N. Boukhatem, D. Buscaldi, L. Liberti, Empirical comparison of semantic similarity measures for technical question answering, in S. Chiusano et al. (eds.), Proceedings of the 26th European Conference on Advances in Databases and Information Systems, accepted.

Empirical comparison of semantic similarity measures for technical question answering

No Author Given

No Institute Given

Abstract. We consider the task of looking for the answer to a given user question by means of identifying the most relevant document in a technical knowledge base. We briefly introduce the NLP fields related to this task, then discuss what we think are the most promising methods to accomplish the task. The main aim of the paper is to benchmark the chosen methods on two different knowledge bases (one proprietary, one public). Every document in each KB consists of a title and a text describing a solution to a technical problem. Our tests point out that the best method for the task at hand is the use on Sentence Transformers, a deep learning based method using pre-trained language models.

1 Introduction

This paper concerns a variant of a well known Natural Language Processing (NLP) task, namely Question Answering (QA) [8, Ch. 25]. QA aims at automatically answering questions posed by humans in natural language (NL). The variant we are interested in is actually a restriction of QA: our output is a relevant document in a Knowledge Base (KB) instead of an answer written in NL.

The reason why we look at this variant is that it represents an important need in industrial contexts. Typically, in our scenario of interest the "user" who asks the question may be an employee of the firm, or a technically skilled client. This scenario is very different from those leading to open-text QA systems yielding NL answers, where the user may be any individual. In the latter case, users may employ more informal language to phrase their questions, and will expect answers in NL.

In our case, we expect the user to pose more precise questions, and, what's more important, we know that he or she will accept the pointer to documents containing an explanation as a valid answer. This effectively sets our task at the intersection of QA and Document Retrieval (DR) [9]. We note, however, that in DR the user queries need not be cast in NL. For later reference, we name our task of interest Natural Language Document Retrieval (NLDR). Methods that can be used to address NLDR have to match the meaning of the user question to relevant semantic indicators in the documents of the KB. This establishes a connection between NLDR and semantic similarity measures and techniques [4]. All NL tasks may either refer to a general language setting, or to a specific language domain [5, 11]. The present research focuses on specific language domains: our motivation stems from the interest of a service firm with several industrial clients: each client is considered a specific domain.

The contribution of the paper is a computational comparison of known methods that can address the NLDR task, based on popular performance measures, on two KBs: a proprietary one belonging to the service firm that motivates this work, and the public IBM TechQA dataset [3]. In the rest of this section we briefly survey the fields of NLP that are relevant to the NLDR task, from the point of view of the methods we benchmark. In the rest of the paper we review the benchmarked methods, and we comment our computational results.

1.1 Document Retrieval

Document retrieval methods are usually structured around a user query (which may be expressed in either formal or NL) and a KB (database, corpus, graph. . .). The goal is to return the KB entry that is the most relevant to the query. The relevance depends on the satisfaction of the user's information need. The document ranking can be obtained in different ways: using word frequency and co-occurrence (e.g. [14]); using several independent syntactical statistics and co-occurrence measures (e.g. [2]); using a weighted variant of TF-IDF called BM25F (e.g. [12]); using sentence transformers (ST) to compare the given query with sentences in the corpus (e.g. [13]).

1.2 Technical Question Answering

By *Technical* QA (TQA) we mean QA over a restricted domain, where user information needs are predominantly oriented to solving technical issues. The interest about these systems has been growing in recent years, as testified by the proliferation of technical forums, some dedicated to developers, such as Stack Exchange[1], while others are related to a particular product or service. Early TQA systems were based on syntactical analysis of sentences, possibly with some elementary form of logical entailment, complemented by semantics stored in a specific-domain ontology [11]. More recently, ST have been used in TQA in [15], where an elaborate pipeline was used to train a neural network to match questions to answers, using training sets constructed from specific domains. In [16], the authors address the difficulty of forming large enough training sets for TQA related tasks. They propose a system relying on a general-purpose QA training set, before applying a transfer learning techniques using the IBM TechQA training set [3].

A TQA system can also be seen as an evolution towards an automatization of a classic technical support system, relieving technical experts from the burden of browsing the documentation necessary to answer users questions. A standard situation in a classic support system can be described as follows: a user encounters an issue with its system, gets in touch with the support team and describes

[1] `https://data.stackexchange.com/`

the situation he or she is facing. The contact can occur by phone, by email or by filling a form directly in the support system.

The user usually tries to describe the situation to help the support agent identify the source of the issue and how to fix it, for example : *"Hello, I've been struggling all day with my internet connection without any improvement. Is it possible to help me please?"*

Depending on the root cause, issues may need to be escalated to qualified agents, investigated by means of technical user guides, and/or compared to known issues. Finding a solution often requires time and effort. Each issue and the corresponding solution is documented, and the information is archived for later reference. The documentation process helps creating the set of documents (or *corpus*) C that will be considered the knowledge base of a TQA system.

Typical users of TQA systems (usually clients or support agents) describe their issues in NL by typing a query Q. The system then compares Q to the documents in the set C so that the most fitting answer (or set K of answers) can be retrieved. If an answer is found, it is returned to the user, who can rate its pertinence by providing a score. In the above example, a typical TQA system might infer issues in the internet connection because of the words *"struggling"*, *"internet"* and *"connection"*. This would lead the system to retrieve a set of documents about internet connection issues.

1.3 Semantic similarity measures

Semantic similarity measures are functions mapping sentence pairs to a similarity measure (usually in $[0, 1]$), by means of semantic considerations. We refer the reader to the recent comprehensive survey [4]. In particular, with reference to [4, Fig. 1], the methods we benchmark are mostly based on edge counting (e.g. WordNet information), information content (e.g. word frequency), transformer models (based on BERT).

2 Compared methods

We chose a set of semantic similarity methods that cover the most important models for the semantic representation of documents: keyword-based similarity methods represent the content of a document by means of selected words, which are matched to the user question. Methods such as BM25, which is at the core of Whoosh and Elasticsearch, rely instead on the fact that all words in a document have some degree of importance. The document itself is represented by a (sparse) vector in the space of words. Finally, ST take advantage from deep learning to obtain a semantic representation of the document as dense vectors.

The TQA methods we test are organized around two pieces of input: Q, which represents a question, and C, which represents a corpus. The representation could be based on words, vectors, or the output of an artificial neural network. The general algorithmic scheme is two-phase: pre-processing (on C) and on-the-fly (on Q). The output is usually a ranked set of documents from C.

2.1 Keyword extraction based methods

In this section we summarize two of the methods we benchmark, where the representation of Q and C is based on keyword extraction (KE). The first phase (pre-processing) extracts keywords from C. The second phase (on-the-fly) extracts them from Q. The methods then match keywords from Q with those from C in order to obtain relevant documents. The difference between the methods is the computation of the weights assigned to keywords from C.

Algorithm 1 Weighting method 1

```
Corpus_KW, Tokens_L, Synonyms_L, Stemmed_Tokens, Scores, Orders : List
for D document in C corpus do
    Corpus_KW.append(KW_extraction(D[“title”]))
Tokens_L ← (Tokenize(Q))
Remove Stopwords from Tokens_L
for T Token in Tokens_L do
    Synonyms_L.append(Syn(T))
    for X word in Synonyms_L do
        Stemmed_Tokens.append(Stem(X))
    Stemmed_Tokens.append(Stem(T))
for i from 0 to len(Corpus_KW) do
    for k Word in Corpus_KW[i] do
        for w Word in Stemmed_Tokens do
            if w == k then
                Scores[i] ← Scores[i] + 1
    Scores[i] ← Scores[i] / len(Corpus_KW[i])
Return the documents of C with the highest scores
```

In the first method, the pre-processing phase scans C, and for each document D in C extracts three keywords from the title and three from the content of the document using a KE algorithm. Each keyword is simply assigned a unit weight. During the on-fly-phase, Q is cleaned from stopwords and tokenized. Each token is stemmed and listed along with its synonyms found in WordNet [7]. Each time a token (or one of its synonyms) is found we increment the document score by the token weight. We divide the score by the number of the document keywords to obtain the final score of the document. The documents with maximum score value are returned to the user.

In the second method, the pre-processing phase scans C, and for each document D in C extracts three keywords from the title, which are given a weight of 2, and three other keywords from the text, which are given a weight of 1, using a KE algorithm. Identical keywords have their weights summed. The three keywords with highest weight values are selected. The on-the-fly phase Q is the same as for the first method. After finding keywords, for both methods we form a list of all corpus documents containing at least one keyword from Q. Documents are then ranked by decreasing keyword weight sums.

Algorithm 2 Weighting method 2

```
Document_KW, Corpus_KW, Query_Words, Tokens_L, Synonyms_L, Stemmed_Tokens, Scores, Orders : List
Title_Weight = 2
Body_Weight = 1
for D document in C corpus do
    // If an element already exists, weights are summed
    Corpus_KW.append(KW_extraction(D["title"]), Title_Weight)
    Corpus_KW.append(KW_extraction(D["body"]),Body_Weight)
    Corpus_KW.append(Max(Document KW,3))
Tokens_L.append(Split(Q))
Remove Stopwords from Tokens_L
for T Token in Tokens_L do
    Synonyms_L.append(Syn(T))
    for X word in Synonyms_L do
        Stemmed_Tokens.append(Stem(X))
    Stemmed_Tokens.append(Stem(T))
for i from 0 to len(Corpus_KW) do
    for k,l Word,weight in Corpus_KW[i] do
        for w Word in Stemmed_Tokens do
            if w == k then
                Scores[i] = Scores[i] + 1
    Scores[i] = Scores[i] / len(Corpus_KW[i])
Return the documents of C with the highest scores
```

For KE purposes we used two well-known systems, namely Rapid Automatic Keyword Extraction (RAKE) [14] and Yet Another Keyword Extraction (YAKE) [2]. Two unsupervised automatic KE methods that are domain, corpus, and language independent.

2.2 Word vector based methods

Whoosh is an open-source Python search engine library for indexing and searching text based on the BM25F algorithm [12]. Whoosh is a native Python alternative to Lucene [1], which strongly inspired its development. Whoosh's index is *fielded*: the user defines a set $\mathsf{fields}(D)$ of fields that represent the structure and content of the document D into the index. For instance, a classic fields referring to scientific papers involve indexing title, abstract and body into different fields. Keywords are selected by means of *analyzers*, which usually provide language-dependent stemming algorithms and stop-word lists. Given a query $Q = \{t_1^{(q)}, \dots, t_n^{(q)}\}$ and a document $D = \{t_1^{(d)}, \dots, t_m^{(d)}\}$, the matching score between them is calculated according to BM25F:

$$\mathsf{BM25F}(D, Q) = \sum_{t \in Q \cap D} \frac{\mathsf{TF}(t, D)}{k_1 + \mathsf{TF}(t, D)} \mathsf{IDF}(t),$$

where t is a term shared by both the query and the document, k_1 a constant set at 1.2, and $\mathsf{TF}(t,D)$ is the *normalized term frequency*:

$$\mathsf{TF}(t,D) = \sum_{c\in\mathsf{fields}(D)} w_c \frac{\mathsf{freq}_c(t,D)}{1+b_c\frac{l_{(D,c)}}{\hat{l}_c}},$$

where $\mathsf{freq}_c(t,D)$ are the occurrences of the term t in the field c of document d, $l_{(D,c)}$ is the length of the field c in document D,and $\hat{l}_c$ is the average length of field c. Moreover,b_c is a field-dependant parameter, usually set at 0.75, and w_c is a boost factor (by default set at 1.0) that can be specified by the user at query time. The inverse document frequency $\mathsf{IDF}(t)$ is usually calculated as $\mathsf{IDF}(t) = \frac{N-df(t)+0.5}{\mathsf{DF}(t)+0.5}$, where N is the number of documents in the collection and $\mathsf{DF}(t)$ is the document frequency of term t .

ElasticSearch (ES) is another open-source search engine built on top of Lucene [1]. It allows users to store, index and search large sets of documents. Unlike Whoosh, explicit mapping schemata are not necessary. ES supports structured queries, full text queries, and complex queries that combine the two. Structured queries are similar to SQL queries, while full-text queries find and return all documents that match the query, sorted by relevance. In addition to searching for individual terms, ES supports phrase searches, similarity searches, and prefix searches. ES, like Lucene [1], also uses the Okapi BM25 algorithm as a default ranking function for documents relevance in the search process. BM25 is a bag-of-words retrieval function that ranks documents based on the terms $q_1,\ldots,q_n$ of a query Q appearing in each document D, regardless of their proximity within the document and given by:

$$\mathsf{BM25}(D,Q) = \sum_{i=1}^{n} \mathsf{IDF}(q_i)\frac{\mathsf{F}(q_i,D)(k_1+1)}{\mathsf{F}(q_i,D)+k_1(1-b+b\frac{|D|}{\mathsf{avgdl}})},$$

where $\mathsf{F}(q_i,D)$ is q_i's term frequency in the document D, $|D|$ is the length of the document D in words, and avgdl is the average document length in the text collection from which documents are drawn. k_1 and b are free parameters.

2.3 Deep learning based methods

ST are a Python framework for calculating sentence, paragraph and image embeddings. It is an implementation of Sentence-BERT (SBERT) [13], a modification of the pretrained BERT network [6] that uses Siamese network structures to derive semantically meaningful sentence embeddings. SBERT uses a pre-trained BERT and RoBERTa networks, and adds a pooling operation to their output in order to derive a fixed sized sentence embedding. It can be described as a document processing method of mapping sentences to real-valued vectors such that sentences with similar meanings are close in vector space.

To achieve this goal, a semantic representation of a sentence is built by adapting a transformer model in a Siamese architecture: sentences are processed pair by pair, with the same neural network structure. A vector is produced by each network, on which a distance is calculated. The loss function for the complete network consists in minimizing the distance between semantically similar sentences, and maximizing the distance between semantically distant sentences.

The obtained word embeddings can be compared using similarity measurements, such as cosine similarity. Such scores can be exploited in different NLP tasks, including information and document retrieval. Cosine similarity is the cosine of the angle between two word vectors E and V (the dot product of the two vectors divided by the product of their lengths):

$$\cos(E, V) = \frac{E \cdot V}{\|E\|_2 \|V\|_2} = \frac{\sum_{i=1}^{n} E_i V_i}{\sqrt{\sum_{i=1}^{n} (E_i)^2} \sqrt{\sum_{i=1}^{n} (V_i)^2}}.$$

The embeddings of the dataset are constructed using two different pre-trained models (RoBERTa and MiniLM-L6), and then stored in D. Cosine similarity ranges in $[0, 1]$, with 0 indicating dissimilarity and 1 identity. Cosine similarity is used in order to find the 3 most similar documents to return to the user.

The RoBERTa model [10] is based on Google's BERT model [6]. It has different key hyperparameters and training data size. It maps sentences and paragraphs to a 1024-dimensional vector space (with dense word vectors). It performs well in tasks like clustering or semantic search.

The MiniLM-L6 model is a sentence-transformer model mapping sentences and paragraphs to a 384-dimensional vector space (with dense word vectors). It can be used as a sentence and short paragraph encoder to capture the semantic information for information retrieval, clustering or sentence similarity tasks.

3 Computational comparison

3.1 Evaluation data

The different methods were evaluated using two different English datasets: one small dataset from the sponsoring company, and a large dataset from IBM.

The company's dataset, which we refer to as the "OT dataset" (from the name of the company), is composed of 76 technical documents, considered as a knowledge base, indicating how to fix common and less-common issues faced by users. In addition, it contains a test set composed of 35 questions, with 19 answerable and 16 non-answerable questions. The test set was collected from interactions between the company's clients and the TQA system we put in place at the company's site. Documents in the dataset, which consist of a title and a body, describe in which situation the solution was applied. The questions consist of a query, a boolean truth value determining if the question is answerable or not, and the ID of the corresponding answer if it is available.

The other dataset is called "TechQA" [3]. It consists of a collection of 28 481 technical documents, called *technotes*, that address specific technical questions, and of annotated questions with answers in the collection. Questions are divided into two sets, which the TechQA documentation describes as “training set”, containing 450 answerable and 150 unanswerable questions, and “development set”, containing 160 answerable and 150 unanswerable questions (by unanswerable we mean that there is no document in the collection containing an answer to the question). We chose the training set for testing purposes in this computational evaluation as the methods that we tested do not require training. Each question consists of a title and a body, and the answerable ones are paired with a set of answers consisting in the technote ID and the start and end offset of the answer within the document referenced by the ID.

3.2 Evaluation measures

The measures selected for the evaluation of the methods are: *Precision*, *Recall*, *F1-Score* and *Mean Reciprocal Rank (MRR)*. We considered for the set Q only the answerable questions, to highlight the position in which the pertinent document is returned when an answer is available. In the following, we refer to this constrained measure as MRR_a.

3.3 Results

The results obtained are detailed in Table 1. It can be noted that, for the keyword extraction methods, RAKE is less accurate than YAKE: in particular, we observed that it is often unable to find appropriate keywords, and, conversely, repeats identified special characters (like “\\” or “&”) as keywords. Both methods, however, were ineffective on the QA dataset, showing that these keyword extraction methods are not particularly fit to extract the technical keywords in that dataset.

Applying the evaluation methods to document titles (instead of contents) turned out to produce better results both with KE and ST methods. While titles contain less information, it appears to be more valuable or better exploitable (by current methods) than the information existing in the content.

ES gave good results on large datasets like TechQA, while Whoosh was better suited to smaller datasets like OT. The reason is related to the indexing technique of both methods: while Whoosh only indexes the two fields “title” and “content”, ES makes use of all the fields of the document, including metadata.

ST gave the best results for both datasets, with a very high MRR score in each case. We can add that the use MiniLM model is less time-consuming (about 3 times less) with a vector size 3 times smaller compared to the use of the RoBERTa model but with similar results. MiniLM even achieved better results on TechQA, as well as a higher MRR score on OT.

We carried out an analysis of the results obtained by ST, focusing on questions for which the answer was either right but with a low score or wrong with a high score. For the ST-MiniLM (T) model, we identified 49 questions of the 1st type and 11 questions of the 2nd type. We observed that in the second case, many questions had acronyms and uncommon words so we carried out an evaluation to compare the number of words of the questions to the number of tokens

Table 1. Results obtained for the chosen evaluation methods, on the OT and TechQA datasets. When present, the labels (C) and (T) indicate that, respectively, only the content and the title have been indexed. For RAKE and YAKE, the -w and +w suffixes indicate the unweighted and weighted versions, respectively.

Method	OT Dataset				TechQA Dataset			
	Precision	Recall	F1-Score	MRR_a	Precision	Recall	F1-Score	MRR_a
RAKE-w (C)	12.86%	23.68%	16.66%	0.175	0.16%	0.22%	0.19%	0.001
RAKE-w (T)	17.14%	31.58%	22.22%	0.254	0.5%	0.67%	0.57%	0.001
RAKE+w	25.71%	47.37%	33.33%	0.228	0.22%	24.17%	0.44%	0.214
YAKE-w (C)	22.86%	42.11%	29.63%	0.333	1.78%	1.33%	1.52%	0.014
YAKE-w (T)	**45.71%**	**84.21%**	**59.26%**	**0.781**	0.5%	0.67%	0.57%	0.001
YAKE+w	42.86%	78.94%	55.55%	0.658	0.5%	0.67%	0.57%	0.006
ElasticSearch	11.76%	21.05%	15.09%	0.211	**26.83%**	**35.78%**	**30.67%**	**0.358**
Whoosh	31.43%	47.37%	37.79%	0.474	0.83%	1.02%	0.92%	0.011
ST-RoBERTa (C)	31.42%	57.89%	40.74%	0.697	12.83%	14.11%	14.67%	0.679
ST-RoBERTa (T)	**51.43%**	**94.74%**	**66.67%**	0.917	21.83%	29.11%	24.95%	**0.917**
ST-MiniLM (C)	37.14%	68.42%	48.15%	0.923	28.5%	38%	32.57%	0.812
ST-MiniLM (T)	**51.43%**	**94.74%**	**66.67%**	**0.963**	**33.83%**	**45.11%**	**38.66%**	0.889

extracted by the SBERT tokenizer. We could observe that the ratio token/words is higher in questions of the 2nd type (1.67) than for questions of the 1st type (1.53), which is also lower than the ratio for the other questions (1.58). This proved our intuition and it also shows that the SBERT model is less effective when dealing with words that are not in the vocabulary and it has to resort to sub-word token to build a semantic representation of the full word.

4 Conclusion and further works

In this paper, we presented an empirical comparison of various semantic similarity measures, based on both sparse and dense vector representations, on the technical QA task. The results confirm for this task the general behaviour that keyword-based and classic models may be useful in small scenarios but they are not particularly useful for large and complex corpora. For the dense representations based on neural models, we observed that the results obtained using only the title information exceed by a large margin those obtained with the content. This may imply that further research on the representation of larger texts is required and SBERT semantic similarity measures are meaningful only at sentence level. We found out also that SBERT models have some problems with the acronyms and rare words that are characteristics of the TechQA task, and their results are worse when these words are not in the dictionary and they have to recur to sub-word tokens.

References

1. Białecki, A., Muir, R., Ingersoll, G.: Apache Lucene 4. In: Proceedings of the SIGIR 2012 Workshop on Open Source Information Retrieval. pp. 17–24 (08 2012)
2. Campos, R., Mangaravite, V., Pasquali, A., Jorge, A., Nunes, C., Jatowt, A.: Yake! keyword extraction from single documents using multiple local features. Information Sciences 509, 257–289 (2020)
3. Castelli, V., Chakravarti, R., Dana, S., Ferritto, A., Florian, R., McCarley, S., Pendus, C., Franz, M., Garg, D., Khandelwal, D., McCawley, M., Nasr, M., Pan, L., Pitrelli, J., Pujar, S., Roukos, S., Sakrajda, A., Sil, A., Uceda-Sosa, R., Ward, T., Zhang, R.: The TechQA dataset. In: Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics. pp. 1269–1278. ACL (2020)
4. Chandrasekaran, D., Mago, V.: Evolution of semantic similarity — a survey. ACM Computing Surveys 54(2), Art. 41 (2021)
5. Cimiano, P., Unger, C., McCrae, J.: Ontology-Based Interpretation of Natural Language. Morgan & Claypool, San Rafael, CA (2014)
6. Devlin, J., Chang, M.W., Lee, K., Toutanova, K.: BERT: Pre-training of deep bidirectional transformers for language understanding. In: Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1. pp. 4171–4186. ACL, Minneapolis, Minnesota (2019)
7. Fellbaum, C.: WordNet: An Electronic Lexical Database. Bradford Books (1998)
8. Jurafsky, D., Martin, J.: Speech and Language Processing. Stanford University, Stanford (Draft 191016)
9. Kruschwitz, U.: Intelligent Document Retrieval. Springer, Dordrecht (2005)
10. Liu, Y., Ott, M., Goyal, N., Du, J., Joshi, M., Chen, D., Levy, O., Lewis, M., Zettlemoyer, L., Stoyanov, V.: Roberta: A robustly optimized BERT pretraining approach. CoRR (2019)
11. Mollá, D., José Luis Vicedo: Question answering in restricted domains: An overview. Computational Linguistics 33(1), 41–61 (2007)
12. Pérez-Agüera, J., Arroyo, J., Greenberg, J., Perez Iglesias, J., Fresno, V.: Using bm25f for semantic search. In: Proceedings of the 3rd international semantic search workshop. pp. 1–8 (2010)
13. Reimers, N., Gurevych, I.: Sentence-bert: Sentence embeddings using siamese bert-networks. In: Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing. ACL (2019)
14. Rose, S., Engel, D., Cramer, N., Cowley, W.: Automatic keyword extraction from individual documents. In: Berry, M., Kogan, J. (eds.) Text Mining: Applications and Theory, pp. 1–20. Wiley, Hoboken, NJ (2010)
15. Yu, W., Wu, L., Deng, Y., Mahindru, R., Zeng, Q., Guven, S., Jiang, M.: A technical question answering system with transfer learning. In: Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP): System Demonstrations. pp. 92–99. ACL (2020)
16. Yu, W., Wu, L., Deng, Y., Mahindru, R., Zeng, Q., Guven, S., Jiang, M.: Technical question answering across tasks and domains. In: Proceedings of the North-American chapter of the Association for Computational Linguistics: Human Language Technologies: Industry Papers. pp. 178–186. ACL (2021)