User's guide for TopicMapping (version 1.0)

December 15, 2013

Thanks for downloading the code which implements TopicMapping. The algorithm finds topics in a set of documents using network clustering (Infomap) [1] and LDA likelihood optimization [2].

This guide will tell you how to compile, what are the input and output format, how to tune the algorithm's parameters, and some more.

Contents

1 Compiling

Open a a Unix (MAC) terminal and type

python compile.py

If you are using Windows, you could still run the program by installing MinGW (Minimalist GNU for Windows, `http://www.mingw.org/`).

2 Input and Output

2.1 Input

Let us start with an example:

./bin/topicmap -f quantum-and-granular-large-stemmed -t 10 -o test_results

The option **-f** is followed by the name of the file where the corpus is recorded. This file is supposed to contain a number of strings separated by newlines. Every string will be considered a different document.

In the example, *"quantum comput predict util . . . "* is the first document, *"develop theori interlay tunnel . . . "* is the second document and so on.

The reason why the documents look a little strange is that we used a stemming algorithm and we removed stop-words. If you want to do the same thing (we recommend it), you can use the following:

python Sources/NatLangProc/stem.py [original_file] [output_file]

which requires the python library called *stemming*.

The list of stop words is in "*Sources/NatLangProc/blacklist129.txt*".

IMPORTANT: Please make sure that your corpus file does not contain empty lines (or lines with just white spaces).

The other options in the example are explained in the next two sections.

2.2 Output

Option **-o** specifies the directory where all the output is redirected to, *test_results* in the example above.

Each document is associated with a probability distribution of topics and each topic is characterized by a distribution of words. These two distributions are written in two separate files:

1. **lda_gammas_final.txt** provides the probability of topics, for each document: $p(\text{topic}|\text{doc})$. Every line refers to a document, in the same order as they appear in the corpus file. Each number is the average usage of the corresponding topics.

   For instance, "0.025 0.01 58.8" means that topic 0 is used 0.025 times, topic 1 is used 0.01 times, and topic 2 is used 58.8 times, on average. To get the probabilities, just normalize this vector.

2. **lda_betas_sparse_final.txt** provides p(word|topic). Every line is a topic. The first number is the topic number. After that, pairs **(word-id, probability)** are sorted starting from the most probable. The word-ids can be mapped to the actual words from the file **word_wn_count.txt** (see below).

Similar files such as **lda_gammas_1.txt**, **lda_gammas_6.txt** etc., are printed every few iterations. Files **plsa_thetas.txt** and **plsa_betas_sparse.txt** also have the same content, obtained before running LDA optimization.

Other supporting files are:

1. **lda_summary_final.txt** gives overall information about the topics, such as their probability $p(t)$, the total number of words which have positive probability given this topic, and their top 100 words.

2. **word_wn_count.txt** contains strings in the format "word word-id occurrences".

3. **infomap.part** contains the (hard) partition of words found by Infomap, where words are represented with the word-id which can be found in **word_wn_count.txt**

4. **infomap-words.part** is the same file as before, written in words.

3 Algorithm options

The basic way to run TopicMapping is to specify just the corpus file and the out-directory.

However, TopicMapping has also a number of options to tune the size of the topics, the execution time and more. The following is an overview of most available options. Other options are available calling the program without arguments.

3.1 Topic size

The algorithm runs without supervision, in particular it does not require that you input a prefixed number of topics. However, two options are available to tune the granularity of the topics to some extent:

1. **-t** [threshold (integer)]. Minimum number of documents per topic. A few topics will likely be very small because of some isolated words. This option allows to get rid of very small topics, such as those used less than the threshold. The threshold is measured in number of documents: for instance **-t 10** means that each topic must be covered by at least 10 documents. **10 (default) or 100 is recommended for fairly large corpuses.** Documents which are entirely isolated from the others, cannot be assigned to any other topic and will still belong to their own topic.

2. **-p** [p-value (float)]. Higher values of the p-value will deliver fewer and more coarse-grained topics because the network of words is more connected. Default is 5%.

Examples:

./bin/topicmap -f quantum-and-granular-large-stemmed -p 0.1 -o test_results -t 10

This does not change the topics very much. But the next example will filter out small topics, so that only two will be left.

./bin/topicmap -f quantum-and-granular-large-stemmed -o test_results -t 100

3.2 Speed vs accuracy

There are two options to set the accuracy of the algorithm. Tuning them, you can get faster or more accurate results:

1. **-r** [number of runs (integer)]. How many times you want the network clustering algorithm to run. Default is 10.

2. **-step** [interval in PLSA local optimization (float)]. Default is 0.01. For example, 0.05 can be used to get faster results. Similarly, selecting **-minf** or **-maxf**, you can narrow the filter range and make the algorithm faster.

Example:

./bin/topicmap -f quantum-and-granular-large-stemmed -r 1 -step 0.05 -o test_results

3.3 Recycling previous runs

If you like to run the algorithm again with a different threshold, you can read the word partition saved in a previous run in the file called "*infomap.part*". This will skip the first part of the algorithm: building the network and running Infomap for the topics. The option is: **-part** [infomap.part (string)].

After running the algorithm without the option, try:

./bin/topicmap -f quantum-and-granular-large-stemmed -o test_results2 -t 100 -part test_results/infomap.part

This allows you to explore how the topics change filtering out more topics (**-t 100**), without running everything from scratch.

3.4 Random number generator

If you do not specify any seed, it will be read from the file **time_seed.dat**, which is updated at each run. If you like to input the seed, the option is **-seed** [integer].

Example:

./bin/topicmap -f quantum-and-granular-large-stemmed -o test_results -seed 101010

4 Subtopics

Sometimes, it is interesting to zoom-in in a topic to find its sub-topics. In order to do that, first we need to decide which topic we want to break further (for that, it is helpful to look at file **lda_summary_final.txt**). Let us say that we would like to zoom-in in topic 0. Running:

python Sources/py_utils/write_sub_corpus.py test_results/lda_word_assignments_final.txt 0,

we get a file called **sub_corpus.txt**, which only contains words which were more likely drawn from topic 0. We can now simply run TopicMapping on the sub-corpus (**./bin/topicmap -f sub_corpus.txt -o sub_results**). The file **doc_list.txt** reports the original ids of the documents which appear in the sub-corpus.

5 Parallelizing (beta)

We also provide a simple python script to parallelize the part of the program which builds the network. Try:

python Sources/py_utils/run_parallel_signet.py quantum-and-granular-large-stemmed

and follows the instructions to run TopicMapping after this.

There is also a script to parallelize the LDA optimization. To use it, you first need to run TopicMapping with option **-skip_lda**, which just provides the file **plsa_betas_sparse.txt**. After that, try:

python Sources/py_utils/run_parallel_lda.py,

where [**intial_model**] should be the path to the file **plsa_betas_sparse.txt**.

6 Acknowledgments

TopicMapping re-uses some of the code which implements the original Infomap [1]. The LDA likelihood optimization code closely follows the original code by David Blei [2]. Xiaohan Zeng curated the stemming algorithm. David Mertens compiled the corpus "*quantum-and-granular-large-stemmed*" pulling abstracts about "quantum computing" and "transitions in granular systems". All the rest was developed by Andrea Lancichinetti.

7 References

[1] M. Rosvall and C. T. Bergstrom, Proc. Natl. Acad. Sci. U.S.A **105**, 1118 (2008).

[2] D. Blei, A. Y. Ng, and M. I. Jordan, The Journal of Machine Learning Research 3 (2003): 993-1022.

If you found this program useful for your research, please cite:
[3] A high-reproducibility and high-accuracy method for automated topic classification. (to be published)