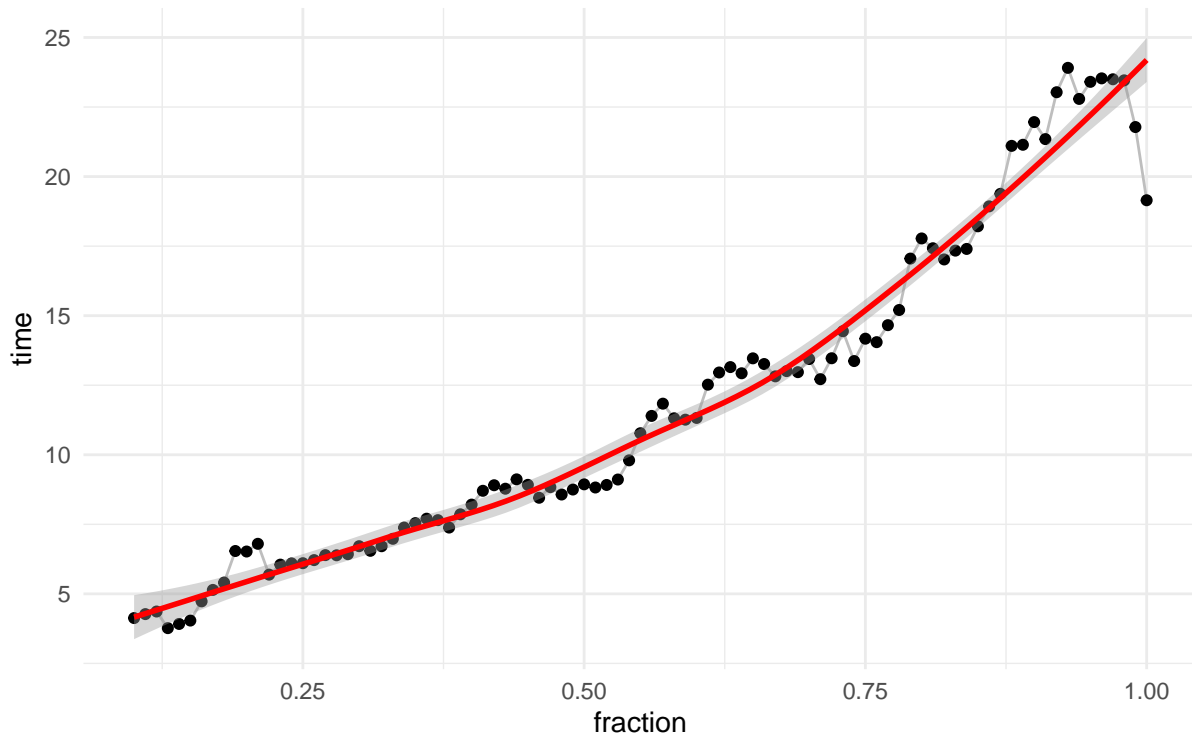
```
ggplot(data=embedding_performance_matrix_sec,
       aes(x=fractions, y=misc.rate)) +
  geom_line(col="grey") +
  geom_point() +
  geom_smooth(col="red", method="loess", span=0.7) +
  theme_minimal() +
  ggtitle("Missclassification rates for different types of tfidf-embeddings \n Example 2 (6 Books)") +
  xlab("fraction")
```

Missclassification rates for different types of tfidf-embeddings
Example 2 (6 Books)



```
ggplot(data=embedding_performance_matrix_sec,
       aes(x=fractions, y=time)) +
  geom_line(col="grey") +
  geom_point() +
  geom_smooth(col="red", method="loess", span=0.7) +
  theme_minimal() +
  ggtitle("Fitting time for different types of tfidf-embeddings \n Example 2 (6 Books)") +
  xlab("fraction") +
  xlab("fraction")
```

Fitting time for different types of tfidf-embeddings Example 2 (6 Books)



4 Definition of Validation

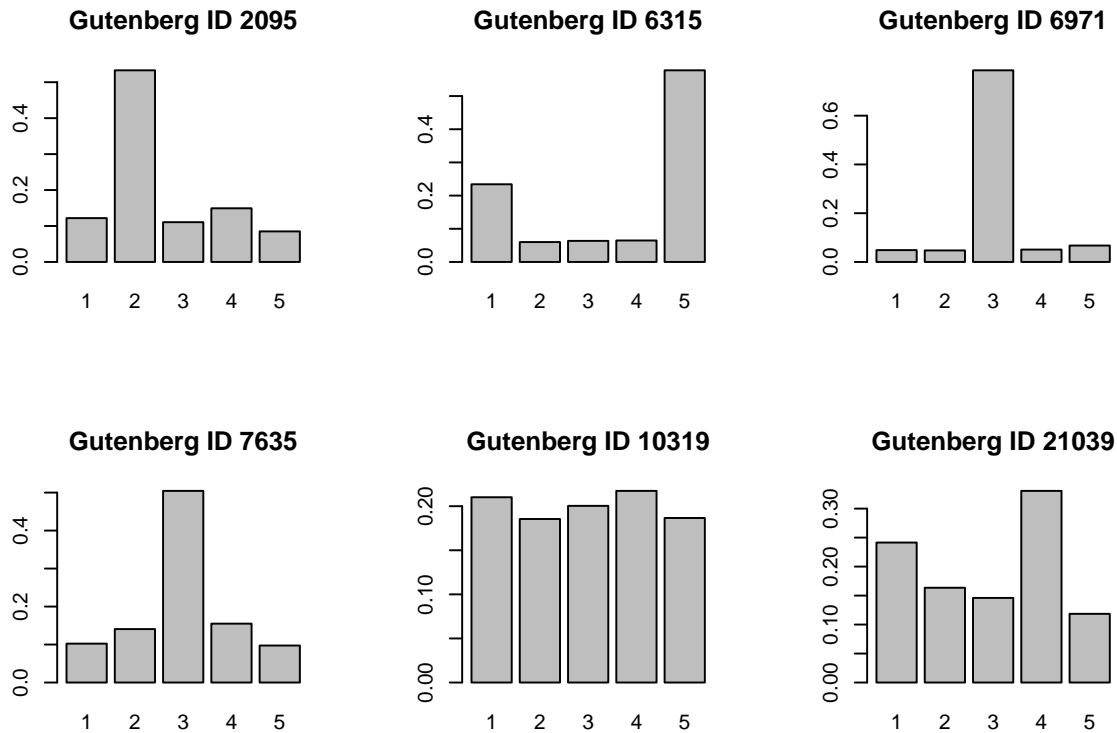
Since the LDA algorithm just clusters into k topics, it is necessary to evaluate which “topic” refers to which book, in order to calculate a missclassification rate. For each of the books we naively derive a distribution of the assignment to the topics. This is done by accumulating the distributions for each chapter of the book. The most likely assignment of the LDA model is chosen as the “correct” topic. Now calculating the missclassification rate is basically the fraction of those chapters/documents not coinciding with the most likely assignment.

```
LDA_model <- LDA(chapters_dtm_tfidf, method="Gibbs",
                 k = n_books, control = list(seed = 1234))
# get gamma matrix for chapter probabilities
chapters_gamma <- tidy(LDA_model, matrix = "gamma") %>%
  # split joint name of book and chapter
  separate(document, c("gutenberg_id", "chapter"), sep = "_", convert = TRUE)
spreaded_gamma <- chapters_gamma %>% spread(topic, gamma)
# get pdfs
plotm <- spreaded_gamma %>%
  group_by(gutenberg_id) %>%
  # note: pdfs are unnormalized
  summarise_at(2:(titles$len+1), sum)
topic_link <- plotm %>%
  apply(1, function(x) which.max(x[2:length(x)])) %>%
  cbind(plotm$gutenberg_id) %>%
  as.data.frame()
```

```

par(mfrow=c(2,3))
for (i in 1:n_books){
  vec <- plotm[i,2:n_books] %>% unlist()
  barplot(vec/sum(vec), main=paste("Gutenberg ID", plotm[i,1]))
}

```



This procedure does not exclude the fact that two books are assigned to the same topic (in this case e.g. 2 books are assigned to chapter 1 and none to chapter 6). But still, from each of the plots of the “distributions” you can obtain how good the chapters of a single book are classified. The more of the mass of the distribution is on a single topic, the better the chapters of this topic are predicted.

5 Example 3 (10 books)

First we will sample the books, the same way as it was done in Example 1.

```

n_books_10 <- 10
by_chapter10 <- set_up_books(n_books=n_books_10, seed=54321)
appended_by_chapter10 <- append_by_chapter(x=by_chapter10, n_books = n_books_10)
word_counts10 <- exclude_stop_words(appended_by_chapter10)

## Joining, by = "word"
titles <- get_titles(appended_by_chapter10, n_books)

## Warning in get_titles(appended_by_chapter10, n_books): --- -4 books have 0
## chapters ---
titles$titles %>% stargazer(summary=FALSE, font.size = "footnotesize",
                           header=FALSE, title="Book-titles Example with 10 Books", rownames=FALSE,
                           label="titles:6books")

```

Table 7: Book-titles Example with 10 Books

gutenberg_id	title	author
1401	Tarzan the Untamed	Burroughs, Edgar Rice
6315	The Awakening of Helena Richie	Deland, Margaret Wade Campbell
6971	Judaism	Abrahams, Israel
7635	The Disowned — Volume 05	Lytton, Edward Bulwer Lytton, Baron
10319	Dave Darrin's Third Year at Annapolis; Or, Le	Hancock, H. Irving (Harrie Irving)
11113	Principal Cairns	Cairns, John
13145	Lippincott's Magazine of Popular Literature a	Various
15735	History of the Negro Race in America From 161	Williams, George Washington
21039	Boycotted, and Other Stories	Reed, Talbot Baines
21710	The Crew of the Water Wagtail	Ballantyne, R. M. (Robert Michael)

```
gbids <- titles$titles$gutenberg_id
categories <- gutenberg_works() %>%
  filter(gutenberg_id %in% gbids) %>%
  select(gutenberg_id, gutenberg_bookshelf)
categories %>% stargazer(summary=FALSE, font.size = "footnotesize",
  header=FALSE, title="Book-categories", rownames=FALSE,
  label="categories:10books")
```

Table 8: Book-categories

gutenberg_id	gutenberg_bookshelf
1401	Adventure/Movie Books
6315	Bestsellers, American, 1895-1923
6971	Judaism
7635	Historical Fiction
10319	Children's Book Series
11113	Famous Scots Series
13145	Lippincott's Magazine
15735	Slavery
21039	School Stories
21710	Children's Fiction

```
# embedding 1
chapters_dtm_10 <- convert_to_dtm(word_counts10, minfq=2)
( M3_f2 <- ncol(chapters_dtm_10) )

[1] 17742

# embedding 2
chapters_dtm_all_10 <- convert_to_dtm(word_counts10, minfq=0)
( M3 <- ncol(chapters_dtm_all_10) )

[1] 29101

# embedding 3
chapters_dtm_tfidf_10 <- convert_to_dtm_2(word_counts10, top=(0.5*M3))

# embedding 1
tim1 <- Sys.time()
chapters_lda_10 <- LDA(chapters_dtm_10, method="Gibbs",
  k = n_books_10, control = list(seed = 1234))
tim2 <- Sys.time()
u_1 <- tim2-tim1
```

```

# embedding 2
tim1 <- Sys.time()
chapters_lda_all_10 <- LDA(chapters_dtm_all_10, method="Gibbs",
                          k = n_books_10, control = list(seed = 1234))
tim2 <- Sys.time()
u_all <- tim2-tim1

# embedding 3
tim1 <- Sys.time()
chapters_lda_tfidf_10 <- LDA(chapters_dtm_tfidf_10, method="Gibbs",
                             k = n_books_10, control = list(seed = 1234))
tim2 <- Sys.time()
u_tfidf <- tim2-tim1

# embedding 1
misc.rate_1_10 <- ext_gamma_matrix(chapters_lda_10) %>%
  validate_LDAClassification()

# embedding 2
misc.rate_all_10 <- ext_gamma_matrix(chapters_lda_all_10) %>%
  validate_LDAClassification()

# embedding 3
misc.rate_tfidf_10 <- ext_gamma_matrix(chapters_lda_tfidf_10) %>%
  validate_LDAClassification()

# overview
performance_matrix_10_Gibbs <- data.frame(freq2.embedding=c(misc.rate_1_10, u_1),
                                           all.embedding=c(misc.rate_all_10, u_all),
                                           tfidf=c(misc.rate_tfidf_10, u_tfidf))
rownames(performance_matrix_10_Gibbs) <- c("misc. rate", "time")
performance_matrix_10_Gibbs %>% stargazer(summary=FALSE, header=F, title = "LDA via Gibbs 10 books exam

```

Table 9: LDA via Gibbs 10 books example

	freq2.embedding	all.embedding	tfidf
misc. rate	0.341	0.322	0.317
time	1.301	1.514	36.493

```

# embedding 1
tim1 <- Sys.time()
chapters_lda_10_VEM <- LDA(chapters_dtm_10, method="VEM",
                           k = n_books_10, control = list(seed = 1234))
tim2 <- Sys.time()
u_1 <- tim2-tim1

# embedding 2
tim1 <- Sys.time()
chapters_lda_all_10_VEM <- LDA(chapters_dtm_all_10, method="VEM",
                               k = n_books_10, control = list(seed = 1234))
tim2 <- Sys.time()
u_all <- tim2-tim1

```

```

# embedding 3
tim1 <- Sys.time()
chapters_lda_tfidf_10_VEM <- LDA(chapters_dtm_tfidf_10, method="VEM",
                                k = n_books_10, control = list(seed = 1234))
tim2 <- Sys.time()
u_tfidf <- tim2-tim1

# embedding 1
misc.rate_1_10_VEM <- ext_gamma_matrix(chapters_lda_10_VEM) %>%
  validate_LDAClassification()

# embedding 2
misc.rate_all_10_VEM <- ext_gamma_matrix(chapters_lda_all_10_VEM) %>%
  validate_LDAClassification()

# embedding 3
misc.rate_tfidf_10_VEM <- ext_gamma_matrix(chapters_lda_tfidf_10_VEM) %>%
  validate_LDAClassification()

# overview
performance_matrix_10_VEM <- data.frame(freq2.embedding=c(misc.rate_1_10_VEM, u_1),
                                         all.embedding=c(misc.rate_all_10_VEM, u_all),
                                         tfidf=c(misc.rate_tfidf_10_VEM, u_tfidf))
rownames(performance_matrix_10_VEM) <- c("missc. rate", "time")
performance_matrix_10_VEM %>% stargazer(summary=FALSE, header=F, title = "LDA via VEM 10 books example")

```

Table 10: LDA via VEM 10 books example

	freq2.embedding	all.embedding	tfidf
missc. rate	0.483	0.497	0.420
time	59.268	55.789	17.245

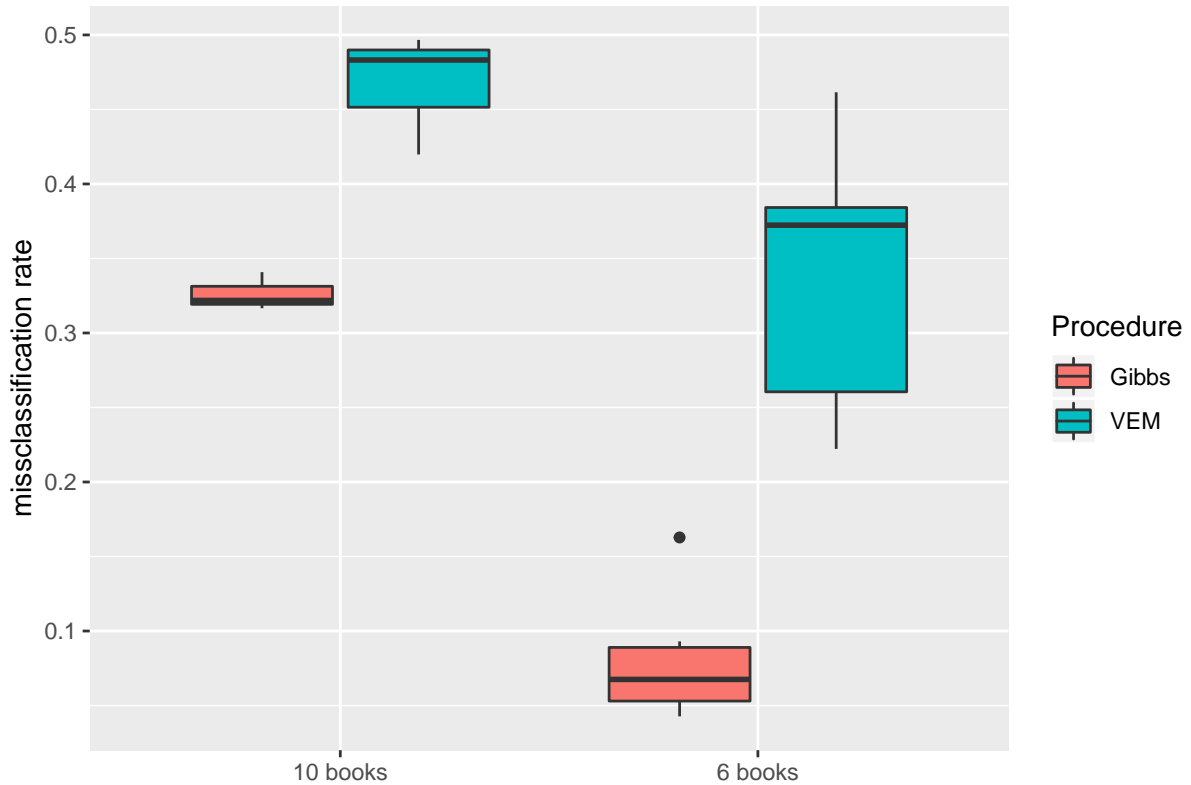
The following box-plot should illustrate the missclassification rate of the LDA algorithm in the different use cases.

```

plot_DF <- data.frame(VEM=c(misc.rate_1_10_VEM, misc.rate_all_10_VEM, misc.rate_tfidf_10_VEM,
                             misc.rate_1, misc.rate_all, misc.rate_tfidf,
                             misc.rate_1_sec, misc.rate_all_sec, misc.rate_tfidf_sec),
                      Gibbs=c(misc.rate_1_10, misc.rate_all_10, misc.rate_tfidf_10,
                              misc.rate_1_Gibbs, misc.rate_all_Gibbs, misc.rate_tfidf_Gibbs,
                              misc.rate_1_Gibbs_sec, misc.rate_all_Gibbs_sec, misc.rate_tfidf_Gibbs_sec),
                      Use_Case=c(rep("10 books",3),
                                  rep("6 books",3),
                                  rep("6 books",3)))
plot_DF2 <- gather(plot_DF, key="Procedure", value="accuracy", VEM, Gibbs)

plot_DF2 %>% ggplot(aes(x=Use_Case, y=accuracy, fill=Procedure))+
  geom_boxplot() + xlab("") + ylab("missclassification rate")

```

5.1 In Model Evaluation

In the first place, we try the in-model-validation procedure, for each of the 3 embeddings. Table 11 displays the results of each validation.

```

chapters_dtm2 <- convert_to_dtm(word_counts, minfq=2)
tim1 <- Sys.time()
chapters_lda_Gibbs2 <- LDA(chapters_dtm2, method="Gibbs",
                           k = n_books, control = list(seed = 1234))
tim2 <- Sys.time()
u_1_Gibbs2 <- tim2-tim1
ncol(chapters_dtm2)

## [1] 8597

chapters_dtm_all2 <- convert_to_dtm(word_counts, minfq=0)
tim1 <- Sys.time()
chapters_lda_all_Gibbs2 <- LDA(chapters_dtm_all2, method = "Gibbs",
                               k = n_books, control = list(seed = 1234))
tim2 <- Sys.time()
u_all_Gibbs2 <- tim2-tim1
M2 <- ncol(chapters_dtm_all2 )

chapters_dtm_tfidf2 <- convert_to_dtm_2(word_counts, top=0.5*M2)
tim1 <- Sys.time()
chapters_lda_tfidf_Gibbs2 <- LDA(chapters_dtm_tfidf2, method="Gibbs",
                                 k = n_books, control = list(seed = 1234))
tim2 <- Sys.time()

```

```

u_tfidf_Gibbs2 <- tim2-tim1
ncol(chapters_dtm_tfidf)

## [1] 1518

misc.rate_1_Gibbs2 <- ext_gamma_matrix(chapters_lda_Gibbs2) %>%
  validate_LDAClassification()

misc.rate_all_Gibbs2 <- ext_gamma_matrix(chapters_lda_all_Gibbs2) %>%
  validate_LDAClassification()

misc.rate_tfidf_Gibbs2 <- ext_gamma_matrix(chapters_lda_tfidf_Gibbs2) %>%
  validate_LDAClassification()

performance_matrix <- data.frame(freq2.embedding=c(misc.rate_1_Gibbs2, u_1_Gibbs2),
  all.embedding=c(misc.rate_all_Gibbs2, u_all_Gibbs2),
  tfidf=c(misc.rate_tfidf_Gibbs2, u_tfidf_Gibbs2))
rownames(performance_matrix) <- c("misc. rate", "time")
performance_matrix %>% stargazer(summary=FALSE, header=F, title="In-Model-Validation for the 3 embeddings")

```

Table 11: In-Model-Validation for the 3 embeddings

	freq2.embedding	all.embedding	tfidf
misc. rate	0.169	0.116	0.042
time	1.804	1.612	39.401

Gutenberg Data via Neural Net Documentation

Sebastian Knigge

5 8 2019

Contents

1	Setup	1
2	Example 1 (6 books)	1
2.1	Get data - Sampling	1
2.2	Reduction of the dimensionality	5
2.3	Splitting and Fitting	5
3	Example 2 (6 Books)	11

1 Setup

Following packages were used in this script:

```
# loading packages
library(keras)
library(gutenbergr)
library(dplyr)
library(tensorflow)
library(tidyr)
library(stringr)
library(tidytext)
library(udpipe)
library(sampling)
```

2 Example 1 (6 books)

2.1 Get data - Sampling

This is the essential step for setting up the neural net. These functions include the sampling procedure of the *gutenbergr* library.

```
sampling_books <- function(seed=1234, n=20){
  # sample n books from the whole library
  set.seed(seed)
  gutenberg_works() %>%
    # select works with title
    dplyr::filter(!is.na(title)) %>%
    # set the sample sitze
    sample_n(n) %>%
    # set a special download link
    gutenberg_download(
      mirror = "http://mirrors.xmission.com/gutenberg/")
}
```

```

}

set_up_books <- function(n_books=4, seed=1992){
  # initial book sample
  books <- sampling_books(n=n_books, seed=seed)
  by_chapter <- books %>%
    group_by(gutenberg_id) %>%
    # split in chapters
    mutate(chapter = cumsum(str_detect(text, regex("^chapter ", ignore_case = TRUE)))) %>%
    ungroup() %>%
    # exclude books without chapters
    dplyr::filter(chapter > 0)
  return(by_chapter)
}

shorten_titles <- function(titles){
  # shorten very long book titles by setting
  # a subset of characters of the first line
  # of the title
  sub_inds <- titles %>%
    regexr(pattern="\\n|\\r")-1
  sub_inds[sub_inds<0] <- nchar(titles)[sub_inds<0]
  titles %>%
    substr(1,sub_inds)
}

get_titles <- function(x, n_books){
  # get the sampled gutenberg_ids
  unique_ids <- x %>%
    select(gutenberg_id) %>%
    unique() %>% unlist()
  # get the titles
  titles <- gutenberg_works() %>%
    dplyr::filter(gutenberg_id %in% unique_ids) %>%
    select(gutenberg_id, title, author) %>%
    mutate(title=shorten_titles(title))
  # get the number of gutenberg ids
  len <- nrow(titles)
  if(n_books!=len) warning(paste("---- ",n_books-len,
                                " books have 0 chapters --- "))
  # the output as a list
  ret <- list(
    titles=titles,
    len=len
  )
  return(ret)
}

append_by_chapter <- function(x=by_chapter, n_books, seed_index=1){
  # append the books matrix until
  # we get the desired number of books n_books
  titles <- get_titles(x, n_books)
  n <- titles$len

```

```

while (n<n_books) {
  book2add <- sampling_books(n=1, seed=seed_index)
  by_chapter_add <- book2add %>%
    group_by(gutenberg_id) %>%
    # split in chapters
    mutate(chapter = cumsum(str_detect(text, regex("^chapter ", ignore_case = TRUE)))) %>%
    ungroup() %>%
    # exclude books without chapters
    dplyr::filter(chapter > 2)
  titles2add <- get_titles(by_chapter_add, 1)
  # adding the book to by_chapter if there are chapters in the
  # book plus it is not in the data already
  if (titles2add$len==1) if(!titles2add$titles$gutenberg_id%in%titles$titles$gutenberg_id) {
    x <- bind_rows(x, by_chapter_add)
  }
  n<-get_titles(x, n)$len
  seed_index <- seed_index+1
}
return(x)
}

exclude_stop_words <- function(x){
  # unite chapter and document title
  by_chapter_word <- x %>%
    unite(document, gutenberg_id, chapter) %>%
    # split into words
    unnest_tokens(word, text)
  # import tibble stop words
  data(stop_words)
  # find document-word counts
  word_counts <- by_chapter_word %>%
    # exclude stop words
    anti_join(stop_words) %>%
    # count each word by chapter
    count(document, word, sort = TRUE) %>%
    ungroup()
  return(word_counts)
}

convert_to_dtm <- function(x, minfq = 2){
  # get into a format lda can handle
  chapters_dtm <- x %>%
    select(doc_id=document, term=word, freq=n) %>%
    document_term_matrix() %>%
    # reduce by low frequencies
    dtm_remove_lowfreq(minfreq = minfq)
  return(chapters_dtm)
}

convert_to_dtm_2 <- function(x, n=n, minfq = 2, top=10000){
  # get into a format lda can handle
  chapters_dtm <- x %>%

```

```

    select(doc_id=document, term=word, freq=n) %>%
    document_term_matrix() %>%
    # reduce by low frequencies
    dtm_remove_tfidf(top=top)
  return(chapters_dtm)
}

# convert x matrix into a form such that it can be used for tensorflow
adjust_tensor_format <- function(x){
  x_chapters <- apply(x, 1, function(x) as.matrix(x)) %>% t()
  topics <- x %>% rownames() %>% as_tibble() %>%
    separate(value, c("gutenberg_id", "chapter"), sep = "_", convert = TRUE) %>%
    select(gutenberg_id) %>%
    # split joint name of book and chapter
    as.matrix %>% as.factor() %>% as.integer()
  # one hot encoding for the chapters (y)
  topics_categorical <- topics %>% -1 %>%
    to_categorical()
  ret <- list(
    x=x_chapters,
    y=topics_categorical,
    topics=topics
  )
  return(ret)
}

```

Now we can use all these functions to get to the initial corpus sample. the corpus of this example is equivalent to the corpus in Example 1 of the LDA Gutenberg Documentation.

```

n_books <- 6
by_chapter <- set_up_books(n_books=n_books, seed=222)
get_titles(by_chapter, n_books)

```

```

## Warning in get_titles(by_chapter, n_books): --- 1 books have 0 chapters ---

## $titles
## # A tibble: 5 x 3
##   gutenberg_id title author
##   <int> <chr> <chr>
## 1      11 Alice's Adventures in Wonderland Carroll, Lewis
## 2     3096 Beatrice Haggard, H. Rider~
## 3    25603 Detailed Minutiae of Soldier life in the~ McCarthy, Carlton
## 4    47402 Along Alaska's Great River Schwatka, Frederi~
## 5    49675 Hawkins Electrical Guide v. 5 (of 10) Hawkins, N. (Nehe~
##
## $len
## [1] 5

```

The function `set_up_books()` (defined above) returns a warning that one book seems to consist of only one chapter. In order to get a corpus consisting out of 6 books, the function `append_by_chapter()` is used, which fills up the corpus to the desired number of books.

```

appended_by_chapter <- append_by_chapter(x=by_chapter, n_books = n_books)
word_counts <- exclude_stop_words(appended_by_chapter)

```

```
## Joining, by = "word"
```

In table 1 the sampled titles for the book sample with the seed 222 are displayed. It appears through the function `append_by_chapter()` one book was added, called “My Novel” — Volume 04“.

```
titles <- get_titles(appended_by_chapter, n_books)
titles$titles %>% stargazer(summary=FALSE, font.size = "footnotesize",
                           header=FALSE, title="Book-titles", rownames=FALSE,
                           label="titles:6books")
```

Table 1: Book-titles

gutenberg_id	title	author
11	Alice's Adventures in Wonderland	Carroll, Lewis
3096	Beatrice	Haggard, H. Rider (Henry Rider)
7705	"My Novel" — Volume 04	Lytton, Edward Bulwer Lytton, Baron
25603	Detailed Minutiae of Soldier life in the Army of Northern Virginia, 1861-1865	McCarthy, Carlton
47402	Along Alaska's Great River	Schwatka, Frederick
49675	Hawkins Electrical Guide v. 5 (of 10)	Hawkins, N. (Nehemiah)

2.2 Reduction of the dimensionality

As mentioned in the documentation for the LDA using Gutenberg data, also here there are different possible embedding methods. The function `convert_to_dtm` takes the parameter `minfq`, which is used to reduce the “bag of words” (i.e. dimensionality). `minfq` is the minimum frequency for the bag of words dictionary. I will refer to this as “embedding”. Let us set it to 2 in this case, meaning that we include a word only if the frequency is 2 or more.

```
chapters_dtm <- convert_to_dtm(word_counts, minfq=2)
adjusted_format <- adjust_tensor_format((chapters_dtm))
( M2 <- ncol(chapters_dtm) )
```

```
## [1] 10685
```

Let us compare it to the case if we include all words.

```
chapters_dtm_all <- convert_to_dtm(word_counts, minfq=0)
adjusted_format_all <- adjust_tensor_format((chapters_dtm_all))
( M <- ncol(chapters_dtm_all) )
```

```
## [1] 17961
```

We also want to compare this to a reduction of the word dictionary by tfidf. We reduce to 50% of the original size of the bag-of-words.

```
chapters_dtm_tfidf <- convert_to_dtm_2(word_counts, top=(M*0.5))
adjusted_format_tfidf <- adjust_tensor_format((chapters_dtm_tfidf))
ncol(chapters_dtm_tfidf)
```

```
## [1] 8980
```

2.3 Splitting and Fitting

The following function splits the sample in an manner, such that each cluster is equally to its size represented in the test data and the validation data.

```
sample_cluster_wise <- function(data, test_ratio=0.1, val_ratio=0.2, seed=1234){
  X <- data$x; y <- data$y
```

```

cluster=data$topics
set.seed(seed)
{
  # setting the absolute number of observations for the sample of each cluster
  n_test <- (table(cluster)*test_ratio) %>% floor() %>%
    # use at least one observation of each cluster
    sapply(., function(x) max(x,1))
  n_val <- (table(cluster)*val_ratio) %>% floor() %>%
    # use at least one observation of each cluster
    sapply(., function(x) max(x,1))
  # function to get the correct sample indices for validation and test sample
  samp_ind <- function(i, n_list) which(cluster==i) %>% sample(n_list[i])
  test_indices <- unique(cluster) %>% sort() %>%
    sapply(function(i) samp_ind(i, n_test)) %>%
    unlist()
  val_indices <- unique(cluster) %>% sort() %>%
    sapply(function(i) samp_ind(i, n_val)) %>%
    unlist()
}
ret <- list(partial_x_train=X[-c(val_indices,test_indices),],
            partial_y_train=y[-c(val_indices,test_indices),],
            x_val=X[val_indices,], y_val = y[val_indices,],

            x_test=X[test_indices,], y_test = y[test_indices,])
return(ret)
}

```

The following two functions are setting up the model in training, validation and testing sets and evaluate the goodness of fit.

```

# The whole model is set up and trained within this function
set_up_n_fit <- function(split, books_n=n_books){
  # starting with 64 neurons and scaling it down to 46 in the
  # mid layer turned out to be a well predicting model
  model <- keras_model_sequential() %>%
    layer_dense(units=64, activation="relu", input_shape=ncol(split$partial_x_train)) %>%
    layer_dense(units=46, activation="relu") %>%
    # we want to classify for as many categories as books
    layer_dense(units=books_n, activation="softmax")

  model %>% compile(
    optimizer="rmsprop",
    loss="categorical_crossentropy",
    metrics=c("accuracy"))

  history <- model %>% fit(
    split$partial_x_train,
    split$partial_y_train,
    # from experience the model tends to
    # overfit for more than 5 epochs
    epochs=5,
    batch_size=512,
    validation_data=list(split$x_val,split$y_val)
  )
}

```



```

return(
  list(history=history,
        model=model))
}

# making a prediction on the test data and calculating the
# misspecification rate; we also want to save the true categories and the predicted ones
evaluate_model <- function(model_fit, y=split$y_test, x=split$x_test) {
  prediction <- model_fit %>% predict(x)
  pred <- apply(prediction, 1, which.max)
  true_value <- apply(y, 1, which.max)
  misspecified <- sum(!pred==true_value)/length(pred)
  ret <- list(misspecified=misspecified,
             # the function also discloses the true and the predicted
             # values for exact evaluation, if needed
             pred=pred, true_value=true_value)
  return(ret)
}

```

Now we can start applying the model. We will measure the misclassification ratio, the fitting time and store the results for evaluation.

```

tim1 <- Sys.time()
n <- 59
results <- rep(NA,n)
for(i in 1:n){
  split <- sample_cluster_wise(adjusted_format, seed=i*2)
  results[i] <- set_up_n_fit(split) %>% .$model %>%
    evaluate_model() %>% .$misspecified
}
tim2 <- Sys.time()

```

We will present some statistics of the results of the evaluation. We consider the misclassification rate als primary measure to evaluate the fit. The results of the misclassification rate over 59 splits and fits of the model are:

```
results %>% summary
```

```
##      Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
## 0.00000 0.00000 0.00000 0.02203 0.00000 0.20000
```

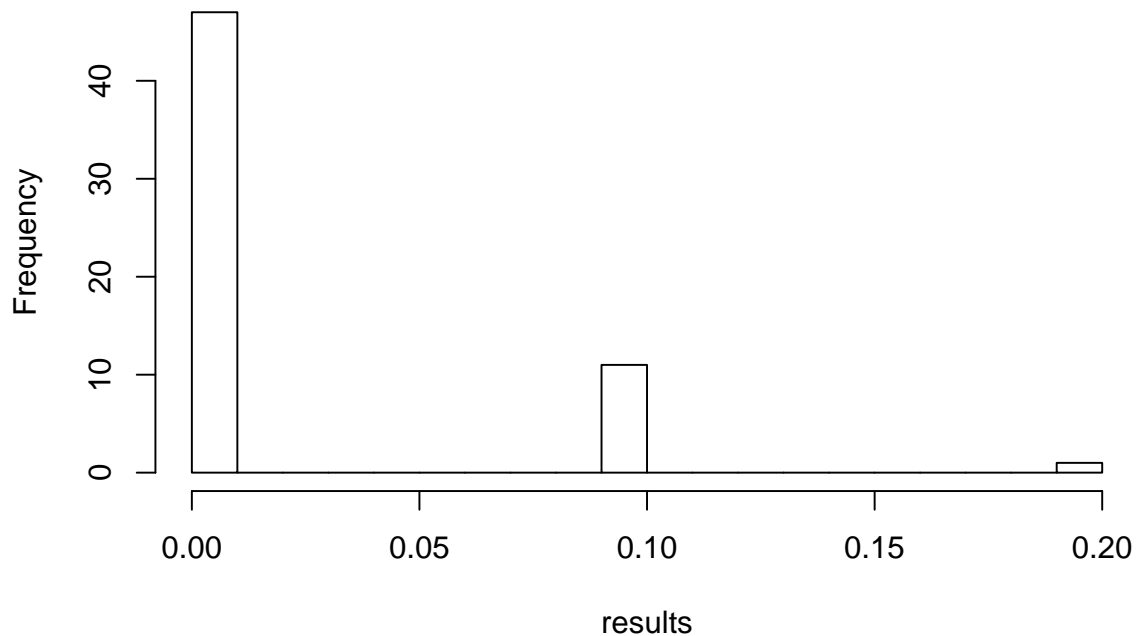
```
results %>% var
```

```
## [1] 0.002092344
```

A histogram might be helpful in visualizing the results.

```
hist(results, breaks = 20)
```

Histogram of results



The fitting time might be interesting as well.

```
# mfreq=2
(u1 <- tim2-tim1)
```

Time difference of 3.646245 mins

We now apply this very same procedure using the full bag of words:

```
tim1_all <- Sys.time()
n <- 59
results_all <- rep(NA,n)
for(i in 1:n){
  split <- sample_cluster_wise(adjusted_format_all, seed=i*2)
  results_all[i] <- set_up_n_fit(split) %>% .$model %>%
    evaluate_model() %>% .$misspecified
}
tim2_all <- Sys.time()
```

```
results_all %>% summary
```

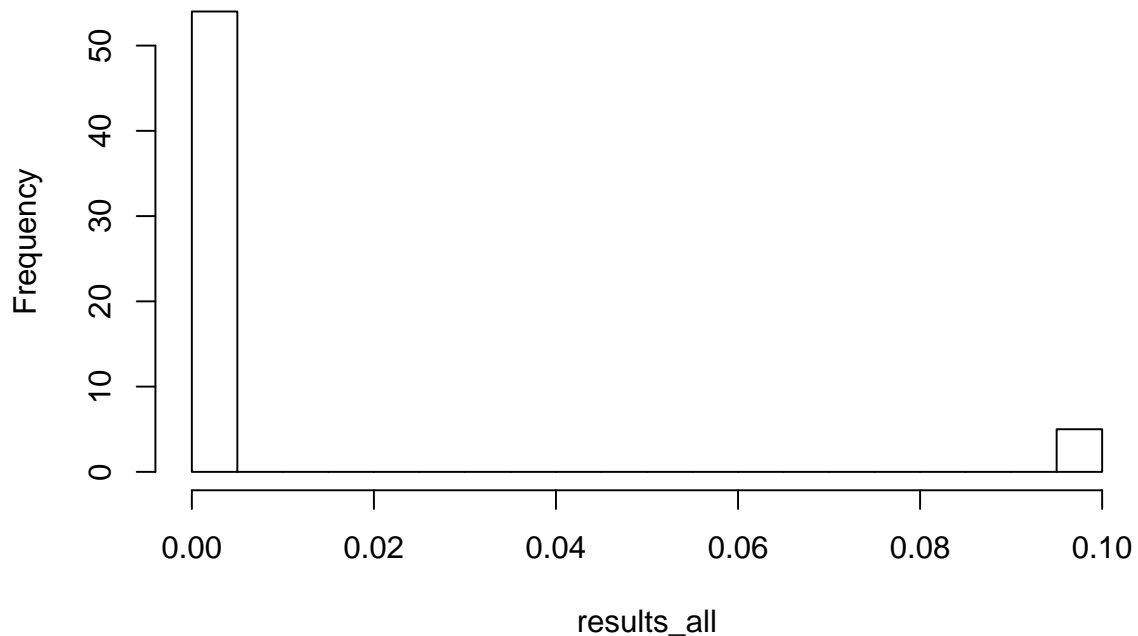
```
##      Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
## 0.000000 0.000000 0.000000 0.008475 0.000000 0.100000
```

```
results_all %>% var
```

```
## [1] 0.0007890123
```

```
hist(results_all, breaks = 20)
```

Histogram of results_all



```
# mfreq=2
(u2 <- tim2_all-(tim1_all))
```

```
## Time difference of 10.28905 mins
```

In this step we use the embedding via *tfidf* and fit the model. The *tfidf* embedding equals a reduction by 50%.

```
tim1_tfidf <- Sys.time()
n <- 59
results_tfidf <- rep(NA,n)
for(i in 1:n){
  split <- sample_cluster_wise(adjusted_format_tfidf, seed=i*2)
  results_tfidf[i] <- set_up_n_fit(split) %>% .$model %>%
    evaluate_model() %>% .$misspecified
}
tim2_tfidf <- Sys.time()
```

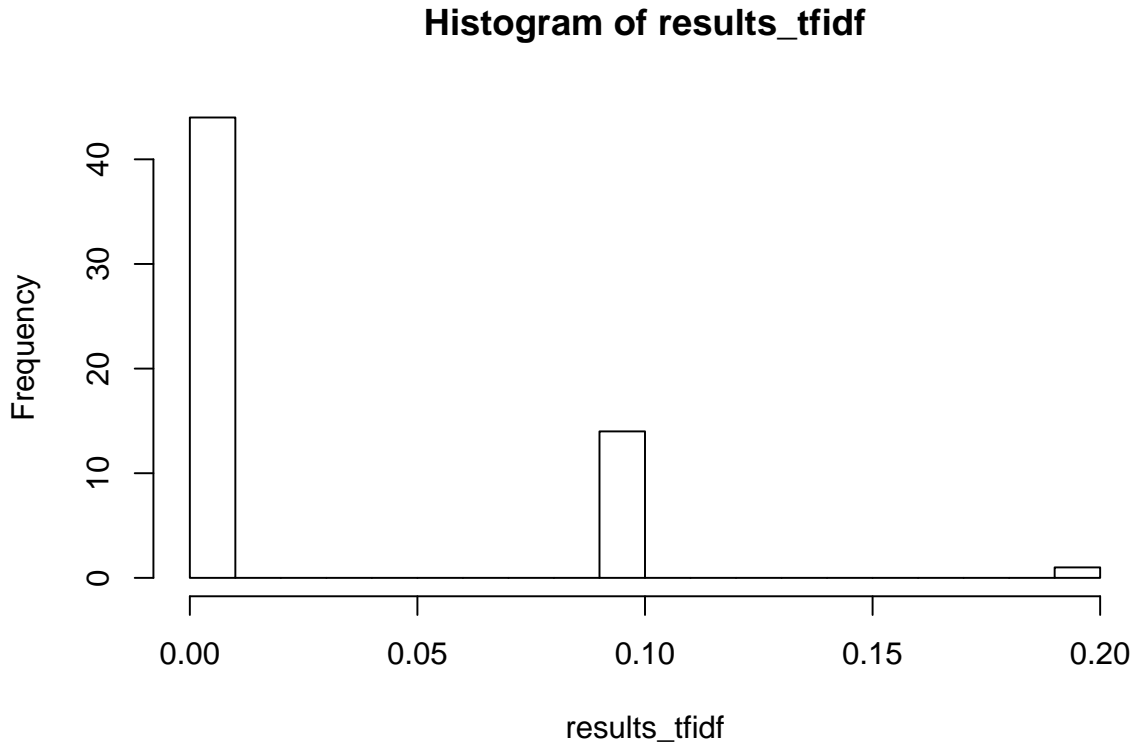
```
results_tfidf %>% summary
```

```
##      Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
## 0.00000 0.00000 0.00000 0.02712 0.05000 0.20000
```

```
results_tfidf %>% var
```

```
## [1] 0.002355348
```

```
hist(results_tfidf, breaks = 20)
```



```
# mfreq=2
(u3 <- tim2_tfidf-(tim1_tfidf))

## Time difference of 4.240429 mins

Table 2 summarises the results of all 3 different embedding methods.

performance_matrix <- data.frame(freq2.embedding=c(results%>%mean, u1),
  all.embedding=c(results_all%>%mean, u2),
  tfidf=c(results_tfidf%>%mean, u3))
rownames(performance_matrix) <- c("misc. rate", "time")
performance_matrix %>% stargazer(summary=FALSE, header=F,
  title="Performance of embeddings",
  label="performancematrix")
```

Table 2: Performance of embeddings

	freq2.embedding	all.embedding	tfidf
misc. rate	0.022	0.008	0.027
time	3.646	10.289	4.240

The time for the embedding using more than frequency 2 is very short. Whereas the embedding via tfidf with the same dimensionality takes very long. It certainly makes sense that the missclassification rate is better for embedding with the full vocabulary than for the other two embeddings.

3 Example 2 (6 Books)

Also the second example is based on the same sample of Example 2 of the LDA Gutenberg Documentation. This is `seed = 101`. We will use the same calculations as done for Example 1. This second example shall help providing more valid results.

```
n_books_sec <- 6
by_chapter_sec <- set_up_books(n_books=n_books_sec, seed=101)
appended_by_chapter_sec <- append_by_chapter(x=by_chapter_sec, n_books = n_books_sec)
word_counts_sec <- exclude_stop_words(appended_by_chapter_sec)
```

```
## Joining, by = "word"

titles_sec <- get_titles(appended_by_chapter_sec, n_books)
gbids_sec <- titles_sec$titles$gutenberg_id
categories_sec <- gutenberg_works() %>%
  filter(gutenberg_id %in% gbids_sec) %>%
  select(gutenberg_id, gutenberg_bookshelf)
```

Table 3 displays the sampled books for this example.

```
titles <- get_titles(appended_by_chapter_sec, n_books)
titles$titles %>% stargazer(summary=FALSE, font.size = "footnotesize",
  header=FALSE, title="Book-titles for Example 2 (6 Books)", rownames=FALSE,
  label="titles:6books_sec")
```

Table 3: Book-titles for Example 2 (6 Books)

gutenberg_id	title	author
2029	Lahoma	Ellis, J. Breckenridge (John Breckenridge)
12035	Progressive Morality: An Essay in Ethics	Fowler, Thomas
14625	Military Instructors Manual	
18633	My Lady of Doubt	Parrish, Randall
34037	Under the Southern Cross	Ballou, Maturin Murray
36697	Revisiting the Earth	Hill, James Langdon

```
chapters_dtm_sec <- convert_to_dtm(word_counts_sec, minfq=2)
adjusted_format_sec <- adjust_tensor_format((chapters_dtm_sec))
( M2 <- ncol(chapters_dtm_sec) )
```

```
## [1] 12704
```

```
chapters_dtm_all_sec <- convert_to_dtm(word_counts_sec, minfq=0)
adjusted_format_all_sec <- adjust_tensor_format((chapters_dtm_all_sec))
( M <- ncol(chapters_dtm_all_sec) )
```

```
## [1] 20634
```

```
chapters_dtm_tfidf_sec <- convert_to_dtm_2(word_counts_sec, top=(M*0.5))
adjusted_format_tfidf_sec <- adjust_tensor_format((chapters_dtm_tfidf_sec))
ncol(chapters_dtm_tfidf_sec)
```

```
## [1] 10317
```

```
tim1 <- Sys.time()
n <- 59
results_sec <- rep(NA,n)
for(i in 1:n){
```

```

split <- sample_cluster_wise(adjusted_format_sec, seed=i*2)
results_sec[i] <- set_up_n_fit(split) %>% .$model %>%
  evaluate_model() %>% .$misspecified
}
tim2 <- Sys.time()
# evaluate fitting time
(u1_sec <- tim2-tim1)

## Time difference of 10.13462 mins

tim1 <- Sys.time()
n <- 59
results_all_sec <- rep(NA,n)
for(i in 1:n){
  split <- sample_cluster_wise(adjusted_format_all_sec, seed=i*2)
  results_all_sec[i] <- set_up_n_fit(split) %>% .$model %>%
    evaluate_model() %>% .$misspecified
}
tim2 <- Sys.time()
# evaluate fitting time
(u2_sec <- tim2-tim1)

## Time difference of 3.773633 mins

tim1 <- Sys.time()
n <- 59
results_tfidf_sec <- rep(NA,n)
for(i in 1:n){
  split <- sample_cluster_wise(adjusted_format_tfidf_sec, seed=i*2)
  results_tfidf_sec[i] <- set_up_n_fit(split) %>% .$model %>%
    evaluate_model() %>% .$misspecified
}
tim2 <- Sys.time()
# evaluate fitting time
(u3_sec <- tim2-tim1)

## Time difference of 9.303475 mins

A overview of the results will be found in Table 4.

performance_matrix <- data.frame(freq2.embedding=c(results_sec%>%mean, u1_sec),
  all.embedding=c(results_all_sec%>%mean, u2_sec),
  tfidf=c(results_tfidf_sec%>%mean, u3_sec))
rownames(performance_matrix) <- c("missc. rate", "time")
performance_matrix %>% stargazer(summary=FALSE, header=F,
  title="Performance of embeddings for Example 2",
  label="tab:sec_results")

```

Table 4: Performance of embeddings for Example 2

	freq2.embedding	all.embedding	tfidf
missc. rate	0.105	0.106	0.119
time	10.135	3.774	9.303