UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Facultat d'Informàtica de Barcelona

FIB

IRRS Lab 4 Report

MapReduce Implementation of K-means

Yasmine Souabi
MSc student
yasmine.souabi@estudiantat.upc.edu

Emmanuel Werr
MSc student
emmanuel.werr@estudiantat.upc.edu

Facultat d'Informàtica de Barcelona
Master of Data Science

December 8, 2022

1 Introduction

The main purpose of this lab is to implement a document clustering algorithm and use it to explore a set of documents. This will help us learn how to use the map-reduce python library MRJob. Throughout the report we will discuss the K-means algorithm implementation using MRJob as well as the effects of choosing different parameters both at the time of Extracting the data and generating Prototypes, as well as during the Reducer step of the MapReduce job.

2 Extracting the Data

The first part of the exercise requires querying the entire index in ElasticSearch in order to create a 'documents.txt' containing a dictionary of all documents in the index along with a list of their contained words. We then use the 'documents.txt' file in order to create a 'prototypes.txt' file with random clusters that we will attempt to fit our entire index into at least one of.

The provided scripts, 'ExtractData.py' and 'GeneratePrototypes.py' allow us to perform the steps detailed above. For extraction we have some parameters to control the available information:

- –minfreq: bottom threshold of global frequency of words selected.
- –maxfreq: upper threshold of global frequency of words selected.
- –numwords: vocabulary size cutoff (by most frequent).

The values we choose for these parameters during extraction will have a profound effect on the result of our clusters at the end of the exercise. Our frequency range will determine how separable our data will be by the generated clusters (selecting a high frequency range will mean each individual cluster will struggle to separate documents into specific classes, and vice versa for selecting a low frequency range).

3 Implementing K-means Algorithm with MRJob

After reading all necessary data and constructing the necessary initial data structures to perform our computations, there are two major parts of the Map Reduce job, which we implement as a single mapper and single reducer.

The mapper takes each line of the 'documents.txt' file as input and calculates the Jaccard Similarity of the list of words in the document to a list of words for each cluster in prototypes. Its purpose is to determine the most similar class of each document, and assign the document to that class. The reducer then takes the output pairs from the mapper, and performs an aggregation by each class of the total words for each document and updates the words' frequency count in the original 'prototypes.txt' file. Once the prototypes list is filled with all (words)+(word frequencies), we sort the list descending by

their calculated frequency and only keep the top words. This way we imitate the natural behavior of the k-means algorithm of attempting to move closer to the spatial mean of a cluster of points after every iteration of k-means.

4 Experiments and Results

We ran the 'MRkmeans.py' for many variations of 'documents.txt' and 'prototypes.txt', mainly by using different parameters for minfreq, maxfreq, and numwords. To keep the report relatively short, we have only included results graphs for the base case (minfreq=0.1, maxfreq=0.3, numwords=200). Throughout our experimentation, we also obtained different results by selecting a different number of total classes generated in the initial prototypes file, we chose 5 for these visualizations because they turned out nicely for the report.

During our trials we found it helpful to use this range of frequencies when it comes to the resulting differentiability of the classes. The line plots in the figure below represent the number of assigned docs of each topic to each particular class at the end of each iteration of k-means from 1 to 10. As we can see, each class has a unique distribution of total docs assigned by topic. We can also see that the classes reach convergence after the 6th iteration of k-means.

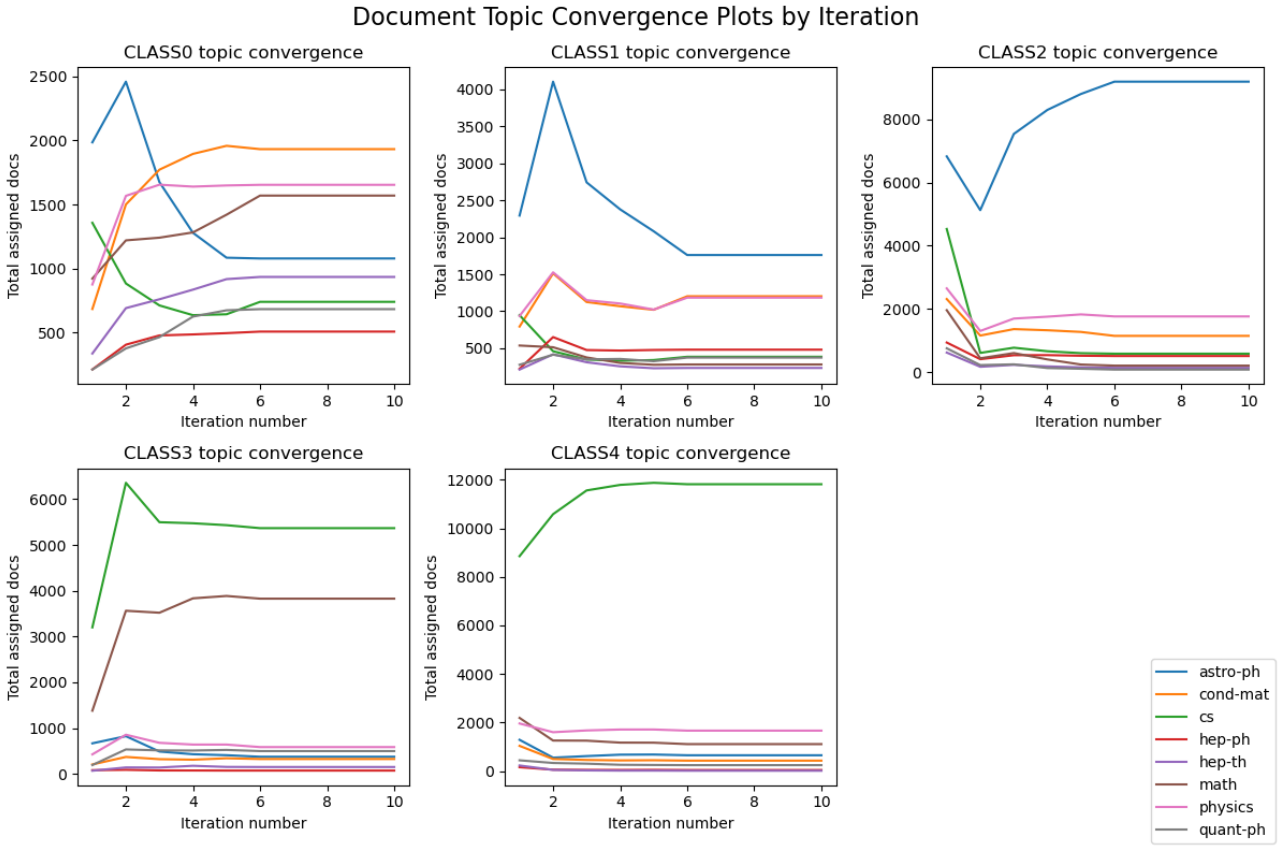

Figure 1: Convergence Summary for each prototype class

Below is also the terminal output for the 'ProcessResults.py' file showing the top words in each of the class prototypes after k-means has converged, and it looks like a decent result. One or two common words are repeated within classes but overall the prototypes are sufficiently different to allow us to draw reasonable conclusions about the topics of the documents they contain.

For example: by combining the results of the terminal output (below) with our topic convergence graphs (above), we could hypothesize that docs in 'CLASS2' contain mostly papers related to astro-physics (with 'mass' and 'star' within the top words of the prototype), while docs in 'CLASS4' are mostly about topics in computer science (with 'comput' and 'network' within the top words of the prototype). Given the terminal output, we could also hypothesize that docs in 'CLASS0' are related mostly with mathematics and physics/quantum physics (given the words 'field', 'theori', 'equat', and 'state'). We are able to confirm this hypothesis by looking at the topic convergence plot for 'CLASS0'.

```
(irrs_env) efwerr@m1Mac lab4 % python processresults.py
CLASS4
[(0.5041051916784987, 'learn'), (0.46113957651706894, 'network'), (0.4092845237360331, 'approach'), (0.38527069572195816, 'howev'), (0.36218285079325885, 'comput')]
CLASS2
[(0.3962501852675263, 'mass'), (0.35630650659552393, 'star'), (0.3405958203646065, 'measur'), (0.3191047873128798, 'compar'), (0.31488068771305766, 'low')]
CLASS3
[(0.5202932704896570, 'problem'), (0.4346687614558785, 'set'), (0.398882779087021, 'general'), (0.394169503360391, 'algorithm'), (0.36746094090948767, 'case')]
CLASS0
[(0.411917684305744, 'field'), (0.386250565355043, 'theori'), (0.3476933514246947, 'equat'), (0.3473541383989145, 'state'), (0.34283129805517865, 'non')]
CLASS1
[(0.37894554283732224, 'measur'), (0.3373222337842525, 'field'), (0.3194588969823101, 'here'), (0.313735691987513, 'physic'), (0.30939993062781823, 'energi')]
(irrs_env) efwerr@m1Mac lab4 %
```

Figure 2: ProcessResults.py terminal output after running

Below we are also showcasing the convergence of the overall size of each cluster (by total documents assigned) after each iteration of the k-means algorithm. We can see that 'CLASS4' is the largest cluster, followed by 'CLASS2', 'CLASS3', 'CLASS0', and finally 'CLASS1' (the smallest cluster).

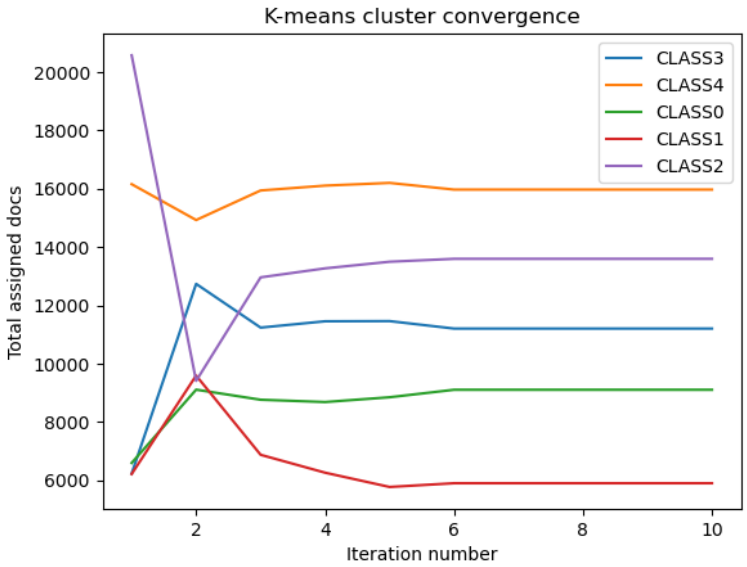

Figure 3: ProcessResults.py terminal output after running

5 Difficulties and Final Thoughts

This lab presented many difficulties, especially in implementing the K-means algorithm properly with the MRJob package in Python. The main difficulty was in the reducer step, where it was tough to find a good way of generating a cluster's new prototype after each iteration of k-means.

One problem with our method of generating the new prototypes in the Reducer is that the final classes are very dependent on the initial prototypes generation. In order to review the code for our implementation, please revise the 'MRKmeans.py' and 'MRKmeansStep.py' files within the zipped folder submission.

There is still much left to learn about using the MRJob package in python for MapReduce jobs, but this lab proved to be a rewarding experience filled with many learning opportunities.