SD201: Evaluation of Clustering Algorithms

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

November 21, 2015

In this project, we are going to run K-Means, K-Means++ and Agglomerative Clustering on a text corpus in order to cluster documents according to their topic. The goal of this project is to evaluate the quality of the solutions computed by the different methods and write a report discussing the results.

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 project. *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.

1.2 Support data & tools

We provide you with a package that contains :

1. a collection of more than 1300 documents as well as a Python program to parse the documents.
2. A Python program that first turns the documents into vectors in the euclidean space and then implements the clustering algorithms we are going to evaluate.

1.3 Corpus

Our corpus consists of an excerpt from the Simple Wikipedia[1]. The documents have been cleaned up (no markup), and stored sequentially in a single file named `corpus` with the following

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

format: the first line contains the title and the next lines the article's text. A blank line is used as a delimiter between articles. A second file is also provided (`labeled_corpus`) where the topics of the documents are specified (in the line after the title of the document).

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 dealing with a same topic. It supports K-Means, K-Means++ and Agglomerative Clustering. Run the following (in the same directory of your code):

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

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

Our software 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

If you are a complete beginner with Python we recommend you to go through the tutorials and perhaps do some basic exercises recommended in the tutorials. Once you are more confident with Python go to the next step.

Then experiment with the algorithms and their parameters so as to improve SSE (sum of squared errors), entropy, and purity. In particular you should do the following and answer the

following questions:

1. evaluate k-means, k-means++ in terms of SSE on the `corpus` file. Plot the SSE as a function of k where k ranges between 3 and 15. Do you expect SSE to decrease or increase as k increases? Does SSE always decrease/increase? If not why this is the case?

2. which algorithm (between k-means and k-means++) performs best in terms of SSE?

3. The output of the algorithm shows for each cluster the 10 longest documents and the 10 most frequent terms in the documents of the cluster. You can use this information to infer the topic of the documents in the corresponding clusters. Try to label each cluster (i.e. assign a topic to each cluster). Which algorithm performs best between k-means, k-means++, agglomerative clustering when this additional information (top 10 most frequent terms) is taken into account?

4. We also provide you with a labeled corpus, where the documents are labeled with a topic (`labeled_corpus`). For each document, a topic is specified after the title of the document. Implement a function in Python for computing purity and entropy. Then, determine which of the three algorithms performs best in terms of entropy and purity, when the external information about the topics of the documents is taken into account. Report for each algorithm the value (or average values if you ran the algorithms multiple times) of entropy and purity. In this case, you can assume that the number of clusters is 3.

5. In order to determine which algorithm performs best, does a single execution of the algorithms suffice? Motivate your answer.

Remember that clustering is an *unsupervised* data mining task, therefore points 1-3 should be done while ignoring the topics of the documents and considering only the `corpus` file. This information should be taken into account only for 4, where you are supposed to use `labeled_corpus`.

Organize yourself in groups of at most 3 students. Write a report (max 3 pages) discussing your findings and including detailed answers to the questions above. Each group should submit one report and the code of the project. You should send us your report by **Friday December 4th at 5pm** by specifying "SD201: Clustering" in the subject of your email. The recipients of the email should be the professor of this class *and* the two teaching assistants. Failing of doing so might give you zero points for this project. All students submitting same reports (or very similar) will receive 0 points!