User's guide for TopicMapping (version 1.0)

July 10, 2013

Thanks for downloading the code which implements TopicMapping (Text Infomap).

The algorithm finds topics in a set of documents using network clustering (Infomap [1]) and a likelihood local optimization.

This guide will tell you (1) how to compile, (2) what are the input and output format, (3) how to tune the algorithm's parameter, and (4) some overall suggestions to get topics with the desired level of coarse-graining.

Contents

1 Compiling

Open a a Unix (MAC) terminal and type

make

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 can start with an example:

./topicmap -f quantum-and-granular-large-stemmed

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 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/stem.py [original_file] [output_file]

which requires the library called *stemming*. The list of stop words is in "*Sources/blacklist129.txt*".

IMPORTANT: Please make sure that your corpus file does not contain empty lines (or lines with just white spaces). They will be skipped by the algorithm, but it could be harder to match topics and documents afterwards.

2.2 Output

By default, the algorithm provides topics and subtopics. 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. **doc_topics.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 line is a string of pairs in the format:

   "topic : $p(\text{topic}|\text{doc})$".

   For instance, "10:**0.3** 22:**0.7**" means that topic 10 is used with probability 0.3 and topic 22 with probability 0.7.

2. **topic_words.txt** provides $p(\text{word}|\text{topic})$. Every line is a topic. Pairs word-probability are sorted starting from the most probable.

doc_topics2.txt and **topic_words2.txt** are the analogous for the subtopics.

There are also two summary files, **topic_summary.txt** and **subtopic_summary.txt** which give overall information about topics and subtopics respectively, such as their probability $p(t)$, and their top 20 words. In **subtopic_summary.txt** it is also possible to see the parent topic. *If these files are too long, you can try to run a simple and quick python script which should make the information more digestible (see Sec. 4).*

Other supporting files are:

1. **word_wn_count.txt** contains strings in the format "word word-id occurrences".
2. **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**
3. **infomap-words.part** is the same file as before, written in words.

3 Algorithm options

The basic way to run TopicMapping, is just to specify the corpus file, as we did before. The option for that is **-f** [filename (string)].

Example:

./topicmap -f quantum-and-granular-large-stemmed

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 all the available options.

If you want to fully learn how to use the algorithm, you can go though this section (two pages) and try the examples at the end of each subsection. **But if you are impatient, go to Sec. 4 and come back here when you want to know how the options work in detail.**

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. **-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%.
2. **-t** [threshold (integer)]. Minimum number of documents per topic (for subtopics the option is **-subt** [threshold (integer)].) 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 mush be covered by at least 10 documents.

Default is 0, but 10 or 100 is recommended for big corpuses. Documents which are entirely isolated from the others, cannot be assigned to any other topic and will still belong to their own topic.

Examples:

./topicmap -f quantum-and-granular-large-stemmed -p 0.1

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

./topicmap -f quantum-and-granular-large-stemmed -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 1.
2. **-conv** [threshold (float)]. If the relative gain of Infomap's objective function gets smaller than this threshold, the optimization is done. Default is 10^{-8}, but 10^{-5} is recommended for big datasets.

Example:

./topicmap -f quantum-and-granular-large-stemmed -r 1 -conv 1e-5

3.3 Overlap

In LDA, high values of the hyperparameter α, provide topic models where documents are assigned to many topics and topics are highly specialized on some words. On the contrary, small values of α will let documents use fewer topics, and topics have more overlap. The latter option is preferable.

Here, we have not implemented the LDA model (yet), but you can decide how much overlap you would like, if you choose a range for the filter parameter η. The filter parameter tries to find models such that, for each document and topic, $p(\text{topic}|\text{doc}) \geqslant \eta$. High values of the filter parameter are the equivalent of small values of α: the overlap is on the words and not on the topics. By default, TopicMapping will scan all possible values of filter parameters from 0 to 0.5, in steps of 0.01. You can reduce the range with options: **-minf** [lower-bound (float)], and **-maxf** [upper-bound (float)].

TopicMapping will figure out which value of the filter parameter is the best. If you like to save some time, you might want to manually set **-minf** 0.3 or similar.

Also, if TInfomap finds that the best value is zero or close, it is likely that your corpus have very short documents. In that case, I would probably increase **-minf** to 0.3 or 0.4.

In the following example we fix $\eta = 0.43$:

./topicmap -f quantum-and-granular-large-stemmed -minf 0.43 -maxf 0.43

3.4 Recycling previous runs

If you like to run the algorithm again with a different filtering range (options **-minf** and **-maxf**) or find the subtopics with different parameters ($p-$value for example), 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. Option is: **-part** [infomap.part (string)].

After running the algorithm without the option, try the example:

./topicmap -f quantum-and-granular-large-stemmed -minf 0.2 -maxf 0.2 -part infomap.part -p 0.3 -t 20

This allows you to explore how the topics change tuning the filtering range (**-minf 0.2 -maxf 0.2**), filtering out small topics (**-t 20**) or changing the $p-$value of the subtopics (**-p 0.3**) without running everything from scratch. Please not that the $p-$value for the topics will be the same as in the previous run (the part where that is used is skipped here).

3.5 Subtopics

TopicMapping will zoom in each topic and partition the documents which use that topic to provide subtopics.

If the subtopic structure is not strong though (for instance there are no significant subtopics), TopicMapping might split the documents in a lot of very small subtopics, whereas you might prefer to be told that no subtopics will be found.

The option **-subdoc** [threshold (integer)] allows you to choose how many documents should be in each subtopic (on average) to say the subtopics are significant. Default is 10. If you are already using option **-subt**, this threshold will be automatically set to be not small than that.

In this example we will find very small subtopics which actually cover less than 10 documents each, on average.

./topicmap -f quantum-and-granular-large-stemmed -subdocs 0

Instead, if you are **not** interested in finding subtopics at all, the option is **-nos**:

./topicmap -f quantum-and-granular-large-stemmed -nos

3.6 More output

Option **-fullout** will print files two additional files: *thetas.txt* and *betas.txt* (*thetas2.txt* and *betas2.txt* for subtopics.)

thetas.txt contains the same information as *doc_topics.txt*: each line is a document and is p(topic|doc) written as an array of length equal to the number of topics: the first number is the probability of using topic 1, the second number is for topic 2 and so on.

betas.txt contains the same information as *topic_words.txt*: Each line here refers to a topic. For each topic, the first number is the **log** of the probability of using the word whose word_id is 0, the second number is for word_id 1 and so on. (We recall that the word_ids

are the second argument of each line in file *word_wn_count.txt*). In writing this file, we used Laplace smoothing with parameter 10^{-4}.

Example:

./topicmap -f quantum-and-granular-large-stemmed -fullout

3.7 Random number generator

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

Example:

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

4 Hands-on suggestions

Here, we suggest an overall set of commands to supervise the algorithm and get results with the desired granularity and overlap.

For fairly large datasets (more than 1000 documents), we find useful the following set of options. We split the search for topics and subtopics.

4.1 Topics

1. **./topicmap -f quantum-and-granular-large-stemmed -minf 0.3 -conv 1e-5 -t 100 -nos -r 1 -p 0.05 -seed 1**

   This will provide only the topics (**-nos** means no subtopics), with the requirement that each topic is covered by at least 100 docs (**-t 100**). Options **-r 1 -p 0.05** are the default ones (1 run and 5% $p-$value) but we wrote them to make them explicit. **-conv 1e-5** is a reasonable convergence criterion (default is smaller though) and **-minf 0.3** requires that documents do not use too many topics. **-seed 1** sets the seed for the random number generator.

2. **mv infomap.part infomap-p005-t100.part**

   We can save the word partition in a file (Infomap's outcome), so that it does not get overwritten.

3. **python Sources/show_topic_info.py topic_summary.txt 100 5 quantum-and-granular-large-stemmed doc_topics.txt**

   This script allow us to quickly overview the results. Two files are generated: (1) *short_summary.txt* shows the topics which are covered by at least **100** docs, and the topics are named with their top **5** words. (2) *doc_topics_text.txt* shows the documents followed by p(topic|doc), written both with the topic ids and the topic names.

4.2 Subtopics

After this, we can run the algorithm for finding subtopics:

1. **./topicmap -f quantum-and-granular-large-stemmed -part infomap-p005-t100.part -minf 0.3 -conv 1e-5 -t 100 -subt 100 -subdocs 100 -r 1 -p 0.05 -seed 1**

   -part infomap-p005-t100.part is for reading the word partition we saved before, so that we do not need to find the topics again. **-subdocs 100** says that if the algorithm was not able to find subtopics which cover at least 100 documents, we prefer to be told that no subtopics were found. Please note that changing the $p-$value will only affect the subtopics and not the topics, because the network for topics was already computed beforehand. We keep the same **-t 100** as before, and we also set the same threshold for subtopics, **-subt 100**.

2. **python Sources/show_topic_info.py subtopic_summary.txt 100 10 quantum-and-granular-large-stemmed doc_topics2.txt**

   As before, we can check how the subtopics look like. Two files will be generated: *short_summary_sub.txt* and *doc_subtopics_text.txt*.

3. **./topicmap -f quantum-and-granular-large-stemmed -part infomap-p005-t100.part -minf 0.3 -conv 1e-5 -t 100 -subt 10 -subdocs 10 -r 1 -p 0.05 -seed 1**

   Here we reduce the minimal size and we eventually get subtopics as well.

5 Acknowledgments

TopicMapping is using some of the code which implements the original Infomap [1] and this code was written by Martin Rosvall. 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.

6 References

Infomap paper:

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

If you used this program for your research, please cite:

[2] A high-reproducibility and high-accuracy method for automated topic classification. (to be published)