Università
Ca'Foscari
Venezia

LEARNING WITH MASSIVE DATA
CM 0622-1

ASSIGNMENT 2

Student:

Giosuè Zannini 873810

Academic year 2022/2023

Contents

Chapter 1

Introduction

1.1 Problem

This assignment is divided in two parts:

1. What do we do when vectors have both a dense subspace and a sparse subspace? In other words, if your documents and queries are of the form $u \oplus v$, where $\oplus$ denotes concatenation, $u \in \mathbb{R}^n$ is a dense vector, and $v \in \mathbb{R}^N$ is a sparse vector where $n << N$, how do we efficiently retrieve the top-k documents with respect to a given query by maximal inner product? One idea is to take advantage of the linearity of the inner product operator: The inner product of a query vector $q = q_{\text{dense}} \oplus q_{\text{sparse}}$ with a document vector $d = d_{\text{dense}} \oplus d_{\text{sparse}}$ is the sum of $q_{\text{dense}}^{\text{T}} d_{\text{dense}} + q_{\text{sparse}}^{\text{T}} d_{\text{sparse}}$. We can now approximate the joint MIPS[1] by breaking it up into two smaller MIPS problems: Retrieve the top-k' documents from a dense retrieval system defined over the dense portion of the vectors, and another set of top-k' documents from a sparse retrieval system operating on the sparse portion, before somehow "merging" the two sets and retrieving the top-k documents from the combined (much smaller) set. My task is to examine the correctness of the algorithm above and study the effect of k' on the quality of the final top-k set, theoretically and/or empirically. I decided

[1] Maximum Inner Product Search

to do the empirically part and for this I designed an experiment to demonstrate this effect on a real-world dataset with off-the-shelf deep learning "embedding" models.

2. Suppose you have implemented the algorithm above to retrieve the top-k set by maximal inner product. Your task in this problem is to demonstrate that this sparse-dense retrieval can be beneficial in practice. In particular, you are asked to study the effect of combining dense and sparse embedding models on the ranking quality on the BeIR datasets. More concretely:

   - Implement a program to perform retrieval over dense embeddings using FAISS.
   - Implement another program to perform top-k retrieval over sparse vectors produced by BM25.
   - Measure the quality of the two rankings separately for some k.
   - Examine if a fusion of the BM25 scores and dense model's scores lead to better ranking quality.

   You are encouraged to think about the right fusion. As candidates, take a convex combination of BM25 and dense scores, $\alpha f_{\text{Dense}} + (1-\alpha) f_{\text{BM25}}$, and study the effect of the α parameter on the final ranking quality. Does the fact that BM25 scores (or the dense scores) are unbounded pose an issue for the fusion? If so, how do you propose to remedy that?

Chapter 2

First part

2.1 Idea of implementation

Being that I do the empirical part I decided to use the Scifact dataset composed by 300 queries and 5183 documents. Instead with regard to models I used BM25 to generate sparse vectors and as sentence-transformers model I used all-minilm-l6-v2 to generate dense vector with a dimension equal to 384. With these models I implemented the algorithm (approximate joint MIPS) for the first part using as merging the sum between the sets of score (one composed by top-k′ documents from sparse retrieval system and the other composed by top-k′ documents from dense retrieval system) where the cardinality of this merging set is at most 2k′. To evaluate the algorithm I used the following metrics (where the ground truth is the joint MIPS):

- NDCG: Normalized Discounted Cumulative Gain is a popular measure for evaluating web search and related task.

$$\text{NDCG} = \frac{\text{DCG}}{\text{IDCG}} \tag{2.1}$$

  where:

  - $\text{DCG} = \text{rel}_1 + \sum_{i=2}^{p} \frac{\text{rel}_i}{\log_2 i}$ (2.2)
  - IDCG is the ideal Discounted Cumulative Gain (max possible DCG).

- MRR: Mean Reciprocal Rank is the multiplicative inverse of the rank of the first retrieved relevant document averaged over many queries.

$$\text{MRR} = \frac{1}{|Q|} \sum_{i \in Q} \frac{1}{\text{rank}} \tag{2.3}$$

  where:

  - rank is the position of the first retrieved relevant document.

- MAP: Mean Average Precision is the average of the precision scores computed after each relevant document is retrieved averaged over many queries.

$$\text{MAP} = \frac{1}{|Q|} \sum_{i \in Q} \frac{\sum_{\text{r}} \text{Precision @r}}{\text{R}} \tag{2.4}$$

  where:

  - r is the position of a relevant document.
  - R is the total number of relevant documents.

- AR: Average Recall is the ratio between the retrieved documents that are relevant and the total number of relevant documents averaged over many queries.

$$\text{AR} = \frac{1}{|Q|} \sum_{i \in Q} \frac{\text{r}}{\text{R}} \tag{2.5}$$

  where:

  - r is the number of retrieved relevant documents.
  - R is the total number of relevant documents.

2.2 MIPS

MIPS is a search problem, with a corresponding class of search algorithms which attempt to maximise the inner product between a query and the data items to be retrieved. MIPS algorithms are used in a wide variety of big data applications, including recommendation algorithms and machine learning. Formally given a query vector $Q \in \mathbb{R}^n$ and a set of document vectors $\mathcal{D} \subset \mathbb{R}^n$, find the top-k documents with maximal inner product with Q:

$$\delta = \arg \max_{d \in \mathcal{D}}^{(k)} \langle Q, d \rangle \tag{2.6}$$

MIPS can be viewed as equivalent to a nearest neighbor search problem in which maximizing the inner product is equivalent to minimizing the corresponding distance metric in the nearest neighbor search problem.

2.3 BM25

In information retrieval, BM25 (BM is an abbreviation of best matching) is a ranking function used by search engines to estimate the relevance of documents to a given search query. It is based on the probabilistic retrieval framework developed in the 1970s and 1980s by Stephen E. Robertson, Karen Spärck Jones, and others. It is a bag-of-words retrieval function that ranks a set of documents based on the query terms appearing in each document, regardless of their proximity within the document. The problem that BM25 tries to solve is similar to that TF-IDF[1], The gist behind TF-IDF is that is relies on two main factors to determine whether a document is similar to the query:

- TF: how often does the term occur in the document.
- IDF: how many documents the term appeared in.

[1]Term Frequency - Inverse Document Frequency

Using these two factors, it measures the relative concentration of a term in a given document. If the term is common in this article, but relatively rare elsewhere, then the TF-IDF score will be high, and documents that have higher TF-IDF score would be considered as very relevant to the search term. BM25 improves upon TF-IDF by casting relevance as a probability problem. A relevance score, according to probabilistic information retrieval, ought to reflect the probability a user will consider the result relevant. BM25 function scores each document in a corpus according to the document's relevance to a particular text query. For a query Q, with terms $q_1, \ldots, q_n$, the BM25 score for document d belonging in the collection of documents $\mathcal{D}$ is:

$$\text{BM25}(d, Q) = \sum_{i=1}^{n} \text{IDF}(q_1, d) \cdot \frac{\text{TF}(q_i, d)(k_1 + 1)}{\text{TF}(q_i, d) + k_1(1 - b + b \cdot \frac{|d|}{l})} \tag{2.7}$$

Where:

- $|d|$ is the number of words in document d.
- l is the average number of words per document.
- b and k_1 are hyper parameters.
- $\text{IDF}(q_1, d) = \log\left(1 + \frac{|\mathcal{D}| - \text{CF}(q_1) + 0.5}{\text{CF}(q_1) + 0.5}\right)$ (2.8).
  Where:
  - $\text{CF}(q_1)$ is the number of documents that contain term q_1.

The hyper parameter k_1 determines the term frequency saturation characteristic, it models non linearity of term frequencies contribution, the higher the value, the slower the saturation.

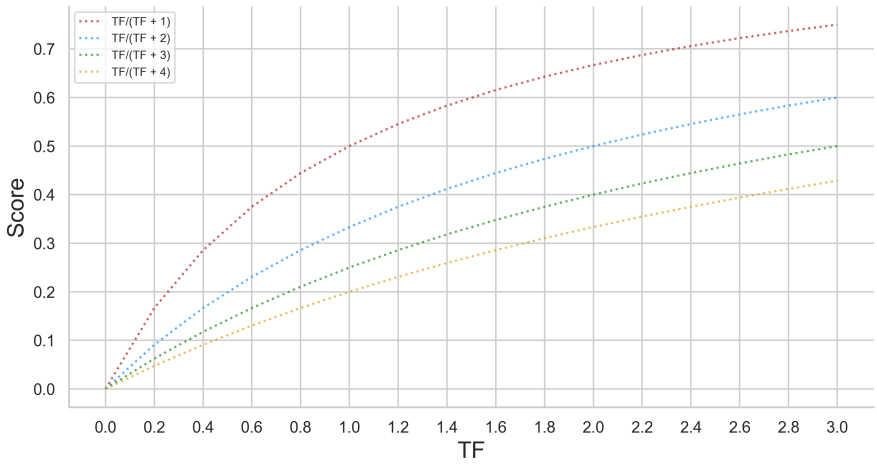

FIGURE 2.1: TF contribution by changing k_1

Then $\frac{|d|}{l}$ part in the denominator means a document that is longer than the average documents will result in a bigger denominator, resulting in a decrease in the score. The intuition is that the more terms in the document that does not match the input query the lower the document's score should be. The hyper parameter $b \in [0, 1]$ helps to encourage this behaviour. $b >> 0$ goes to penalize long documents because they naturally have a better chance to match any query, instead $b = 0$ removes the penalization given by the length of documents.

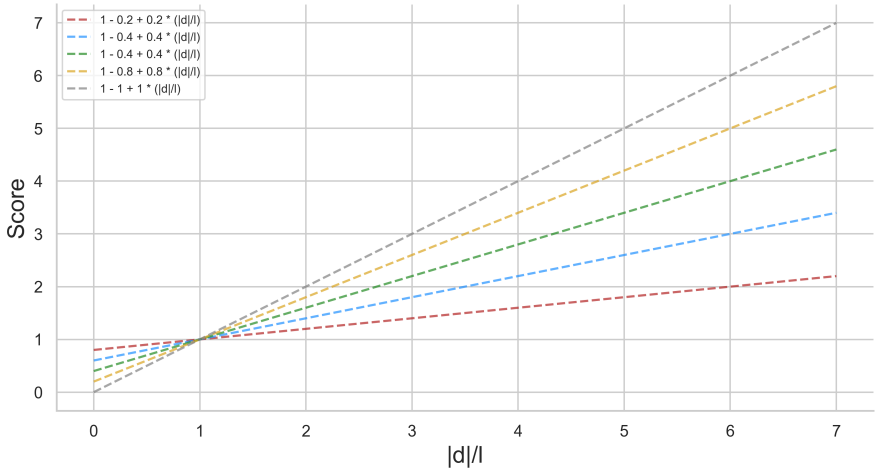

FIGURE 2.2: denominator contribution by changing $\frac{|d|}{l}$ with different b

The inverse document frequency part is very similar to that of TF-IDF, whose role is to make sure that rarer words will have a higher score and contribute more to the final score, the $\log$ function is used to decrease the effect of rarer words. IDF affects the ranking of documents for queries with at least two terms. Being that I use MIPS to retrieve the top-k documents I must express BM25 as the inner product of two vectors.

Proof. Since that BM25 is similar to TF-IDF I can write it in the following way:

$$\text{Score}(Q, d) = \sum_{q \in Q} \text{TF}(q, d) \cdot \text{IDF}(q) \tag{2.9}$$

Let V be a vocabulary, $\vec{d} \in \mathbb{R}_+^{|V|}$ be a vector whose i^{th} coordinate records the importance of the i^{th} term in V.
Let $\vec{Q} \in \mathbb{R}_+^{|V|}$ be a vector whose i^{th} coordinate records the count of the i^{th} term of V in the query.
Then I can simply express the BM25 as dot product of two vectors.

$$\text{Score}(Q, d) = \langle \vec{Q}, \vec{d} \rangle \tag{2.10}$$

□

2.4 sentence-transformers model

SentenceTransformers is a Python framework for state-of-the-art sentence, text and image embeddings. I can use this framework to compute sentence/text embeddings for more than 100 languages. These embeddings can then be compared with cosine-similarity (so I can use MIPS) to find sentences with a similar meaning. The framework is based on PyTorch and Transformers and offers a large collection of pre-trained models tuned for various tasks. Before sentence transformers, the approach to calculating accurate sentence similarity

with BERT was to use a cross-encoder structure. This meant that I would pass two sentences to BERT, add a classification head to the top of BERT and use this to output a similarity score. The BERT cross-encoder architecture consists of a BERT model which consumes sentences A and B, both are processed in the same sequence, separated by a [SEP] token. All of this is followed by a feedforward NN classifier that outputs a similarity score. The cross-encoder network does produce very accurate similarity scores, but it's not scalable.
The solution of this lack was the introduction of sentence-transformers library. It produces sentence embeddings, so I don't need to perform a whole inference computation for every sentence pair comparison. The training process of sentence transformers is especially designed with semantic similarity in mind.

2.5 Evaluations and considerations

After implementing the required algorithm I wanted to verify using different metrics the behavior of approximate joint MIPS going to modify both k and k' with the same step, this is the result that I achieved:

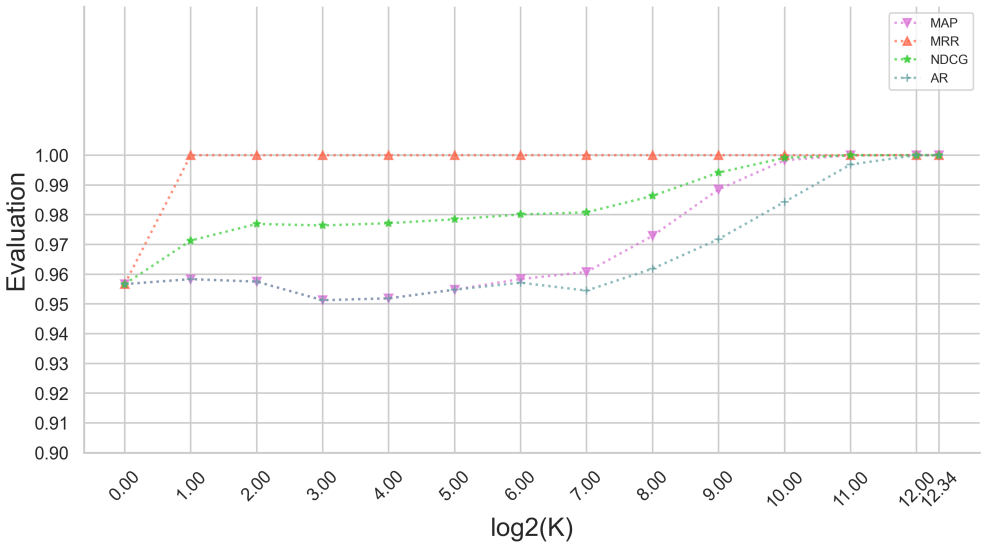

FIGURE 2.3: Behaviour changing both k and k'

From it I can notice that with k and k' equal the size of documents all metrics

reach 1 being that in this case the approximate joint MIPS converges to the exact MIPS.

Looking at the MRR metric I notice that its value immediately reaches 1 as soon as the number of k and k' reaches 2, this is due to the fact that at least one of the two sets (sparse and dense) containing the top-k' has inside the most relevant document in the ground truth. This can be seen from the graph below which represents with a k' fixed at 2 the two different sets and goes to check if at least one set contains the most relevant document present in the joint MIPS (being that the number of queries was too high I decided to randomly select fifteen of them so as to have a graph cleaner and easier to view).

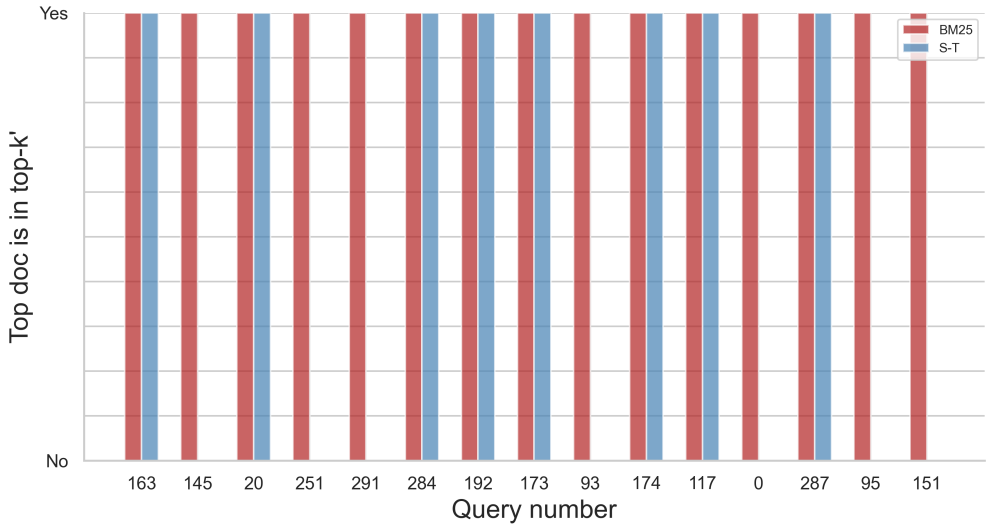

FIGURE 2.4: Check if one of the two metrics contain in the top-2 the top doc for some random queries

Looking at the NDCG metric I can see that by increasing k and k' the value continues to rise until it reaches its maximum, this because increasing k' I go to increase the size of the two approximate sets and executing the union through the sum of the scores I go to get a set of dimension at most 2k' and this makes so that my approximate set contains a greater number of documents present in the ground truth, where the order of the two sets (joint and approximate MIPS) is respected, i.e. $\forall d_i, d_j \in \mathcal{D}, i \neq j \mid \text{J-MIPS}_q(d_i) > \text{J-MIPS}_q(d_j) \Rightarrow$

APPR-MIPS$_q(d_i) >$ APPR-MIPS$_q(d_j)$, where q is a query.

Looking at the recall metric I note that it has an initial growth, followed by a small decrease and then have a positive trend until reaching the maximum. This behavior in my opinion is due to the fact that the two metrics give scores belonging to different sets, BM25 $\in \mathbb{R}^+ \cup \{0\}$ and sentence-transformers $\in [-1, 1]$ and the fact that BM25 doesn't consider semantic information and sentence-transformer instead considers it. Also taking into account a high k the first metric using a sparse vectors will make a great contribution to the documents in the upper part of the ranking list but will not give any contribution to the documents in the lower part of the list as those values will be zero, instead the other metric will make its contribution for both the top and the bottom of the list, of course the biggest contribution will be for the bottom because the range of values of the score is clearly lower than the range of the score of the BM25. This behavior can be seen from the following graph:

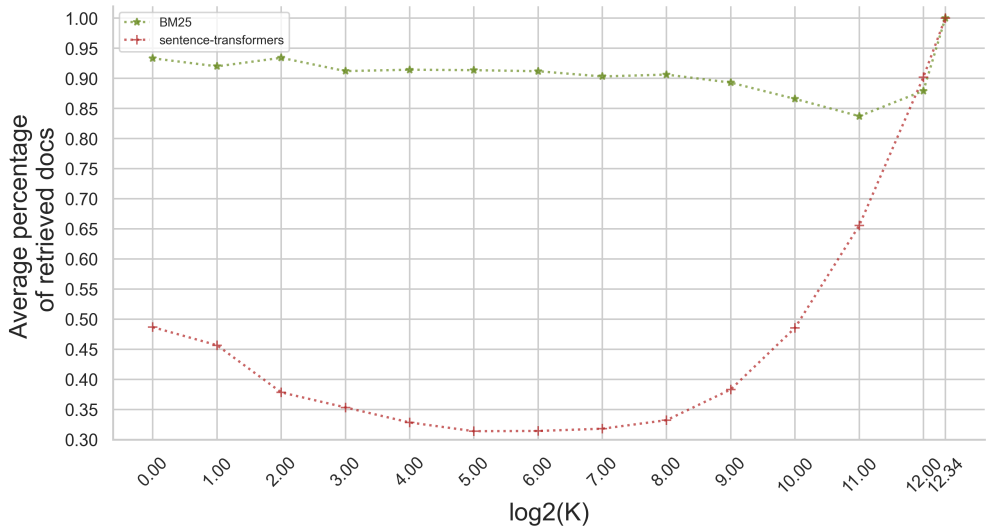

FIGURE 2.5: Behaviour of BM25 and sentence-transformer changing both k and k'

Looking at the MAP metric I note that it follows the same path of recall and at some point its value grows more than the recall because in my opinion the ordering of the two sets (joint and approximate MIPS) is respected and therefore

the value grows faster.

Another behavior I want to analyze is to change k' using all possible scale while keeping k fixed, this is the result that I achieved:

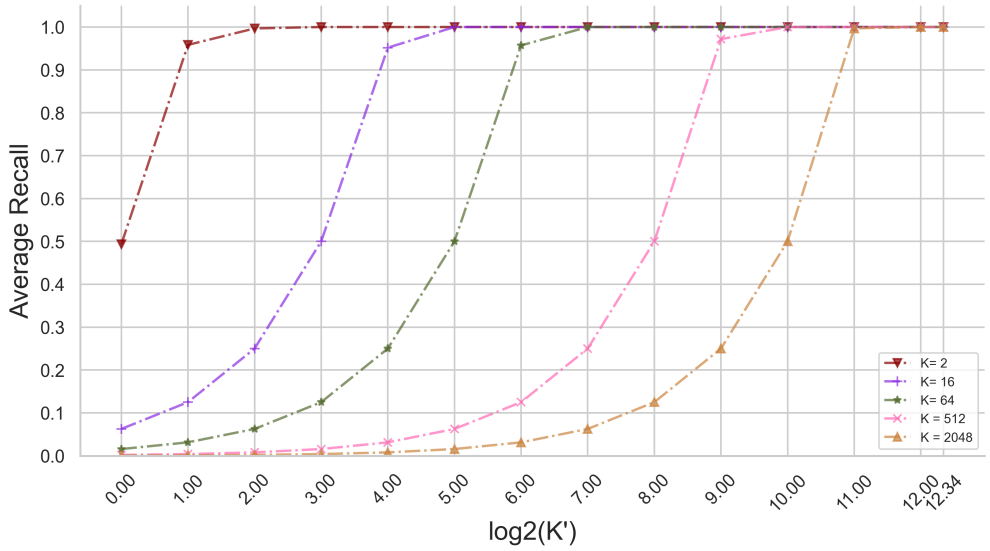

FIGURE 2.6: Behaviour changing k' while k is fixed

Looking at the recall metric in this graph I can see that at the same value for k and k' the recall is higher than 90%, obviously with k' < k the recall is lower because it is impossible that all documents are retrieved.

In addition, there may be a case where the intersection between approximate and joint MIPS is null, due to the fact that neither of the two sets (dense and sparse) contains any document belonging to the top-k in joint MIPS, here it is very difficult to happen as the BM25 dominates (especially on the top of the ranking list) on the determination of the top-k.

Chapter 3

Second part

3.1 Idea of implementation

In this second part I decided to use the same dataset (Scifact) of the first part. As program to retrieve the top-k documents over sparse vectors I used the TAAT[1] algorithm, instead regarding dense vectors I used FAISS with safe ranking version in order to retrieve the real top-k documents. As metrics I decided to use the same used in the first part, so:

- NDCG.
- MRR.
- MAP.
- AR.

3.2 TAAT

TAAT is a technique to determine the top-k documents using a structure called inverted index where it is a data structure that allows efficient, full-text searches in the database. It is a very important part of information retrieval systems and search engines that stores a mapping of words (or any type of search terms) to

[1] Term At A Time

their locations in the database table or document. The TAAT algorithm reduces the cost of determining top-k documents for sparse vectors. As for the difference of the algorithm MIPS that provides for the calculation of the dot product between the query and all the documents in the collection it goes to compute the score of the only documents that have in their body the terms present in the query. As the name says the idea of this algorithm is to compute for each term the score for each document that presents such term and to accumulate the sum for all the other terms of the query that will be computed.
the downside of this algorithm is about memory waste, being that computes the score of the document for each term will not know the real value of the document until the end of the algorithm and for this reason it is necessary to keep in memory all the documents that will be computed ($O(|\mathcal{D}|)$) to then determine the top-k.

3.3 FAISS

FAISS is a library developed by Facebook AI that enables efficient similarity search.
So, given a set of vectors, I can index them using FAISS then using the query vector, I can search for the most similar vectors within the index.
IndexFlatL2 measures the Euclidean distance between all given points between my query vector, and the vectors loaded into the index. This algorithm alone is computationally expensive and it doesn't scale well. When using this index, I am performing an exhaustive search, meaning I compare my query vector to every other vector in the index, FAISS allows my to add multiple steps that can optimize the search using many different methods. A popular approach is to partition the index into Voronoi cells. Using this method, I would take a query vector, identify the cell it belongs to, and then use IndexFlatL2 algorithm to search between the query vector and all other vectors belonging to that specific

cell. In this way I am reducing the scope of our search, producing an approximate answer, rather than exact (as produced through exhaustive search).

3.4 Evaluations and considerations

3.4.1 TAAT

After implementing the required algorithm I wanted to verify the performance of this retrieval system, this can be seen from this plot:

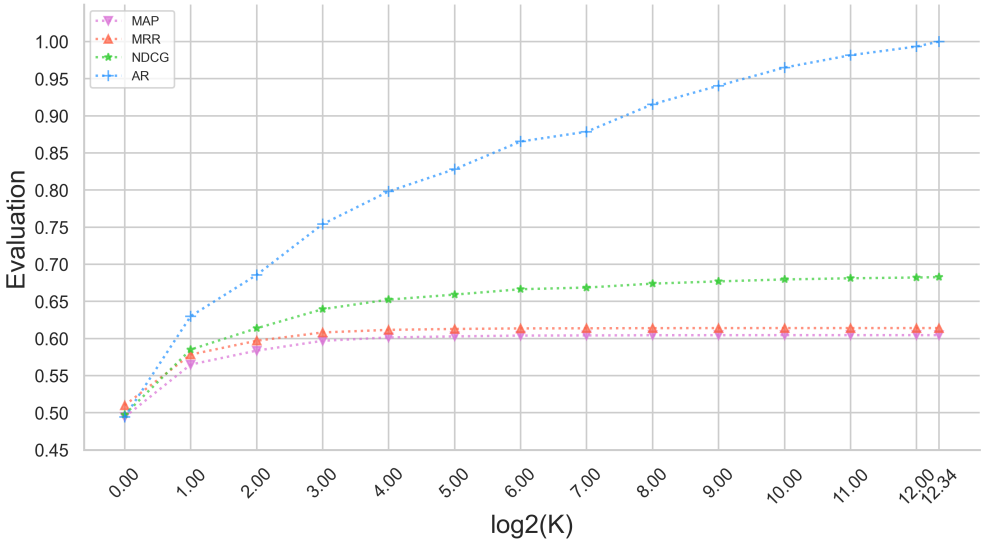

FIGURE 3.1: Evaluation of TAAT algorithm by varying the value of k

From this note that to the increase of k the recall goes to 1, this is obvious being that I go to retrieve all the documents in the collection. Instead as far as MRR and MAP both grow up to $k = 16$ and then stabilize, this behavior in my opinion is due to the fact that in most of the queries in this dataset the relevant documents are recovered within the first 16 positions.

Instead as for the metric NDCG initially has a rapid growth and then go to continue this positive trend but very slightly until reaching its peak with $k = |\mathcal{D}|$,

this in my opinion is due to the fact that for some queries my algorithm retrieves the relevant documents only in the lower part of the ranking list, in this graph is the verification of what I just said:

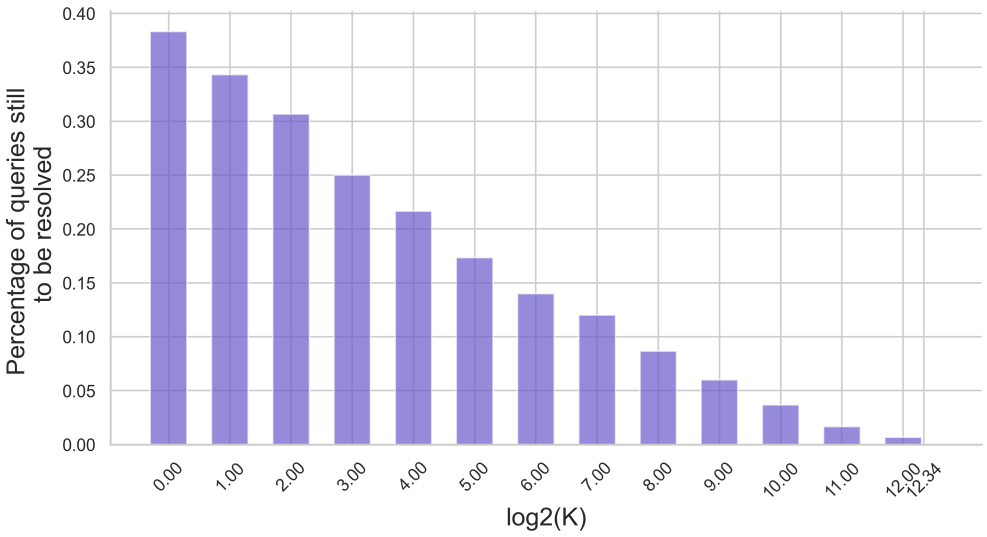

FIGURE 3.2: Percentage of queries which documents have yet to be retrieved

3.4.2 FAISS

After implementing the required algorithm I wanted to verify the performance of this retrieval system, this can be seen from this plot:

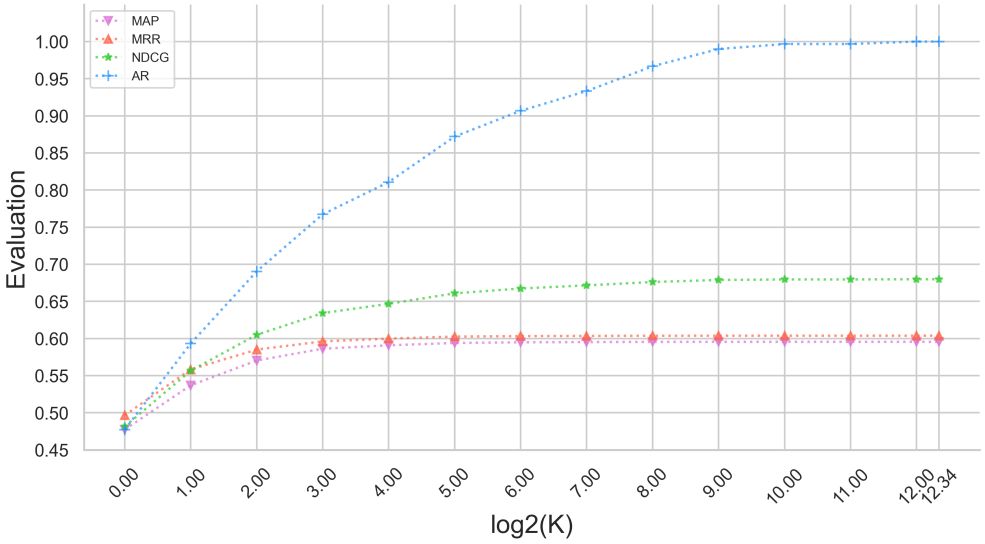

FIGURE 3.3: Evaluation of FAISS algorithm by varying the value of k

From this graph I note that it has almost the same behavior of the plot regarding the performances of the algorithm TAAT even if MAP, MRR and NDCG have a lower score. One difference concerns the recall metric which rises more quickly to 1, probably due to the fact that towards the end of the list the scores of BM25 are at zero and therefore there isn't a preference for recovering one document at the expense of another, unlike the sentence-transformer which will always have (or almost) a preference between two documents. The fact that this algorithm has slightly lower scores is due to the fact that the TAAT resolves $\approx 62\%$ of the queries with a $k = 2$ and this increases the other metrics more as the relevant documents appear at the top of the list.

3.4.3 Fusion between BM25 and Sentence-transformer

In this subsection I wanted to discover if a fusion between the two methods lead to a better result in terms of score. As first stage I decided to verify the performance of this new retrieval system using as fusion the following function without any standardization:

$$\text{Score}(Q, d) = \alpha \cdot f_{\text{Dense}}(Q, d) + (1 - \alpha) \cdot f_{\text{Sparse}}(Q, d) \tag{3.1}$$

Where the parameter $\alpha \in [0, 1]$ is an hyper-parameter.
The plot below show the performance of this merge.

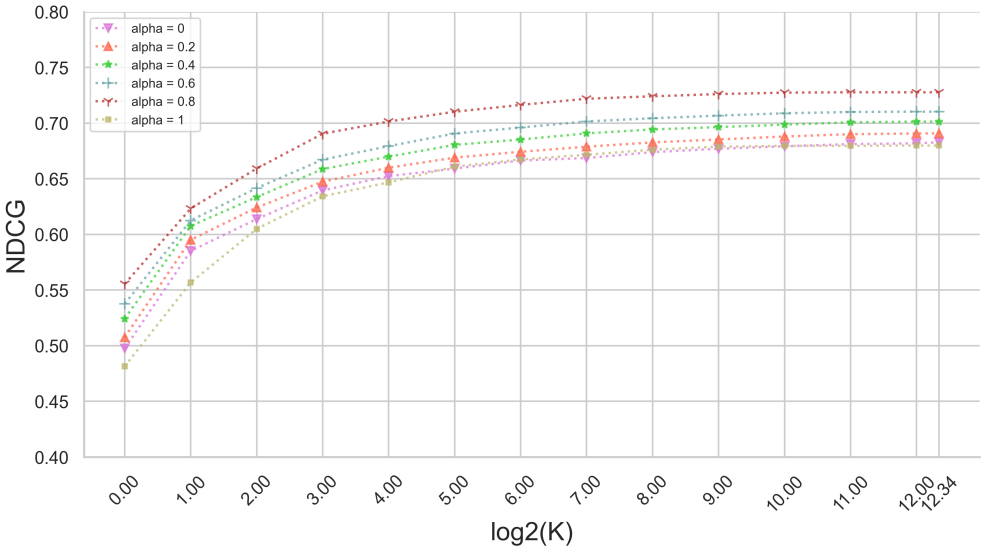

FIGURE 3.4: Evaluation of fusion without standardization by varying the value of k

From it I can see that the behavior of varying k is the same for all α. The best result is $\alpha = 0.8$, the worst performance are when only one of the two methods is used ($\alpha \in \{0, 1\}$) and this makes me think that a merger is the correct thing to do.

Now I want to visualize the same graph but using standardized metrics, as they don't have a common set of values.

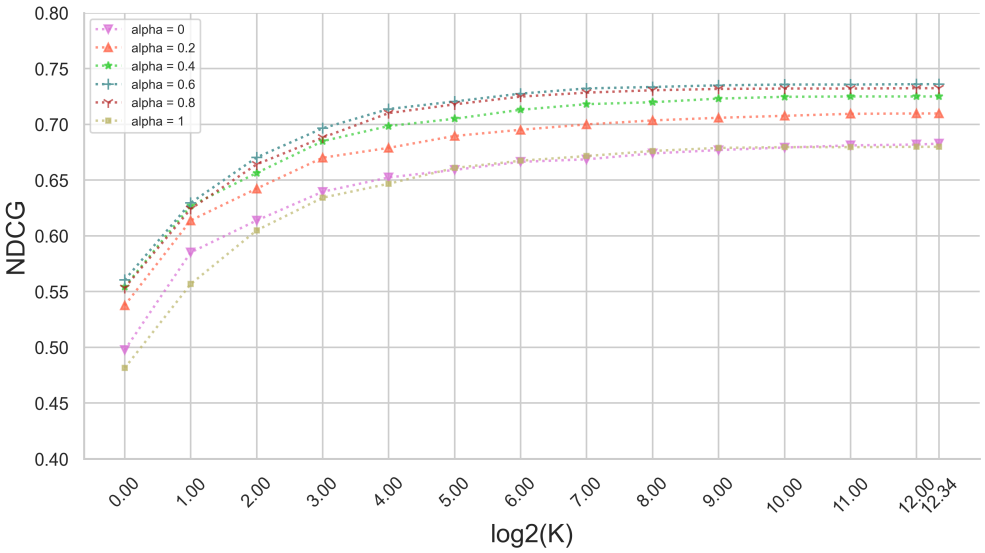

FIGURE 3.5: Evaluation of fusion with standardization by varying the value of k

As before the behavior of varying k is the same for all α. The best result is $\alpha = 0.6$, and again the worst performance are when $\alpha \in \{0, 1\}$ since even using standardization the order of documents for the individual method remain unchanged.

Now I want to check which of these two method is better, to do so I decided to print two curves, one with standardization and one without keeping k fixed. Here are present the result that I achieved.

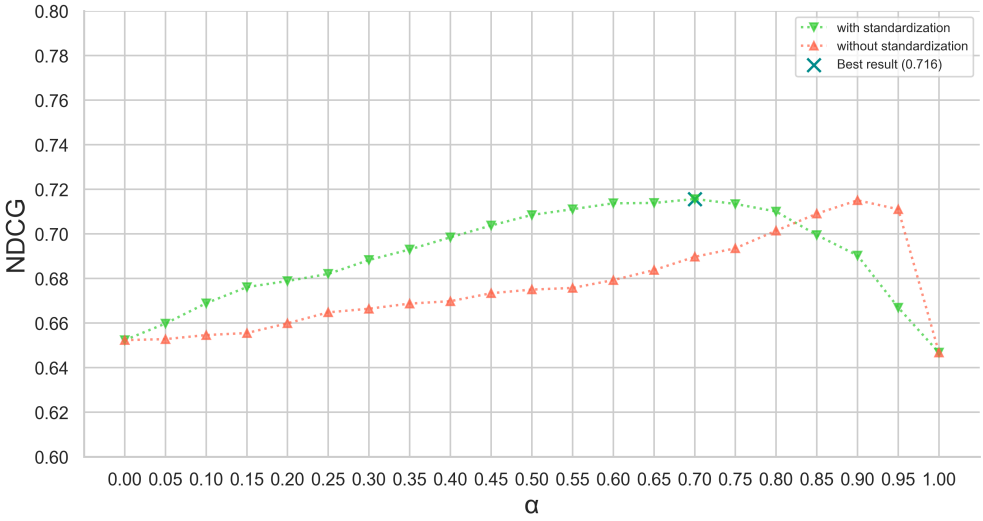

FIGURE 3.6: Comparison between method with and without standardization with $k = 16$

From this I know that the best result is obtained using standardization, even if the curve without standardization achieves almost the same result with an $\alpha = 0.9$, this is probably due to the fact that doing so gives more weight to the sentence-transformer (as its scores have lower weight) achieving the same performance (and scores) of the method with standardization.
I think that the fusion with standardization is the right fusion to be made because in this way it also gives a more marked meaning to the hyper-parameter (in this way $\alpha = 0.5$ means giving the same weight to both methods). Moreover, since the BM25 is not limited above and the sentence-transformer is limited in the set $[-1, 1]$ with standardization it is possible to give a greater emphasis to one of the two methods, being that BM25 doesn't use contextual information and instead sentence-transformer uses it.