Usage guide for TMM.jar, a software package that provides implementations of various topic detection methods

Giorgos Petkos and Symeon Papadopoulos
CERTH

July 16, 2013

1 Introduction

This is a usage guide for TMM.jar, a software package that contains Java implementations of the following topic detection methods that are presented in [1]:

- Document-pivot topic detection with LSH indexing.
- Graph-based feature-pivot topic detection with the use of the SCAN algorithm.
- Latent Dirichlter Allocation. In fact this is a wrapper around the mallet implementation.
- Soft frequent itemset mining.
- BNgram.

In order to use this library, please put the file TMM.jar to the classpath of your project. For each of the 5 algorithms there is a java package that contains (among other classes) a class named TopicDetector and this class has a public method named createTopics. For each of these topic detection methods, there is a parameter file, with a fixed name, which has to reside on the execution folder. In the following, it will be presented, how one may use these classes in order to perform topic detection.

Additionally, the jar can be used by itself in order to test the topic detection algorithms without the need to write any further code. In order to do this, one has to determine the parameters in a main parameter file, which is used in order to select which topic detection algorithm will be run, on which data and determines various output options. It should be noted that in the provided main function, the input data are json files which contain tweets retrieved using the Twitter API. However, it is possible to write code that works on data coming from different sources, more on this will be mentioned later.

2 Using the jar as a standalone application.

As mentioned, one may use the jar file to test the topic detection algorithms without running any additional code. In order to do this, one needs to fill in the parameters in the appropriate parameters files. In the main parameters file (main_parameters.properties), one may determine where the data on which the topic detection algorithm will be applied reside. In particular, one has to determine the parameters TWEETS_DIRECTORY and TWEETS_FILE. As mentioned, the data in the current implementation are taken from json files containing tweets in the json format, as returned from the Twitter API, but it is possible to work with different types of data as well. Subsequently, one has to determine the topic detection method that will be used. Only one of the 5 options has to be uncommented. In the following example, the BNgram approach has been selected.

```
#TOPIC_DETECTION_METHOD = LDA
TOPIC_DETECTION_METHOD = BNGRAM
#TOPIC_DETECTION_METHOD = DOC_PIVOT
#TOPIC_DETECTION_METHOD = GRAPH_BASED
#TOPIC_DETECTION_METHOD = SOFT_FIM
```

Finally, there are options that determine if the resulting topics will be displayed on the standard output and if they will be written to a text file. The results will be saved as a list of space separated keywords, with one line per detected topic. This is the format expected by the automated evaluator that has been provided together with the ground-truth topic files.

Additionally, in order to utilize the jar as a stand-alone application, one needs to fill in the parameters for the chosen topic detection method in tha appropriate file. More details on the parameters for each topic detection method can be found in the sections where the use of each topic detection algorithm is discussed.

3 Using the classes of the project in a separate project

3.1 Basic classes

Before proceeding with discussing the details of using the classes that implement each of the topic detection methods, it is necessary to present a few central classes of the package.

The first is the class Tweet. Despite the fact that the class is called Tweet, it can essentially be used in order to represent any document. In particular, the class Tweet has member variables for the text of the document, a unique identifier, the name of the uploader / author and a timestamp. It can be created either from a json string that should be in the form returned by the Twitter API, or it can be created using a constructor that takes as input values for each of the member variables. This allows the user to create documents which do not correspond to Tweets and therefore, this can be used to test the algorithms in different types of documents.

The second important class is the class Topic. This represents a topic as a set of keywords and also comes with fields that represent a score for the topic (that could be used for ranking), an id and a list of relative tweets. Each topic detection method returns a list of topics. Some algorithms may assign a score to the topic or also assign a set of relevant documents / tweets, but not necessarily all.

Having seen these classes, we may now proceed to examine how the code may be used

3.2 LDA

The code that implements Latent Dirichlet Allocation is essentially a wrapper around an existing implementation (mallet [1]). The class that provides this wrapper can be found in the package tmm.lda and is named TopicDetector. A very simple example of its use is the following:

```
import tmm.lda;
...
//We first load a set of documents, this can be done using
//a simple provided utility or some other user-defined procedure.
    List<Tweet> tweets=DataLoader.loadData(dataDirectory+tweetsFile);
//We then create the topic detector
TopicDetector topicDetectorLDA=new TopicDetector();
//And finally, we get a list of topics by calling createTopics.
List<Topic> topics=topicDetectorLDA.createTopics(tweets);
```

This is pretty straightforward. The parameters of the topic detection method are read from the file lda_parameters.properties. In this file, one has to determine the number of topics (which is not selected automatically), the number of training iterations and the number of keywords for each topic.

3.3 Document-pivot

This is an implementation of the document-pivot approach for topic detection. In particular, an incoming document / tweet is assigned to the same cluster as its most similar document, as long as their similarity is above some threshold. If it is not, a new cluster is created. In order to speed up the search for the most similar document, an implementation of Locality Sensitive Hashing (LSH is used).

[1] http://mallet.cs.umass.edu/

The code that implements this can be found in the package tmm.documentpivot. Again, there is a class named TopicDetector and one needs to call the function create topics. A code snippet for using the class is the following:

```
import tmm.documentpivot;
...
TopicDetector topicDetectorDocPivot=new TopicDetector();
topics=topicDetectorDocPivot.createTopics(tweets);
```

Where we do not show again, the costruction of the tweets list. The parameters for this method can be found in the file doc_pivot_parameters.properties. The only parameter that needs to be defined in there is the similarity threshold for assignment to a cluster.

3.4 Graph-based

This is an instance of a feature-pivot method, i.e. it groups together terms instead of documents. This approach in particular, organizes terms according to their cooccurrence patterns in a graph and applies a community detection algorithm (SCAN) on the graph, in order to come up with the resulting set of topics. The set of terms that will be used to construct the graph is selected using the ratio of likelihood of appearance in the provided corpus over the likelihood of appearance in a reference corpus.

The code that implements this can be found in the package tmm.graphbased. As for the previous methods, there is a class named TopicDetector and one needs to call the function create topics. A simple code snippet for utilizing the class is the following:

```
tmm.graphbased.TopicDetector topicDetectorGraph=new tmm.graphbased.TopicDetector();
topics=topicDetectorGraph.createTopics(tweets);
```

Importantly, there is a large number of parameters that need to be set and which determine how the terms will be selected and how the graph will be constructed. The parameters file is named graph_parameters.properties. More particularly:

- The parameter TERM_SELECTION_METHOD, determines the term selection method that will be used. This can be to select all terms with likelihood ration above some threshold (as defined by the parameter TERM_SELECTION_RATIO_THRESHOLD), a specific number of terms with the highest likelihood ratio (as defined by the parameter TERM_SELECTION_TOP_N) or a percentage of the terms with the highest likelihood ratio (TERM_SELECTION_TOP_PERCENTAGE).

- The parameter TERM_SIMILARITY_METHOD, which defines the similarity metric between terms that will be used. There are many options for this, the more important of which are absolute number of cooccurrences, Jaccard and cosine similarity.

- The parameter CORRELATION_SELECTION_TYPE, determines how the graph will be constructed. One may choose a percentage of the top possible links (GLOBAL_PERCENTAGE / CORRELATION_SELECTION_GLOBAL_RATIO), the N top possible links (GLOBAL_N / CORRELATION_SELECTION_GLOBAL_N), a fully connected graph (FULL), a simple threshold on the similarity between terms (THRESHOLD / CORRELATION_THRESHOLD) and even more. For more details please see the comments in the parameters file.

- Finally, there are parameters that are relevant to the clustering procedure. In particular SCAN_EPSILON and SCAN_MU are the main parameters of the SCAN algorithm and HUB_LINKING_THRESHOLD determines what is the threshold for assigning a hub to its best matching adjacent community (according to the chosen TERM_SIMILARITY_METHOD).

3.5 Soft frequent itemset mining

The soft frequent itemset mining method is another feature pivot method. It attempts to take into account cooccurrence patterns of degree larger than two (as compared to the graph-based approach). It's implementation can be found in the package tmm.sfim. A simple piece of code that uses the relevant class is the following.

```
import tmm.sfim;
...
TopicDetector topicDetectorSFIM=new TopicDetector();
topics=topicDetectorSFIM.createTopics(tweets);
```

Like the graph based approach, soft frequent itemset mining first selects the terms that will be processed. The selection mechanisms are the same as the ones for the graph based approach. The preferred term selection mechanism has to be defined in the parameter file sfim_parameters.properties. Moreover, the expansion of the topics depends on the shape of a sigmoid function (for more details please see the paper) and this is determined by the value of the parameters B_SIGMOID and C_SIGMOID.

3.6 BNgram

This is an another feature-pivot approach. It utilizes the $dfidf_t$ measure to quantify the burstiness of n-grams and subsequently clusters the most bursty n-grams. The implementation of the algorithm can be found in the package tmm.bngram. A small code snippet for using it is the following:

```
import tmm.bngram;
...
TopicDetector topicDetectorBNgram=new TopicDetector(dataDirectory,tweetsFile);
topics=topicDetectorBNgram.createTopics(tweets);
```

Please note that in order to use this method, one also has to explicitly define during the construction of the instance of the class the directory to which the json files are located. The reason is that the algorithm needs to compute burstiness scores for the terms that appear in the target corpus. The implementation assumes that the data for previous timeslots are available and placed in the same directory as the target corpus, therefore they can be computed. More particularly, in the parameters file (bngram_parameters.properties), one also needs to define the filenames which contain the tweets for the previous t timeslots. This is the parameter PREVIOUS_TIMESLOTS_FILES and needs to be a string of comma-separated filenames. For an example, please see the provided parameter file.

References

[1] Luca Maria Aiello, Georgios Petkos, Carlos Martin, David Corney, Symeon Papadopoulos, Ryan Skraba, Ayse Goker, Yiannis Kompatsiaris, and Alex Jaimes. Sensing trending topics in twitter. *IEEE Transactions on Multimedia*.