1 Introduction

In this assignment, the goal was to parse a large set of Wikipedia articles, to extract features into a sparse feature matrix and to cluster them with a clustering algorithm of our choice. One motivation to perform automated clustering on unstructured, unlabeled data is to detect correlations between data points; for instance, in the case of Wikipedia, one might be able to automatically group articles into a small set of larger categories such as “Politics”, “People”, etc.

Although Wikipedia does provide category labels, labels are not centrally controlled or restricted (any user can create a category), which results in a very large amount of category labels. Also, there is no documentation page that shows a list of labels, and some online resources point out that Wikipedia’s current set of category labels by no means form a tree, but rather a directed acyclical graph.

2 Methodology

We chose a slightly different route than the other groups: Instead of using Hadoop on the iCluster, we used Spark. Since the Spark framework is closely integrated with Scala, this allowed us to write our code in Scala, thus simplifying it and allowing for interactive data exploration. Furthermore, Spark is generally faster than Hadoop.

The Wikipedia articles were first read from the file, and the Mahout library was used to split the XML file and to extract every article with the help of the <page> and </page> tags. The articles were then passed on to the Scala XML parser that extracted the actual text body. The text body is formatted using the Wikipedia markup language and contains a lot of semantic information. Using some regular expressions, the markup elements were removed from the text. After that, the articles were passed through a first filter in which all articles with less than 30 words were eliminated, in order to remove stubs and reference pages. The text was then passed to a Porter word stemmer [1] from the ScalaNLP project, and the unique tokens were mapped to an index.

To reduce the computational burden, we used the following procedure: First all the tokens (i.e. stemmed words) in an article were identified and entered
into a bag of words with the respective count for that article. They were then sorted by occurrence in every article and summed over articles. The index on the words was constructed according to this order. Then, rare words were eliminated by removing all tokens with a count of less than 5; we observed that this group contained mostly tokens like URLs embedded in the text that were not useful in categorizing the text. Of the original 10.6 million unique tokens, this yielded 1.1 million unique tokens that occurred more than 5 times in the whole corpus, which significantly reduced computational complexity. These tokens were then locally sorted and indexed by decreasing order of occurrence. Using this mapping from words to the index, each article was transformed into a sparse vector following a bag-of-word model.

Using these vectors as input, we ran a standard K-means clustering algorithm on the data. Given that the number of categories listed in the wikipedia articles was very large, we needed to manually set a number of clusters smaller than that, which we chose to be 100. The K-means procedure yields a partitioning of the set of articles, while Wikipedia classifies all its articles using an elaborate directed acyclic graph, so the output is not directly comparable. Spark is an in-memory cluster (although recent versions can offload the data on disk). Live memory is an expensive resource so we processed 1.1 million articles due to budget constraints, which still represents a little more than 10% of all articles in Wikipedia. After pre-processing, we tokenized and clustered 635985 articles. About 518984 articles were discarded because they were too short.

3 Clustering results

We found that the k-means algorithm performed well in the first few iterations, but converged very slowly. Figure 3 shows a plot of the entropy of the clustering vs. number of iterations. The entropy of the clustering is the entropy of the distribution of the articles if one were to classify the article by randomly assigning them to each cluster, based on the number of articles in the cluster. As such, it is a good convergence indicator of the K-Means algorithm. One can see that the algorithm makes the maximum progress during the first 6 iterations, after which convergence toward the maximum value of ln(100) slows down markedly. This can be attributed to the very high-dimensional space in which we are operating (>1 million).

The results of the clustering were manually inspected. For each center of a cluster, most of the weight was concentrated on the first 20 tokens of each center (which is a high-dimensional vector). From a practical perspective, this could lead to some important computation savings.

The majority of the 100 clusters were topical and contained relevant keywords. We also noted that, as expected, they were representative of typical trends on Wikipedia: Few tokens were in foreign languages; articles from the United States, the United Kingdom, France, Germany dominated. In particular, each of these nations was clearly associated to a cluster. Regarding topics, multiple distinct clusters were related to sports and the Olympic games, people...
and religion, with a smaller number on science (Astronomy in particular) and politics.

3.1 Computational issues

This code uses all the advanced functionalities provided by Spark: map, reduce, reduceByKey, groupByKey. We encountered some stumbling blocks in unexpected areas, both due to the relative youth of spark and due to some issues with the underlying libraries. In order to diagnose performance issues, we used a tool released by Twitter called Ostrich, which lets the developer gather statistics (like counters and gauges) from the running code and stream them to a
Memory management in Spark is still an issue and its use varied wildly from 36GB (aggregated) to 130GB during the construction of the sorted list. Since there is no way to change dynamically the memory allocated to the cluster, we were forced to allocate a conservative amount (200GB).

Our Wikipedia parser, which uses regular expressions to find some link patterns in the text, managed to crash the regular expression analyzer in the Java library, in some corner cases. This happened only in a few articles which were discarded. We suspect that while the Scala API for regular expressions looks thread-safe, the underlying implementation in the JVM may have some shared mutable states.

To represent the sparse vector, we first used the SparseVector class provided by the Maillet library, a library written in Java to perform NLP and statistical analysis. This class was one of the few classes written in Java that had an efficient dot product implementation. However, computing the new centers of each cluster required the addition of sparse vectors and we found this default implementation to be very inefficient: this computation was happening at a rate of less that 650 summations per second on 20 cores. An inspection of the sources revealed a very inefficient implementation. Unrolling our own implementation of sparse vectors raised the speed of this step to more than 70000 summations per second.

References