universität wien

MAGISTERARBEIT / MASTER'S THESIS

Titel der Masterarbeit / Title of the Master's Thesis

„A Generic Approach for Clustering and Classification of Text Documents"

verfasst von / submitted by

Sebastian Knigge, B.Sc., B.Sc.

angestrebter akademischer Grad / in partial fulfilment of the requirements for the degree of

Magister der Sozial- und Wirtschaftswissenschaften (Mag. rer. soc. oec.)

Wien, 2019 / Vienna 2019

Studienkennzahl lt. Studienblatt / degree programme code as it appears on the student record sheet:	UA 066 951
Studienrichtung lt. Studienblatt / degree programme as it appears on the student record sheet:	Magisterstudium Statistik
Betreut von / Supervisor:	Univ.-Prof. i.R. Dr. Wilfried Grossmann
Mitbetreut von / Co-Supervisor:	

Abstract

Many scientific disciplines overlap in the field of Natural Language Processing. This paper examines statistical approaches to Natural Language Processing. It analyses the statistical models Latent Dirichlet Allocation and Artificial Neural Networks. A theoretical introduction will be given to the reader and two different examples will be used to illustrate the models and their use.

This thesis aims at developing and evaluating a generic methodology for clustering and effectively classifying documents. In the first step I use textbooks as documents to adapt two models. These documents are already categorized by the initiators of the free online library Project Gutenberg. Thus it is possible to evaluate the clustering algorithm on a man-made grouping principle. As second step, the models developed in this manner will be tested on regulatory documents and guidelines of EUROSTAT (an authority of the EU) and compared with an already existing grouping.

The open source software used is R. All codes used can be found in the appendix of this paper and are entirely reproducible.

Acknowledgement

I would like to express my thanks and esteem to my supervisor and mentor of this thesis Prof. Grossmann. Many suggestions, extensive and inspiring conversations have made this work possible.

Many thanks to my friends from Duke University (Durham, NC) Catherine and Joseph, to my sister Fiona and of course to Sarah-Ann from University of Vienna for proof reading.

After all, my deepest gratitude goes to my parents, who always supported me in all situations during my studies. I owe you way more than dedicating this thesis to you.

Contents

ANN Artificial Neural Net

API Application-Programming-Interface

CPU Central Processing Unit

DNA Deoxyribonucleic Acid

EM algorithm Expectation–Maximization algorithm

ESS European Statistical System

EUROSTAT European Statistical Office

GPU Graphics Processing Unit

IR Information Retrieval

KL divergence Kullback–Leibler divergence

LDA Latent Dirichlet Allocation

M-P neuron McCulloch-Pitts neuron

NLP Natural Language Processing

tf term frequency

tf-idf term frequency inverse document frequency

VEM algorithm Variational Expectation–Maximization algorithm

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 *k-means* [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

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

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 an LDA model is adapted to cluster the data. In 4.2 a neural network is built up piece by piece, in order to classifiy 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 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 topicmodels. 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, the evolution of natural language processing is discussed. 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 $D = (\mathbf{w}_1, \ldots, \mathbf{w}_M)$. It consists of M documents $\mathbf{w} = (w_1, \ldots, w_N)$ 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 $j \in \{1, ..., V\}$ 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 **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 $j \in \{1, \ldots, k\}$. 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 **w** is generated by the following process:

1. Choose $z_i \sim Mult(\theta)$
2. Choose $w_i \sim Mult(\beta_i)$ 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].

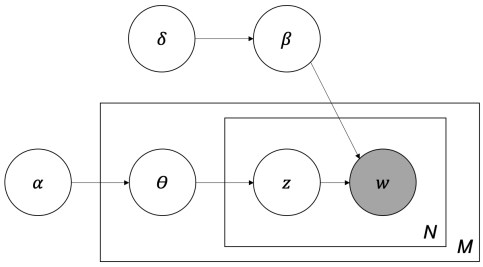


Figure 1: Well-established plate diagram for the standard LDA model extended by the parameter δ. The slightly bigger box represents the generative model of the corporis M documents. The smaller plate represents the iterative generation process of the N words of each document with the aid of the topics. 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.

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

Note that the Dirichlet distribution is the conjugate prior to the multinomial distribution. Conjugate prior means, that it is possible - using a certain distribution - to update the prior in such a way, that the posterior is of the same family of distribution as the prior. The so called, theory of conjugate prior distributions was introduced by [Raiffa and Schlaifer, 1961]. In many cases a closed form solution of the posterior can be easily derived. For a multivariate distribution, which is a multivariate extension of the binomial distribution it is intuitive that the Dirichlet distribution is the conjugate prior, since Dirichlet is again a multivariate extension of the beta distribution. Note that the beta distribution is the conjugate prior for a Bernoulli distribution. In this case the posterior can't be derived in closed form, despite the conjugate property of the Dirichlet distribution, as shown below.

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

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

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.

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

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.

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

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

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

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

Here $q(\theta,\mathbf{z})$ is an arbitrary distribution which can be called the variational distribution. When factorizing the latent variable $\boldsymbol{z}$ into N groups, the calculation of $q(\theta,\mathbf{z})$ with the aid of the product of N individual probabilities $q(z_i|\phi_i)$ is feasible. Since independence needs to be assumed in this approach, the resulting probability is rather an approximation than a true posterior (so called mean field approximation).

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

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:

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

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]).

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

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

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.

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

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

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

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

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 Artifical 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 bivariate 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 mtrix 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 $\mathbf{w} = w_1, \ldots, w_k$ 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 percepron 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 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).

Although the researchers Minsky and Papert showed that the perceptron in this form cannot solve the "xor problem", they argued that extending the

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

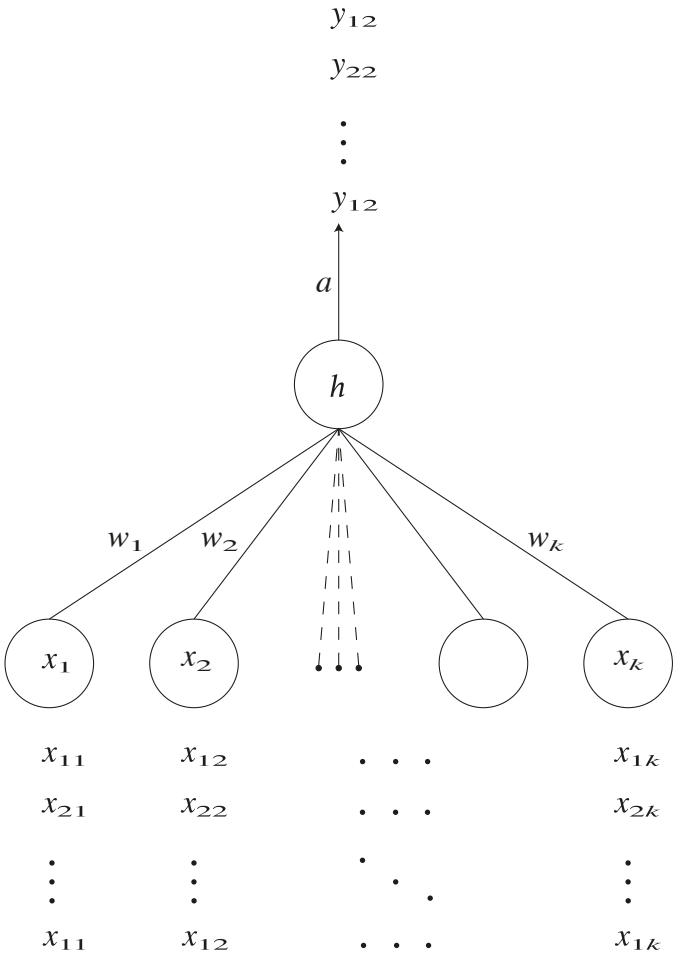


Figure 2: Schematic diagram of a simple perceptron by [Rosenblatt, 1958]

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 an network. However, training a model using Rosenblatt's naive optimization algorithm of the perceptron would not have been feasible.

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 Webos - 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

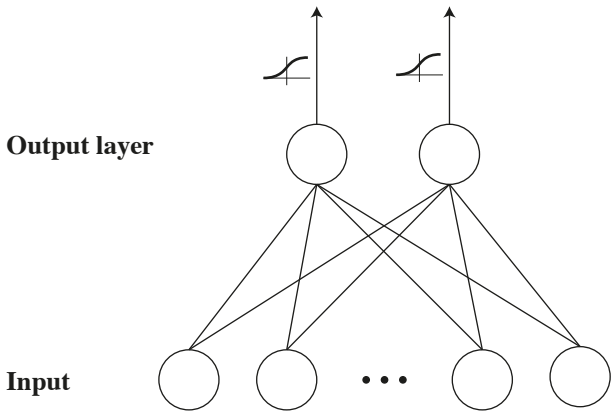


Figure 3: Schematic diagram of an adapted Rosenblatt-perceptron network

function which is used in the perceptron, as this function is differentiable. Assume a random initial distribution of the weights $w_1, \ldots, w_d$ 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.

$$\hat{y_i} = f_{w_d}(f_{w_{d-1}}(...(f_{w_1}(x_i)))) = f_{w_d} \circ f_{w_{d-1}} \circ \cdots \circ f_{w_1}(x_i) \tag{15}$$

The goal is to update the weights of each layer in such a way as to minimise a selected loss function $L(w_1, \ldots, w_d)$. 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].

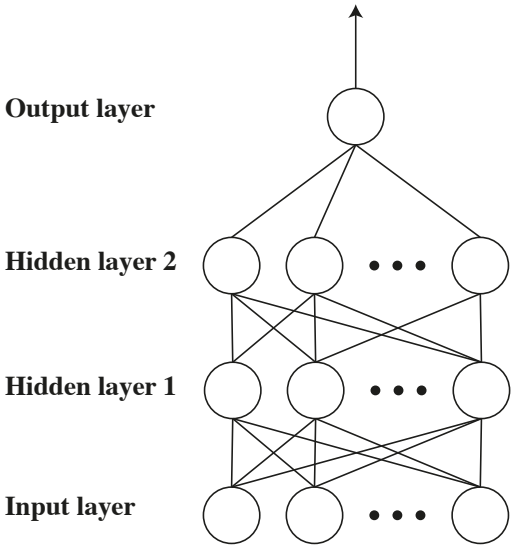


Figure 4: Schematic diagram of an multi layer perceptron (MLP)

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 are 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 for the analysis of a large number of text documents with a secure source and en extensive 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. This sample represents actually the six books sampled in one of the studies examples (Example 1). After splitting the books into chapters, the total number of documents amounts to 117 in this example. This number depends highly on the sampled books. In the course of this thesis a second sample of six different books was drawn for Example 2. In this case one can obtain only 86 books even if the same number of books was sampled. The last sample studied, a sample of ten books was drawn. The number of documents for this example (Example 3) amounts to 230. All these samples and tables of the sampled books (like Table 1) can be found in the appendix.

Table 1: Example book corpus

gutenb. ID	title	author	gutenberg bookshelf
2095	Clotelle: 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 may be examined on the basis of these simple text documents. **Firstly (Q1), are LDA cluster text documents similar to human-driven classification?** In this case, this could be validated by the categorization into Gutenberg bookshelves. **And secondly (Q2), what is the best approach in reproducing such a classification using an ANN as a classification model?**
Since large amounts of data have to be processed and analyzed in order to model the bookshelf classification for a vast 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 suggest an example, using N books and by separating their chapters and shuffling them. Is it possible to use models to reassemble the chapters into stacks that could be assigned to individual books?

The raw text record is now transformed into the orderly format of tidy-text (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 three sources, "onix", "SMART" and "snowball", whereas the latter two are pulled from the `tm` package [Silge, 2019]. Additionally the Onix stop words are taken from the publicly accessible site lextec.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 studied in this thesis use bag-of-words data sets 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. Firtsly, a dimension reduction may decrease the fitting time of the model, secondly, the diversity of the documents may be increased by skilfully reducing certain words, which occur equally as often in all documents. This requires a special measure in which we decide which words to exclude. One may call this reduction "embedding". Throughout the course of this work three 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.

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

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

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

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 will 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 six books are sampled. That is, there are six categories, because as mentioned in 4 all these six have been taken from different bookshelves.

As described in Chapter 3.1, it is possible to calculate the LDA model by means of various algorithms. The two most common ones 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*.

To be more specific, in the first example studied the full bag of words contains 15186 words whereas the set is reduced to 8597 and 7593 words with *tf*-2 and *tf-idf* embedding respectively. In the course of this work a second example with six books was studied. The dimensions of the bag-of-words with the three different embedding methods were similar to the first example (i.e. 15328, 8565 and 7664). Another example was studied when sampling 10 books from the Gutenberg library. In this case the bag-of-words amounted to 29101 for the full embedding 17742 via *tf*-2 and 14550 for the *tf-idf* embedding.

These three embedding approaches are reviewed in regard to the goodness

of the fit and the speed of the computation. One may intuitively assume that the calculation takes longer for data with higher dimensionality.

For a clustering model there are two contrasting methods to evaluate the fit. First, it is possible to fit the model with training data and to evaluate the goodness of the fit using the test data. Second, 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. When starting this analysis, the second approach will be 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 purpose 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 somewhat longer 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 often 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

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

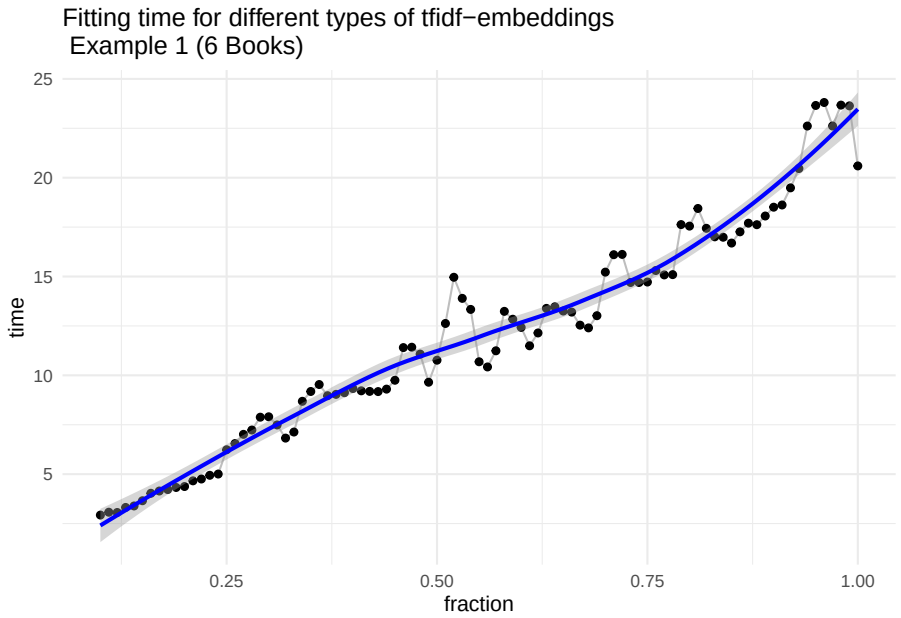


Figure 5: Fitting time of Example 1 (6 Books) depending on the reduction of the bag-of-words. A big number of fraction stands for a bigger bag-of-words, i.e. less reduction. The blue smoothed curve is a Loess kernel. The drop in fitting time for the last few values of fraction might result from parallelization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

in Chapter 5.1.

Surprisingly, the models with *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 some of 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 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). This

may also be overfitting of the parameter fraction, if for other samples regarding 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 using either *tf-idf* with fixed 50% or term frequency 2 as embedding method, if there is no further information about the data set.

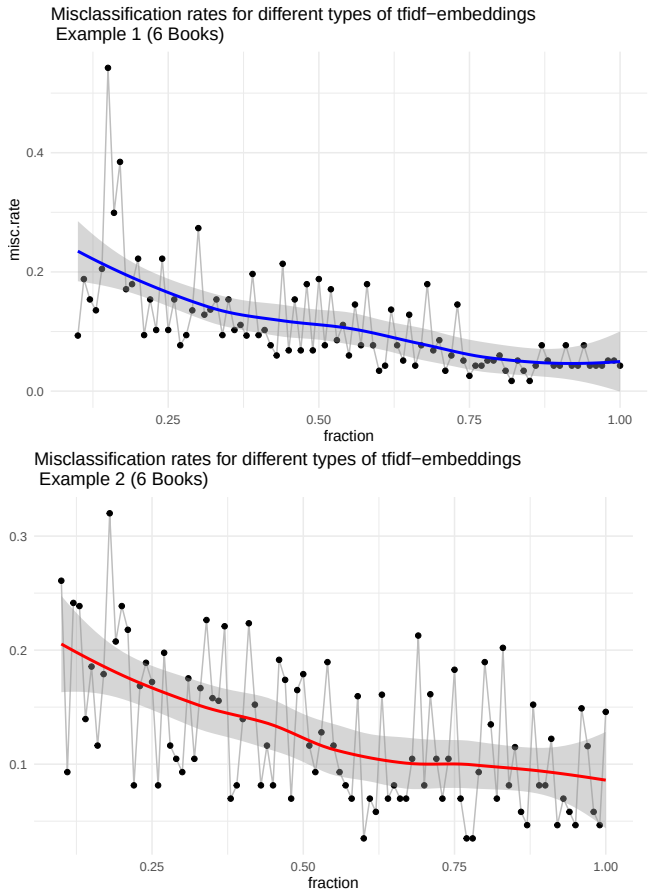


Figure 6: Misclassification rate depending on the reduction of the bag-of-words. Examples 1 and 2 with six books each are considered here. A big number of fraction stands for a bigger bag-of-words, i.e. less reduction. The colored smoothed curves are a Loess kernels.

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 far 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 when more topics were used.[4] This seems 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. 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.

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

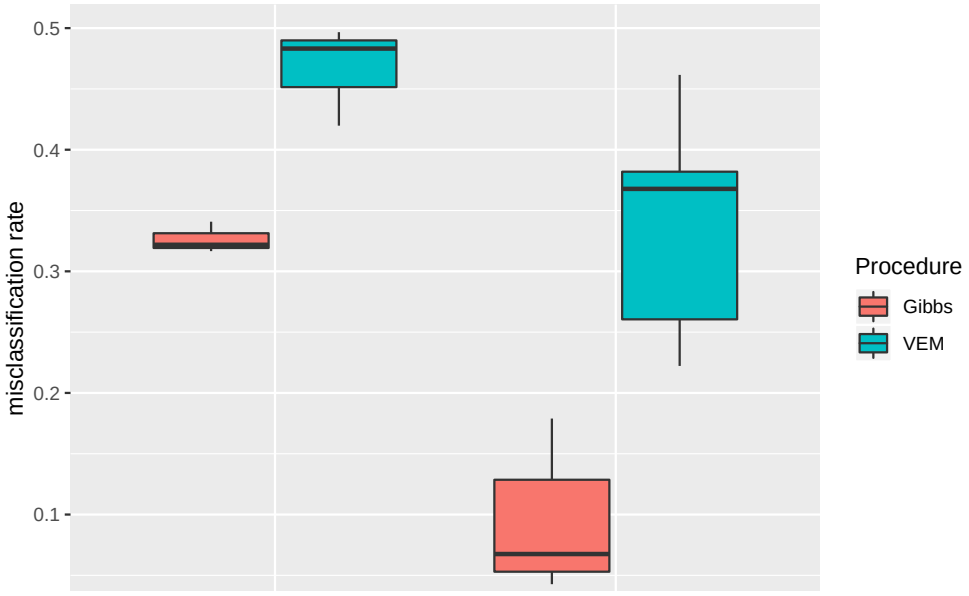


Figure 7: Boxplot of the all examples analyzed. Split for the number of categories (books) as well as for the calculation procedures (VEM algorithm and Gibbs sampling).

When plotting the misclassification rate of Example 3 versus the *fraction* parameter, the shape of the curve is significantly different. For the sample of 10 books, the same analysis with regards to different embedding methods were performed. Figure 8 shows the equivalent plot as in Figure 6. This time the misclassification rate of 10 books (Example 3) was plotted for different *fractions* of *tf-idf* embeddings. Surprisingly, we can't obtain a continuous

[4]For the LDA model accuracy is defined as how good the model reproduces the categorization given by the books (see section).

negative trend for bigger values of *fraction*. The loess estimator indicates a local maximum for values around 0.65 and a local minimum in the area 0.45. Even if volatility is quite big at this point, materially smaller average values of misclassification rate can be found compared to embeddings with slightly higher dimension. This drop will be referred to as " low-dimension embedding anomaly". It is remarkable that this phenomenon only occurs with models featuring more categories. We will return to the different form of this plot when we analyze the EUROSTAT documents in section 5.1.

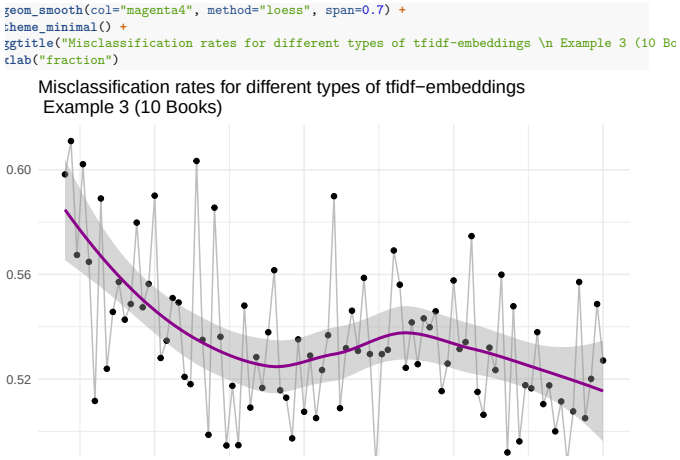


Figure 8: Misclassification rate depending on the reduction of the bag-of-words. Example 3 using 10 books is considered here. A big number of fraction stands for a bigger bag-of-words, i.e. less reduction. The violet smoothed curve is a Loess kernel. Nothe, that the local minumum in the misclasfication rate will be referred to as "low-dimension embedding anomaly"

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 to the training data and to evaluate it with the aid of the test data. To calculate valid results, 59 random clustered samples were used with splits of 90% training sample and 10% test sample. The results for the models which were only fitted to a subset of the data were significantly worse than the comparable models that were trained and tested on the entire dataset. However, since 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 to evaluate 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 (chapters) 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 purpose 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 9 for comparison). The used network essentially consists of three layers. The low-

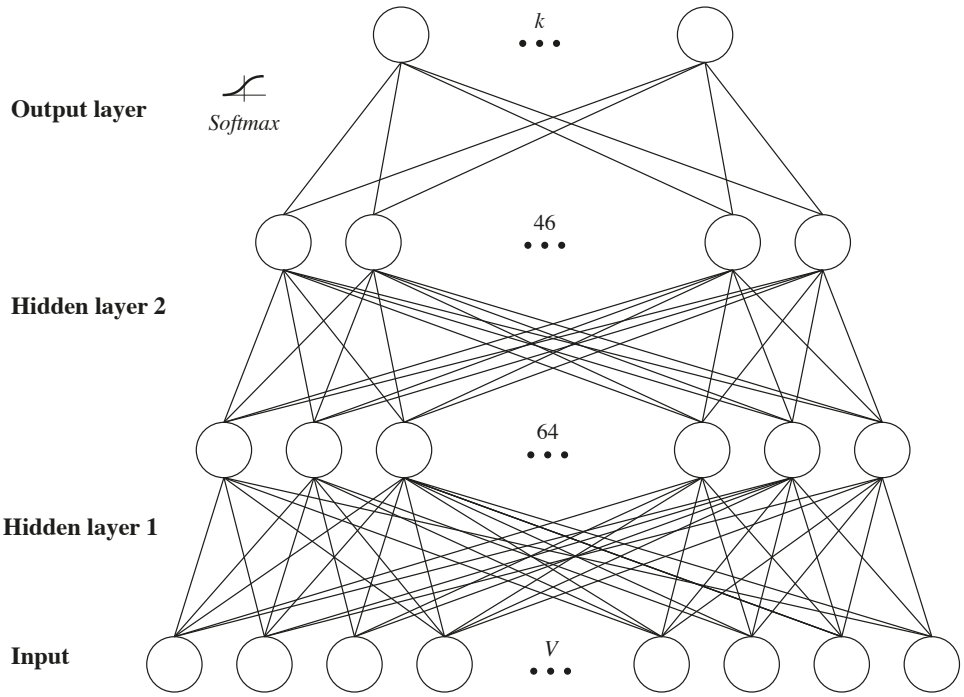


Figure 9: The network contains 3 layers. The input data is the the bag-of-words, which has dimension V. The first two layers are hidden, containing 64 and 46 neurons respectively. The last layer leads to the k classes (here the number of topics or books). The activation function is softmax.

est V circles represent the dimension of the input. I.e. V is 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.

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

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 five 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.

Recap: The evaluation procedure of the model for this data set was conducted as follows:

1. Bring the whole corpus into the *tidytext* format. Split at each chapter to receive a list of words for each chapter of each book labelled with the book ID (Gutenberg ID) and the chapter.

2. Exclude stop words

3. Convert the labeled list of words into a "document term matrix". This is a word count for each document (i.e. combination of chapter and Gutenberg ID) including all unique words in the corpus. Either use term frequency (tf), tf-2 or *tf-idf* as measure.

4. Convert the documents to the tensor format and the dependent variable (Book) to a one-hot encoded format.

5. Evaluate the model iteratively 59 times. Each time the documents are split randomly in 10% test data, 70% training data and 20% validation data, by adjusting the randomness parameter (seed) in every iteration. Thus, each model is trained, validated and tested (i.e. evaluated) at different documents.

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

embeding method	*tf*-2	full bag-of-words	*tfidf* 0.5
misc. rate	0.014	0.007	0.034
time	4.353	11.241	17.4170

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

embeding method	*tf*-2	full bag-of-words	*tfidf* 0.5
misc. rate	0.121	0.103	0.100
time	24.758	31.371	37.566

The results of the repeated sampling show that a large part of the classifications by the ANN have an exact match to the true values of the test data set. In very few samples was the classification incorrect. In these few cases the miscalssification rate amounted 10-20% (i.e. almost perfect classification). Therefore, misclassification rates over all samples are in the range of 1-12%. Classification for Example 1 shows better results. The variances of each simulation amounts to 0.1-0.2%. The 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. We can observe that embedding via *tf*-2 and full embedding gives very similar results. In Example 1 the full embedding shows the lowest misclassification ratio, whereas in Example 2 *tf-idf* embedding appears to be the best method. Note that there comes a small amount of uncertainty with the evaluation of the accuracy of ANNs, despite the iterative sampling approach used in this case. In other simulations I conducted the *tf*-2 embedding was found to be the most accurate. 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 to "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. In this case, the research questions are slightly modified and work towards 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, however differs 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. Since the aim of the example tested and proven in Chapter 4 was to find an optimal generic architecture for problems of this kind, the same hyper-parameters and fitting methods will be used here

Only a very small amount of preprocessing was necessary for the documents that were taken directly from the website. The table of contents had to be removed (for certain documents) in order to prevent their reoccurrence. The same stop words were excluded as in Chapter 4.[5] Consideration must be given to further circumstances of the regulatory documents. It is important to note 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]The stop word dictionaries "onix", "SMART" and "snow- ball" are used, as they are provided by the `tm` package [Silge and Robinson, 2017].

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_matching.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

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 degree of uncertainty depending on the aspects according to which it is clustered. Therefore, it is reasonable to compare the statistical clustering to a human-made grouping. This corresponds to the procedure of Chapter 4, where the clustering of books by the LDA algorithm was compared with the man-made classification of Gutenberg bookshelves (in fact - due to the long computing time - 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
7. Frames

The model was used with the same specifications as already known from the example in Chapter 4. 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, the comparison to a man-made grouping is not trivial for clustering, since 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 is reasonable to determine the aspects responsible for the algorithm clustering, 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 differentiate when deploying *tf-idf*, especially since the documents all originate from the underlying topic statistics.

Figure 10: Wordclouds for the clustered topics via LDA – using tfidf word proportions

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%.

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 once again 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-word's dimension remaining when shrinking via *tf-idf* embedding. This is visualized in the plot of Figure 11. 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. Similar to figure 8, the "low-dimension embedding anomaly" can also be observed in this plot. A range of lower misclassification rates for values of fraction between 0.3 and 0.4 shows slightly lower misclassification rates. 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-ifd* 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 rate. 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 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. This is consistent to the Gutenberg examples *tf*-2 embedding outperforms *tf-idf* embedding with fraction 0.5. Due to the lack of 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.

Still, these results are worse than what experienced in the Gutenberg data examples. But note that the use case differs considerably from the first ex-

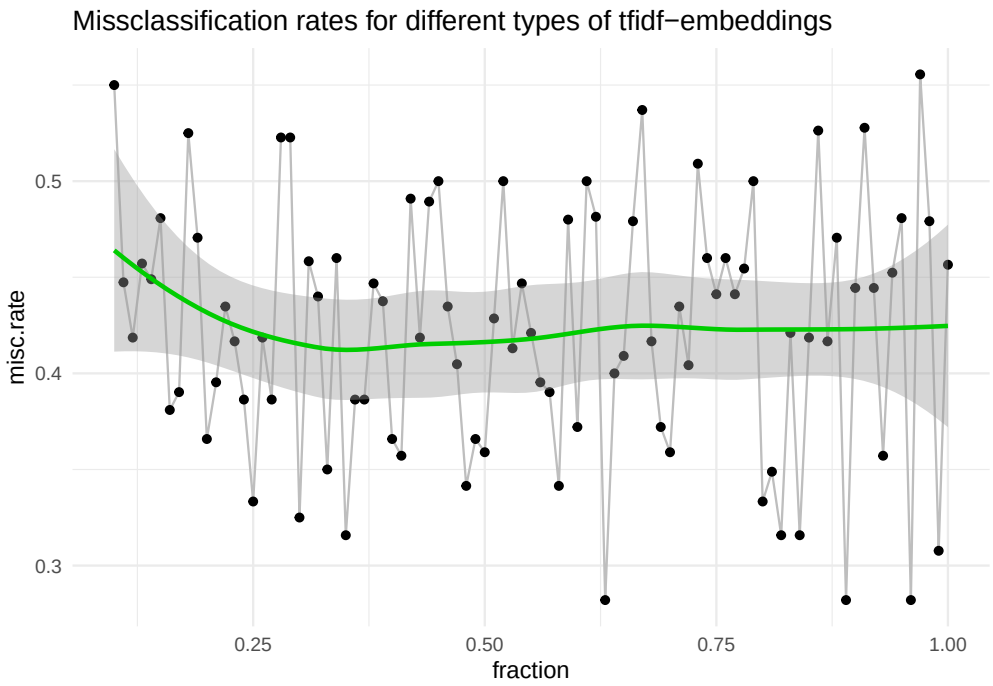


Figure 11: Fitting LDA via *tf-idf* embeding to EUROSTAT documents. Misclassification rates depending on the reduction of the bag-of-words. A big number of fraction stands for a larger bag-of-words, i.e. less reduction. The green smoothed curve is a Loess kernel.

Table 6: LDA via Gibbs Sampling - different embedding methods

embedding method	*tf*-2	*tf-idf* 0.50	*tf-idf* 0.35
misc. rate	0.342	0.359	0.316
time	14.818	17.878	14.818

ample, 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 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 instance 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]". It is recommended 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 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 five epochs and a batch size of 512. To illustrate the fitting and evaluation process I refer to the recap in Chapter 4.2. In this process there are some minor changes within step 5) (Iterative evaluation of the model). Due to the number of documents a split of 10% testing data, 17% validation data and 63% training data was used. To improve the computing time only 49 iterations instead of 59 iterations were used. Certainly, 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 results of the fits are promising considering, that we face a rather small

corpus. The misclassification rates for the different approaches amount between 0.37 and 0.42 (see Table 7). Compared to section 4.2, in this example it was even more apparent that the models with tf-2 and full embedding provide very similar results. Athough these results are satisfying, this model may be further optimized in order to potentially obtain even better results. More documents might even have a positive impact on the accuracy.

Table 7: ANN for EUROSTAT documents - different embedding methods

embedding method	*tf*-2	full *bag-of-words*	*tf-idf*
misc. rate	0.418	0.418	0.367
variance	0.108	0.149	0.133
time	2.666	7.668	11.792

The learning process of the architecture of a neural network found here as the most promising is illustrated in Figure 12.In the Chapter 3.2 the learning of neural networks has only been discussed at a very high level. To illustrate this process, the learning for the EUROSTAT document classification network that I proposed was visualized once again. In Figure 12 there are two plots, showing the trial of the accuracy and of the loss for each epoch (iteration in the learning process), for the training- and the validation set. Even if a single path of such a net is hardly meaningful - because a new fit can differ clearly from it - the plot still shows, for example, that the net does not yet overfit for five epochs.

Since the purpose of this paper is to find a generic approach for clustering and classifying documents, this 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 classification model (in this case an ANN) as desired. As with any type of statistical model, it must naturally 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 sets. 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.

6 Conclusions

In the course of this thesis the reader has already received a comprehensive introduction to NLP and its importance has been outlined. Two statistical

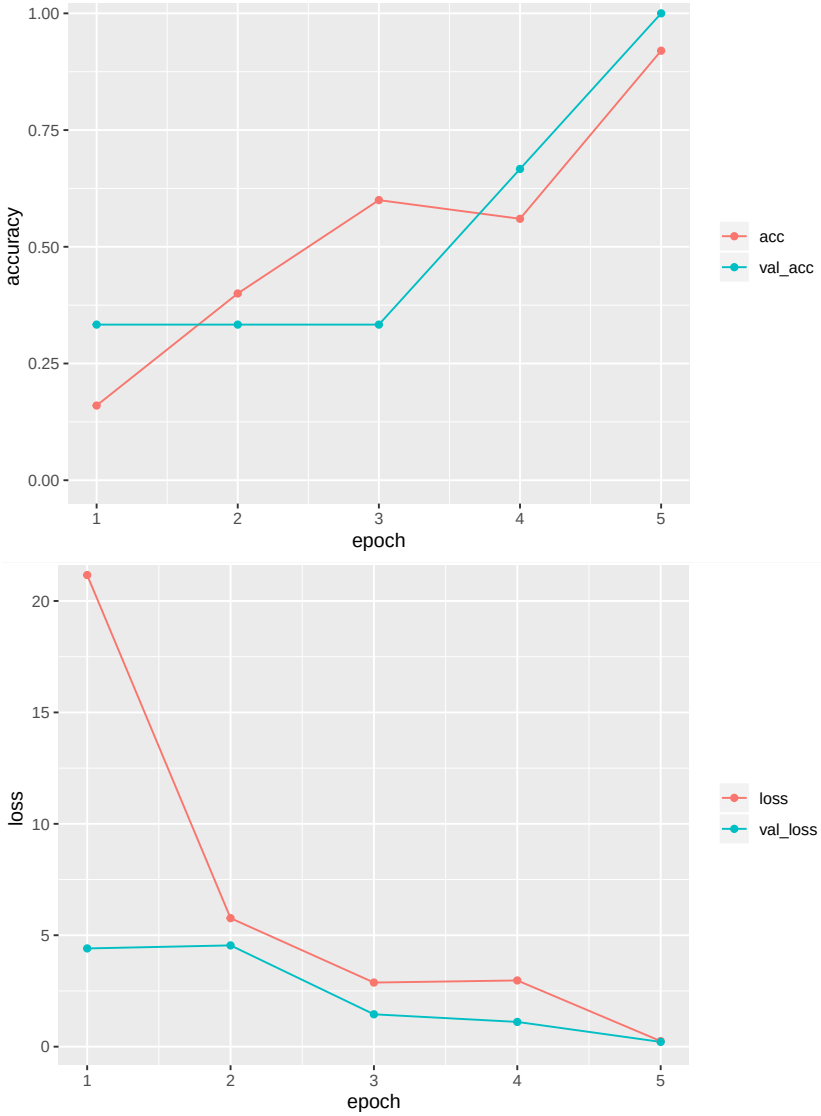


Figure 12: Plot of the learning process for the used neural network to classify the EUROSTAT documents.

models - NLP and ANN - to analyze text documents were explained in detail. At this point, the attentive reader should know the exact differences, weaknesses, and strengths of each model. The individual advantages and disadvantages were explained using two examples. In Chapter 4 exemplary textbooks were analyzed and in Chapter 5 technical, regulatory documents and guidelines of the ESS Vision 2020 ADMIN project of EUROSTAT were examined.

The two cases are not only examples to illustrate the methods, but serve for developing a generic approach for clustering and classifying text documents. Based on the Gutenberg book-documents, both models - LDA for clustering and artificial neural net for classifying new documents - were optimized. The free project Gutenberg offers a very good basis as it allows you to work with many different text documents of different sizes. Validation of a clustering model is challenging. Through the use of the Gutenberg documents and their metainformation, it was possible to tackle this issue and set up a well-proven LDA model. In the course of this analysis it was possible to show the weak performance of the LDA model for classifying new documents. For this reason, I applied a new, very versatile model for the classification of text documents. A simple but highly effective neural network has been adapted to classify text documents. The second use case - regulatory EUROSTAT documents - served to evaluate the previously established approach. In spite of the different nature of the documents, the methods were shown to be successful.

It can be concluded that the approach presented in this thesis - as a combination of methods - is a new process in the field of natural language processing. LDA is a well-known model in statistic text analysis. Neural networks in the form as applied in this thesis, are established and supported by different packages. Still, the combination of both methods is most likely scientifically evaluated the first time in this context. Although I believe this approach is a very promising process for special cases, more research in this field might help improving and optimizing it.

Further research in this field is required to improve this approach in the future. With this thesis only one approach to cluster and classify text documents was created, there is still a gap in knowledge to be explored. Research could be directly linked to the presented process, or it could concern the individual two models. However, in my opinion the following questions are very interesting and should be addressed in more detail.

- The impact of the bag-of-word's embedding method – by optimizing the *fraction* parameter for *tf-idf* embedding - on the clustering method's performance was analyzed. This analysis could not be con-

ducted with regards to the classification ANN due to a lack of computational power. It would be a worthwhile task to examine this hyperparameter and its effect on the performance of the classification model.

- In the work [Winter, 2017] *k-means* was used to classify text documents. It would be very interesting to investigate a direct comparison to the LDA model.

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 Rojal 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 Retrival - Algorithms and Heuristics*. Springer, 2 edition.

[Harris, 1951] Harris, Z. (1951). Methods in structural linguistics. *Linguistic Society of America*.

[Hart,] Hart, M. Project gutenberg.

[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.

[Raiffa and Schlaifer, 1961] Raiffa, H. and Schlaifer, R. (1961). *Applied Statistical Decision Theory.* Harvard University Press.

[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 'itdytext'.*

[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

List of Tables

Gutenberg Data via LDA - Documentation

Sebastian Knigge

6 8 2019

Contents

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 %>%
    regexpr(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 choosen.

```
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
##   gutenberg_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_chaper()* 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. gutenberg bookshelfes) are different. See Table (2) for comparison.

```
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",
```

Table 1: Book-titles

gutenberg_id	title	author
2095	Clotelle: 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
```

```
# number of documents
nrow(chapters_dtm)
```

```
## [1] 117
```

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 *tf-idf*. 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=(0.5*M))
ncol(chapters_dtm_tfidf)
```

```
## [1] 7593
```

2.3 Application of LDA Model to Full Corpus

2.3.1 Fitting via VEM

We set up the LDA model for the shrinked 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 shrinked dictionary/bag of words by *tf-idf*.

```
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 census and topic
    transmute(consensus = gutenberg_id, topic)

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

Now we exclude for each of the 3 embedings the - so called - beta matrix and compare the most likely result with the real results. Depending on how good the LDA will seperate 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("misc. rate", "time")
performance_matrix %>%  stargazer(summary=FALSE, header=F, title = "LDA via VEM")
```

Table 3: LDA via VEM

	freq2.embedding	all.embedding	tfidf
misc. rate	0.385	0.462	0.374
time	12.800	9.911	1.882

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 embeding 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 <- gutenberg_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

```
# number of documents
nrow(chapters_dtm_sec)
```

[1] 86

```
# 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))
ncol(chapters_dtm_tfidf_sec)
```

[1] 7664

```
# 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("misc. 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
misc. rate	0.227	0.362	0.222
time	7.986	7.496	1.468

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("misc. 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
misc. 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" (misclassification 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("misc. 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	0.146	0.179
2	15.460	20.365	7.150

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$train, method="Gibbs",
                             k = k, control = list(seed = 1234))
  # use the predict function of udpipe
  # the topic predict funtion already extract the most likely topics
  prediction <- predict(LDA_model, newdata=split$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)
  # misclassification 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 embeding via *tf-idf* 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 *tf-idf* 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
```

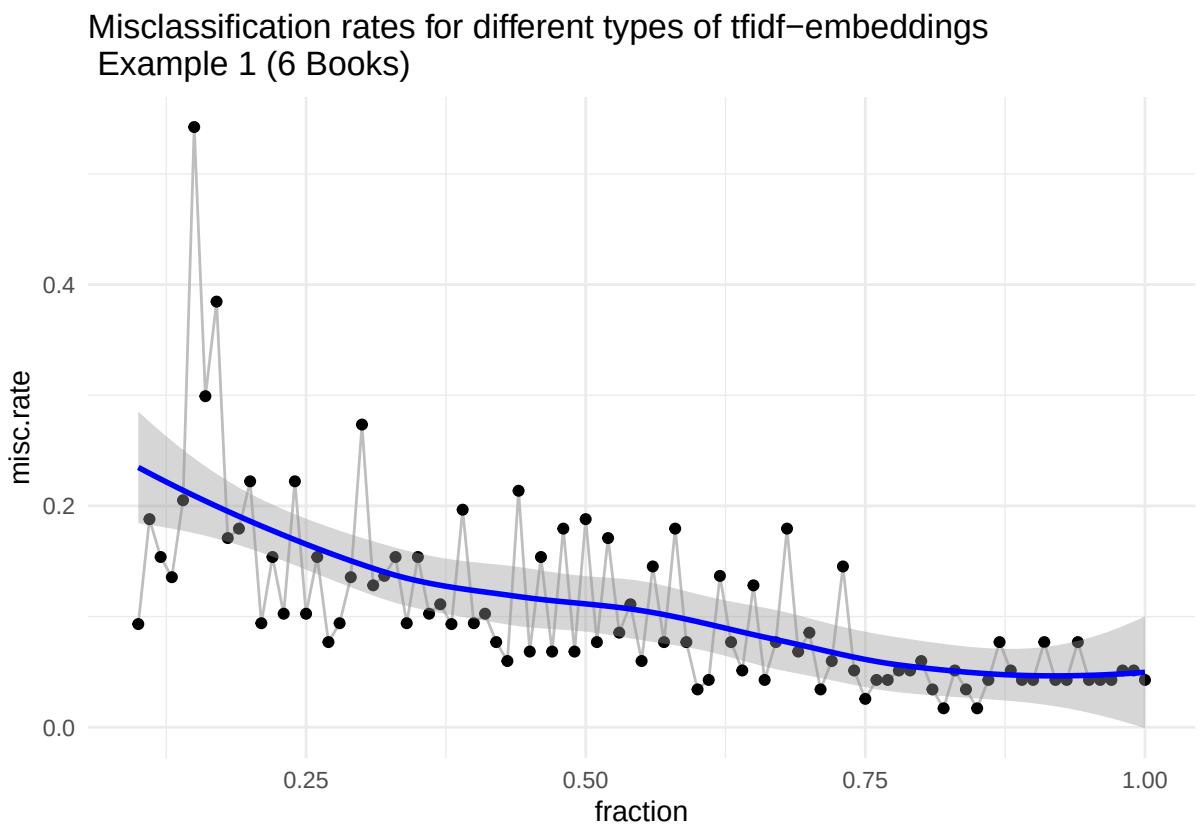
```
  tim1 <- Sys.time()
  chapters_lda_tfidf_Gibbs <- LDA(chapters_dtm_tfidf,  method="Gibbs",
                    k = n_books, control = list(seed = 1234))
  tim2 <- Sys.time()
  u_tfidf_Gibbs <- tim2-tim1
  # calculating misclassification rate
  misc.rate_tfidf_Gibbs <- ext_gamma_matrix(chapters_lda_tfidf_Gibbs) %>%
    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(.Glob


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


}
```

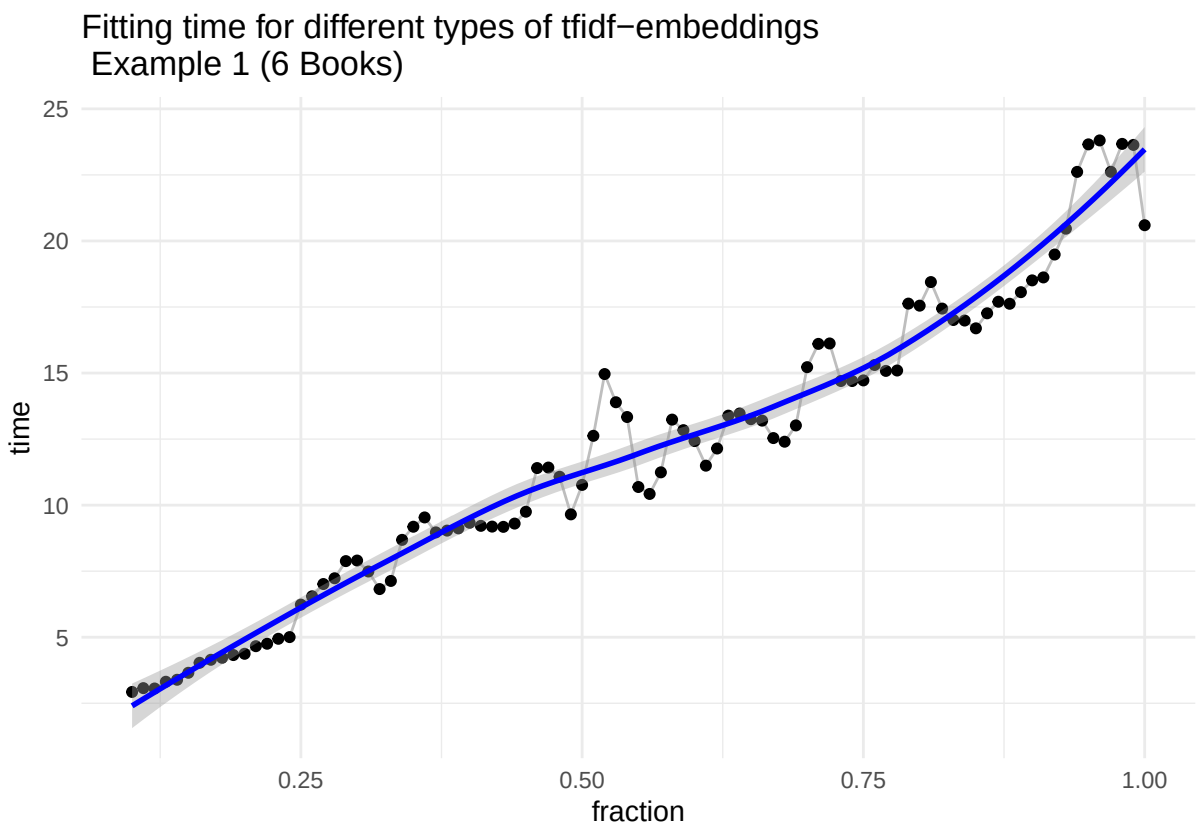
```
## Warning in e$fun(obj, substitute(ex), parent.frame(), e$data): already
## exporting variable(s): convert_to_dtm_2, evaluation_for_embedding,
## ext_gamma_matrix, fractions, M2, n, n_books, validate_LDAclassification,
## word_counts
```

```
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("Misclassification rates for different types of tfidf-embeddings \n Example 1 (6 Books)") +
  xlab("fraction")
```




```
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")
```



Now let us study Example 2.

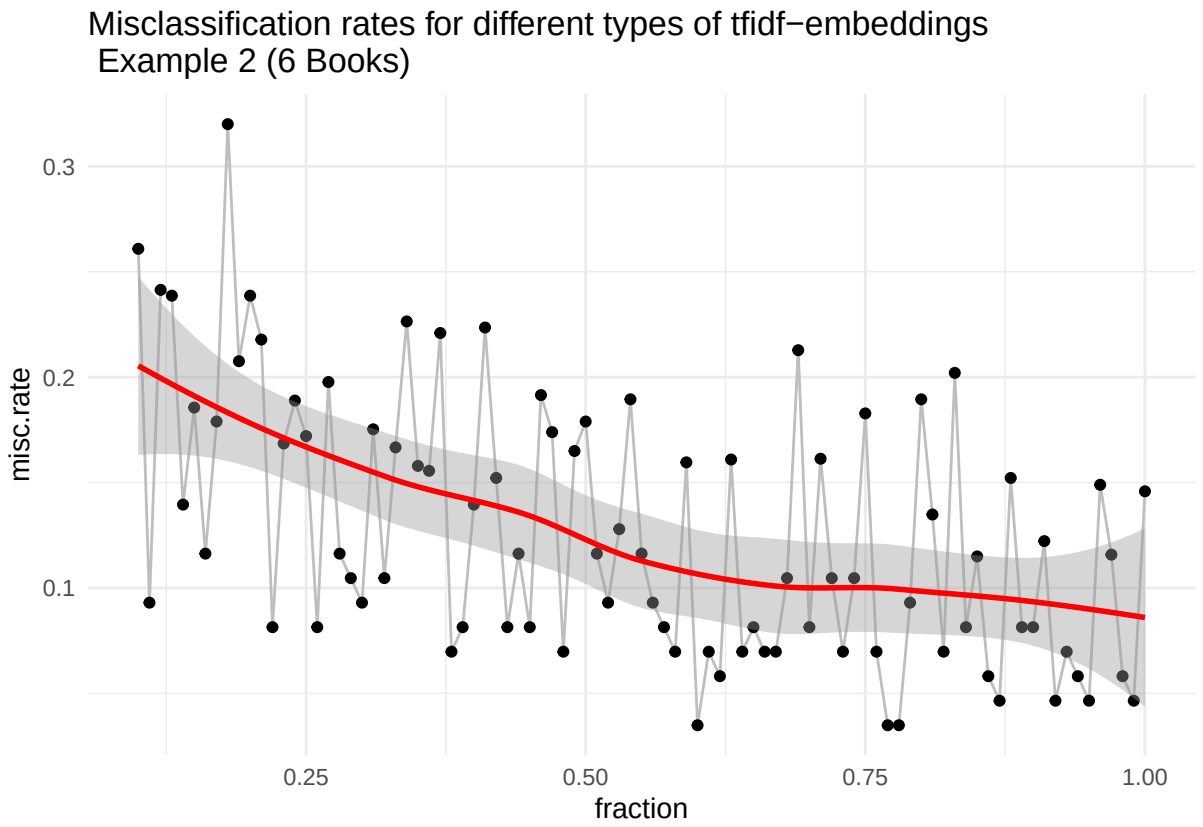
```
# set up the fractions we want to use
# we use 0.1, 0.2, ..., 1
fractions <- seq(0.1,1,0.01)

# 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_LDAclassification,
## word_counts_sec
```

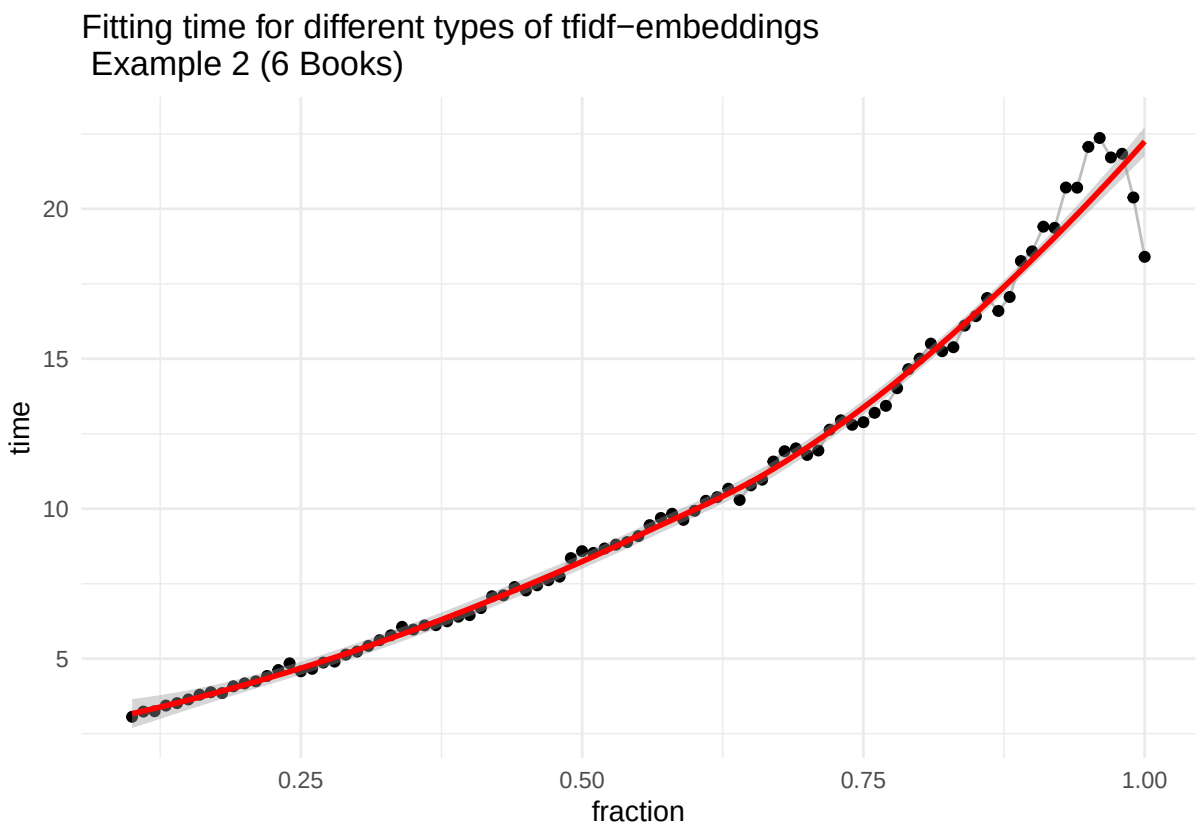
```
parallel::stopCluster(cl)

# 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("Misclassification rates for different types of tfidf-embeddings \n Example 2 (6 Books)")+
  xlab("fraction")
```




```
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")
```

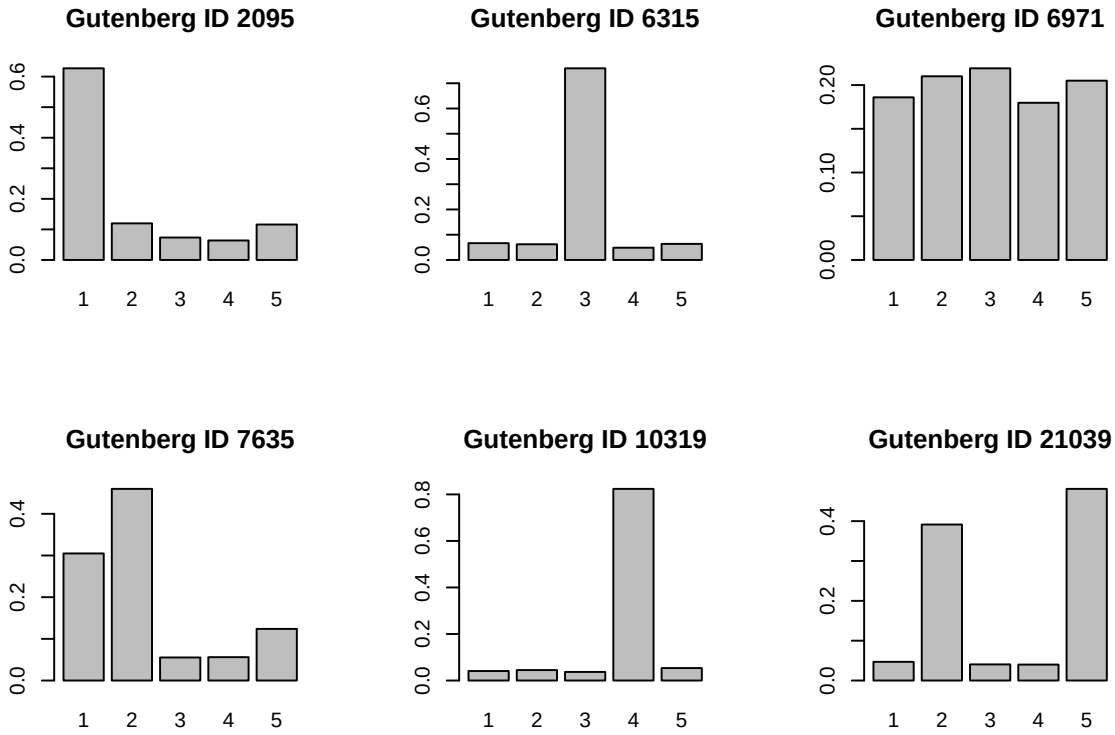


4 Definition of Validation

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 misclassification 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 misclassification 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 is 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

```
# number of documents
nrow(chapters_dtm_10)
```

[1] 230

```
# 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))
ncol(chapters_dtm_tfidf_10 )
```

[1] 14550

```r
# 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 exa
```

Table 9: LDA via Gibbs 10 books example

	freq2.embedding	all.embedding	tfidf
misc. rate	0.341	0.322	0.317
time	1.227	1.332	32.043

```r
# 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
```

```r
# 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("misc. rate", "time")
performance_matrix_10_VEM %>%  stargazer(summary=FALSE, header=F, title = "LDA via VEM 10 books example
```

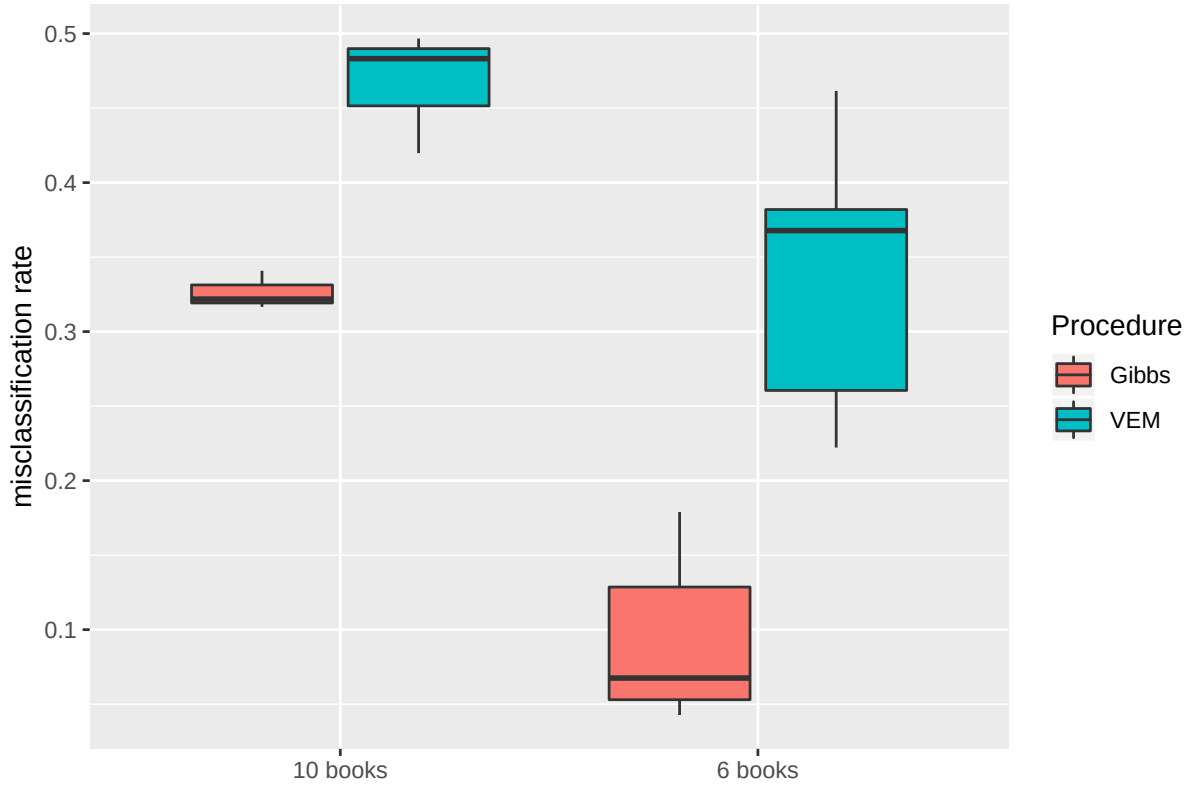
Table 10: LDA via VEM 10 books example

	freq2.embedding	all.embedding	tfidf
misc. rate	0.483	0.497	0.420
time	53.366	51.926	16.110

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

```r
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("misclassification rate")
```



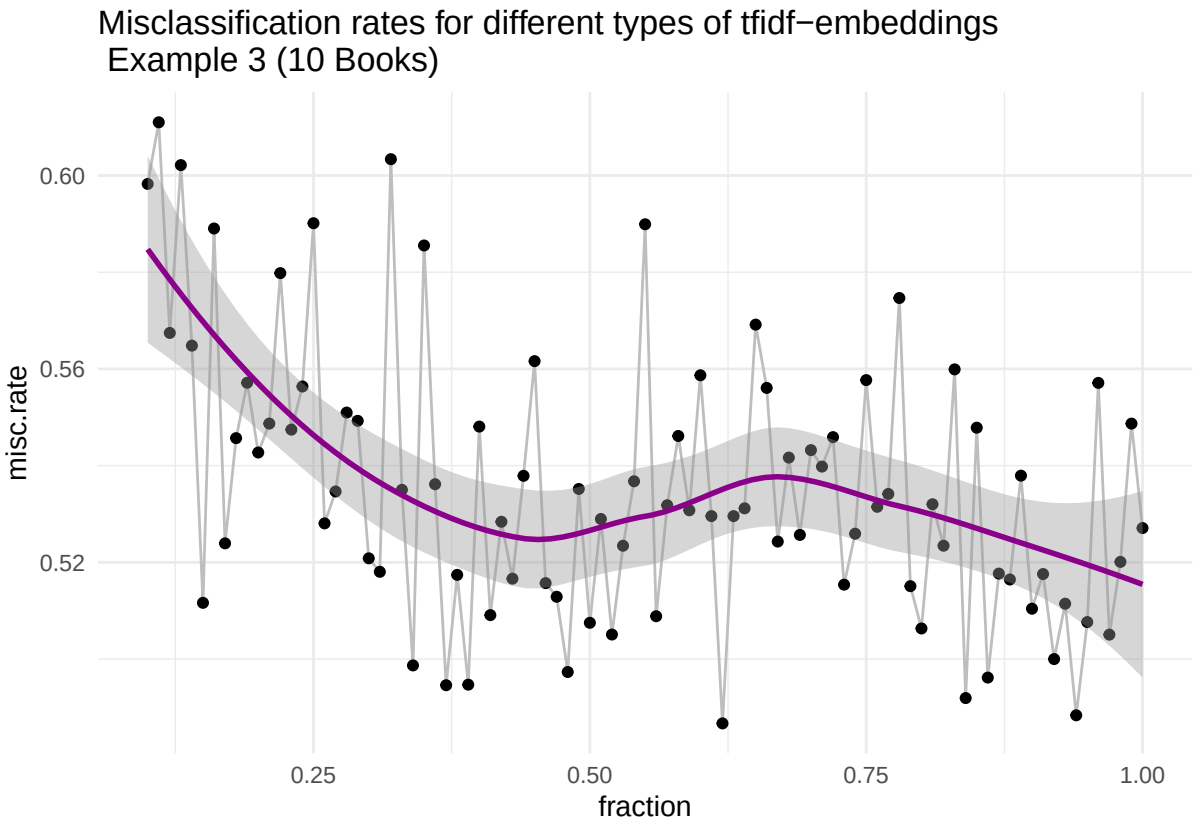
For this example we can study one more time the optimal value of "fraction". I.e. which fraction should remain after *tfidf* embedding.

```
# 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_10 <- foreach(i = 1:length(fractions), .combine = 'rbind', .export = ls(.G


  evaluation_for_embedding(word_counts10, 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_counts10
```

```
parallel::stopCluster(cl)

# adjust the resulting matrix to a data frame for plotting
colnames(embedding_performance_matrix_10) <- c("misc.rate","time","fractions")
embedding_performance_matrix_10 <- as.data.frame(embedding_performance_matrix_10)
```

```
ggplot(data=embedding_performance_matrix_10,
       aes(x=fractions, y=misc.rate)) +
  geom_line(col="grey") +
  geom_point() +
  geom_smooth(col="magenta4", method="loess", span=0.7) +
  theme_minimal() +
  ggtitle("Misclassification rates for different types of tfidf-embeddings \n Example 3 (10 Books)")+
  xlab("fraction")
```



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 embeddi
```

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

	freq2.embedding	all.embedding	tfidf
misc. rate	0.169	0.116	0.042
time	22.425	21.171	7.622

Gutenberg Data via Neural Net Documentation

Sebastian Knigge

5 8 2019

Contents

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 %>%
    regexpr(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 copus consisting out of 6 books, the function append_by_chaper() 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 *tf-idf*. 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 eually 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
# mispecification 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)
  mispecified <- sum(!pred==true_value)/length(pred)
  ret <- list(mispecified=mispecified,
              # the function also dicloses 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) %>% .$model %>%
    evaluate_model() %>% .$mispecified
}
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
```

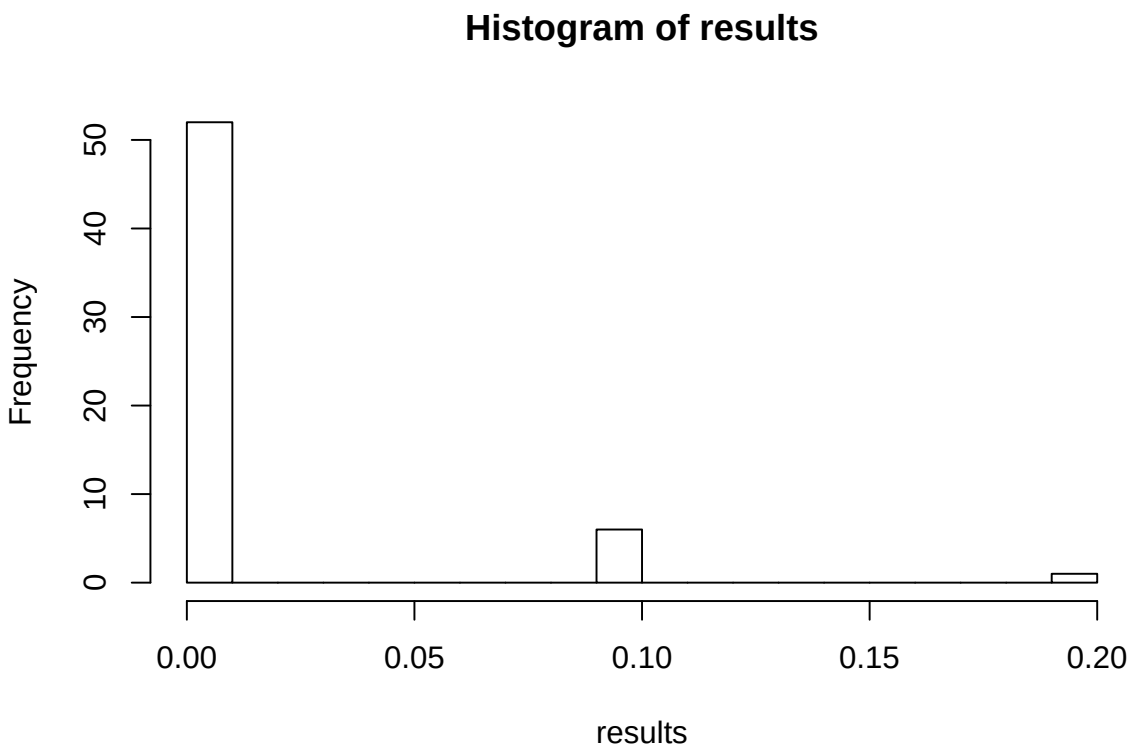
```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
## 0.00000 0.00000 0.00000 0.01356 0.00000 0.20000
```

```
results %>% var
```

```
## [1] 0.001537113
```

A histogram might be helpful in visualizing the results.

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



The fitting time might be interesting as well.

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

```
## Time difference of 4.353287 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) %>% .$model %>%
    evaluate_model() %>% .$mispecified
}
tim2_all <- Sys.time()
```

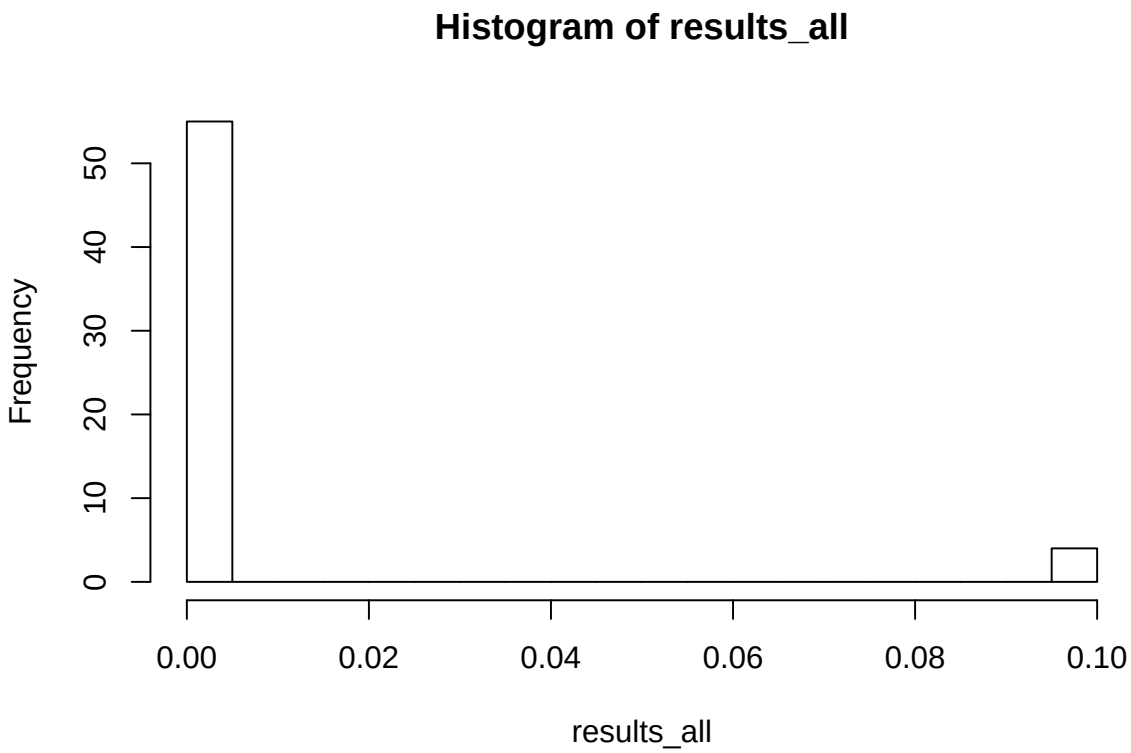
```
results_all %>% summary
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
## 0.00000 0.00000 0.00000 0.00678 0.00000 0.10000
```

```
results_all %>% var
```

```
## [1] 0.0006428989
```

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




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

```
## Time difference of 11.24102 mins
```

In this step we use the embeding via *tf-idf* and fit the model. The *tf-idf* 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) %>% .$model %>%
    evaluate_model() %>% .$mispecified
}
tim2_tfidf <- Sys.time()
```

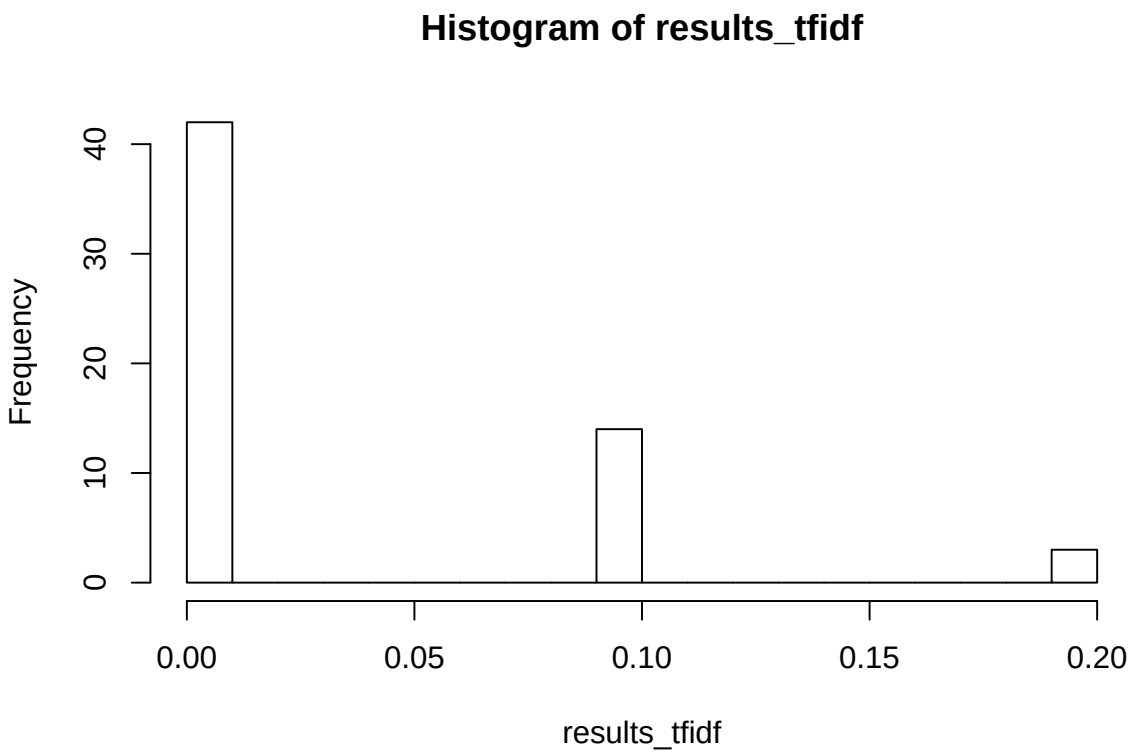
```
results_tfidf %>% summary
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##  0.0000  0.0000  0.0000  0.0339  0.1000  0.2000
```

```
results_tfidf %>% var
```

```
## [1] 0.003313852
```

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




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

```
## Time difference of 17.41653 mins
```

Table 2 summarises the results of all 3 different embeding mthods.

```
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	0.014	0.007	0.034
time	4.353	11.241	17.417

The time for the embedding using more than frequency 2 is very short. Whereas the embedding via *tf-idf* with the same dimensionality takes very long. It certainly makes sense that the mislcassification 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 halp 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) %>% .$model %>%
    evaluate_model() %>% .$mispecified
}
tim2 <- Sys.time()
# evaluate fitting time
(u1_sec <- tim2-tim1)
```

```
## Time difference of 24.758 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) %>% .$model %>%
    evaluate_model() %>% .$mispecified
}
tim2 <- Sys.time()
# evaluate fitting time
(u2_sec <- tim2-tim1)
```

```
## Time difference of 31.37081 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) %>% .$model %>%
    evaluate_model() %>% .$mispecified
}
tim2 <- Sys.time()
# evaluate fitting time
(u3_sec <- tim2-tim1)
```

```
## Time difference of 37.56571 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("misc. 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
misc. rate	0.121	0.103	0.100
time	24.758	31.371	37.566

Regulatory Documents via LDA - Documentation

Sebastian Knigge

18 8 2019

Contents

1 Setup

Following libraries are used in the code:

```
library(dplyr)
library(tidytext)
library(pdftools)
library(tidyr)
library(stringr)
library(tidytext)
library(udpipe)
library(topicmodels)
library(ggplot2)
library(wordcloud)
library(tm)
library(SnowballC)
library(RColorBrewer)
library(RCurl)
library(XML)
library(openxlsx)
library(keras)
library(foreach)
library(doParallel)
```

2 Import data

The Documents had to be preprocessed. For the documents wp2.5 all list of contents had to be deleted, because they were the same in each of these documents. No more adjustments had to be made.

In this code reulatory documents are red in and processed via LDA. This first part focusses on reading in the pdf documents.

```
# getting the right order
setwd('..')
documents <- read.xlsx("Docs_classes.xlsx")[,2]
classes <- read.xlsx("Docs_classes.xlsx")[,c(1,3)]
documents <- paste0(documents,".pdf")
documents %>% as.data.frame() %>% stargazer(summary=FALSE,
                                            header = FALSE,
                                            title="Document Titles")
```

Table 1: Document Titles

	.
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_matching.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

```
# getting the right directory
library(here)
setwd("../")
path <- getwd() %>%
  file.path("TextDocs")
```

```
setwd(path)
```

Following functions are used to set up and analyze the pdfs. When cleaning up data, we have to take into account certain circumstances of the regulatory documents. For example, there are many formulas and technical abbreviations in the documents. Every variable, every estimator, and every 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, we exclude all mixed words with characters and numeric values, as well as all terms with special characters (e.g. Greek letters).

```
read_pdf_clean <- function(document){
  # This function loads the document given per name
  # and excludes the stop words
  pdf1 <- pdf_text(file.path(path, document)) %>%
    strsplit(split = "\n") %>%
    do.call("c",.) %>%
    as_tibble() %>%
    unnest_tokens(word,value) %>%
    # also exclude all words which include numbers and special characters
    filter(grepl("^[a-z]+$", word))
  # load stopword library
  data(stop_words)
  # stop words are excluded via anti_join
  pdf1 %>%
    anti_join(stop_words)
}

plot_most_freq_words <- function(pdf, n=7){
  # plots a bar plot via ggplot
  pdf %>% count(word) %>% arrange(desc(n)) %>% head(n) %>%
    ggplot(aes(x=word,y=n)) +
    geom_bar(stat="identity")+
    # no labels for x and y scale
    theme(axis.title.y=element_blank(),
          axis.title.x=element_blank())
}
```

Now we can read in all documents using a for loop:

```
setwd(path)
# inital set up for the corpus
pdf1 <- read_pdf_clean(documents[1])
corpus <- tibble(document=1, word=pdf1$word)
# adding the documents iteratively
for (i in 2:length(documents)){
  pdf_i <- read_pdf_clean(documents[i])
  corpus <- tibble(document=i, word=pdf_i$word) %>% bind_rows(corpus,.)
}
```

3 LDA

The LDA model is applied. First the document term matrix has to be set up.

```
dtm <- corpus %>% count(document, word, sort = TRUE) %>%
  select(doc_id=document, term=word, freq=n) %>%
```

```
  document_term_matrix()
# dimensions
c(N,M) %<-% dim(dtm)
N; M
```

```
## [1] 28
```

```
## [1] 8254
```

We use term frequency 2 embedding because in the example with the Gutenberg Data, it turned out to be advantageous with regard to the "predictive power" of the LDA algorithm.

```
dtm_tf2 <- dtm %>%
    # reduce by low frequencies
    dtm_remove_lowfreq(minfreq = 2)
ncol(dtm_tf2 )
```

```
## [1] 5842
```

Using the function LDA sets up the model and prediction/evaluation is done via *predict*(). But first of all it shall be verified whether the Predict function actually delivers the same classification as the export of the gamma matrix directly from the LDA model. Therefore both gamma matrices of the single functions are compared. Table 2 displays the output of the gamma matrix received by the *predict*() function and Table 3 displays the gamma matrix returned by the LDA model itself.

```
tim1 <- Sys.time()
set.seed(123)
documents_lda <- LDA(dtm_tf2, method = "Gibbs",
                     k = 7, control = list(seed = 1234))
tim2 <- Sys.time()
u1 <- tim2 - tim1
```

```
prediction5 <- predict(documents_lda, newdata=dtm_tf2, type="topic")

prediction5 <- merge(prediction5, classes, by.x="doc_id", by.y="No")

prediction5 %>%
  select(doc_id,topic_001,topic_002,topic_003,topic_004,topic_005,
         topic_006, topic_007) %>%
  mutate_each(funs(as.numeric),
              doc_id,topic_001,topic_002,topic_003,topic_004,
              topic_005, topic_006, topic_007) %>%
  arrange(desc(-doc_id)) %>%
  round(2) %>%
  stargazer(summary=F, rownames = F, header = F,
            title="Gamma matrix for predict function", label="predict")
```

```
ext_gamma_matrix <- function(model=documents_lda){
  # get gamma matrix for chapter probabilities
  chapters_gamma <- tidy(model, matrix = "gamma")
  # get matrix with probabilities for each topic per chapter
  spreaded_gamma <- chapters_gamma %>% spread(topic, gamma)
  spreaded_gamma %>%
    mutate_each(funs(as.numeric), document,1,2,3,4,5,6,7) %>%
  arrange(desc(-document))
}
```

Table 2: Gamma matrix for predict function

doc_id	topic_001	topic_002	topic_003	topic_004	topic_005	topic_006	topic_007
1	0	0.950	0	0.030	0.010	0	0.010
2	0.010	0.760	0.010	0.160	0.030	0.030	0.010
3	0.160	0.020	0.060	0.030	0.630	0.050	0.050
4	0.180	0.010	0.030	0.020	0.710	0.020	0.040
5	0.720	0	0.080	0.010	0.160	0.020	0.010
6	0.600	0.010	0.230	0.030	0.120	0.010	0.010
7	0.830	0.010	0.080	0.010	0.040	0.020	0.020
8	0.760	0.020	0.030	0.010	0.130	0.030	0.020
9	0.750	0.010	0.090	0.030	0.060	0.030	0.030
10	0.900	0	0.030	0.010	0.050	0.010	0.010
11	0.690	0.010	0.170	0.010	0.100	0.020	0.010
12	0.800	0.010	0.030	0.010	0.120	0.030	0.010
13	0.010	0.080	0.010	0.410	0.470	0.020	0.010
14	0.170	0.010	0.030	0.030	0.730	0.020	0.010
15	0.010	0.020	0.010	0.020	0.180	0.750	0.010
16	0.010	0	0.770	0	0.120	0.080	0.010
17	0.010	0.010	0.100	0.030	0.050	0.040	0.760
18	0.010	0	0.770	0	0.130	0.080	0.010
19	0.060	0	0.450	0.010	0.100	0.320	0.060
20	0.030	0.010	0.040	0.020	0.040	0.840	0.020
21	0.010	0	0.940	0.010	0.020	0.010	0.010
22	0.010	0.010	0.910	0.020	0.030	0	0.010
23	0.010	0.010	0.820	0.030	0.040	0.050	0.050
24	0.010	0.010	0.860	0.010	0.080	0.010	0.010
25	0.130	0	0.820	0.010	0.010	0.020	0.010
26	0.010	0.010	0.010	0.100	0.160	0.060	0.650
27	0	0.100	0.010	0.690	0.020	0.010	0.180
28	0.010	0.060	0.010	0.670	0.020	0.180	0.050

```
ext_gamma_matrix(documents_lda) %>%
  round(2) %>%
  stargazer(summary=F, rownames = F, header=F,
            title="Gamma matrix extracted from model",
            label="extract")
```

Table 3: Gamma matrix extracted from model

document	1	2	3	4	5	6	7
1	0	0.94	0	0.03	0.01	0	0.01
2	0.01	0.75	0	0.16	0.03	0.03	0.01
3	0.15	0.02	0.07	0.03	0.62	0.05	0.05
4	0.18	0.01	0.03	0.03	0.7	0.02	0.04
5	0.71	0.01	0.09	0	0.16	0.02	0.01
6	0.59	0.02	0.23	0.02	0.12	0.01	0.01
7	0.8	0.01	0.07	0.01	0.04	0.03	0.03
8	0.73	0.01	0.05	0.01	0.14	0.04	0.02
9	0.73	0.01	0.09	0.02	0.07	0.03	0.05
10	0.88	0	0.04	0.01	0.06	0	0.02
11	0.67	0.01	0.19	0.01	0.1	0.01	0.01
12	0.79	0.01	0.04	0.01	0.12	0.02	0.01
13	0.01	0.08	0.01	0.4	0.47	0.02	0.01
14	0.18	0.01	0.03	0.04	0.72	0.02	0.01
15	0.02	0.02	0.02	0.02	0.18	0.73	0.02
16	0.01	0.01	0.76	0.01	0.13	0.08	0.01
17	0.01	0.01	0.1	0.03	0.05	0.05	0.74
18	0.01	0	0.76	0	0.13	0.08	0.01
19	0.07	0.01	0.45	0.01	0.1	0.31	0.06
20	0.05	0.01	0.03	0.02	0.05	0.82	0.03
21	0.01	0	0.92	0.01	0.02	0.02	0.01
22	0.01	0	0.89	0.02	0.04	0.01	0.02
23	0.01	0.01	0.82	0.03	0.04	0.05	0.04
24	0.02	0.01	0.87	0.01	0.08	0.01	0.01
25	0.13	0.01	0.81	0.02	0.01	0.02	0.01
26	0.01	0.02	0.01	0.11	0.16	0.06	0.64
27	0.01	0.09	0.01	0.68	0.03	0.01	0.17
28	0.01	0.06	0.02	0.66	0.02	0.19	0.05

The tables below summarize which document refers to which topic, according to the LDA model (term frequency 2 embedding).

3.1 Wordclouds

To check what topics tackle which context, we produce wordclouds using the *tf-idf* and the *tf* itself.

```
plot_wordcloud <- function(corpus, selection="ALL",
                           max.words=50, i, freq="tfidf",
                           scale=c(3,0.2)){
  # setting up a tibble which returns tfidf and tf and frequency for
  # the whole corpus
  tfidf <- corpus %>% count(document, word, sort = TRUE) %>%
    bind_tf_idf(word, document, n)
```

Table 4: Documents for Topic 1

Topic	doc_id	Group
1	10	4
1	11	4
1	12	4
1	5	2
1	6	3
1	7	4
1	8	4
1	9	4

Table 5: Documents for Topic 2

Topic	doc_id	Group
2	1	1
2	2	1

Table 6: Documents for Topic 3

Topic	doc_id	Group
3	16	5
3	18	5
3	19	5
3	21	6
3	22	6
3	23	6
3	24	6
3	25	6

Table 7: Documents for Topic 4

Topic	doc_id	Group
4	27	7
4	28	7

Table 8: Documents for Topic 5

Topic	doc_id	Group
5	13	3
5	14	4
5	3	2
5	4	2

Table 9: Documents for Topic 6

Topic	doc_id	Group
6	15	5
6	20	5

Table 10: Documents for Topic 7

Topic	doc_id	Group
7	17	5
7	26	7

```
  # include all documents for selection if selection="ALL"
  if (all(selection=="ALL")) {
    selection <- corpus %>%
      select(document) %>%
      unique() %>%
      unlist() %>%
      sort()}
  # filter for all selected documents
  # use either ft or tfidf
  if (freq=="tfidf"){
    dtm_selected <- tfidf %>% filter(document%in%selection) %>%
      select(word, tf_idf) %>% count(word, wt=tf_idf, sort=TRUE)
  } else {
    dtm_selected <- tfidf %>% filter(document%in%selection) %>%
      select(word, tf) %>% count(word, wt=tf, sort=TRUE)
  }
  # plotting
  wordcloud(words = dtm_selected$word, freq = dtm_selected$n,
            min.freq = 1,max.words=max.words, random.order=FALSE,
            colors=brewer.pal(8, "Dark2"), scale=scale,
            main="Title", use.r.layout = TRUE)
  # set a title = document
  text(x=0.5, y=1, paste("Topic", i))
}
```

A second possibility is to extract the *tf-idf*s of the words linked to the topics directly. First you have to map the topics to the documents within the tidytext format. This is the only way the tfidf_tf matrix can be set up for the individual topics.

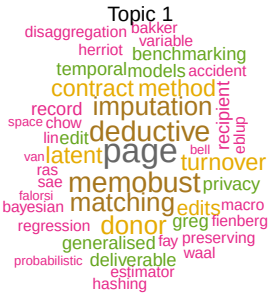
```
plot_wordcloud_topic <- function(corpus, topic_select=1, prediction=prediction5,
                           max.words=50,
                           scale=c(3,0.2)){
  # get the correct topic mapping via the first two columns
  # of the prediction matrix
  new_corpus <- prediction %>%
    transmute(doc_id=as.numeric(doc_id), topic) %>%
    right_join(corpus, c("doc_id"="document"))
  # setting up a tibble which returns tfidf and tf and frequency for
  # the whole corpus
```

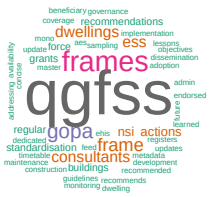
```
  tfidf <- new_corpus %>% count(topic, word, sort = TRUE) %>%
    bind_tf_idf(word, topic, n) %>%
    # filter for all selected topics
    # use either ft or tfidf
    filter(topic==topic_select)
  # plotting
  wordcloud(words = tfidf$word, freq = tfidf$tf_idf,
            min.freq = 1,max.words=max.words, random.order=FALSE,
            colors=brewer.pal(8, "Dark2"), scale=scale,
            main="Title", use.r.layout = TRUE)
  # set a title = document
  text(x=0.5, y=1, paste("Topic", topic_select))
}
```

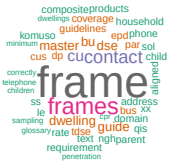
3.1.1 Wordclouds using *tf-idf*

For getting specific and more individual words for each cloud, we use the *tf-idf* in the first step. Now there are two methods to create the word clouds for the *tf-idf* measure. Once by aggregating the individual *tf-idf*s of the documents, as it is done in the following.

```
par(mfrow=c(2,4))
par(mar=c(1,1,1,1))
set.seed(123)
plot_wordcloud(corpus, selection=ind1[,1], i=1, scale=c(1.7,0.001),
               max.words = 45)
plot_wordcloud(corpus, selection=ind2[,1], i=2)
plot_wordcloud(corpus, selection=ind3[,1], i=3, scale=c(1.6,0.005),
               max.words = 45)
plot_wordcloud(corpus, selection=ind4[,1], i=4)
plot_wordcloud(corpus, selection=ind5[,1], i=5, scale=c(1.8,0.001),
               max.words = 40)
plot_wordcloud(corpus, selection=ind6[,1], i=6, scale=c(1.8,0.001),
               max.words = 40)
plot_wordcloud(corpus, selection=ind7[,1], i=7, scale=c(2.1,0.3),
               max.words = 45)
```







Now using the second approach, when applying the *tf-idf* measure to the mapped corpus.

```
par(mfrow=c(2,4))
par(mar=c(1,1,1,1))
set.seed(123)
plot_wordcloud_topic(corpus, topic_select=1, scale=c(1.9,0.0006),
                max.words = 40)
plot_wordcloud_topic(corpus, topic_select=2)
plot_wordcloud_topic(corpus, topic_select=3, scale=c(1.6,0.005),
                max.words = 40)
plot_wordcloud_topic(corpus, topic_select=4, max.words = 50, scale=c(2.3, 0.2))
plot_wordcloud_topic(corpus, topic_select=5, scale=c(1.8,0.01),
                max.words = 40)
plot_wordcloud_topic(corpus, topic_select=6, scale=c(2,0.1),
                max.words = 45)
plot_wordcloud_topic(corpus, topic_select=7, scale=c(2.5,0.5),
                max.words = 50)
```

Although the second procedure, using *tf-idf* distributions for the individual topics, seems to be more intuitive, the two figures are surprisingly similar.

3.1.2 Wordclouds using *tf*

The same can be done using the regular term frequency.

```
par(mfrow=c(2,4))
par(mar=c(1,1,0.5,1))
set.seed(123)
plot_wordcloud(corpus, selection=ind1[,1], i=1, freq="tf", scale=c(2,0.05))
plot_wordcloud(corpus, selection=ind2[,1], i=2, freq="tf", scale=c(2,0.05))
plot_wordcloud(corpus, selection=ind3[,1], i=3, freq="tf", scale=c(2,0.03),
               max.words = 40)
plot_wordcloud(corpus, selection=ind4[,1], i=4, freq="tf", scale=c(2,0.3))
plot_wordcloud(corpus, selection=ind5[,1], i=5, freq="tf", scale=c(2.5,0.1))
plot_wordcloud(corpus, selection=ind6[,1], i=6, freq="tf")
plot_wordcloud(corpus, selection=ind7[,1], i=7, freq="tf", scale=c(2.5,0.2),
               max.words = 45)
```

3.2 Embedding via *tf-idf*

Now it is interesting to see if embedding via *tf-idf* will cluster other groups or the same. So we will reduce the Document Term Matrix to 5717 words, which amounts to a reduction by 50%.

```
tim1_tfidf <- Sys.time()
dtm_50 <- dtm %>% dtm_remove_tfidf(top=(0.5*M))
set.seed(123)
documents_lda_2 <- LDA(dtm_50, method="Gibbs",
                       k = 7, control = list(seed = 1234))
tim2_tfidf <- Sys.time()
u2 <- tim2_tfidf - tim1_tfidf
```

```
prediction5_2 <- predict(documents_lda_2, newdata=dtm_50, type="topic")
prediction5_2 <- merge(prediction5_2, classes, by.x="doc_id", by.y="No")
# compare topic 1 with topic 2, 3, 4 and 5
ind1_2 <- prediction5_2 %>% filter(topic==1) %>% select(doc_id, Group)
ind2_2 <- prediction5_2 %>% filter(topic==2) %>% select(doc_id, Group)
ind3_2 <- prediction5_2 %>% filter(topic==3) %>% select(doc_id, Group)
ind4_2 <- prediction5_2 %>% filter(topic==4) %>% select(doc_id, Group)
ind5_2 <- prediction5_2 %>% filter(topic==5) %>% select(doc_id, Group)
ind6_2 <- prediction5_2 %>% filter(topic==6) %>% select(doc_id, Group)
ind7_2 <- prediction5_2 %>% filter(topic==7) %>% select(doc_id, Group)
```

```
ext_gamma_matrix(documents_lda_2) %>%
  round(2) %>%
  stargazer(summary=F, rownames = F, header=F,
            title="Gamma matrix extracted from model for embedding with tfidf",
            label="extract2")
```

Table 11: Documents for Topic 1

Topic_embedding_0.8	doc_id	Group
1	10	4
1	11	4
1	12	4
1	5	2
1	6	3
1	7	4
1	8	4
1	9	4

Table 12: Documents for Topic 2

Topic_embedding_0.8	doc_id	Group
2	15	5
2	25	6

Table 13: Documents for Topic 3

Topic_embedding_0.8	doc_id	Group
3	13	3
3	14	4
3	3	2
3	4	2

Table 14: Documents for Topic 4

Topic_embedding_0.8	doc_id	Group
4	1	1
4	2	1

Table 15: Documents for Topic 5

Topic_embedding_0.8	doc_id	Group
5	16	5
5	18	5
5	19	5
5	21	6
5	22	6
5	23	6
5	24	6

Table 16: Documents for Topic 6

Topic_embedding_0.8	doc_id	Group
6	20	5
6	28	7

Table 17: Documents for Topic 7

Topic_embedding_0.8	doc_id	Group
7	17	5
7	26	7
7	27	7

Table 18: Gamma matrix extracted from model for embedding with tfidf

document	1	2	3	4	5	6	7
1	0	0	0	0.99	0	0	0
2	0.01	0.02	0.03	0.86	0.01	0.07	0.01
3	0.16	0.02	0.54	0.01	0.18	0.03	0.05
4	0.2	0.02	0.68	0.02	0.02	0.02	0.04
5	0.85	0.03	0.04	0.01	0.07	0.01	0.01
6	0.69	0	0.02	0.01	0.25	0.01	0.01
7	0.85	0.04	0.02	0.01	0.05	0.01	0.03
8	0.86	0.02	0.03	0.02	0.03	0.01	0.03
9	0.72	0.06	0.08	0.01	0.06	0.03	0.05
10	0.95	0	0.01	0	0.02	0.01	0
11	0.73	0.01	0.03	0.01	0.19	0.01	0.01
12	0.9	0.02	0.03	0.01	0.02	0.01	0.01
13	0.01	0.02	0.76	0.15	0	0.01	0.04
14	0.24	0.02	0.68	0.02	0.02	0.01	0.01
15	0.01	0.83	0.09	0.02	0.02	0.01	0.02
16	0.01	0.06	0.01	0.01	0.9	0.01	0.01
17	0.01	0.05	0.01	0.01	0.07	0.01	0.84
18	0.01	0.06	0.01	0	0.89	0.01	0.02
19	0.04	0.42	0.02	0.01	0.43	0.02	0.07
20	0.01	0.05	0.02	0.01	0.01	0.88	0.01
21	0.01	0.01	0.01	0.01	0.96	0.01	0.01
22	0.01	0.01	0.01	0.01	0.94	0.01	0.01
23	0	0.02	0.03	0.01	0.9	0	0.03
24	0.02	0.01	0.02	0.01	0.92	0.01	0.01
25	0.01	0.91	0.01	0.01	0.05	0.01	0.01
26	0.01	0.01	0.06	0.03	0.02	0.05	0.82
27	0.01	0.01	0.03	0.19	0	0.02	0.75
28	0.01	0.03	0.04	0.05	0.01	0.73	0.13

We want to give an overview over the clustered documents using the *tf-idf* embedding.

Table 19: Documents for Topic 1

Topic	doc_id	Group
1	10	4
1	11	4
1	12	4
1	5	2
1	6	3
1	7	4
1	8	4
1	9	4

Table 20: Documents for Topic 2

Topic	doc_id	Group
2	15	5
2	25	6

Table 21: Documents for Topic 3

Topic	doc_id	Group
3	13	3
3	14	4
3	3	2
3	4	2

Table 22: Documents for Topic 4

Topic	doc_id	Group
4	1	1
4	2	1

Table 23: Documents for Topic 5

Topic	doc_id	Group
5	16	5
5	18	5
5	19	5
5	21	6
5	22	6
5	23	6
5	24	6

We want to produce wordclouds again. This time using the *tf-idf* embedding for clustering via LDA. The first plot shows the aggregated *tf-idf*s.

Table 24: Documents for Topic 6

Topic	doc_id	Group
6	20	5
6	28	7

Table 25: Documents for Topic 7

Topic	doc_id	Group
7	17	5
7	26	7
7	27	7

```
par(mfrow=c(2,4))
par(mar=c(1,1,0.5,1))
set.seed(123)
plot_wordcloud(corpus, selection=ind1_2[,1], i=1, scale=c(1.7,0.002),
               max.words = 35)
plot_wordcloud(corpus, selection=ind2_2[,1], i=2, scale=c(2.5,0.1))
plot_wordcloud(corpus, selection=ind3_2[,1], i=3, scale=c(1.5,0.001),
               max.words = 30)
plot_wordcloud(corpus, selection=ind4_2[,1], i=4, scale=c(1.9,0.05),
               max.words = 40)
plot_wordcloud(corpus, selection=ind5_2[,1], i=5, scale=c(2,0.02),
               max.words = 35)
plot_wordcloud(corpus, selection=ind6_2[,1], i=6, scale=c(2,0.1),
               max.words = 40)
plot_wordcloud(corpus, selection=ind7_2[,1], i=7, scale=c(2,0.03),
               max.words = 35)
```

The second plot shows the *tf-idf*s for each individual topic.

```
par(mfrow=c(2,4))
par(mar=c(1,1,1,1))
set.seed(123)
plot_wordcloud_topic(corpus, topic_select=1, scale=c(1.9,0.0006),
                max.words = 40, prediction=prediction5_2)
plot_wordcloud_topic(corpus, topic_select=2, prediction=prediction5_2,
                     scale=c(1.9,0.0006))
plot_wordcloud_topic(corpus, topic_select=3, scale=c(1.6,0.005),
                max.words = 40, prediction=prediction5_2)
plot_wordcloud_topic(corpus, topic_select=4, max.words = 40,
                     scale=c(2.3, 0.02), prediction=prediction5_2)
plot_wordcloud_topic(corpus, topic_select=5, scale=c(1.8,0.01),
                max.words = 40, prediction=prediction5_2)
plot_wordcloud_topic(corpus, topic_select=6, scale=c(2,0.1),
                max.words = 45, prediction=prediction5_2)
plot_wordcloud_topic(corpus, topic_select=7, scale=c(2.5,0.5),
                max.words = 50, prediction=prediction5_2)
```

3.3 Missclassification Rates

Now we use the validation measure we used for Example 1.

```
validate_LDAclassification <- function(predict_table){
  # gamma_matrix ... an object of the function ext_gamma_matrix()
  # First we'd find the topic that was most associated with
  # each chapter
  conversion <- predict_table %>%
    select(Group, topic) %>%
    group_by(Group) %>%
    top_n(1,topic) %>%
    unique()

  consensus <- predict_table %>%
    left_join(conversion, by=c("topic"))
  consensus %>% filter(Group.x!=Group.y) %>%
    nrow()/nrow(consensus)
}
```

On both full bag of words and 50% embedding via *tf-idf*

```
predict_table <- prediction5 %>% select(doc_id, topic) %>%
  merge( y=classes, by.x=1, by.y=1)

( misc.rate_embedding2 <- validate_LDAclassification(predict_table) )
```

```
## [1] 0.3421053
```

```
predict_table2 <- prediction5_2 %>% select(doc_id, topic) %>%
  merge( y=classes, by.x=1, by.y=1)
```

```
( misc.rate_embedding_tfidf <- validate_LDAclassification(predict_table2))
```

```
## [1] 0.3589744
```

```
performance_matrix <- data.frame(freq2.embedding=c(misc.rate_embedding2, u1),
           tfidf.embedding=c(misc.rate_embedding_tfidf, u2))
rownames(performance_matrix) <- c("missc. rate", "time")
performance_matrix %>%  stargazer(summary=FALSE, header=F, title = "LDA via Gibbs Sampling")
```

Table 26: LDA via Gibbs Sampling

	freq2.embedding	tfidf.embedding
missc. rate	0.342	0.359
time	41.608	17.878

4 Optimal value of Reduction via tfidf

In this example we want to know if there is a better value for the reduction of tfidf than 50%. We will examine it using the same approach as for the Gutenberg examples.

```
evaluation_for_embedding <- function(word_counts, frac=0.1, M=8254) {
  # embedding 3
  chapters_dtm_tfidf <- dtm %>% dtm_remove_tfidf(top=(frac*M))
  # fitting
  tim1 <- Sys.time()
  chapters_lda_tfidf_Gibbs <- LDA(chapters_dtm_tfidf,  method="Gibbs",
                    k = 7, control = list(seed = 1234))
  tim2 <- Sys.time()
  u_tfidf_Gibbs <- tim2-tim1
  prediction5_2 <- predict(chapters_lda_tfidf_Gibbs,
                           newdata=chapters_dtm_tfidf, type="topic")
  prediction5_2 <- merge(prediction5_2, classes, by.x="doc_id", by.y="No")
  # calculating missclassification rate
  predict_table <- prediction5_2 %>% select(doc_id, topic) %>%
  merge( y=classes, by.x=1, by.y=1)

  misc.rate_tfidf_Gibbs <-validate_LDAclassification(predict_table)

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

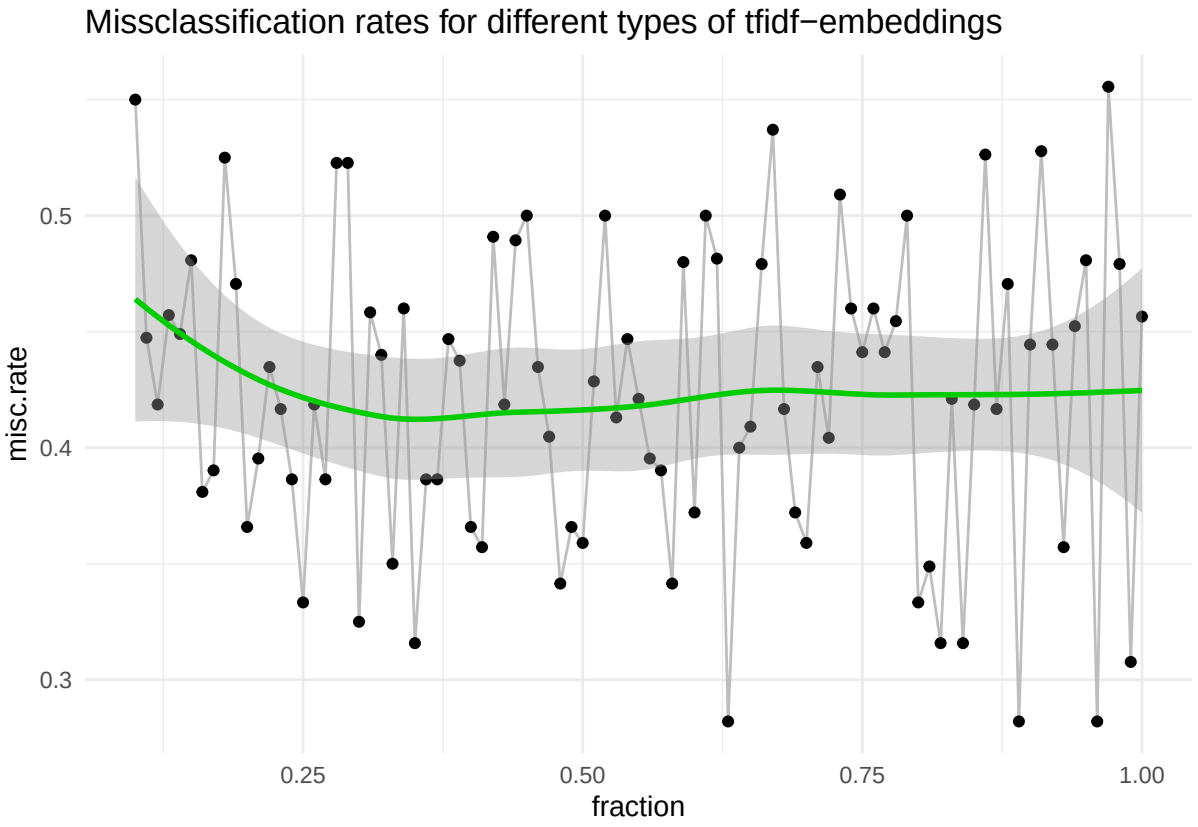
```
# set up the fractions we want to use
# we use 0.1, 0.2, ..., 1
fractions <- seq(0.1,1,0.01)

# Use parallelization
# setting up how many cores to be used
useable_cores <- parallel::detectCores() - 1
# registering cluster
```

```
cl <- parallel::makeCluster(useable_cores)
doParallel::registerDoParallel(cl)
embedding_performance_matrix <- foreach(i = 1:length(fractions), .combine = 'rbind', .export = ls(.Globa


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


}
```

```
## Warning in e$fun(obj, substitute(ex), parent.frame(), e$data): already
## exporting variable(s): classes, dtm, evaluation_for_embedding, fractions,
## validate_LDAclassification
```

```
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="green3", method="loess", span=0.7) +
  theme_minimal() +
  ggtitle("Missclassification rates for different types of tfidf-embeddings") +
  xlab("fraction")
```



We can find an optimal value for *fraction* at the value aroung 0.35, so it might be interesting to campare the results of a fit using this optimized value of fraction to the other embeddings.

```
tim1_tfidf0.85 <- Sys.time()
dtm_0.85 <- dtm %>% dtm_remove_tfidf(top=(0.35*M))
set.seed(123)
documents_lda_0.85 <- LDA(dtm_0.85, method="Gibbs",
                    k = 7, control = list(seed = 1234))
tim2_tfidf0.85 <- Sys.time()
u0.85 <- tim2_tfidf0.85 - tim1_tfidf0.85
```

```
prediction0.85 <- predict(documents_lda_0.85, newdata=dtm_0.85, type="topic")
prediction0.85 <- merge(prediction0.85, classes, by.x="doc_id", by.y="No")
```

```
predict_table0.85 <- prediction0.85 %>% select(doc_id, topic) %>%
  merge( y=classes, by.x=1, by.y=1)

( misc.rate_embedding_0.85 <- validate_LDAclassification(predict_table0.85))
```

```
## [1] 0.3157895
```

In Table 27 you will find an overview of the performace of the different embedding methods including *tfidf* embedding of fraction 0.35.

```
performance_matrix <- data.frame(freq2.embedding=c(misc.rate_embedding2, u1),
           tfidf_0.5.embedding=c(misc.rate_embedding_tfidf, u2),
           tfidf_0.85.embedding=c(misc.rate_embedding_0.85, u0.85))
rownames(performance_matrix) <- c("missc. rate", "time")
performance_matrix %>%  stargazer(summary=FALSE, header=F,
```

```
                                    title = "LDA via Gibbs Sampling", label="comp0.85")
```

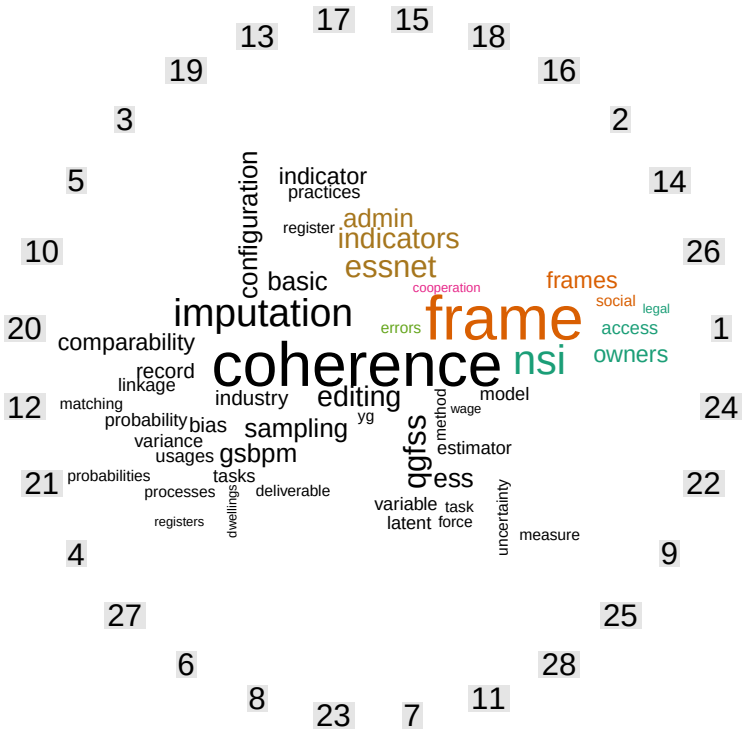
Table 27: LDA via Gibbs Sampling

	freq2.embedding	tfidf_0.5.embedding	tfidf_0.85.embedding
missc. rate	0.342	0.359	0.316
time	41.608	17.878	14.818

5 Coherence Cloud

```
dtm_50 %>% as.matrix() %>% t() %>% comparison.cloud(scale=c(2,.05),
                                                    max.words=50, title.size = 1)
```

```
## Warning in brewer.pal(max(3, ncol(term.matrix)), "Dark2"): n too large, allowed maximum for palette
## Returning the palette you asked for with that many colors
```



Regulatory Documents via Neural Nets - Documentation

Sebastian Knigge

18 9 2019

Contents

1 Setup

Following libraries are used in the code:

```
library(dplyr)
library(tidytext)
library(pdftools)
library(tidyr)
library(stringr)
library(tidytext)
library(udpipe)
library(topicmodels)
library(ggplot2)
library(wordcloud)
library(tm)
library(SnowballC)
library(RColorBrewer)
library(RCurl)
library(XML)
library(openxlsx)
library(keras)
```

2 Import data

The Documents had to be preprocessed like for the LDA part. This first part - again - focusses on reading in the pdf documents.

```
# getting the right order
setwd('..')
documents <- read.xlsx("Docs_classes.xlsx")[,2]
classes <- read.xlsx("Docs_classes.xlsx")[,c(1,3)]
documents <- paste0(documents,".pdf")
```

Following functions are used to set up and analyze the pdfs.

```
# getting the right directory
library(here)
setwd("../")
path <- getwd() %>%
  file.path("TextDocs")
setwd(path)
```

When cleaning up data, we have to take into account certain circumstances of the regulatory documents. For example, there are many formulas and technical abbreviations in the documents. Every variable, every estimator, and every 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, we exclude all mixed words with characters and numeric values, as well as all terms with special characters (e.g. Greek letters).

```
read_pdf_clean <- function(document){
  # This function loads the document given per name
  # and excludes the stop words inclusive numbers
  pdf1 <- pdf_text(file.path(path, document)) %>%
    strsplit(split = "\n") %>%
    do.call("c",.) %>%
    as_tibble() %>%
    unnest_tokens(word,value) %>%
    # also exclude all words including numbers and special characters
    filter(grepl("^[a-z]+$", word))
  # load stopword library
  data(stop_words)
  # add own words to stop word library - here the numbers from 1 to 10
  new_stop_words <- tibble(word=as.character(0:9),
                           lexicon=rep("own",10)) %>%
                           bind_rows(stop_words)
  # stop words are excluded via anti_join
  pdf1 %>%
    anti_join(new_stop_words)
}

plot_most_freq_words <- function(pdf, n=7){
  # plots a bar plot via ggplot
  pdf %>% count(word) %>% arrange(desc(n)) %>% head(n) %>%
    ggplot(aes(x=word,y=n)) +
    geom_bar(stat="identity")+
    # no labels for x and y scale
    theme(axis.title.y=element_blank(),
          axis.title.x=element_blank())
}
```

Now we can read in all documents in a for loop:

```
# inital set up for the corpus
pdf1 <- read_pdf_clean(documents[1])
corpus <- tibble(document=1, word=pdf1$word)
# adding the documents iteratively
for (i in 2:length(documents)){
  pdf_i <- read_pdf_clean(documents[i])
  corpus <- tibble(document=i, word=pdf_i$word) %>% bind_rows(corpus,.)
```

```
}
```

3 Apply ANN

```
dtm <- corpus %>% count(document, word, sort = TRUE) %>%
  select(doc_id=document, term=word, freq=n) %>%
  document_term_matrix()
c(N,M) %<-% dim(dtm)
```

We will reduce the dimensionality again (i.e. performing embedding).

```
dtm_embedding2 <- dtm %>%
  dtm_remove_lowfreq(minfreq = 2)
# ordering
dtm_embedding2 <- dtm_embedding2[order(as.numeric(rownames(dtm_embedding2))),]

dtm_embedding_all <- dtm %>%
  dtm_remove_lowfreq(minfreq=0)
# ordering
dtm_embedding_all <- dtm_embedding_all[order(as.numeric(rownames(dtm_embedding_all))),]

dtm_embedding_tfidf <- dtm %>%
  dtm_remove_tfidf(top=M*0.5)
# ordering
dtm_embedding_tfidf <- dtm_embedding_tfidf[order(as.numeric(rownames(dtm_embedding_tfidf))),]
```

```
# convert x matrix into a form such that it can be used for tensorflow
adjust_tensor_format <- function(classes, dtm){
  x_chapters <- apply(dtm, 1, function(x) as.matrix(x)) %>% t()
  # one hot encoding for the Groups (y)
  topics_categorical <- classes$Group %>% -1 %>%
    to_categorical()
  ret <- list(
    x=x_chapters,
    y=topics_categorical,
    topics=classes$Group
  )
  return(ret)
}
```

```
dtm_tensor_2 <- adjust_tensor_format(classes, dtm_embedding2)
dtm_tensor_all <- adjust_tensor_format(classes, dtm_embedding_all)
dtm_tensor_tfidf <- adjust_tensor_format(classes, dtm_embedding_tfidf)
```

3.1 Splitting and Fitting

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

```
sample_cluster_wise <- function(data, test_ratio=0.2, 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()
  n_val <- (table(cluster)*val_ratio) %>% floor()
  # 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 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, epochs=5, batchsize=2^9){
  # 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 7 categories
    layer_dense(units=7, 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=epochs,
    batch_size=batchsize,
    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, split) {
  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 dicloses 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. Here we use a split of 20% validation-, 17% test- and 60% training-set. All 3 embeddings are evaluated. We use 49 simulations to test each embedding.

```
tim1 <- Sys.time()
n <- 49
results <- rep(NA,n)
for(i in 1:n){
  split <- sample_cluster_wise(dtm_tensor_2, seed=i*201, test_ratio = 0.17)
  results[i] <- set_up_n_fit(split) %>% .$model %>%
    evaluate_model(split=split) %>% .$misspecified
}
tim2 <- Sys.time()
```

```
u_2 <- tim2-tim1
results_2 <- results
misc.rate_2 <- results_2 %>% mean
var_2 <- results_2 %>% var
```

```
tim1 <- Sys.time()
n <- 49
results <- rep(NA,n)
for(i in 1:n){
  split <- sample_cluster_wise(dtm_tensor_all, seed=i*201, test_ratio = 0.17)
  results[i] <- set_up_n_fit(split) %>% .$model %>%
    evaluate_model(split=split) %>% .$misspecified
}
tim2 <- Sys.time()
```

```
u_all <- tim2-tim1
results_all <- results
misc.rate_all <- results_all %>% mean
var_all <- results_all %>% var
```

```
tim1 <- Sys.time()
n <- 49
results <- rep(NA,n)
for(i in 1:n){
  split <- sample_cluster_wise(dtm_tensor_tfidf, seed=i*201, test_ratio = 0.17)
  results[i] <- set_up_n_fit(split) %>% .$model %>%
```

```
    evaluate_model(split=split) %>% .$misspecified
}
tim2 <- Sys.time()
```

```
u_tfidf <- tim2-tim1
results_tfidf <- results
misc.rate_tfidf <- results_tfidf %>% mean
var_tfidf <- results_tfidf %>% var
```

```
# overview
performance_matrix <- data.frame(freq2.embedding=c(misc.rate_2, var_2, u_2),
           all.embedding=c(misc.rate_all, var_all, u_all),
           tfidf=c(misc.rate_tfidf, var_tfidf, u_tfidf))
rownames(performance_matrix) <- c("missc. rate", "variance", "time")
performance_matrix %>%  stargazer(summary=FALSE, header=F, title = "ANN for Reg. Docs.")
```

Table 1: ANN for Reg. Docs.

	freq2.embedding	all.embedding	tfidf
missc. rate	0.418	0.418	0.367
variance	0.108	0.149	0.133
time	2.666	7.668	11.792

3.2 Different Architecture

In this section we want to try a different architecture. Using batch size 2^5 instead of 2^9 In Table 2 the results of this architecture are shown, including the variances of the missclassification rates over all 49 simulations.

```
n <- 49
results_2_d <- rep(NA,n)
for(i in 1:n){
  split <- sample_cluster_wise(dtm_tensor_2, seed=i*201, test_ratio = 0.17)
  results_2_d[i] <- set_up_n_fit(split, batchsize = 2^5) %>% .$model %>%
    evaluate_model(split=split) %>% .$misspecified
}
```

```
n <- 49
results_all_d <- rep(NA,n)
for(i in 1:n){
  split <- sample_cluster_wise(dtm_tensor_all, seed=i*201, test_ratio = 0.17)
  results_all_d[i] <- set_up_n_fit(split, batchsize = 2^5) %>% .$model %>%
    evaluate_model(split=split) %>% .$misspecified
}
```

```
n <- 49
results_tfidf_d <- rep(NA,n)
for(i in 1:n){
  split <- sample_cluster_wise(dtm_tensor_tfidf, seed=i*201, test_ratio = 0.17)
  results_tfidf_d[i] <- set_up_n_fit(split, batchsize = 2^5) %>% .$model %>%
    evaluate_model(split=split) %>% .$misspecified
}
```

```r
# overview
performance_matrix <- data.frame(freq2.embedding=c(results_2_d %>% mean, results_2_d %>% var),
           all.embedding=c(results_all_d %>% mean, results_all_d %>% var),
           tfidf=c(results_tfidf_d %>% mean, results %>% var))
rownames(performance_matrix) <- c("missc. rate", "variance")
performance_matrix %>%  stargazer(summary=FALSE, header=F, title = "ANN for Reg. Docs. Different Archit
```

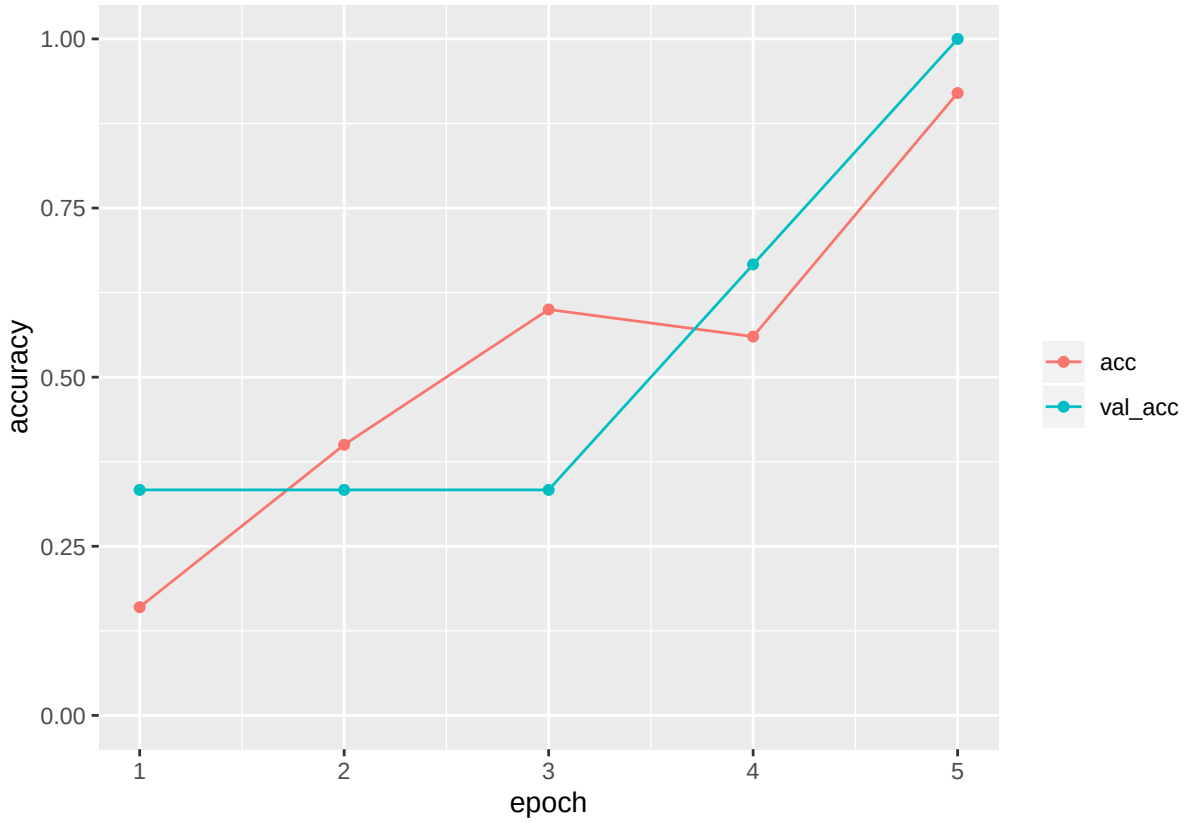
Table 2: ANN for Reg. Docs. Different Architecture (7 epochs)

	freq2.embedding	all.embedding	tfidf
missc. rate	0.418	0.418	0.388
variance	0.118	0.118	0.125

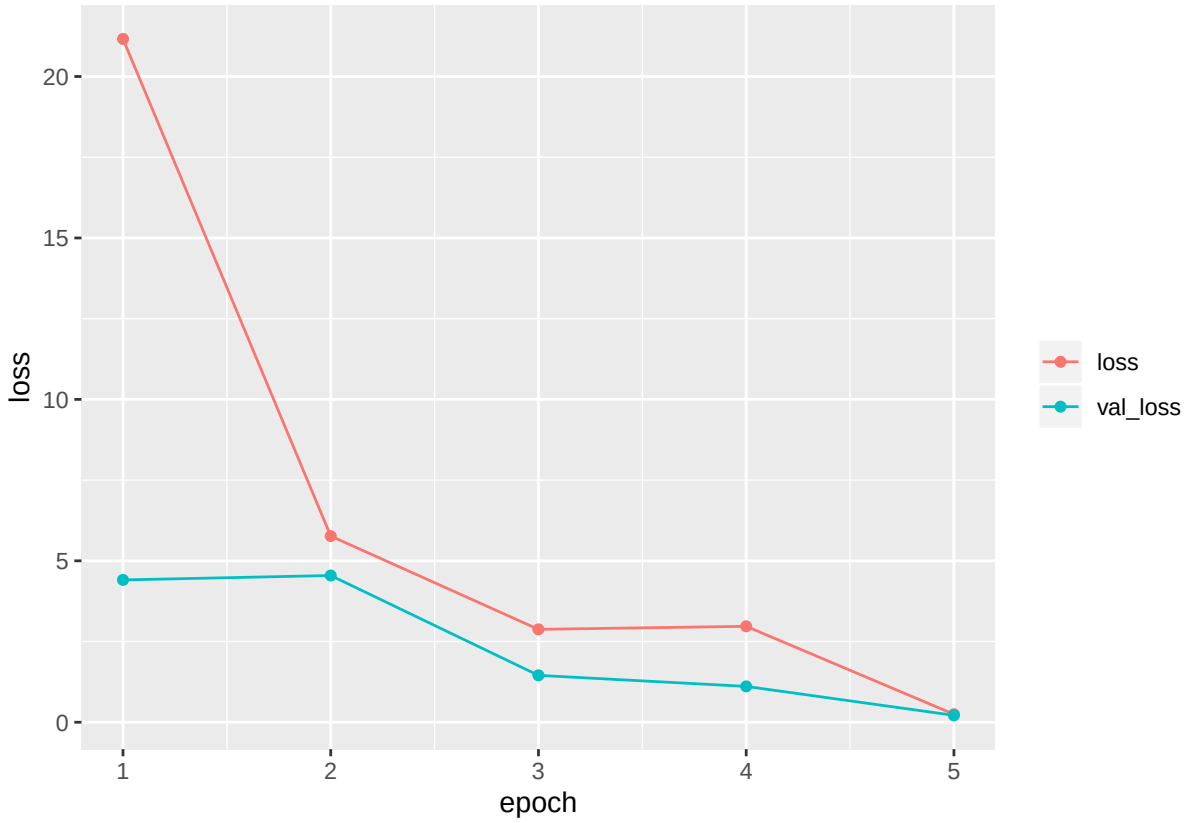
4 Training of the most Promising Structure

In the course of training and optimizing, many hundret models were fitted. In the following I want to show the training-plot of the most promising approach of the ANN for the regultory documents.

```r
{
set.seed(1234)

split <- sample_cluster_wise(dtm_tensor_2, test_ratio=0)
hist <- set_up_n_fit(split, batchsize = 2^9) %>% .$history

plot_df <- hist$metrics %>% do.call("cbind",.) %>% cbind(.,1:5) %>% as.data.frame()

plot_df <- plot_df %>% gather(value, key=measure, acc, loss, val_acc, val_loss)

}


ggplot(subset(plot_df, measure %in% c("acc", "val_acc")), aes(V5, value)) +
  geom_line(aes(color=factor(measure))) +
  geom_point(aes(color=factor(measure))) +
  ylim(c(0,1))+
  xlab("epoch")+
  ylab("accuracy")+
  theme(legend.title = element_blank())
```




```
ggplot(subset(plot_df, measure %in% c("loss", "val_loss")), aes(V5, value)) +
  geom_line(aes(color=factor(measure))) +
  geom_point(aes(color=factor(measure)))+
  xlab("epoch")+
  ylab("loss")+
  theme(legend.title = element_blank())
```


20
15
10
5
0
loss
1
2
3
4
5
epoch
loss
val_loss

B Zusammenfassung

Im Forschungsfeld Natural Language Processing überschneiden sich viele wissenschaftliche Disziplinen. Diese Arbeit beleuchtet statistische Herangehensweisen des Natural Language Processing. Untersucht werden die beiden Modelle Latent Dirichlet Allocation und Künstliche Neuronale Netze. Dem Leser wird eine Einführung in die Theorie gegeben und anhand zweier unterschiedlicher Beispiele sollen die Modelle und deren Einsatz veranschaulicht werden.

Ziel der Arbeit ist die Entwicklung und Evaluierung einer generischen Methodik zum Clustern und effektiven Klassifizieren von Dokumenten. Dazu verwende ich in einem grundliegenden Schritt Textbücher als Dokumente, um die beiden Modelle anzupassen. Diese Dokumente sind bereits durch die Initiatoren der freien Online-Bibliothek Project Gutenberg kategorisiert. Somit ist es möglich den Clustering Algorithmus an einem menschgemachten Gruppierungsprinzip zu evaluieren. Die so entwickelten Modelle werden im zweiten Schritt an regulatorischen Dokumenten und Anleitungen der EUROSTAT (einer Behörde der EU) letztlich getestet und ebenfalls mit einer bereits existierenden Einteilung abgeglichen.

Bei der verwendeten Open Source Software handelt es sich um R. Alle verwendeten Codes sind im Anhang dieser Arbeit zu finden und vollständig reproduzierbar.