III ASSIGNMENT

LEARNING WITH MASSIVE DATA

Luca Bizzotto 875814

May 30, 2023

Data

For this analysis we choose to use the SciFact dataset that contain 5 k documents that is one of the BeIR datasets. Since the SciFact consist of a large collection of documents that for the task in question is computationally onerous for the resources at our disposal, and the task required to test the various algorithms with data sets that were different in size and different from each other. We randomly created three different data set subsets from the SciFact then later we will use during our analysis. Besides being then the collections of documents created not huge and therefore with high probability having low levels of similarity between documents, we have moreover made some data augmentation for each data set created, so in this way we are sure when we will vary the threshold of the similarity between documents, as a task requirement we will have some documents that will overtake it.

Representation of the documents

First of all we create a sparse representations of the documents. We used the bag of words model using as a dictionary the vocabulary compose by documents words and then we calculate for each doc in BOW its tf_idf and then we normalize it by its norm2 $\|d\|_2$.

Sequential implementation

Since the similarity measure is symmetric we choose to calculate only the similarity of the upper diagonal of the matrix. In addition to speed up the computation of the sequential algorithm we first sort all documents by their length and then we calculate K. K indicate the minimum length that a document must have to have a similarity measure that is greater or equal of the fixed similarity threshold. In this way when we calculate K and since all the document are in decreasing order by length if at any point we find a document that has a length that not satisfy the requirement K length, we can skip it and all the other documents below it that has less or equal length. **To compute** K for a given document we had the following reason: since we want to do safe skip we want be sure that if we skip a document we are sure that the similarity measure is not good enough. So we sort first the tf_idf value of the document in decreasing order then we calculate progressively the inner product with itself and we subtract to the threshold until we go bellow zero, at that point we know for sure the minimum length that a document must have, since no one document can reach the threshold value earlier then K.

Map Reduce first method implementation

The idea that we used to compute the cosine similarity for each pair of document in the novel paradigm of map reduce is the following:
Observation we know that the minimum requirement to have a similarity measure above 0 between two documents is that at least they have one term in common.
Map: the map function that each mappers execute will do the following: it get as an input doc_id and the vector of tf_idf representing the document, and for each term in the document will produce a pair of the kind(term, [doc_id, tf_idf]).
Reduce: the reduce function will take as input a term and the list of document containing it and for each pair we will compute the similarity, and if the similarity overtake the threshold it count the pair as similar document.
On the spark framework to implement the function that we just describe we choose to work mainly with transformation and a then at the end we used an action to collect in the master all the partial result.

Map Reduce second method implementation

With the first method we are emitting for each term in a document a pair of the kind (term, [doc_id, tf_idf]) so each each document is replicate many times as the number of term it has. Now in this implementation we want to reduce it.
What we did is the following: in pre-processing we sort in decreasing order the column of the tf_idf matrix (where each row represent a document in tf_idf format) based on tf_idf value, so in this way we get term with high tf_idf value at the begining. Then we calcualte a vector, we call it d_star that store the maximum score of each column. Then for each document, we calculate what we call the

$b(d)$. The $b(d)$ represent the largest term such that a document similar to d must share with d to have a cosine similarity that satisfy the threshold.
Now after the pre-processing thanks to the sort and the b(d) index, we will not anymore produce a pair for each term in the document as before, but we will produce just pair for term that are over the b(d) indices,so thanks to this improvement we reduce the number of pair that we emit and then as a result we saved computation to calculate the similarity measure.

Map Reduce third method implementation

Now the improvement we wanna bring in this method arrive from the following consideration. If two documents share terms after filtering, their similarity measure similarity is computed many times as the number of terms the two documents share. To overtake this problem we want to create a general rule that each reducer will use without the need of communicate with any other reducer. Each reducer know (doc_id, tf_idf, b_d) for each document, so it compute the cosine similarity only on the term the respect the following: we take first the maximum between the b(d) of the two documents and we know per construction that for sure the term that they have in common is over that index, then when both documents share a term over the max(b(d)) index we check if is the term that the reducer has in charge and if is that one we compute the cosine similarity.

Analysis

Plot analysis you can find all the plot that we are speaking about during this analysis as an attachment file called **Analysis.png** in the Assignment directory submitted for evaluation.
Following the directives required by the assignment we evaluated the performance of all the implementations of the above mentioned algorithms for each data set, varying the threshold, and also varying the number of workers.
The datasets used in our analysis are characterise by different size of the collection of document, in particular $|corpus\_100| \leq |corpus\_200| \leq |corpus\_500|$.
From the results obtained we can say the following:
From a **threshold** perspective we can see that increasing the threshold will bring an improvement in performance, and we can see this behaviour for all type of algorithm and for all data-set used. We can therefore say that the optimization passage we used in our algorithms to skip document that are not relevant for our purpose are working as we wanted and help us to save computational time.
From a **algorithm** perspective we can see the sequential algorithm seams to be the one that offer better performance, but changing from one data set to another one that has a larger collection the performance between the sequential implementation and the second_method are going to approach, in particular in the data set corpus_500 that is the one with the larger collection of document we can notice that the performance of the second method are going to be better then the sequential one. This behavior lead us to say that using collection of document that are characterized by a larger set of documents then the corpus_500 the second_method is the one that will offer better performance.
Our opinion about the third method that are not giving the best performs despite the improvement we are trying to bring, is that the cost of the linear scansion of the tf_idf vector of the two documents that share the term, to find if is the right reducer that has to calculate the similarity between those two documents is greater then compute multiple times the similarity between those two vector.
At end respect varying the number of **worker** we can say that contrary from what we are expecting the computational time are not going to improve, but instead the plots in the file Analysis.png show us worst time performance, probably this is due a limitation given from our hardware used in the analysis.