INF230 TP: Evaluation of Clustering Methods

Luis Galárraga (TA), Mauro Sozio, Oana Balalau (TA)
name.lastname@telecom-paristech.fr

October 23, 2014

In this TP session we are going to apply K-Means, K-Means++ and Agglomerative Clustering on a text corpus to cluster documents of the same topic. The goal is to evaluate the performance of the different methods for this particular scenario.

1 Software

1.1 Python and scikit-learn

We use Python 2.7.x and the library *scikit-learn* (`http://scikit-learn.org`) for this TP. *scikit-learn* provides implementations for the most common data mining and machine learning algorithms, including the clustering methods we are interested in; scikit-learn is already installed on the machines of the lab.

For those of you who are not familiar with Python we recommend `https://developers.google.com/edu/python/` which is used within Google by the software engineers who are beginners in Python. A more comprehensive tutorial can be found here `https://docs.python.org/2.7/`. Common editors such as kate, edit, vim, emacs provide syntax highlighting for Python. Overall it should be fairly easy to find the documentation for the most common modules in Python with Google. If you don't know Python we recommend you to use this lab to get familiar with it.

1.2 Support data & tools

We provide you with a package that contains :

1. a collection of more than 2500 documents each one being labeled with a topic as well as a Python program to parse the documents.
2. A Python program that first turns the documents into vectors into the euclidean space and then implements the clustering algorithms we are going to analyze.

1.3 Corpus

Our corpus consists of a dump of the Simple Wikipedia[1]. The articles have been cleaned up (no markup), labeled by topic and stored sequentially in a single file named `labeled_corpus` with

[1] http://simple.wikipedia.org

the following format: the first line contains the title, the second line the topic of the article (music, sports, etc.) and the next lines the article's text. A blank line is used as delimiter between articles.

We provide two supporting classes to parse the corpus: `Document` and `WikiCorpus`. They are implemented in the file `corpus.py`. See the main routine in the file for an example of how to use them.

1.4 Clustering

The program `docs_clustering.py` is a simple utility that takes the Wikipedia corpus as input and finds clusters of documents about the same topic. The program also calculates the Sum of Square Error (SSE) of the resulting clustering. It supports K-Means, K-Means++ and Agglomerative Clustering. Open a command line and from the location of the code, run

```
$ python docs_clustering.py [options] CORPUS-FILE
```

For a detailed description of the program's options, run `python docs_clustering.py -h`.

Our program converts each document in the corpus to a vector in the euclidean space using the class `sklearn.feature_extraction.text.TfidfVectorizer`. Each word in a document becomes a component of the vector associated to the corresponding document. The coefficients of the vector are the tf-idf scores of the words, which roughly speaking measure how frequent is a word (or term) in the document (tf, which stands for term frequency) and how frequent is that word across the whole corpus (idf, which stands for inverse document frequency). The main idea is that if a word is very frequent in a document but not frequent in the whole corpus then it gives much more information about the topic of the document (the tf-idf score will be larger). More information can be found here: `http://nlp.stanford.edu/IR-book/html/htmledition/term-frequency-and-weighting-1.html`.

The vectors for each document are then put together into a document-term matrix (each row is the vector representing the document). For big corpora with thousands of words, the dimension of the vector space may lead to scalability problems. However, we observe that most of the words will not appear in the same document which means our vectors contain many zero entries, providing an opportunity for compression. TfidfVectorizer leverages this fact and uses document-term sparse matrix representation. Notice also that words that are very frequent such as 'the', 'or', 'of' (so called stopwords) are not informative and should be removed. For more details about the code, see the function `vectorize` in `docs_clustering.py`.

Notice that one of the parameters of K-means is the maximum number of iterations (which in our code is 300). This means that K-Means will be iterated at most 300 times or less if the algorithm converges earlier (SSE does not change).

2 Tasks

Depending on your experience with Python we recommend you the following tasks. If you are a complete beginner with Python go through the tutorials and perhaps do some basic exercises recommended in the tutorials. Once you are more confident with Python you might want to implement your own function for purity, entropy, SSE. Then you can check if these are correct

by comparing it against our implementation. Then go to the following step. If you are familiar with Python we recommend you to experiment with the algorithms and their parameters so to improve SSE, entropy, purity. In particular we recommend you to experiment with:

1. different algorithms, K-means, K-means++, agglomerative clustering
2. their parameters, such as number of iterations, centroid selection, k (number of clusters), etc. You can also filter out most frequent words in the first phase (when the document-term matrix is built). In particular notice how SSE changes as k increases.
3. Our code shows also for each cluster the most frequent words in the cluster so that you should be able to infer whether there is a clear topic in the cluster and which one. Try to determine what each cluster of documents is about.
4. Our code also measures purity and entropy with respect to a "good" clustering of the documents. Try to understand and interpret those values. Remember that clustering is an *unsupervised* task so you should ignore the topics of the documents, entropy and purity when devising an effective clustering algorithm, as in real life you would have no information about the topics of the documents.

If you know Python very well you might try to implement your own heuristic. For example, you might notice that some clusters have a well defined topic while for others this is not the case. You can think to remove the good clusters and iterate on the remaining part of the documents. When clustering documents it is also important to select the most "informative" words. For all these issues you could also use any of the heuristics we discussed in class or try your own idea!