# Information retrieval - Topic modeling

Quentin De Haes - S0172574 - quentin.dehaes@student.uantwerpen.be
Stijn Rosaer - S0172195 - stijn.rosaer@student.uantwerpen.be

December 2020

## 1 Introduction

In this assignment, we try to implement a basic topic model for text articles. This is done by implementing a variation on the topic model Latent Dirichlet Allocation [1], namely Gibbs sampling.

We chose to implement this algorithm ourselves based on the provided Youtube lectures. Before applying the algorithm we performed preprocessing to normalize the dataset of news articles. Next, we executed our algorithm for a differing number of topics. This allowed us to see the effect of changing this parameter and we tried out different number of iterations over the data to see the improvements that were made. Finally we build a list with the documents that are mostly related to each topic.

## 2 Implementation

Since we did not use a preexisting library for the Latent Dirichlet Allocation, we will be discussing the implementation of our algorithm in this section. It was mentioned to us that the speed of the algorithm was not that important, but we did need to make sure that a significant amount of documents was processed. We choose to limit the amount of documents that are passed through the algorithm to 10000 which resulted in an acceptable run-time.

### 2.1 Preprocessing

In order to make the data more usable, we performed many different preprocessing techniques on it [2]. Such as the removal of punctuation and

special characters, the capitalization was changed to lower case and excessive spaces were removed. We also removed letters that stand on their own because they have no meaning and were present in the original text.

The lemmatization of words was originally attempted,the singularization of words to be precise, but this took too much time for it to be use-able.

```
import re
df["content"] = df["content"].str.lower() # to lower case

df['content'] = df['content'].map(lambda x: re.sub(r'[,!.;+-@
    !%^&*)(_\\\'\"-]', '', str(x))) # remove special characters

df['content'] = df['content'].map(lambda x: re.sub(r'\W+', '␣',
    str(x))) #remove exess white spaces

df['content'] = df['content'].map(lambda x:  re.sub('(\\b[A-Za-z
    ]␣\\b|\\b␣[A-Za-z]\\b)', '', str(x))) # remove single letters
```

Code Listing 1: text preprocesing

Along with that, in order to make the words we topic actually mean anything, we remove stopwords from the documents as well. We use the NLTK library to get our list of stopwords to remove. We also added some words to this list ourselves, because, after some testing, we found some words which lacked any meaning still among our processed data.

## 2.2 Gibbs sampling

the first step in our algorithm is assigning each word a uniformly random topic. This is needed because we need to have a basic understanding of what the topic of a word in a document is. Notice that this is has no meaning at the moment, but by subsequently executing the algorithm, a more meaningful topic will be assigned to each word.

Gibbs sampling [3][4] works on a word by word basis meaning that we change the previous topic assignment of a single word. If we pretend to not know the topic assignment of this specific word, but do know the assignment of all the other words, we can compute the probability that a topic should be assigned to this unknown word. This is done by making assumptions about a word having the same topic regardless of context, and a document having a single topic. When doing this approach is done for each word, once per iteration.

We will find a conditional probability distribution of a single words topic assignment, based on the rest of the words' topics.

$$p(z_{d,n} = k|\vec{z}_{-d,n}, \vec{w}, \alpha, \gamma) = \frac{n_{d,k} + \alpha_k}{\sum_i^K n_{d,i} + \alpha_i} \frac{v_{k,w_{d,n}} + \gamma_{w_{d,n}}}{\sum_i v_{k,i} + \gamma_i}$$

The first part of the right side indicates how much a topic is related to a document and the second part gives information about how much a word is related to a topic.

- $n_{d,k}$ is the number of times document d uses topic k.
- $v_{k,w_{d,n}}$ is the number of times topic k uses word type $w_{d,n}$.
- $\alpha_k$ is a Dirichlet parameter for document to topic distribution. It makes sure that if a document never used a topic before, it still has a chance to introduce it.
- $\gamma_{w_{d,n}}$ is a Dirichlet parameter for topic to word distribution. It makes sure that if a word type never used a topic before, it still has a chance to introduce it.

```
def gibbs_iter():
    for index, row in df.iterrows(): #1

        for i, word in enumerate(row["content"]):#2
            t = assigned[index][i] #3
            word_topics[word][t] -= 1
            doc_topics[index][t] -= 1
            sum_topics[t] -= 1

            topic_scores = []
            for top in range(nr_topics):  #4
                n_dk = doc_topics[index][top]
                v_kwdn = word_topics[word][top]

                s1 = doc_len[index]
                s2 = sum_topics[top]
                topic_scores.append(((n_dk + alpha)/s1) * ((
                    v_kwdn + gamma)/s2))

            z_new = random.choices(topics, weights=topic_scores,
                k=1)[0] #5
            word_topics[word][z_new] += 1 #6
            doc_topics[index][z_new] += 1
            sum_topics[z_new] += 1
            assigned[index][i] = z_new
```

```
return 0
```

Code Listing 2: Gibbs sampling

The above code implements a single iteration of the gibbs algorithm

1. We run over each of the documents.
2. Per document, we run over all words in the document.
3. We remove the current word from the topic it was originally assigned to.
4. We calculate the probability of a word having a specific topic based on the formula above.
5. We randomly pick a topic to assign the word based on the probabilities.
6. Update the tables based on the selected topic

# 3 Results

We were pleased to find out that our model is correctly generated and that clear and useful topics are extracted. This makes it possible to take a look at the results and compare the different k-values with each other.

To build these results, we had to take a subset of the data-set and chose to limit it to 10000 documents in order to speed up the algorithm.

## 3.1 K-value

If we compare the different k-values, it is clear that 50 topics is to much for this model. A lot of topics are double or without any meaning. We can conclude that this is not a good model.

The different between 10 and 20 topics is not that significant. all the topics that are present with k equal to 10 can be found in the results for k equal to 20. We noticed that there are no topics occurring twice or extreme overlap between two topics for k equal to 20. This is because 10 topics might not have been enough for this data-set and thus we are missing details in the model.

### 3.2 Iterations

We also tried to figure out if the amount of iterations over the data-set would effect the result.

It is very clear that there is a minimum amount of iterations required to generate a sufficient model. But after a decent amount of iterations, the difference between each iteration will decrease. We did this experiment for a k-value of 10 and with both 50 and 100 iterations. The difference between both results is insignificant and thus not worth the extra time needed to perform these calculations.

This is why it would be a good idea to measure the improvement made between each iteration, and when this measurement converges to a value stop the algorithm.

### 3.3 Picking an optimal k-value

There are multiple options for determining the optimal number of topics (k) to perform Gibbs sampling. A to low k-value could render an LDA model that is too coarse to identify accurate classifiers. If there are a to large number of topics, the result may be to complex making the interpretation and subjective validation difficult [5].

The most easy way is by using an existing library that do the calculations for us. One downside with this method is that it is very computationally expensive because most of them try out every possible option and compares them. This iterative trail and error approach needs a verification method that indicates if a specific number of topics is indeed better than another. This is also a way to validate a topic model.

A usable option is the perplexity of a model [7]. This is a widely used metric for evaluation a language model. It captures how surprised a model is of new data is has never seen before. This will give a value that indicates how well the model represents the statistics of the held-out data. A lower perplexity score indicates a better generalization performance.

The perplexity can be calculated with

$$perplexity(D_{test} = exp\left\{-\frac{\sum_{d=1}^{M} log(p(w_d)}{\sum_{d=1}^{M} N_d}\right\}$$

Using the held-out likelihood might not be the best method. We try to determine the comprehensibility of a topic by inserting a random extra word

in this topic and allowing humans to figure out what the extra word is. This way we can see if a topic is well defined. But with a model, higher likelihood does not always mean higher interpretability. This is why we can not use interpretability as a metric to define the correctness of a model [6].

We also propose a different methodology for evaluating the correctness of a model and thus determining an optimal k-value. It is a variation on the distance between two documents, but now applied to the topics themselves. This value would rely on the amount of words two topics have in common and the importance of these words in the topics. If the distance between two topics is below a given threshold, we could merge those two since the meaning of these is close to one another. We did not work out this theory in detail but believe that it is a good metric to determine if the k-value is optimal. This seems to be a unknown method for evaluating models.

It is very important to understand that there is no one optimal way to evaluate a model. We need to take into account what needs to be measured an thus measure what you care about [6].

# 4 Assigning documents to a topic

In order to assign documents to a topic, we must give each document a score for how well it fits that topic. To calculate this score, we pass over each word in document and look how often this word occurs in this topic. To ensure a word isn't unrelated and divided among all topics, we divide this value by the total occurrence of the word. This calculation gives a value between 0 and 1, but since words that occur more in a topic are more important, we multiply by the occurrences of the word in that topic once more. We do this for all words in this document, and sum these calculations. To ensure longer documents do not have an unfair advantage, we normalize this value and get our score of a document for a specific topic. After we've calculated the score for all our documents, we sort them to get the highest scores.

```
for topic in range(nr_topics):
    print(topic)
    top_list = []
    for index, row in df.iterrows():
        top_doc_score = 0
        nr_words = len(row["content"])
        if nr_words == 0:
            continue
        for word in row["content"]:
```

```
            word_topic_score = word_in_topic_score(topic, word)
            word_all_topics_score = 0
            for topic2 in range(nr_topics):
                word_all_topics_score += word_in_topic_score(
                    topic2, word)

            word_topic_avg_score = (word_topic_score /
                word_all_topics_score) * word_topic_score

            top_doc_score += word_topic_avg_score

        top_doc_score /= nr_words
        top_list.append([index, top_doc_score])

    topic_top_lists.append(sorted(top_list, key= lambda x: x[1],
        reverse=True)[:100])

topic_top_lists
```

Code Listing 3: topic scoring

We could extrapolate this method to give scores to other documents that were not part of the data set used to build the topic model. This would be very useful if we know that the new documents have similar topics as the model.

Since scores of different topics of documents are somewhat independent of one another, it is possible for a document to be in more than one topics top 100 documents.

## 5 Conclusion

We can conclude that gibbs sampling is a realy good algorithm for extracting topics from a large data set. Our implementation wasn't optimal but we know what could be improved.

The main problem we have is the speed of our algorithm. This could be drastically improved by not using a Jupyter notebook, but a precompiled language such as java or C++. Another improvement would be by using an existing library because a lot of time was taken to make this implementation as performant as possible, something that we could not improve further in the amount of time given.

Something that might even improve our results is the optimization of the $\alpha$ and $\gamma$. These parameters are used to ensure that a topic has the chance to be introduced into a document or word. We did not do this since our results were already quite good.

## 6 Project information

We chose to make this implementation in a Jupyter notebook because this allowed us to evaluate the result after each step and tweak the implementation. This was very useful, especially because in earlier versions some steps took a lot of time to execute.

The algorithm was implemented ourselves without the use of external libraries. The only libraries used are

- nltk: for a list of stopwords
- regex: to compactly substitute unwanted characters
- pandas: as dataframe

The code and resulting csv can be found in the Github repository:
`https://github.com/stijnrosaer/topic-modeling-IR`

## References


[1] BOYD-GRABER, Jordan. *Topic Models: Introduction (13a)*. 2018. Available at: https://www.youtube.com/watch?v=fCmIceNqVog.

[2] DAVYDOVA, Olga. *Text Preprocessing in Python: Steps, Tools, and Examples*. 2018. Available at: https://medium.com/@datamonsters/text-preprocessing-in-python-steps-tools-and-examples-bf025f872908.

[3] GRIFFITHS, Tom. *Gibbs sampling in the generative model of latent dirichlet allocation*. 2002.

[4] BOYD-GRABER, Jordan. *Topic Models: Gibbs Sampling (13c)*. 2018. Available at: https://www.youtube.com/watch?v=u7l5hhmdc0M.

[5] ZHAO, Weizhong; ZOU, Wen; CHEN, James J. *Topic modeling for cluster analysis of large biological and medical datasets*. In: BMC bioinformatics. BioMed Central, 2014. p. S11.

[6] BOYD-GRABER, Jordan. *Topic Models Evaluation (13b)*. 2018. Available at: https://www.youtube.com/watch?v=rfHCronRgQU.

[7] BLEI, David M.; NG, Andrew Y.; JORDAN, Michael I. *Latent dirichlet allocation*. Journal of machine Learning research, 2003, 3.Jan: 993-1022.

# 7 APPENDIX

## 7.1 k = 20, 50 iterations

'one', 'want', 'art', 'briefing', 'last', 'see', 'day', 'first', 'look', 'posted'

'school', 'children', 'university', 'students', 'black', 'many', 'family', 'women', 'college', 'public'

'new', 'york', 'times', 'city', 'california', 'years', 'today', 'san', 'people', 'los'

'like', 'people', 'dont', 'think', 'going', 'know', 'im', 'one', 'get', 'time'

'news', 'media', 'twitter', 'breitbart', 'facebook', 'social', 'fox', 'follow', 'reported', 'people'

'united', 'states', 'president', 'american', 'security', 'china', 'russia', 'officials', 'russian', 'north'

'world', 'time', 'two', 'years', 'one', 'olympic', 'athletes', 'sports', 'last', 'first'

'bill', 'tax', 'change', 'administration', 'federal', 'plan', 'house', 'state', 'government', 'could'

'company', 'percent', 'million', 'companies', 'business', 'year', 'money', 'financial', 'workers', 'billion'

'like', 'home', 'one', 'water', 'food', 'day', 'around', 'small', 'town', 'street'

'police', 'man', 'officers', 'people', 'one', 'two', 'officer', 'death', 'told', 'killed'

'show', 'book', 'like', 'film', 'music', 'shows', 'series', 'first', 'best', 'also'

'health', 'dr', 'people', 'care', 'many', 'medical', 'study', 'found', 'percent', 'research'

'court', 'case', 'law', 'justice', 'federal', 'judge', 'department', 'state', 'legal', 'investigation'

'use', 'like', 'also', 'could', 'one', 'technology', 'system', 'new', 'used', 'service'

'team', 'game', 'first', 'two', 'last', 'players', 'one', 'season', 'back', 'three'

'trump', 'trumps', 'clinton', 'president', 'campaign', 'republican', 'house', 'white', 'election', 'republicans'

'one', 'years', 'like', 'could', 'time', 'first', 'even', 'might', 'also', 'two'

'state', 'military', 'government', 'islamic', 'united', 'war', 'american', 'syria', 'forces', 'attacks'

'political', 'european', 'country', 'party', 'union', 'people', 'may', 'europe', 'british', 'britain'

### 7.2 k = 10, 50 iterations

'trump', 'president', 'trumps', 'clinton', 'campaign', 'house', 'republican', 'obama', 'white', 'election'

'like', 'people', 'one', 'dr', 'dont', 'get', 'think', 'even', 'know', 'time'

'law', 'court', 'state', 'case', 'federal', 'justice', 'department', 'judge', 'states', 'students'

'united', 'state', 'american', 'states', 'military', 'government', 'security', 'islamic', 'officials', 'attack'

'company', 'percent', 'million', 'new', 'year', 'companies', 'business', 'money', 'also', 'years'

'one', 'first', 'team', 'show', 'game', 'last', 'two', 'best', 'play', 'time'

'like', 'one', 'new', 'family', 'life', 'time', 'women', 'years', 'first', 'work'

'police', 'city', 'people', 'one', 'new', 'two', 'officers', 'man', 'york', 'around'

'united', 'states', 'china', 'american', 'european', 'world', 'government', 'country', 'union', 'chinese'

'news', 'media', 'twitter', 'times', 'president', 'russian', 'new', 'intelligence', 'information', 'former'

### 7.3 k = 10, 100 iterations

'united', 'states', 'american', 'government', 'state', 'military', 'war', 'china', 'islamic', 'security'

'percent', 'million', 'company', 'new', 'year', 'companies', 'business', 'money', 'years', 'people'

'police', 'city', 'people', 'one', 'two', 'officers', 'man', 'family', 'home', 'black'

'news', 'media', 'twitter', 'times', 'officials', 'information', 'also', 'intelligence', 'new', 'russian'

'new', 'like', 'one', 'water', 'also', 'food', 'may', 'could', 'many', 'use'

'trump', 'president', 'trumps', 'clinton', 'campaign', 'house', 'republican', 'white', 'obama', 'election'

'like', 'people', 'one', 'dont', 'get', 'think', 'know', 'going', 'time', 'im'

'law', 'court', 'state', 'case', 'federal', 'states', 'justice', 'judge', 'university', 'public'

'dr', 'team', 'first', 'game', 'two', 'one', 'last', 'years', 'games', 'three'

'show', 'new', 'one', 'like', 'first', 'years', 'book', 'also', 'york', 'film'

### 7.4 k = 50, 50 iterations

'trump', 'trumps', 'president', 'clinton', 'campaign', 'house', 'republican', 'white', 'obama', 'election'

'people', 'new', 'times', 'could', 'year', 'two', 'one', 'public', 'according', 'last'

'new', 'one', 'back', 'also', 'last', 'two', 'year', 'work', 'made', 'still'

'think', 'dont', 'going', 'know', 'im', 'like', 'thats', 'people', 'get', 'really'

'two', 'one', 'first', 'like', 'many', 'years', 'time', 'could', 'way', 'new'

'first', 'may', 'even', 'people', 'years', 'many', 'last', 'new', 'one', 'also'

'also', 'new', 'even', 'last', 'years', 'people', 'made', 'many', 'part', 'york'

'police', 'officers', 'man', 'city', 'officer', 'black', 'one', 'killed', 'shot', 'shooting'

'new', 'one', 'many', 'also', 'last', 'still', 'made', 'including', 'even', 'time'

'media', 'also', 'former', 'new', 'first', 'public', 'may', 'called', 'last', 'people'

'one', 'also', 'could', 'first', 'state', 'people', 'new', 'last', 'called', 'wrote'

'one', 'year', 'time', 'like', 'new', 'even', 'work', 'day', 'people', 'two'

'people', 'last', 'new', 'one', 'may', 'even', 'part', 'also', 'time', 'take'

'new', 'make', 'first', 'people', 'also', 'one', 'two', 'back', 'year', 'last'

'also', 'two', 'new', 'like', 'way', 'make', 'made', 'last', 'time', 'know'

'school', 'children', 'family', 'students', 'life', 'mother', 'schools', 'college', 'university', 'parents'

'new', 'one', 'show', 'people', 'years', 'like', 'two', 'day', 'could', 'even'

'like', 'one', 'people', 'time', 'going', 'even', 'see', 'could', 'make', 'first'

'york', 'also', 'year', 'new', 'many', 'times', 'last', 'first', 'week', 'several'

'one', 'new', 'could', 'like', 'york', 'last', 'another', 'two', 'show', 'since'

'years', 'new', 'one', 'like', 'last', 'make', 'also', 'people', 'around', 'get'

'also', 'many', 'people', 'former', 'could', 'even', 'say', 'long', 'part', 'week'

'new', 'york', 'one', 'time', 'made', 'say', 'many', 'called', 'like', 'year'

'million', 'money', 'tax', 'financial', 'pay', 'bank', 'business', 'paid', 'year', 'income'

'law', 'court', 'federal', 'case', 'department', 'justice', 'state', 'states', 'judge', 'order'

'one', 'people', 'new', 'also', 'like', 'time', 'prince', 'since', 'long', 'king'

'people', 'new', 'first', 'may', 'one', 'also', 'many', 'say', 'members', 'two'

'united', 'american', 'state', 'states', 'officials', 'government', 'security', 'military', 'russia', 'russian'

'times', 'also', 'years', 'last', 'could', 'new', 'without', 'time', 'one', 'week'

'news', 'media', 'twitter', 'fox', 'network', 'times', 'statement', 'comey', 'breitbart', 'investigation'

'percent', 'china', 'states', 'united', 'european', 'economic', 'trade', 'union', 'chinese', 'workers'

'company', 'companies', 'business', 'executive', 'chief', 'billion', 'year', 'technology', 'industry', 'employees'

'water', 'also', 'new', 'years', 'still', 'one', 'time', 'part', 'like', 'day'

'one', 'like', 'years', 'time', 'also', 'much', 'world', 'new', 'first', 'people'

'north', 'south', 'states', 'korea', 'nuclear', 'united', 'park', 'national', 'korean', 'president'

'health', 'dr', 'care', 'people', 'medical', 'study', 'insurance', 'percent', 'drug', 'patients'

'new', 'people', 'last', 'one', 'also', 'two', 'take', 'week', 'could', 'year'

'like', 'time', 'one', 'way', 'even', 'show', 'much', 'day', 'get', 'go'

'like', 'new', 'also', 'one', 'years', 'many', 'could', 'may', 'year', 'two'

'new', 'people', 'may', 'first', 'last', 'many', 'used', 'two', 'york', 'like'

'people', 'new', 'last', 'time', 'one', 'two', 'even', 'like', 'year', 'country'

'one', 'could', 'people', 'according', 'years', 'time', 'new', 'world', 'first', 'also'

'team', 'women', 'game', 'games', 'season', 'players', 'first', 'play', 'sports', 'athletes'

'also', 'people', 'new', 'two', 'even', 'many', 'well', 'years', 'year', 'world'

'new', 'still', 'back', 'also', 'even', 'including', 'york', 'twitter', 'year', 'could'

'also', 'one', 'people', 'even', 'national', 'last', 'first', 'since', 'still', 'members'

'also', 'last', 'could', 'like', 'time', 'one', 'national', 'even', 'new', 'two'

'new', 'one', 'last', 'may', 'people', 'week', 'including', 'years', 'times', 'two'

'many', 'american', 'one', 'years', 'two', 'country', 'people', 'long', 'even', 'united'

'time', 'years', 'one', 'work', 'two', 'like', 'could', 'many', 'first', 'made'

### 7.5 Top documents per topic with 20 topics

The CSV file was too large to be displayed here but can be found in *topic_document_rank.csv*.