# Project - Latent Dirichlet Allocation

Till Späth 171085

16/1/2019

## Introduction

This project is about finding a set of topics in the Simple English Wikipedia [2] and about a simple recommender system based on the topics found in the articles. The project comprises three parts:
- implementation of a document generator to create documents based on the LDA model,
- implementation of a LDA inference algorithm based on Gibbs sampling,
- document exploration and recommendation.
The implementations are done in python 3.6. The detailed problem description is given in [4].

## The LDA model

The LDA model [1] describes how documents are created based on distributions for document length, topics and terms.

The documents consist of words which are positions for terms. In the following 'word' stands for a position in a document that can be filled with a 'term' from the vocabulary. The document length (number of words) is described as random number from a poisson distribution.

The vocabulary is a set of all terms that occur in the documents.

The topics $k$ are described as distributions $\varphi_k$ on the vocabulary drawn from a dirichlet distribution each.

Each document $m$ has a distribution on the topics $\vartheta_m$ also drawn from a dirichlet distribution.

The topics for the words in the documents are drawn from multinomial distributions according to the distribution $\vartheta_m$ on the topics for the document $m$. The terms for the words in the documents are drawn from multinomial distributions according to the distribution $\varphi_k$ for the topic $k$ that is assigned to the word. So every word is assigned to one topic and one term eventually

## Implementation of the generative LDA model

The generative model is used to produce a relatively small number of documents according to the LDA Model [1]. The generated documents are used to test the implemented Gibbs sampler for correctness, later. A version of the 'Lorem Ipsum' text is used as base for the vocabulary.

For every topic $k$ a random distribution $\varphi_k$ over the terms in the vocabulary is drawn from a dirichlet distribution with parameter $\beta$

For every document $m$ a random length (number of words in the document) is drawn from a poisson distribution. Then for the document a random distribution $\vartheta_m$ over the topics is drawn from a dirichlet distribution with parameter $\alpha$.

For every word in the document a topic is drawn randomly from a multinomial with distribution $\vartheta_m$. After a topic is assigned to the word, a term for the word is drawn randomly from a multinomial with distribution $\varphi_k$ according to the topic $k$ assigned to the word before.

Here is the short pseudocode for the document generator:

```
extract vocabulary from given text

for every topic k:
   draw ($\varphi_k$) from dirichlet($\beta$)

for every document m:
   draw length from poisson(..)
   draw $\vartheta_m$ from dirichlet($\alpha$)
   for every word:
      draw topic k from multinom.($\vartheta_m$)
      draw term from multinom.($\varphi_k$)

write vocabulary and documents to files
```

## The Gibbs sampler as inference algorithm for the LDA Model

Gibbs sampling is a general technique for sampling from a multivariate distribution [9]. It comprises the following steps [9]:

```
Choose some initial point $x^{(1)}$=($x_1^{(1)}$,...,$x_n^{(1)}$)
For j=2 to k:  (k is the num. of iterations)
   For i=1 to n:
      Set $x_i^{(j)}$=$x^{(i)}$ with probability
            p($x_i$|$x_1^{(j)}$,...,$x_{i-1}^{(j)}$,$x_{i+1}^{(j-1)}$,...,$x_n^{(j-1)}$)
```

The derivation and theory of the Gibbs sampler can be found in [1] and [5] in detail, it is not covered here completely. Only the equations that are represented in the implementation are described here.

Gibbs sampling is depending on conditional probabilities for each of the variables that should be sampled. In our case the variables to be sampled are the topics for the words in the documents. [1] gives the conditional probabilities for the topics as (slightly modified for readability):

$$p(z_i = k|\vec{z}_{\neg i}, \vec{w}\ ) \propto \frac{n^{(t)}_{k,\neg i} + \beta_t}{\Sigma_{\tau=1}^{V} (n^{(\tau)}_{k,\neg i} + \beta_\tau)} (n^{(k)}_{m,\neg i} + \alpha_k) \quad (1)$$

where:
$t$ is the term of word $i$ ($w_i$=$t$)
$z_i$ is the topic for word $i$
$\vec{z}_{\neg i}$ is the vec. of topics, excl. topic for word $i$
$\vec{w}$ is the vector of words
$n^{(t)}_{k,\neg i}$ is the number of times that topic $k$ occurs with term $t$, excl. topic for word $i$
$n^{(k)}_{m,\neg i}$ is the number of times that topic $k$ occurs in document $m$, excl. topic for word $i$
$V$ is the number of terms in the vocabulary
$\alpha$ is the parameter for the priori distribution of topics for documents
$\beta$ is the parameter for the priori distribution of terms for topics

The distributions of interest, $\vartheta$ and $\varphi$, can now be calculated from the counters (from [1], but simplified):

$$\vartheta_{m,k} = \frac{n^{(k)}_m + \alpha_k}{normalization} \quad (2)$$

$$\varphi_{k,t} = \frac{n^{(t)}_k + \beta_t}{normalization} \quad (3)$$

where:
$\vartheta_{m,k}$ is the distribution over topics of document $m$, entry for topic $k$
$\varphi_{k,t}$ is the distribution over terms of topic $k$, entry for term $t$

# Implementation of the Gibbs sampler

For efficiency reasons the documents are not represented as arrays of words but as arrays of indices of the corresponding terms in the vocabulary. A dictionary is set up for the correspondence of terms to indices, then for every document the corresponding array of indices is computed.

The Gibbs sampler is implemented according to [1] and as a function that is executed in parallel on the chosen number of processors. For the initialization step each instance of the function gets another seed for the random number generation. The parameters $\alpha$ and $\beta$ are set to uniform distribution vectors with according size.

The implementation uses counter variables, in every instance of the function, which are listed here:

| **Var.** | **Description** |
|---|---|
| *NMK* | Counter for the occurrence of topic $k$ in document $m$ |
| *NKT* | Counter for the occurrence of term $t$ as word with topic $k$ |
| *nk* | Counter for occurrence of topic $k$ |

In the initialization step for every word in every document a topic is chosen randomly from a multinomial distribution with parameter $\alpha$. The counters are incremented accordingly.

The Gibbs samplers function consist of an inner loop that is updating the topics assigned to the words in the documents. This inner loop is run several times before the resulting *NMK* and *NKT* are accumulated in the outer loop. This outer loop, containing the inner loop, is also run several times to get multiple samples.

In the inner loop each word in each document is processed as follows. The counters *NMK*, *NKT* and *nk* are decremented according to the term and topic assigned to the word before the update. The vector for the conditional distribution of the topics for the word is calculated based on $\alpha$, $\beta$, *NKT*, *NMK* and *nk*, see equation (1). The new topic for the word is drawn randomly from a multinomial distribution based on this vector. The counters *NMK*, *NKT* and *nk* are incremented according to the result of the draw. This inner loop is executed sufficiently often to get practically independent samples used in the outer loop.

In the outer loop *NMK* and *NKT* are accumulated to get the matrices $\theta_{sum}$ (for *NMK*) and $\phi_{sum}$ (for *NKT*).

After all the runs of the outer loop are finished the vectors $\alpha$ and $\beta$ are multiplied by the multiplicity of the runs of the outer loop and added to every row of $\theta_{sum}$ respectively $\phi_{sum}$. The resulting matrices $\theta_{sum+\alpha}$ and $\phi_{sum+\beta}$ are then normalized row-wise (l1-norm) to get the matrix $\theta_p$ for the $\vartheta_i$ and $\phi_p$ for the $\varphi_i$ as result of the instance p of the Gibbs sampler, equations (2) and (3).

The instances of the Gibbs samplers run independently with different seeds for the random number generator. So even if the instances find very similar distributions the columns of $\theta_p$ are probably not in the same order. To align the results of the instances the matrices $\theta_p$ have to be permutated. To do the permutation the similarity of the columns of $\theta_1$ from the first instance and $\theta_2$ of the second instance are computed pairwise. As measure the cosine similarity is used. The first permutation of columns of $\theta_2$ is done according to the highest similarity to a column of $\theta_1$. The second permutation is done according to the second highest similarity found and so on. $\theta_1$ and the permutation of

$\theta_2$ are then combined and the similarity of this combination to the result of the third instance is calculated and the permutation done. This is repeated for all the instances. The same has to be done for the $\phi_p$, row-wise.

The sum (after permuations) of all the $\theta_p$ respectively $\phi_p$ of the instances is normalized and written to files as finial result from the Gibbs sampler.

Here is the pseudo code for the Gibbs sampler implementation:

```
load docs.  and vocabulary from files
build up term-index dictionary
convert docs.  from strings to indices
K=40 ..  number of topics
P=36 ..  number of processors

func gibsSampling(seed,docs,vocab,..)
   start randomNumGen(seed)
   $\theta_{sum}$=zeroMatrix
   $\phi_{sum}$=zeroMatrix
   M=length(docs)
   V=length(vocab)
   $\alpha$=uniformVec(K)
   $\beta$=uniformVec(V)
   $\alpha$Count=0
   $\beta$Count=0
   $\beta$Sum=sum($\beta$)
   for m in M:
      for n in length(docs[m]):
         k=multinomial($\alpha$)
         topics[m,n]=k
         t=docs[m,n]
         NMK[m,k]+=1
         NKT[k,t]+=1
         nk[k]+=1
   for iOut outer cycles:
      for iIn inner cycles:
         for m in M:
            for n in length(docs[m]):
               k=topics[m,n]
               t=docs[m,n]
               NMK[m,k]-=1
               NKT[k,t]-=1
               nk[k]-=1
               p=(NKT[:,t]+bet[t])/
                  (nk+$\beta$Sum)*
                  (NMK[m,:]+$\alpha$)
               normalize(p)
               k=multinomial(p)
               topics[m,n]=k
               NMK[m,k]+=1
               NKT[k,t]+=1
               nk[k]+=1
      $\theta_{sum}$+=NMK
      $\phi_{sum}$+=NKT
   $\theta_{sum+\alpha}$+=iOut*matrix($\alpha$)
   $\theta_p$=normalize($\theta_{sum+\alpha}$)
   $\phi_{sum+\beta}$+=iOut*matrix($\beta$)
   $\phi_p$=normalize($\phi_{sum+\beta}$)
   return $\theta_p$, $\phi_p$

execute gibsSampling() P times parallel
               -> Thetas, Phis

finalTheta=Thetas[0]
for p=1 to P:
   for i in K:
      for ii in K:
         simMat=cosineSim
         (finalTheta[:,i],Thetas[p][:,ii])
   for i in K:
      maxI,maxJ=argmax(simMat)
      permutation[maxI]=maxJ
      simMat[maxI,:]=0
      simMat[:,maxJ]=0
   finalTheta=finalTheta+
               Tetas[p][:,permutation]

finalTheta=normalize(finalTheta)

..  permutation for Phis -> finalPhi

write finalTheta, finalPhi to files
```

## Application of the Gibbs sampler to the generated data

To check the correctness of the implementation, the Gibbs sampler is applied to the artificially generated documents according to the LDA model. The $\vartheta$ for the generator are not drawn randomly but set in a pattern so for each document mainly one topic is very dominant. In the result from the Gibbs sampler this pattern can clearly be found which is a good reason to assume the correctness of the implementation.

## Preprocessing of the wikipedia data

The data to be used with the Gibbs sampler eventually is taken from the 'Simple English Wikipedia' [2], the xml document is downloaded and parsed for the parts that are of interest. Only the introductory text of each article is used as a 'document'. First stop words are removed according to a manually defined list. Then the documents are preprocessed with gensim [3], special characters, multiple white spaces, numbers, further stop words and short words are removed, the remaining words are stemmed. Irregular expressions are also removed. From the result very frequent and very infrequent (based on the remaining corpus) words are removed. From the preprocessed documents very small ones are removed. The length of the documents is then cut after 20 words. From the preprocessed documents the vocabulary is extracted. The resulting documents, titles and the vocabulary is then saved to files for further use.

As result of the preprocessing we get 129407 documents with an average length of 19.34 words. The vocabulary consist of 8370 terms.

## Application of the Gibbs sampler to the Wikipedia data

The preprocessed data from the 'Simple English Wikipedia', documents and vocabulary, is loaded from files as input for the implemented Gibbs sampler. The number of topics to find is set to 40. The number of documents is very large (129407 documents), to have enough computing power to solve the problem in decent time, cloud computing is used (AWS EC2 from Amazon) with 36 virtual processors, so the number of processors is set to 36.

## Simple recommender system

A simple recommender system is implemented based on the similarity of the $\vartheta$ of the documents by cosine similarity. To find the $m$ documents most similar to document $i$, the $\vartheta_j$ of all documents (except document $i$) are compared to $\vartheta_i$. The similarities are sorted in descending order and the $m$ highest values are picked.

$$sim_{(i,j)} = \frac{< \vartheta_i, \vartheta_j >}{|\vartheta_i||\vartheta_j|} \tag{4}$$

## Results from the Gibbs sampling

The results from Gibbs sampling are the $\varphi_i$ for the distributions of terms for the topics, and the $\vartheta_i$ for the distribution of topics for the documents. Those distributions can be evaluated visually.

Few examples for the $\varphi_i$ are displayed here. The terms are sorted in descending order of weight in the topic:

topic 0: war militari, armi, battl, forc, gener, germani, german, air, lang, civil, conflict, attack, british, center, soldier, union, russian, march, empir, ...

topic 1: king, princ, hous, royal, queen, emperor, reign, england, monarch, henri, royalti, duke, charl, kingdom, roman, iii, father, mari, princess, ...

topic 2: greek, mean, ancient, word, god, mytholog, centuri, call, roman, english, book, lang, religion, christian, old, known, latin, come, dai, greec, ...

topic 10: univers, school, colleg, institut, public, student, educ, establish, size, field, scientist, high, studi, art, person, scienc, alma, build, professor, ...

Those topics should also make sense for the human reader.

The assignment of the documents to the topics can be evaluated indirectly by finding similar documents with regard to their topics. The similarity is determined by cosine similarity, as described above. Here are few examples of groups of similar documents, listed with their title in descending similarity:

Similar to 'Angela Merkel': Heinz Fischer, Anton Cermak, Robert Schuman, Roman Herzog, Friedrich Ebert, Rudolf Kirchschlger, Richard von Weizscker, Heinrich Lbke, Alexander Van der Bellen, Franz Vranitzky, Gnter Schabowski, Erich Honecker, Konrad Adenauer, Baldur von Schirach, Kurt Schuschnigg, Kurt Eisner, Anton Drexler, Paul von Hindenburg, ...

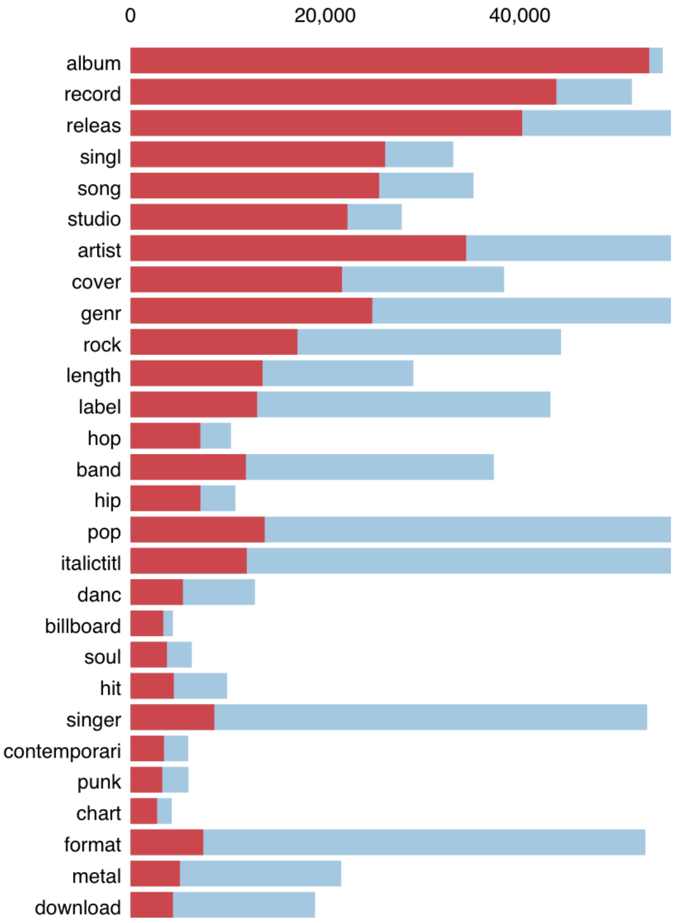
Similar to 'Banana': Passionfruit, Strawberry, Capsicum, Watermelon, Potato, Pomegranate, Podocarpaceae, Camellia, Parsley, Plum, Prune, Gooseberry, Grape, Muscadine, Sugarcane, Bean, Peyote, Avocado, Cyperaceae, ...

Similar to 'Science': Supersymmetry, Grand unification theory, Big Bang, Antineutron, Stereochemistry, Graduated cylinder, Fluid dynamics, Hydraulics, Universal indicator, Fermion, Quantum gravity, Mechanician, Damping, Nucleosynthesis, Shapley Supercluster, Krypton, Intergovernmental Panel on Climate Change, Critical density, Selenocysteine, ...

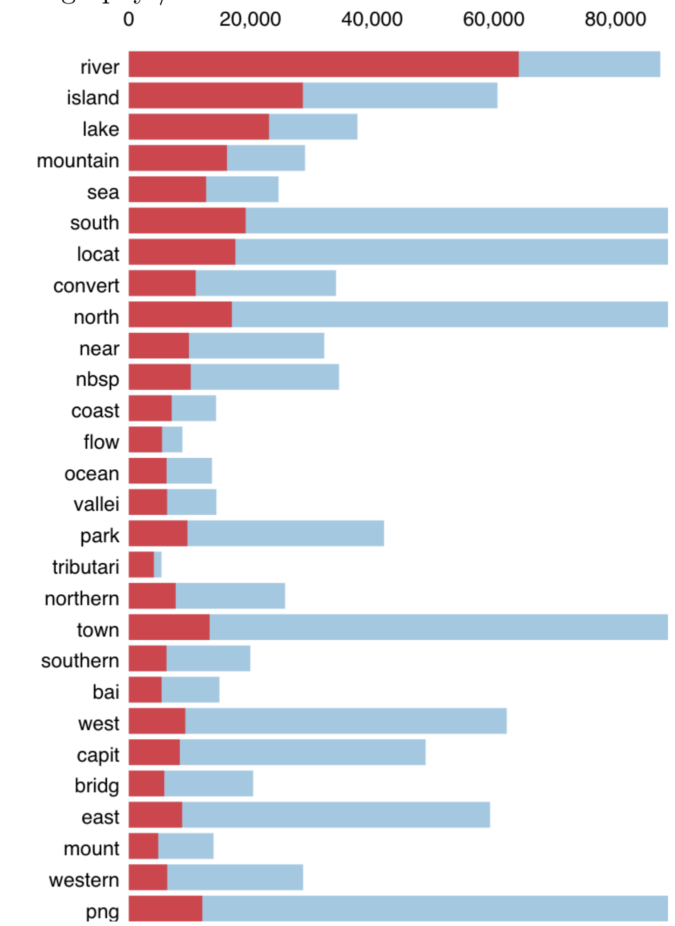
The similarity of those articles can be seen by a human reader.

The results from the Gibbs sampling can also be evaluated with the Python pyLDAvis module [8]. A html site containing the visualization created with pyLDAvis is delivered with this document.

'Music industry':



'Geography / location':



Here two example are shown for a topic that could be called 'Music industry' and a topic that could be called 'geography / location' or similar. The blue bars shows the overall term frequency while the red bards show the relevance of the 28 most important terms to the topic. $\lambda$ is set to 0.6 here. The relevance is calculated as:

$$\begin{aligned} &relevance = \\ &\lambda \cdot log(p(term|topic)) + \\ &(1-\lambda) \cdot log(p(term|topic)/p(term)) \quad [8] \end{aligned} \tag{5}$$

The longest blue bars are slightly cut so the diagrams fit into the document.

On the last page a bar chart is shown to visualize the accumulated occurrence of the topics in all the documents. The $\vartheta_j$ are summed and the sum is normalized. The topic with the highest accumulated weight is topic 7 (in order of descending weight):

topic 7: mean, word, number, person, call, differ, us, usual, wai, thing, exampl, law, like, group, mathemat, refer, inform, form, set, chang, ...

Probably this is what could be called a 'remainder topic'. Many of the terms present in that topic would qualify as stop words.

This does not hold for the topic with the second highest accumulated weight, topic 17.

topic 17: known, televis, actor, plai, actress, best, award, ndash, march, june, februari, juli, septemb, april, role, august, octob, novemb, seri, decemb, ...

Topic 17 makes sense, even so it looks strange. It looks like a combination of the topics 'television', and 'month'.

## Conclusion

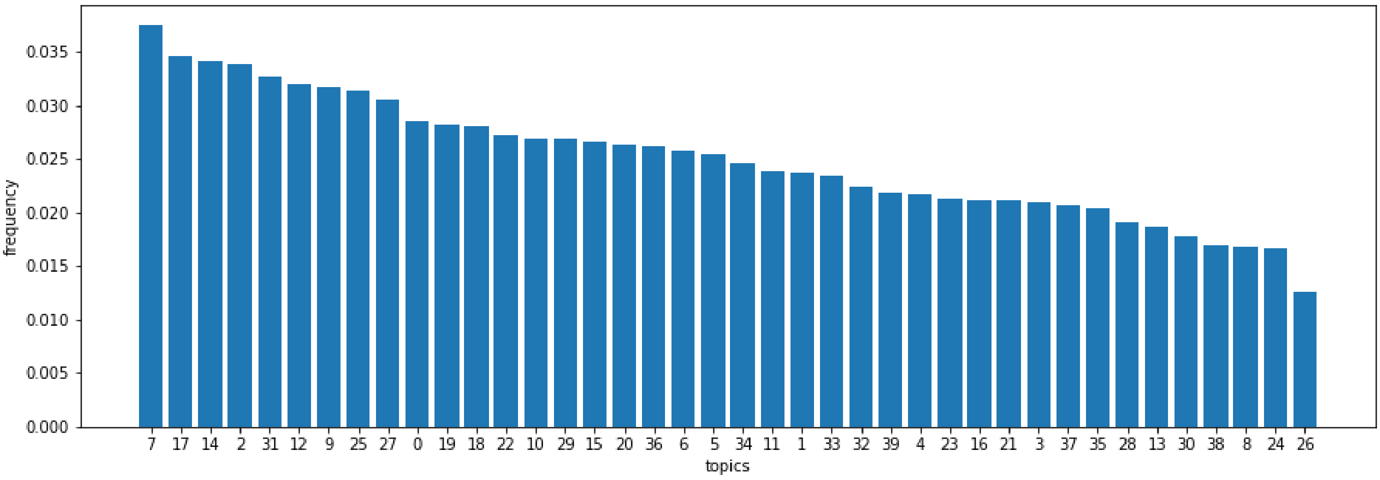
The LDA Algorithm delivers very good results with regards to the found topics for the documents, the found topic distributions look reasonable. Also the recommender system based on the similarity of the $\vartheta$ for the documents delivers good results with little effort.

The LDA Algorithm can be distributed on multiple processors very easily, communication between the instances of the Gibbs sampler function is not necessary.

But the computation power necessary to receive those results is very high, this has to be considered. If the goal is only to find groups of similar documents, a comparison of documents based on the cosine similarity of the word frequency vectors for example seems more efficient.

The found topics still contain some words that better should have been removed in the preprocessing of the raw documents. But since the computation is very expensive a second run with improved stop word list was not done.

For the solution to be of practical use it should be implemented in some compiled language, for example C++. Probably a much shorter running time could be achieved.



# References

[1] G. Heinrich, “Parameter estimation for text analysis’, tech. rep., 2009.
[2] ’Simple English Wikipedia - Archive’, https://dumps.wikimedia.org/simplewiki/ 20181120.
[3] ’Python gensim module’, https://radimrehurek.com/gensim/parsing/ preprocessing.html.
[4] M. Mitterreiter, ’Project - Latent Dirichlet Allocation Graphische Modelle (Lab)’, 2018.
[5] P. Resnik, E. Hardisty, ’Gibbs sampling for the uninitiated’, 2010
[6] ’Python XML module’, https://docs.python.org/3.7/library/xml.etree. elementtree.html.
[7] ’Python gensim module’, https://radimrehurek.com/gensim/parsing/ preprocessing.html.
[8] ’Python pyLDAvis module’, https://github.com/bmabey/pyLDAvis.
[9] J. Giesen, ’Probabilistic Modeling’, lecture script, 2018.