Textual Information Retrieval
Machine Learning for Natural Language Processing 2022

Hamdi BEL HADJ HASSINE
3A-Data Science Applied Statistics
hamdi.belhadjhassine@ensae.fr

Georges SARR
3A-Data Science Applied Statistics
georges.sarr@ensae.fr
georgesmbissanes@gmail.com

Abstract

The quality of a search engine depends on the textual information retrieval model implemented. The underlying models generally belong to one the following family of retrieval methods: set theoretic methods (Boolean model), algebraic methods (Vector Space model (VSM), Latent Semantic Indexing (LSI)), probabilistic methods (Query Likelihood Model (QueryLklhd), Binary Independence Model (BIM) and its three extensions Two-Poisson, BM11 and BM25) and semantic methods using embeddings and deep learning. We implemented Vector Space Model with TF-IDF weights (VSM) and with embeddings (W2Vsm), along with LSI, QueryLklhd, BIM, BM11, BM25. We also implemented newer Deep Learning methods such as MatchPyramid (Pang et al., 2016) and BERT (Nogueira and Cho, 2019). The models were compared using the Mean Reciprocal Rank at 10 (MRR@10) and MAP (Mean Average Precision). The experiments data is a subset of the MSMARCO[1] dataset loaded from huggingface API [2]. The implementation code is available on github[3].

1 Problem Framing and Models

The task at hand is to retrieve the relevant documents d_i for a given query q. To tackle this problem, as explained in (Manning et al., 2009), each model attributes a representation f_d and f_q for document d and query q respectively, along with a relevance score $r(d, q)$ that can be used to rank the documents

For VSM, a query q and a document d are represented by a vector of term weights $w_t^p = TF(t, p) \times IDF(t)$ with $p \in \{q, d\}$ and t a term belonging to the vocabulary, with f_d, f_q vectors of size $|T|$ the vocabulary length. The model W2Vsm takes f_p equal to the TF-IDF weighted average of the embeddings of the words in $p \in \{q, d\}$. The idea behind LSI model is to build a Singular Value Decomposition (SVD) of the word-document raw frequency matrix[4] $WD = USV$, then $f_q = U^T q$ and $f_d = (SV)_{i_d}$ are the projections of q and d on the latent space, where i_d is the column index for document d and U^T designates the transpose of U. The documents are ranked in a decreasing order of their relevance $r(d, q) = \text{cosine}(f_d, f_q)$. Regarding BIM, the documents and the query are considered as random vectors of terms to model the uncertainty about the query understanding and the relevance of the documents for a query. The objective is to estimate $r(d, q) = P(R = 1|d, q)$ the probability that d is relevant for q, where $R \in \{0, 1\}$ is a random variable. Under some assumptions (found in (Manning et al., 2009) chapter 11-3 The Binary Independence Mode), this boils down to $r(d, q) = \sum_{t_q} \frac{1}{2} \log \left[\frac{|D|}{N_{t_q}}\right]$ where t_q is a term appearing in q, N_{t_q} is the number of documents where t_q appears, $|D|$ is the number of documents in the collection. The extensions of BIM adjust the relevance with some weights to account for term frequencies (Two Poisson), term frequencies and document length (BM11), term frequencies and dampened document length (BM25) known to be the best among the three. The approach of QueryLklhd model is to estimate the probability $P(q|d) = r(d, q)$ of a q to be generated from a document d, hence the name query likelihood. To do so, a language model is estimated for each document. We implemented the unigram model for which $P(t|d) = \frac{n_{t,d}}{n_d}$, where $n_{t,d}$ is the total number of occurences (raw frequency) of t in document d, and n_d is the sum of the $n_{t',d}$ s, and it is assumed that $P(q|d) = \prod_{t_q=t_d} P(t_q|d)$. Besides, the MatchPyramid model (Pang et al., 2016) is a deep learning model that first computes in a forward pass an interaction matrix of a pair (*q*,*d*) (or batch of pairs) using term embeddings. This matrix is scanned over to build a convolution layer followed by a max pooling layer followed by successive convolution layer, max pooling layer pairs and finally a multilayer perceptron with softmax output [5] estimates the probability of relevance of d for q, then the binary cross-entropy is computed and propagated backwards to fine tune the weights. Once the model is trained and validated, it can be used to rerank the top K documents ranked initially by one of the above models, in a decreasing order of their probability of relevance

Another method that recently gained significant popularity is the use of a general purpose attention-based

[1] https://microsoft.github.io/MSMARCO-Question-Answering/

[2] https://huggingface.co/datasets/ms_marco

[3] https://github.com/geosarr/search-engine

[4] Its coefficients are the number of occurrence of a term in a document, terms in rows and documents in columns

[5] we used a sigmoid instead

deep learning model such as BERT and fine-tuning it for information retrieval. To do this we adopt the approach proposed in (Nogueira and Cho, 2019) which is based on (Devlin et al., 2018): The input is constructed by concatenating a classification token ([CLS]), a tokenized query, a separator token, and a tokenized document (truncated or padded to a specific length). Additional learned embeddings are then added to represent the two parts of the input (d,q) and the positions of the tokens as illustrated in Fig. 6. We use a pretrained BERT model to which a classification layer is appended, taking as input the final hidden state of the [CLS] token and outputting the relevance scores. In our implementation to accelerate the model training and inference we used a lighter version of BERT (DistilBERT) and we used it to rerank the top 50 results returned by BM25 (ranking the whole corpus with DistilBERT would be slow).

2 Experiments Protocol

A small subset of MSMARCO made of 9650 english question answering units is preprocessed to build the collection of documents (a document is an answer), the questions are the queries. An inverted index[6] is constructed from the documents, and relevance judgments are given in the dataset[7]. The key to build the search engine is to compute as many useful features as possible before querying: TF, IDF, unigram models, word embeddings[8], document embeddings. To speed up retrieval for W2Vsm, a document K-means clustering is performed[9]. The collection is made of more than 70 thousand documents and more than 100 thousand terms, so computing the SVD of WD for LSI was impossible on the environment provided by Google Colab. As an alternative, we randomly subsampled 1000 documents of the collection to build a so called subinverted index and worked with it for searching and benchmarking purpose. The metrics MRR@K for a model is the average, over $|Q|$ queries, of the inverse rank of the most relevant document for each query among the K top retrieved documents[10]. We computed MAP also, which is the average, over $|Q|$ queries, of the rate of relevant documents among the top K_q retrieved documents[11]. The higher these metrics are, the better the model.

[6] it associates each term to the documents in which they appear, gives term occurrences, it is the basis of almost all the models

[7] We just add +1 to the relevance score: 1 means relevant, 2 means the most relevant, the remaining documents are assumed to be irrelevant, i.e score=0

[8] english google news word2vec model with 300-dimensional embeddings is used for searching purpose, but english Glove model with 50-dimensional embeddings is used to train the MatchPyramid model

[9] A query will be compared to the documents that belong to the clusters with centroids that are the most similar to the query.

[10] If the most relevant document does not appear among the top K, then the inverse rank is 0. We measured MRR@K with $K = 10$ and $|Q| = 100$

3 Results

In our implementation, there are many hyperparameters: the number of clusters for K-means, the words embeddings, the formula used to compute TF-IDFs, etc. We only focused on the hyperparameters k and b of BIM extensions given by

$$r(d,q) = \frac{1}{2}\sum_{t_q} \frac{n_{t_q,d}(k+1)}{n_{t_q,d} + k\frac{l_d}{l_{avg}}b + k(1-b)} \log\left[\frac{|D|}{N_{t_q}}\right]$$

where l_d is the number of distinct terms (document length) in d and l_{avg} is the average of the l_ds. We took a range of 10 values of k varying from 1 to 2 and 10 values of b from 0 to 1, Two poisson corresponds to the case $b = 0$ and $k \neq 0$, BM11 to $k \neq 0$ and $b = 1$ and BM25 to $k \neq 0$ and k between 0 and 1

best model	MRR@10	MAP
VSM	0.308	0.57
BIM	0.264	0.554
QueryLklhd	0.0	0.002
W2Vsm	0.225	0.356
LSI	0.005	0.009
Two poisson	0.34	0.555
BM11	0.347	0.571
BM25	0.358	0.572
BM25+BERT	0.456	0.527

Table 1: Performance of the best model for each category of model

The overall best category of classical models is BM25 with respect to both metrics. For MRR@10, BM25 with $k = 2$ and $b = 0.65$ is the best. For MAP, BM25 with $k = 1$ and $b = 0.75$ is the best. Reranking the top 50 BM25 results with a DistilBERT model improves its performance for MRR.

4 Discussion

The probabilistic models (especially the extensions of BIM) perform well compared to their counterparts. However implementation of QueryLklhd leads to many false positives[12] due the assumption $P(q|d) = \prod_{t_q=t_d} P(t_q|d)$. Furthermore the bad performance of LSI is due to the subsampling of documents. We point out that our work did not focus on the search duration for each model, but throughout the experiments, we noticed that VSM , W2Vsm (without preclustering the documents) take longer to retrieve results than the rest. We also observe that new deep learning models such as BERT outperform classical models wrt MRR metric.

[11] K_q is the number of relevance judgments for q

[12] Some smoothing methods exist (Laplace, Jelinek-Mercer or Dirichlet smoothing) to account for more information

References

Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze. 2009. An introduction to information retrieval. *https://nlp.stanford.edu/IR-book/pdf/irbookonlinereading.pdf*.

Liang Pang, Yanyan Lan, Jiafeng Guo, Jun Xu, Shengxian Wan, and Xueqi Cheng. 2016. Text matching as image recognition. *https://arxiv.org/abs/1602.06359*.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. BERT: pre-training of deep bidirectional transformers for language understanding. *CoRR*, abs/1810.04805.

Rodrigo Nogueira and Kyunghyun Cho. 2019. Passage re-ranking with BERT. *CoRR*, abs/1901.04085.

A Descriptive statistics on the dataset

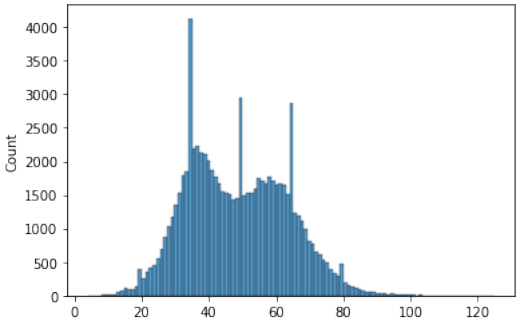

Figure 1: Distribution of the document lengths l_ds, i.e number of distinct word in a document, there are roughly 79000 documents in the collection

stats	doc freq	average raw freq
mean	29	1
median	2	1
std	571	0

Table 2: The mean, median and standard deviation of the terms document frequencies (doc freq) and terms average raw frequencies (average raw freq): on average a term appears once in a document and appears in 29 documents. Half of the terms appear in less than 2 documents, That is because we do not tokenize the terms (like stemming), we instead applied a simple rule based preprocessing (to speed up inverted indexing), doing so gives many rare terms

B Optimality conditions

B.1 Best number of clusters

In order to choose the number of clusters for K-means, we allowed training a range of K-means model, compute the corresponding penalized Inertia given by

$$\text{Penalised Inertia}(K) = \text{Inertia}(K) - \text{penalty} \times K$$

The optimal K minimizes this penalized Inertia

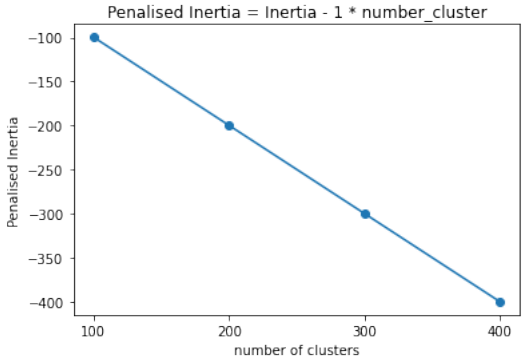

Figure 2: penalized inertia with penalty=1, optimal $K = 400$

B.2 Best number of singular values

To speed up the retrieval a little bit for LSI, one chooses the K largest singular values of the word-document occurrences matrix WD instead and project the query vector on the space engendered by the corresponding latent vectors, not computing $\cos(U'q, (SV)_{i_d})$ but computing $\cos(U'_K q, (S_K V_K)_{i_d})$ to rank the documents, where $U_K \in R^{|T| \times K}$, $S_K \in R^{K \times K}$, $V_K \in R^{K \times |D|}$.

The optimal K is computed by maximizing with respect to K the penalized sum of singular values given by:

$$PSSV(K) = \sum_{k=1}^{K} \sigma_k - \text{penalty} \times K$$

where σ_k is the k^{th} largest singular value of WD

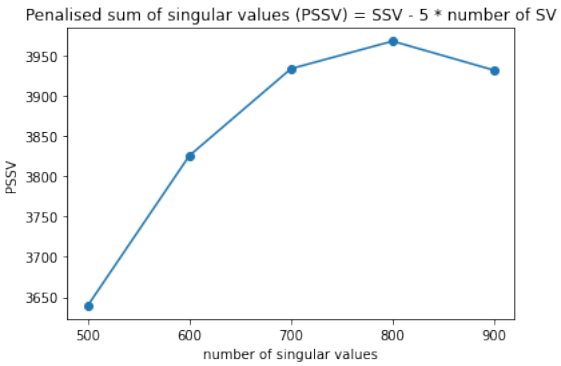

Figure 3: PSSV with penalty=5, optimal $K = 800$

C MatchPyramid model

The architecture of the model is given in the image below, we implemented it with pytorch without fine tuning the hyperparameters (convolution filters, kernel size, etc). We do not exactly follow the authors choices, but were inspired by their work. An instance (q, d) has

a prediction=1 if MatchPyramid outputs a probability ¿ threshold=0.7. The cross entropy evolution over the epochs is given below. The average accuracy on the validation set is 0.67 and on the training set is 1, on the the test set the loss is 0.69 and the accuracy is equal to 0.67. Something looks off with the current model because the accuracy on training and validation sets are equal, a thorougher examination should be conducted to get a stable and high performance model.

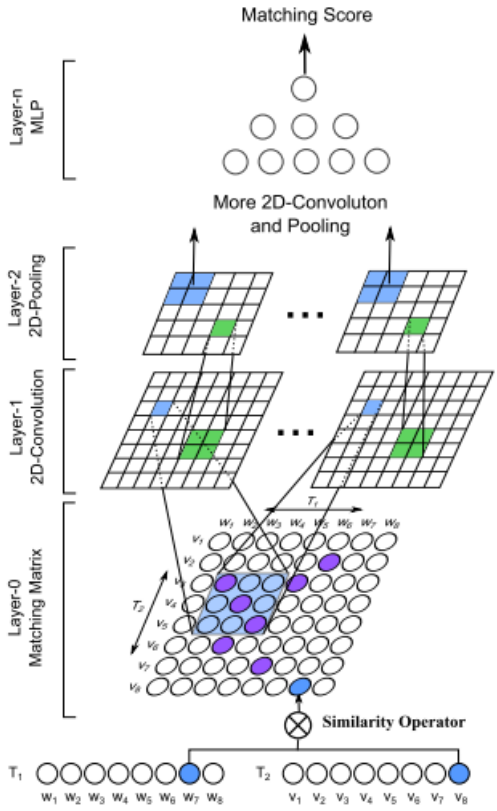

Figure 4: MatchPyramid architecture, source (Pang et al., 2016)

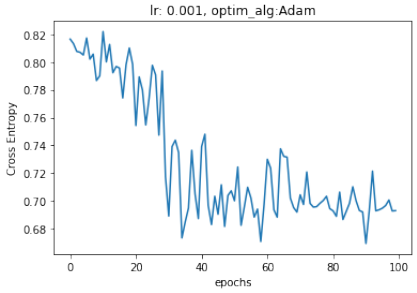

Figure 5: MatchPyramid Loss function variation against the epochs, with learning rate=0.001, optimizer Adam

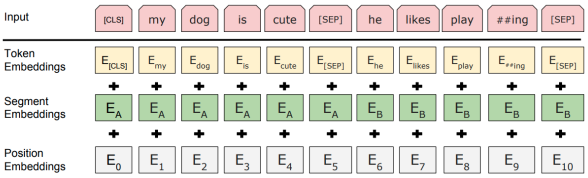

Figure 6: BERT input representation. Segment A is replaced by the query and segment B by the document text