11-641 Machine Learning for Text Mining

Leonardo Neves
Andrew ID: lneves @andrew.cmu.edu

Homework 2

1) Corpus Exploration

The output for the corpus exploration part program was as follows:

==== Development Set ====
There are 942 documents, 254852 words and 14063 unique terms on the development set, averaging 174 unique terms per document
The word ids that occur twice on the first doc are 2, 5, 10, 18, 23, 27, 28, 30, 32, 42, 44, 45, 46, 50, 52, 60, 62, 69, 79, 87, 91, 99, 102, 107, 114, 141 and there are 161 unique terms on it

==== Test Set ====
There are 942 documents, 249516 words and 13924 unique terms on the test set, averaging 173 unique terms per document

2) Experiments

For the experiments, I first written the Kmeans algorithm and Kpp. Kpp made the algorithm to converge faster, going from around 20-30 iterations for Kmeans to 8-10 iterations for Kpp. After that, for my custom algorithm, I have used TF-IDF instead of only term frequency, testing it with both Kmeans and Kpp. To access cluster performance, together with the F1 measure, I have tried the silhouette metric first, with bad results, and the sum of the intracluster similarity. Inserting TF-IDF made the algorithm to converge much faster than Kmeans and a little faster than Kpp, with an average of 5-7 iterations, but got higher values for the cosine similarity.

Determining the parameters for K was not easy. I have first used a heuristic of computing K by the formula (V*D)/L, where V is the size of the vocabulary, D is the number of documents and L the number of non-zero terms in the corpus. This gave me the value of 80 for K. From this, I have tried using the silhouette metric, which did not give me interesting results, and then I have plotted the sum of the intra-cluster similarity, looking for the “elbow pattern” on the graph. One example is image 1 at the Appendix. From this graph, I could not find anything that would make it clear what the value should be. I have, then, plotted the log-variation between cluster numbers. One idea that came from this was to see where the last peak before variation got stable. This happens on the cluster 69. We decided on using this cluster, since it is not too far from the heuristic and the results look stable for this value, when comparing different methods.

K = 69

a. **Baseline Approach**
Intra-Cluster Similarity: 589.815985946
Mean F1: 0.539754434582
STD: 0.0281455543876
Max F1: 0.584579458415

b. **K++ approach**
Intra-Cluster Similarity: 594.755553083
Mean F1: 0.567469881969
STD: 0.0173684236221
Max F1: 0.59404105856

c. **Custom Algorithm**

As the custom algorithm, I have introduced the TF-IDF metric. My idea was that, since we would now highlight the relevance of each term for each document, it would improve the computation of similarity, making the algorithm to be more precise and converge faster. I have tried it on the two algorithms.

i. **Kmeans with TF-IDF**
Intra-Cluster Similarity: 499.99882841
Mean F1: 0.589758849979
STD: 0.0263435290717
Max F1: 0.630221409178

ii. **K++ with TF-IDF**
Intra-Cluster Similarity: 511.085687389
Mean F1: 0.594758492291
STD: 0.0232267758973
Max F1: 0.637822030997

3) **Analysis**

We can see, from the results of last section, that our best results came from the custom algorithm. Not only the convergence was faster, but also the F1 measure was the highest. On the other hand, the intra-cluster similarity was not as large as the KPP without TF-IDF. This might have happened because now the values for less important terms are lower, making the overall value to be smaller. The variance for the results were a little larger too, compared to KPP.

The pure Kmeans, using term frequency as metric, was the worst result. Not only it was much slower, frequently taking more than 20 iterations to converge, but also less accurate, as the F1 measures and similarity were lower. For the selected number of 69 clusters, Kmeans with TF-IDF had a performance similar to Kpp with TF-IDF, but the similarity was a little lower, the variance of the results, probably because of the randomness of the initialization, was a little higher, and it would take a little longer (2-4 iterations more, on average) to converge.

From this we conclude that, using TF-IDF made the algorithm more precise and faster. The initialization of KPP was important not only for how fast the algorithm converged, but also for the accuracy of our result.

4) The software implementation & data preprocessing

For the experiments, I have used Python as the programming language with the libraries Numpy and Scipy for handling sparse matrices and computing matrices operations, and sklearn for computing the cosine similarity. I have also adapted the eval.py script so that I can call the function to compute the F1 for the development set from my script. This way, it is easy to generate the files with the statistics about each experiment.

I have created the docs sparse matrix, where each row represents a document and each column represents a term. I have also created the centroids sparse matrix, where each row is a centroid and each column is a term. This made possible for my code to take advantage of Numpy's extremely fast matrix computation as all the operations could be vectorized. I had then a main function that would call the kmeans function. To isolate each experiment, I have created a function for running one experiment of kmeans, and the kmeans function would compute the average of results and create result files.

The function to initialize centroid would initialize it according to the method passed as parameters, being it kpp or kmeans.

For the data preprocessing, I have used the given document vectors and would get the information like document frequency, term frequency, vocabulary size, etc. all from the given files before starting computation.

My first idea was to create classes for clusters and documents, which did not take advantage of the matrix structure provided by Numpy. This first version was extremely slow compared to the final version. I think the design takes advantage of all the libraries I have used and was a good choice for this problem.

5) Source Code

As explained, the script eval.py was modified to be used as a function on my code. To run the code, you should pass first the method argument (kmeans or kpp), the set you are using (HW2_dev or HW2_test) and the metric (tf or tfidf). In addition, it is also possible to select start and end (exclusive) values of K for the experiments. An example of how to run the experiments for kpp and tfidf on the dev set only for cluster 69 would be *python kmeans.py kpp HW2_dev tfidf 69 70*

Be sure to have both sim and output folders on the same folder as the script, so it can store the results.

6) Results

The result file for the test set has been attached. The same idea of using the heuristic and the plot for deciding the number of clusters was used and the decided number was 78.

[1] Link for Heuristic

https://en.wikipedia.org/wiki/Determining_the_number_of_clusters_in_a_data_set

Appendix

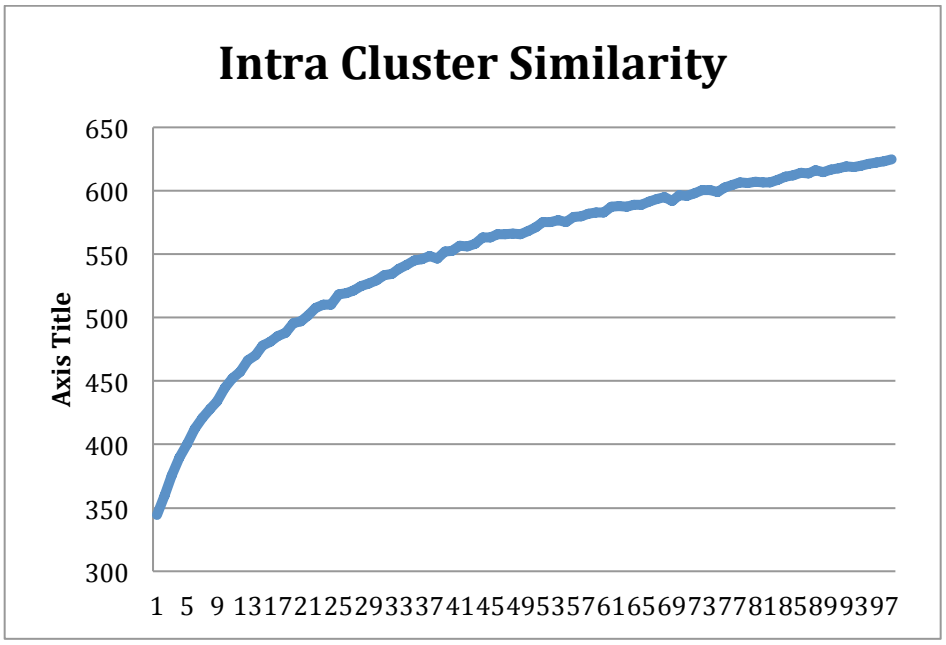

Image 1: Intra-cluster similarity

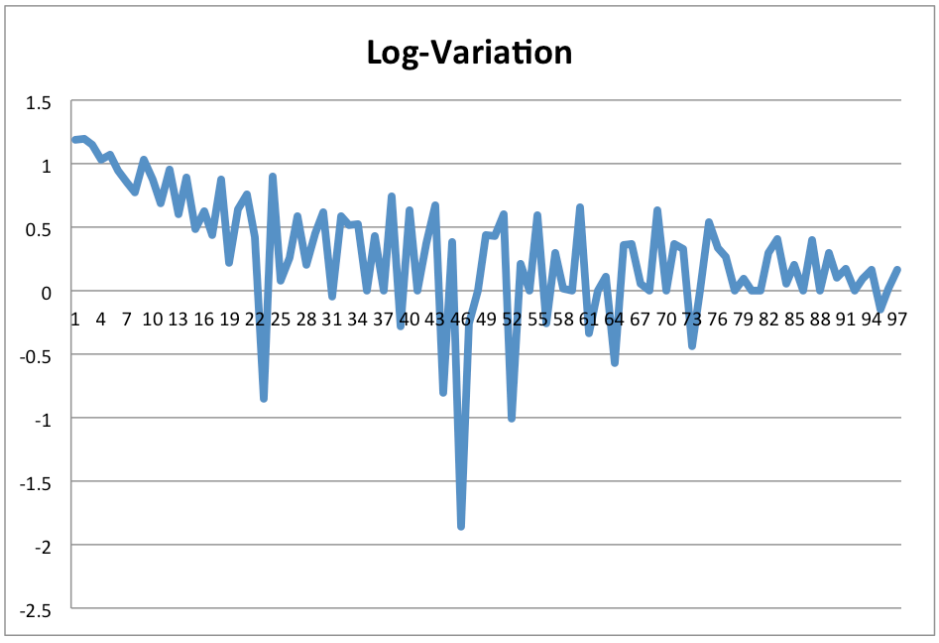

Image 2: Log-Variation of the similarity