Project - Latent Dirichlet Allocation
Graphische Modelle (Lab)

Release: November 28, 2018 Submit: December 30, 2018.

In this project you will implement a document exploration system for the Simple English Wikipedia[1]: Given an article, we want to obtain a number of similar articles, that can be used for recommendation and exploration. For this you will implement the Latent-Dirichlet-Allocation inference algorithm, based on on Gibbs sampling as described in [2]. Finally you will have to write a project report where you present your implementation and results.

Notes:

- You can either chose Python or C++ as a programming language for your implementation.
- You are allowed and encouraged to work in teams.
- You must submit your code and your individually written project report. The oral exam will be based on your submissions.
- The baseline to pass the exam is a working implementation of the LDA inference algorithm.
- Start early with the implementation process!
- Ask, if there are problems.

1 Latent Dirichlet Allocation

Problem 1. The LDA - Generative model
Implement the LDA generative model as described in [2]. Thus, write a function that creates documents, given a number of topics. Additional information canbe found in the original LDA paper [3].

Problem 2. Implement the LDA inference algorithm, based on Gibbs sampling as described in [2]. Use your generative model to test the correctness of your implementation. First, focus on correctness but note that you will also need to optimize you code for performance for the later project.

2 Topic modeling with Wikipedia

The Simple English Wikipedia is an edition of the English Wikipedia written in basic English. The articles on the Simple English Wikipedia are, in general, shorter, use a limited vocabulary and grammatically less complex. Currently (November 2018) it contains a set of 140.794 articles. You will need to obtain a copy of the whole archive, preprocess the document and learn a model upon it.

Problem 3. Data gathering
Get the latest XML dump of the Simple English Wikipedia from here [4]. We are interested in the "All pages, current versions only" which excludes media files. Once you have downloaded the archive ($\sim$ 200.0 MB), unpack the XML file ($\sim 1.2GB$).
The XML file is structured in the following way:

```
<mediawiki>
  <siteinfo> ... </siteinfo>
```

```
  <page> page 1: content </page>
    ....
  <page> page M: content </page>
</mediawiki>
```

Each page corresponds to one Wikipedia article. E.g. the page for article "April" looks as follows:

```
<page>
  <title>April</title>
  <ns>0</ns>
  <id>1</id>
  <revision>
    <id>6286716</id>
    <parentid>6213898</parentid>
    <timestamp>2018-10-24T19:26:39Z</timestamp>
    <contributor>
      <username>CommonsDelinker</username>
      <id>5295</id>
    </contributor>
    <comment>...</comment>
    <model>wikitext</model>
    <format>text/x-wiki</format>
    <text>
        HERE IS THE RELEVANT PART FOR US!
    </text>
    <sha1>h0cu2m6igv5zi77ekvich7a0byi1zuv</sha1>
  </revision>
<page>
```

For us there are two relevant tags:

1. $< ns >$: The namespace tag [5]. We are only interested in articles in the namespace "0" which are the set of main articles.
2. $< text >$: Contains the article content. Note here that for simplicity we only want to process the "introductory" text for each article (see fig. 2).

Problem 4. Preprocessing
Extract the introductory text for each article. In Python you can use the XML module [6] to parse the document and extract the relevant text segments. Once obtained, you have to process the raw text segments that will make your documents:

- Tokenization: Split text documents into words, lowercase the words and remove punctuation.
- Lemmatization: Words in third person are changed to first person and verbs in past and future tenses are changed into present. Words are reduced to their root form.
- Remove stopwords, since they are not expressive in our model.
- Remove wikipedia specific expressions and tags. E.g "[[Some Text]]" should be transformed into "Some text". and "[[File:ColorfulSpringGarden.jpg—thumb—180px—right—[[Spring]]" should be transformed into "Spring".
- Remove numbers.
- ...

Your aim should be reduce the vocabulary size and ambiguity in your documents. You can also try to filter words that are rarely or frequently used. Most of the relevant functions are already implemented and you can make used of them. In Python you can checkout the gensim module [7]. It offers broad support for most preprocessing requirements.

Figure 1: The Simple English Wikipedia article for "April". We are interested in the introductory text (highlighted).

Problem 5. Training the model
Train a model with your implementation of the LDA inference algorithm. To start with, use a properly sized subset of your whole document corpus. Since the key problem with unsupervised learning techniques like LDA is the validation of your results you will need manual validation to make sense of the results. Additionally there are various result visualization tools like pyLDAvis [8] that deliver great insights into your training results (see 2).

Problem 6. Document exploration and recommendation
Given your model we want to able to answer queries of the form: Given article A, what are the k most similar articles $B_1, ..., B_k$. As an use case, you can think of article recommendation for users of the Wikipedia.
For this task you need a similarity measure, based on the document-topic distribution to compare documents. One possible measure is the Jensen–Shannon divergence. Once done, you can use this tool to ask and answer further questions ;)

3 Project report

Problem 7. Write a project report of approx. 5 - 10 content pages. Introduce the LDA model and write about your approach to solve the inference problems. Don't be overly verbose and stay formal whenever you can, but the report should be self-contained. Write about your implementation. Deliver performance results and give key insights. Write about your experiments on the Simple English Wikipedia dataset. Possible questions. Also reference ALL your resources.

Happy Coding!

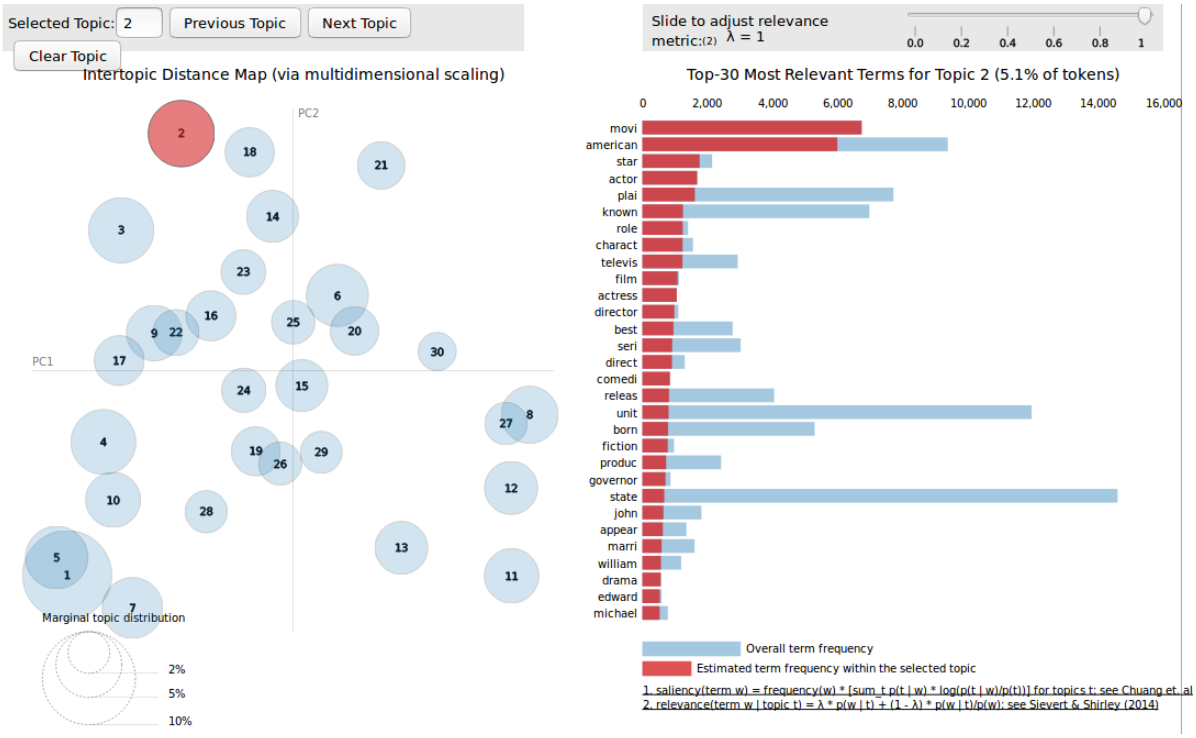

Figure 2: pyLDAvis in action.

References

[1] "Simple English Wikipedia." `https://simple.wikipedia.org/wiki/Main_Page`.

[2] G. Heinrich, "Parameter estimation for text analysis," tech. rep., 2004.

[3] D. M. Blei, A. Y. Ng, and M. I. Jordan, "Latent dirichlet allocation," *J. Mach. Learn. Res.*, vol. 3, pp. 993–1022, Mar. 2003.

[4] "Simple English Wikipedia - Archive." `https://dumps.wikimedia.org/simplewiki/20181120/`.

[5] "Wikipedia - Namespace." `https://en.wikipedia.org/wiki/Wikipedia:Namespace`.

[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/pyLDAvisl`.