**Data cleaning and processing**

For the ATHENS course "Mining of massive data sets", we were asked to process and analyse previously extracted twitter data. This report serves its purpose by guiding the reader through the different steps that need to be taken to conquer this issue.

**Pre-processing**

The provided data consisted of thousands of tweets that were partitioned according to their date and/or location (New York, Oscars, Paris January and Paris February). The format of the files in which they resided was JSON, for which an external java library was readily available to assist us with the parsing.

Once extracted from a file, only the user ID, tweet ID, text and hashtags of a tweet were considered relevant for further analysis. The user ID and tweet ID were needed to extract relevant user and tweet data from the obtained results, whereas the text and hashtags were converted to keywords in the following way:

* text was split into words that were subsequently tagged by a trained, language specific tagger that resides in an external Stanford library for natural language processing (http://nlp.stanford.edu/software/tagger.shtml). Out of all these tagged words, only nouns, verbs and adjectives were used as keywords.
* hashtags were simply treated as keywords themselves.

It is not hard to see that some of those keywords can share certain semantic properties even though they all differ syntactically (and hence would be categorized as different keywords). A partial solution for this problem is to calculate the Levenshtein distance for these keywords and merge those that are close together. We decided to implement this partial solution by overriding the equals(Object) method in the respective keyword class with a call to the levenshteinDistance(String, String) method (http://en.wikibooks.org/wiki/Algorithm\_Implementation/Strings/Levenshtein\_distance) while ensuring a maximum keyword distance of 2.

At the end of the pre-processing phase, a graph was built. This graph's vertex set consisted of all distinct (according to the equals() method discussed above) keywords. Subsequently, an edge between two keywords was added to this graph if and only if these keywords appeared in the same tweet at least once. The weight of an edge was incremented every time an edge with the same vertices needed to be added while it was already present in the graph.