Universitat Politècnica de Catalunya

Facultat d'Informàtica de Barcelona

Master in Data Science

MapReduce and Document clustering

Information Retrieval and Recommender Systems

Fourth laboratory report

Daniel Esquina and Mohana Fathollani

Professor: Ramon Ferrer

December 8, 2022

Contents

List of Figures

List of Tables

1 Introduction

In this laboratory project we are asked to fill an incomplete implementation of the k-means algorithm with a map-reduce framework. After implementing and testing this program, an experimentation task will be performed to better understand the effects of both the map-reduce framework and document clustering.

2 k-means Implementation

The idea behind the K-means algorithm is to set some points as the representatives of each cluster and then act as if they were the centroid (also known as prototypes in this project) of each clusters. A distance measure, in this case the Jaccard similarity will be used to assign the data points to the closer centroid and this centroid will be recomputed as the mean value of all the points from that particular cluster. This iterative process is finished when the number of iterations reaches a maximum or it converges.

From the Map-Reduce point of view, the implementation is straight-forward. The `assign_prototype()` function acts as the mapper and the `aggregate_prototype()` function as the reducer. The second part of the framework, the reducer, receives the lists of all documents for a prototype and computes the new prototype. In the aggregate_prototype we should consider efficient structure to compute the frequency of each word. We compared collections.counter() with dictionary() but we did not find considerable difference between them, in some cases collections.counter() was 3 seconds faster than dictionary, therefore we prefer to use dictionary.
In contrast, the aim of the first part, the mapper, is to return the closest prototype to a document and it receives the document and the list of all prototypes. To obtain it we will build two vectors, one for the documents and another one for the prototype, which contain the presence of a token and will allow us to compute the Jaccard similarity.

For this laboratory project, we needed to implement a function that computed the Jaccard similarities between the documents and the centroids. We tried to make this implementation more efficient by scanning both vectors (document and prototypes) only once in the worst case. In addition, it will stop scanning if the sufficient conditions are met.

The `MRKmeans.py` script was also changed to properly save the results and use the appropriate data structures. Notice that we had to include an addition argument in the `MRKmeansStep()` call which is `-no-bootstrap-mrjob` in order for our windows system to execute the code. It seems to work perfectly without it in other operating systems but the MrJob library crashes when reading from the temporary files created by itself in some systems. This argument however, fixed our problem.

3 Experiments

On this section we will follow the guidelines prompted by the statement in order to fully comprehend and learn about document clustering and the map-reduce framework. The first step for this experimentation task was creating the Elasticsearch index with the `IndexFiles.py` script. This index uses a letter tokenizer and some filters. The next step will be to generate a couple of datasets carefully deciding the min and max frequencies to use in the `ExtractData.py` file so this will become our first experiment. After that, we will investigate on different values of k for those dataset. Once we have found an appropriate value, some research on the execution time depending on the number of nodes will be done. Finally, we will check the results obtained after all these decisions and try to analyze them from a clustering perspective. For this section we have used additional scripts like `experiment_results.py` and `Similarity_Measure.ipynb`, in addition to some non-intrusive changes to the previous files like `ExtractData.py`.

3.1 Min and Max frequency

We considered different values for minimum and maximum frequencies. On a first instance we tried using ranges of 25% between 0 and 1. However, this idea was quickly discarded as the range between 0 and 25% contained most of the instances. This yielded the vocabulary sizes that can be seen in table 1.

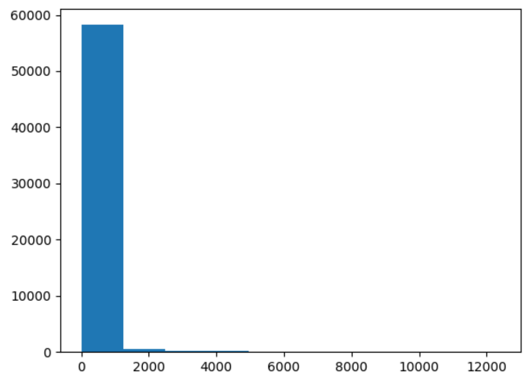
Table 1: Vocabulary sizes for different chosen frequencies.

	0 - 0.25	**0.25 - 0.5**	**0.5-0.75**	**0.75 - 1**
Vocabulary size	59287	55	8	1

From looking at this table it is clear that these ranges will not be useful to determine the clusters, especially if we take a look at the tokens from the range 0.5-0.75. These tokens can be seen in table 2 and they don't seem to provide any meaning that would separate two documents into different clusters. In addition, we know there is a power law affecting this measure from the first laboratory project of the subject and once again we can see it on figures 1.

Table 2: Example vocabulary from 0.5 to 0.75

Token	can	from	model	our	result	show	use	which
Frequency	30399	36470	28192	27671	29030	25304	35396	35403


(a) Plot in normal scale.

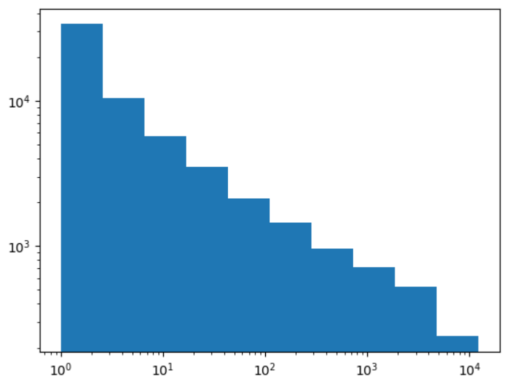

(b) Plot in log scale.

Figure 1: Visual representation of the Zipf's law (frequency vs rank).

We know that the majority of tokens have a very low frequency and we want to achieve a trade-off between two cases. On the one hand we have the case of choosing very frequent words. In that case, all the documents will be represented by the same words and one single cluster could entail them all. On the other hand we could choose low frequency words that would be very different for each document which does not help us to divide them into meaningful clusters.

As the words over 50% did not seem very representative of different clusters, we will continue investigating until we find a low frequency upper bound that satisfies our needs. Another metric that we took into account was the average words per document. In table 3.

Table 3: Vocabulary size and average word per document by frequency range.

	0.025 - 0.05	**0.05 - 0.1**	**0.1 - 0.15**	**0.15 - 0.20**	**0.20 - 0.25**
Average words	15.1130	21.3868	13.1493	11.3394	7.1378
Vocabulary size	472	343	123	74	36

Notice that the first interval is twice smaller than the other ones, therefore the average is not on the same scale. However, upon further investigation we have decided to use the minimum frequency of 0.05 and the maximum frequency of 0.15. The reasoning behind this decision being a compromise between two factors. The first one is that we want to exclude all common words which are in most documents not to bias the clustering method and we have found 0.15 to be a valid option for that matter. The second factor is that we also need to filter the low frequency words which add noise and may not be of use to create meaningful clusterings. A relatively low bound we have found for that matter is the 0.05 value.

We will generate two datasets, one with the a limited vocabulary size of 100 and the other with 250. This will allow us to do some comparisons on the following experiments.

3.2 Analyze number of clusters

As a first guess, we used 8 as a number of clusters. These clusters should be specific and documents that assigned to each of them should belong to same topic. Moreover, we used 10 and 15 clusters to see which one of them yielded meaningful results.

To compare the performance of each cluster, we used Jacard distance to find the similarity between documents assigned to a class with the prototype of that class. Codes for this part provided in `Similarity_measure.ipynb`.

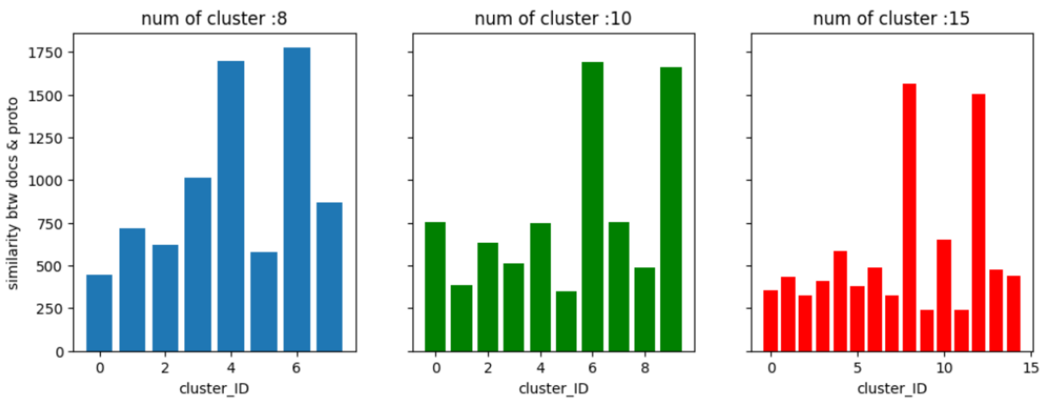


Figure 2: Similarity between documents in each class and prototype of that class

Based on figure 2, belonging to the execution of the 250 dataset and fmin = 0.05 and fmax = 0.15, in all options we have 2 clusters that have the highest similarity between words in documents and prototypes compared to other clusters. For other classes we have more or less same similarity.

When we are using 15 clusters similarity in most of classes decreased compared to 10 and 8 clusters. It means that we do not need to put documents in 15 classes and probably most of classes will have very related topics.

Although we have higher mean similarities in 8 clusters compare to 10 clusters, we could cover more topics in 10 clusters with negligible difference in mean similarities. If we had more data, probably we could reach a higher mean similarity for 10 clusters.

We did the same procedure for 100 words and we saw same trend while changing the number of classes.

3.3 Number of parallel processes

At this point we have experimented about the word frequencies to use and its effects on vocabulary size and representation of documents. We have also investigated about the number of cluster to choose for our particular data. The next thing we are going to analyze is the how changing the number of cores, thus changing the number of mappers/reducers affects iteration time. We will use both generated datasets for this purpose in order to compare them and check the differences.

We have executed the k-means algorithm with this framework setting the `-ncores` argument with values from 1 to 8 cores. In figure 3 we can see the average iteration time of each execution (not including the first iteration).

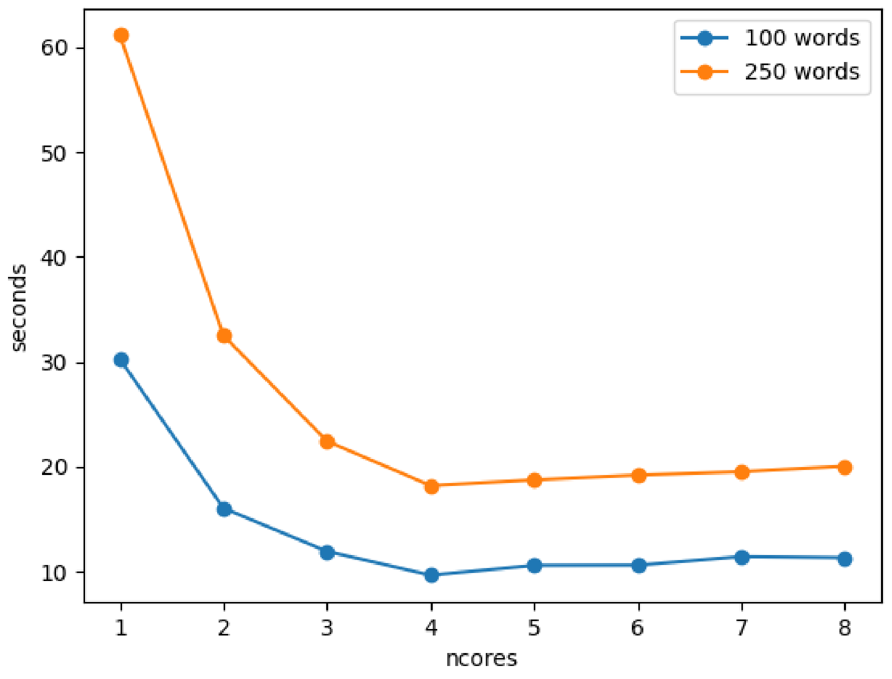


Figure 3: Average iteration time vs number of cores.

Comparing the two datasets of different sizes we can draw some interesting conclusions. The size of the dataset does not seem to have a significant effect on the amount of improvement of this map-reduce technique. It seems reasonable to think that using map-reduce for big amounts of data would equally improve performance, so it looks like a very powerful tool. Looking at the plot we can see how in both cases the best performance is found when the number of cores is 4. We will keep knowledge for future experiments. It is not weird to see that the amount of time starts to increase again after this point, as the parallelization overhead also increases with the number of threads. The higher the number of parallel processes, the higher the time the merging step takes. The tendency of this plot, however, will change for different machines and will yield better or worse performance depending of the number of cores it has.

We have also noticed that the time for the first iteration is lower that the rest and we have not included it for this analysis, as stated in the guidelines. We believe this is the case because of the difference in prototypes used for the computation of the first iteration versus the following.

4 Final clustering approach

On table 4 we can take a look at the different words ranked by frequency of appearance in each cluster. This will be the result of the selected frequencies for the dataset with a vocabulary size of 250 words. Using four parallel processes and 25 iterations of the algorithm we got those classes.

Table 4: Top frequent words of each cluster for the execution with parameters: minfreq = 0.05, maxfreq = 0.15, numwords = 250, iter = 25, ncores = 4

Cluster	**Top frequent words**
CLASS0	region, temperatur, line, sourc, emiss, ratio, identifi, sim, veloc, origin
CLASS1	mechan, understand, test, various, found, were, control, out, character, dure
CLASS2	without, train, need, challeng, dataset, evalu, novel, task, real, thus
CLASS3	second, same, interest, theoret, give, coupl, them, analyz, about, prove
CLASS4	oper, quantum, correspond, implement, extend, construct, defin, novel, evalu, challeng
CLASS5	current, imag, thus, multi, class, address, object, challeng, train, dataset
CLASS6	approxim, equat, analyt, dimension, solv, integr, describ, same, standard, finit
CLASS7	accuraci, error, mean, train, evalu, dataset, accur, neural, independ, deep
CLASS8	lower, bound, size, higher, prove, best, found, random, constant, factor
CLASS9	star, stellar, format, galaxi, survey, evolut, suggest, sim, line, cluster

Upon inspection of the obtained clusters we can check that some classes make sense and the topic of the documents from the cluster can be guessed. For instance, if we take a look at the seventh cluster, all top ten frequent words which appear in the table have some relationship with machine learning processes and their evaluation. In contrast, cluster nine includes some tokens like: star, stellar, galaxi and cluster that may indicate a grouping of documents related to astronomy. Furthermore, on the cluster zero most of the frequent words seem to imply a topic related to physics while on cluster six we could identify a topic concerning a mathematical branch.

In some of the cases the topic of the documents can indeed be guessed and the assignments make sense so we are happy with the obtained results. Out of curiosity we tried including words of higher frequency in dataset and just as expected these high frequency tokens were included in all clusters, the token "we" being the the most common one in every case. The quality of the clusters we have obtained is much better as a consequence of the experimentation steps on this laboratory.