LSA Is not Dead: Improving Results of Domain-Specific Information Retrieval System Using Stack Overflow Questions Tags

Szymon Olewniczak[1,3][a], Julian Szymanski[1][b], Piotr Malak[2,3][c], Robert Komar[1,3][d] and Agnieszka Letowska[3]

[1]*Faculty of Electronics, Telecommunications and Informatics, Gdansk University of Technology, Gdansk, Poland*

[2]*Institute of Media and Information Science, University of Wroclal, Wroclaw, Poland*

[3]*Scalac sp. z o.o., Gdansk, Poland*

Keywords: Word Embeddings, Latent Semantic Analysis, Information Retrieval, Neural Information Retrieval.

Abstract: The paper presents the approach to using tags from Stack Overflow questions as a data source in the process of building domain-specific unsupervised term embeddings. Using a huge dataset of Stack Overflow posts, our solution employs the LSA algorithm to learn latent representations of information technology terms. The paper also presents the Teamy.ai system, currently developed by Scalac company, which serves as a platform that helps match IT project inquiries with potential candidates. The heart of the system is the information retrieval module that searches for the best-matching candidates according to the project requirements. In the paper, we used our pre-trained embeddings to enhance the search queries using the query expansion algorithm from the neural information retrieval domain. The proposed solution improves the precision of the retrieval compared to the basic variant without query expansion.

1 INTRODUCTION

The idea of combining neural networks with classical information retrieval algorithms has become more and more popular in recent years (Mitra et al., 2018). The spectacular success of neural networks in many natural language processing tasks (Deng and Liu, 2018), inspired search engine authors to apply those techniques to the information retrieval field. Currently, the neural networks-based solutions are not only used in upstream IR tasks such as named entity recognition (Li et al., 2020) or part-of-speech tagging (Chiche and Yitagesu, 2022), but they are also gaining popularity in many downstream problems, such as relevance matching (Guo et al., 2016; Mitra et al., 2017), ranking (Liu et al., 2009) and representation creating (Huang et al., 2013; Mitra et al., 2016), that until now were the domain of the classical machine learning algorithms.

One of the very promising topics in the domain of neural information retrieval is the usage of semantic term embeddings for document and query representations. The term embeddings, popularized by Miklov's word2vec (Mikolov et al., 2013) and then further improved in gloVe (Pennington et al., 2014) and fastText (Bojanowski et al., 2016), can be used in search engines to go beyond the exact matching between query and document terms and capture the similarity not only literally but also on the semantic level.

These popular pre-trained term embeddings serve well for general-purpose search engines, but may not give satisfactory results for domain-oriented information retrieval systems. In this paper, we present the Teamy.ai system which works in the specific realm of information technology where general term embeddings are not sufficient. To tackle the problem we demonstrate an alternative solution for training the domain-oriented latent term representations and prove its relevance to our system.

The paper makes the following contributions:

1. We present the Teamy.ai system which implements a domain-specific IR pipeline to match IT project prospects with potential candidates.
2. We propose a method of creating domain-specific

[a] https://orcid.org/0000-0002-9387-8546
[b] https://orcid.org/0000-0001-5029-6768
[c] https://orcid.org/0000-0002-8701-1831
[d] https://orcid.org/0000-0003-2657-1744

Olewniczak, S., Szymanski, J., Malak, P., Komar, R. and Letowska, A.
LSA Is not Dead: Improving Results of Domain-Specific Information Retrieval System Using Stack Overflow Questions Tags.
DOI: 10.5220/0012358400003636
Paper published under CC license (CC BY-NC-ND 4.0)
In *Proceedings of the 16th International Conference on Agents and Artificial Intelligence (ICAART 2024) - Volume 3*, pages 446-453
ISBN: 978-989-758-680-4; ISSN: 2184-433X
Proceedings Copyright © 2024 by SCITEPRESS – Science and Technology Publications, Lda.

IT-related terms embeddings using Stack Overflow question tags.

3. We extend the proposed IR baseline method with an additional Query Expansion step that improves the results of the system.
4. We evaluate the extended pipeline with the human-created gold standard. We show that proposed embeddings improve the IR system results.

The paper is organized as follows. Section 2 describes the most important aspects of the Teamy.ai system. Section 3 outlines the IR component that matches the candidates with the projects. In the next section, we provide a relevant theory for utilizing the term embeddings in the context of information retrieval. Section 5 presents the formal definition of the algorithm for calculating candidates' relevance. Section 6 describes the issues that we facing in the specific context of Teamy.ai and the problem with general-purpose embeddings. The next section describes our training procedure for domain-oriented embeddings based on Stack Overflow question tags. In Section 8 we describe the evaluation procedure for our IR system. The next section compares the results of our IR algorithm with the gold standard dataset. Finally, we conclude our results and present some ideas for the future.

2 TEAMY.AI

Teamy.ai is a web-based application that helps software houses match possible team members and projects. The main motivation behind the application is to facilitate the process of composing the project team, which can consist of many people from different countries, time zones, and with diverse skills.

There are two main objects in Teamy.ai. First is the candidate which represents a possible team member. The second is the project prospect which defines the project requirements that the team is built for.

Each candidate has one or many professions assigned (for example frontend, backend, tester). The possible professions are software house specific. Further, each profession consists of skills (for the backend profession this might be Java, PHP, Python, etc.). Every candidate ranks his or her knowledge for each skill (on a 0 to 10 scale, where 0 means no knowledge and 10 means an expert). Optionally he or she can also rank skill for enjoyment (from -10 to 10, where -10 means that he or she hates the technology, 0 is neutral, and 10 is the most positive). The example candidate object may looks like this:

```
{
  "professionRatings": {
    "frontend": {
      "angularjs": {
        "knowledge": 6,
        "enjoyment": 2
      },
      "reactjs": {
        "knowledge": 8,
        "enjoyment": 8
      }, ...
    },
    "quality_assurance": {...}
  }
}
```

Another important object of Teamy.ai is the project prospect which defines the skills required by the project. Since the project might require many candidates with different professions and skills, the skills are grouped in separate *need* objects. Each prospect can consist of one or more needs. Each *need* defines the required profession and skills. The skills are grouped in two sets: *mustHaveTechStack* and *niceToHaveTechStack* which defines respectively the skills that are required for the project and skills that might be useful but are less important. The example project prospect may look like this:

```
"needs": [
  {
    "profession": "frontend",
    "quantity": 4,
    "mustHaveTechStack": [
      "emberjs",
      "angular7",
      "javascript"
    ],
    "niceToHaveTechStack": [
      "typescript"
    ]
  }
]
```

3 CANDIDATES RETRIEVAL IN TEAMY.AI

Candidates matching problem can be considered an IR problem where the prospect *need* forms the query q over the set of all possible candidates that might be treated as documents in our IR system: D. The set of all possible terms that may appear in query and candidate profiles (often called dictionary) will be denoted as T. Our task is to find the ranked list of candidates that match best the given need.

MOST WIEDZY Downloaded from mostwiedzy.pl

3.1 Exact Matching Algorithm

The exact matching algorithm is the simplest Candidate Ranking algorithm that we developed in our retrieval component and it can serve as a good baseline for testing the query expansion. For each candidate on the list, we calculate the matching score using the following metric:

$$exact(d,q) = \frac{\sum_{i=1}^{|q|} w_i d_{q_i}}{\sum_{i=1}^{|q|} w_i} \quad (1)$$

where q is the set of must-have and nice-to-have skills defined in *need* plus the additional skills provided by the Query Expansion component. q_i is the ith skill in the query. w_i is the weight of the ith skill. For must-have skills, it equals 1.0 and for nice-to-have and expanded skills, it is defined by global hyper-parameters: *nice_to_have_factor* and *expand_factor*, which will be explained later. Finally, d_{q_i} is the candidate's knowledge for the q_i skill, normalized between 0 and 1.

After calculating the matching score for each candidate on the list, we return the top N results with the highest metric.

4 TERM EMBEDDINGS FOR IR

One of the promising idea for using the term embeddings in IR is based on the concept of query expansion. In this approach, we use term embeddings to extend the original query and then perform the classical exact matching algorithm between the extended query and the documents in the corpus. One of the possible metrics that allow us to calculate the candidate term t relevance to the query is based on the arithmetic average of cosines distances between the candidate and all terms in the original query (Roy et al., 2016):

$$rel_t(t,q) = \frac{1}{|q|} \sum_{t_q \in q} cos(\overrightarrow{v_t}, \overrightarrow{v_{t_q}}) \quad (2)$$

To perform the query expansion, we calculate the score for each term in our dictionary and then select n-top results.

We must also remember that all algorithms using term embeddings are as good, as good are the embeddings themselves. There are two primary kinds of similarity that the embeddings can capture. The first is *typical* similarity, while the second is *topical*. The embeddings that capture the *typical* similarity group together the words that are of a similar type. For example, in the context of information technology, this may mean that terms such as *Python* and *PHP* (which are both programming languages and hence are of the same type) may be closer to each other than *Python* and *Django* (the first is the programming language while the other is web framework). On the other hand, the topical similarity may consider the *Python* and *Django* closer to each other than *Python* and *PHP* since they are parts of the same programming ecosystem.

The type of similarity that is captured by different embeddings is determined by the way they are built. The models that represent documents in the training corpus as bag-of-words (bag-of-terms) such as LSA (Deerwester et al., 1990) or LDA (Blei et al., 2003) are closer to *topical* sense of similarity. On the other hand, the models that during training use small spans of text from the documents (such as word2vec, gloVe, or fastText) usually capture more the *typical* sense of similarity (Sahlgren, 2006). The decision on which embedding scheme we should use in our solution is problem-dependent. Usually, we should start from pre-trained embeddings and then switch to the custom solution when they are not satisfying.

5 IR ALGORITHM FOR TEAMY.AI

In this paper, we present the Teamy.ai IR algorithm which uses exact matching together with term embeddings Query Expansion. Formally, for each candidate, we calculate the relevance score:

$$rel(q,d) = exact(d, q \cup argmax_n(\{rel_t(t_1), \cdots, rel_t(t_{|T|-|q|}\})) \quad (3)$$

where $argmax_n(\cdots)$ returns n best matching terms from the terms' dictionary T, that are not part of the query q. Then the candidates list is sorted according to recalculated relevance.

The presented algorithm is parameterized by three parameters:

- $nice\_to\_have\_factor \in [0.0, 1.0]$ – defines w_i in the exact algorithm (1) to weight candidate's knowledge for nice to have skills,
- $expand\_factor \in [0.0, 1.0]$ – defines w_i in exact algorithm (1) to weight candidate's knowledge for skills returned by query expansion,
- $expand\_limit \in \mathbb{N}$, – defines the number of terms that we want to add to the query in the query expansion step.

MOST WIEDZY Downloaded from mostwiedzy.pl

6 TERM EMBEDDINGS FOR TEAMY.AI

There were two main challenges that we are facing in the Teamy.ai search engine. Firstly our dictionary is relatively small but domain-specific with many terms that are not commonly used. Additionally, the dictionary may vary between different Teamy.ai instances (not all software houses need all the terms). Secondly, the differences between different candidates can be quite small which makes it hard to rank them properly.

The Scalac company which is the creator and the first user of Teamy.ai defined a dictionary with $|T| = 900$ elements, grouped into 5 professions: Scala, Data, Frontend, DevOps, Quality Assurance, and Project Management.

Our initial attempt was to try preexisting embeddings: word2vec and fastText-wiki-news-300d-1M. Unfortunately, there were several problems with that approach. Firstly many domain-specific terms are missing in those general-purpose embeddings, partially because they generally ignore multi-word terms. The word2vec has only 419 out of 900 terms, while fastText has 399. Secondly, some ambiguous terms, such as *perl*, *make*, *ant*, or *apache* are not well represented since they have not only IT-related meanings. Thirdly, the pre-trained embeddings were trained at some point in history and do not reflect the latest trends in IT. Finally, the training procedure used in these embeddings captures more the typical notion of similarity where we are more interested in topical one.

Considering those challenges, we decided to create embeddings for our specific use case, which will incorporate as much domain knowledge as possible.

7 TRAINING STACK OVERFLOW EMBEDDINGS

The most important challenge in creating good embeddings is finding a proper dataset for the training. In our case of IT-related terms, there cannot be a better choice than Stack Overflow[1].

Stack Overflow is a successful question-answer portal where users can post IT-related questions. The community can answer the questions and additionally vote for the best questions and answers. From our perspective, there is also a very important mechanism that allows the posters to add tags to their questions. The tags reflect the topics that the question is related to (for example python, java, django, machine-learning, etc.).

Table 1: Some examples from dictionary-tags mapping. Some dictionary entries were mapped with a single tag, others with an entire group of tags.

Dictionary entry	Stack Overflow tag regexp
ActionScript	actionscript.*
Angular 2+	angular2.*
Bash	bash
Bootstrap	bootstrap[0-9.\-]*
C	(ansi-)?c
Classification algos	.*classification
DevOps	devops
ES6	ecmascript-6
Java	java[0-9\-]*

The anonymized Stack Overflow data can be downloaded from Internet Archive[2] in the form of XML dumps. For training our embeddings we used the dump publicized on 2022-03-07. The dump contained the questions asked between 2008-07-31 and 2022-03-06. The total number of questions in the dump was 22,306,171 with 62,706 unique tags.

The tags are a very useful resource of information about the *topical* similarity of the various IT-related terms. The tags that are used frequently together probably reflect the technologies that are usually used jointly and therefore the candidate that declares the knowledge of one of these technologies can also have some experience with the others.

With that hypothesis in mind, we decided to train our embeddings using only the question tags. Our first step in that way was the creation of mappings between the terms in our dictionary and the tags in Stack Overflow. In the result, we obtained key-value pairs, where the key was the technology name in the dictionary, while the value was the regular expression that matched the corresponding tags. Some sample mappings are presented in Table 1. Our mappings contain the tags for 822 out of 900 technologies from the dictionary. The missing values result from either very broad terms (like *Application Architecture* or *Code Quality*) or very rare and specific technologies (like *GitFusion* or *ISTQB*). Nevertheless, we considered it a very promising result.

The next step in building our embeddings was training data preparation. It was done by iterating through all the questions and selecting only the questions whose tags reflect the dictionary entries according to the mapping. Additionally, only the questions that have 2 or more dictionary entries were selected (the question with a single term doesn't provide any useful information). As a result, we constructed the

[1] https://stackoverflow.com

[2] https://archive.org/details/stackexchange

MOST WIEDZY Downloaded from mostwiedzy.pl

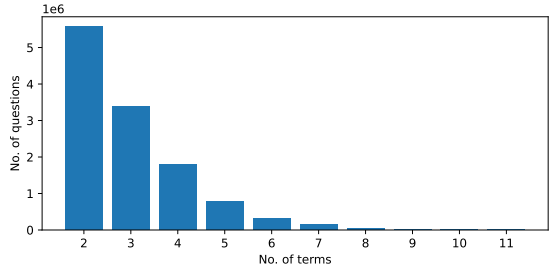

Figure 1: The distribution of dictionary entries (terms) per question in our training dataset. The most popular are the questions with 2 terms, then the popularity shrinks exponentially.

training dataset of 11,989,658 samples. The distribution of dictionary entries per question is presented in Figure 1.

Our final step was the training of the embeddings themselves. Since our terms related to questions are from their definition unordered, we represented our training samples as bag-of-terms vectors with a "1" for the terms present in the question and a "0" otherwise. Then we executed the LSA algorithm over our training dataset to find latent representations of our dictionary entries. We experimented with different vector sizes, where the 300-dimensional vectors proved to be most useful.

The LSA algorithm allows us to transform sparse observed vector spaces into latent vector spaces. In our case, we want to transform our terms represented by "in-document" vectors of size equal to the number of documents in our corpus (11,989,658) into 300-dimensional topical vectors. The LSA algorithm achieves this by performing singular value decomposition (SVD) over our input terms-documents matrix. In the result we obtain the term-topic matrix which represents the correlation between terms and topics according to terms' coexistence in the same documents. What is also important the generated topics are sorted by their importance. The most meaningful topics are on the left (they differentiate our terms the most) and when we are moving to the right we obtain fewer important ones. Considering this we usually take only n-top the most meaningful topics (300 in our case) and leave others, treating them as information noise.

In our training procedure, we used an LSI[3] Model from Gensim library[4] which is optimized for working with large datasets (Řehůřek, 2011). As a result of our training, we obtained 300-dimensional vectors for every entry in our dictionary, which represented the *topical* similarity between our terms.

[3]Latent Semantic Indexing (LSI) is just another name for LSA.

[4]https://radimrehurek.com/gensim/models/lsimodel.html

8 EVALUATION

The most important question for our research was: Does the proposed embedding scheme improve the results of our IR pipeline? Answering this question is not easy since the decision of who is the best candidate for the given project is subjective. Nevertheless, we asked Project Managers from Scalac to provide some real-life project prospects paired with candidates who in their opinion fit best for the job. As a result, we received 7 project prospects (one need per prospect) that represent some recent projects that were developed by the company.

For each prospect, the company's Project Managers assigned the ranked list of best-matching candidates from the current Scalac candidates database (which contains 206 candidates at the moment of writing). The list length varied across the projects between 2 and 5 candidates. We asked the PMs to create the gold standard in two steps. Firstly, they were asked to select the candidates for each prospect who in their opinion would be a good choice for the project. Secondly, they were asked to sort them in the order of relevance. In Table 2 we present the projects together with their Must Have and Nice to Have tech stacks.

Since we received ranked lists of candidates as a gold standard we decided to use Mean Average Precision (mAP) as our performance metric which is common in the IR field. The mAP is calculated using the following formula:

$$mAP(Q) = \frac{\sum_{i=1}^{|Q|} AveP(Q_i)}{|Q|} \tag{4}$$

where Q is the set of all queries in our evaluation dataset (7 in our case), and $AveP(q)$ is the average precision metric for the single query, defined as follows:

$$AveP(q) = \frac{1}{GTP} \sum_{k=1}^{n} P@k \cdot rel@k \tag{5}$$

where GTP refers to the total number of ground truth positives for the query, n is the number of documents in the IR system, $P@k$ is the precision calculated for the first top-k results and $rel@k$ is a boolean function that returns 1 when the kth element on the result list is present in the gold standard and 0 otherwise.

Usually, when the number of documents in the IR system is very huge, we are not calculating the exact $AveP$ for the given query but limit the result list to top n elements and refer to this metric as $AveP@n$. In our case, because the total number of candidates is not very big, we calculate the value of $AveP$ where n is

MOST WIEDZY Downloaded from mostwiedzy.pl

Table 2: Gold standard prospects. N means the number of selected candidates.

Id	Must Have Tech Stack	Nice to Have Tech Stack	n
1	Scala, Akka, Akka HTTP, Akka Clustering, Kafka, MySQL	HashiCorp Valut, Ansible	5
2	Kafka, Docker, Docker-compose, Scala, Akka HTTP	HTTP4S	2
3	Scala, Hadoop, Spark, Python, SQL, Unix, Oozie, Bash	Jenkins, Cats	4
4	Scala, MongoDB, Lift, Rest API	AWS, Kubernetes, Grafana, OpenTelemetry	3
5	Akka HTTP, Akka Streams, Akka Actors, Rust, Cats, Monix, Kotlin	gRPC	5
6	REST, Scala, Tapir, HTTP4S, Kafka	Event Sourcing	5
7	EmberJS, Angular 7, JavaScript	TypeScript	5

equal to the number of all candidates in the database: $n = |D|$.

9 RESULTS

We used the relevance function (3) to retrieve a sorted list of candidates for each of our gold-standard queries. Then we calculated mAP (4) for the entire evaluation dataset. In our evaluation process, we tested both the algorithm variants with and without Query Expansion. Additionally, we also tested the results of QE using general-domain FastText embeddings instead of our domain-oriented Stack Overflow-based. In Tables 3, 4, 5, 6, 7 we present the calculated mAP metric for different algorithm parameters. Each of the tables presents results for different *nice_to_have_factor* parameters. In each table, the rows represent the different values of *expand_factor* parameter while the columns represent *expand_limit* parameter both for Stack Overflow and FastText embeddings. The *expand_factor*=0.0 means no Query Expansion. The results show that the introduction of Stack Overflow expansion improves the algorithm results while FastText embeddings do not change the performance.

We can observe that we obtain globally best result ($mAP = 0.33$) for *nice_to_have_factor*=1.0, *expand_factor*=1.0 and *expand_limit*=3. This gives us 0.06 absolute gain and 22% of the relative gain over the variant without Query Expansion. For different values of *nice_to_have_factor* the gain is similar but proportionally lower according to the result without QE. The interesting conclusion is that scaling down the *nice_to_have_factor* and *expand_factor* worsens the results, so we should weight Nice-To-Have and Extended Skills the same as the Must-To-Have.

To make our experiments complete, we additionally calculated the *AveP* metrics separately for each of the 7 prospects. The results are presented in Fig. 2.

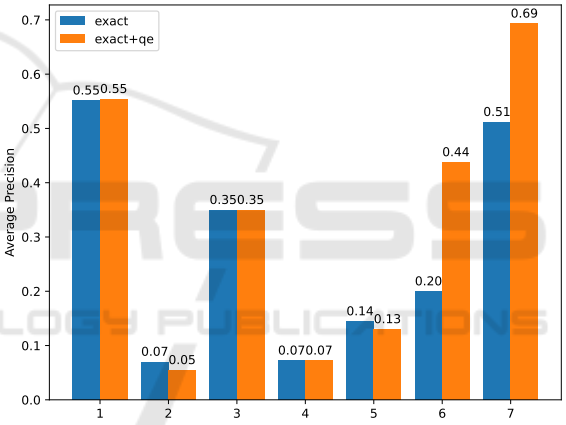

Figure 2: The *AveP* metric calculated for each of the gold standard prospects.

In Fig. 2 we can observe that the additional Query Expansion does not change the results for prospects 1-5 but improves the result significantly for the 6th and 7th prospects. We investigated these two queries and observed that these queries have the shortest list of skills compared to the other five. We think that this may explain the results - the semantic query expansion works best for the less precise queries.

10 DISCUSSION AND FUTURE WORKS

In the paper we presented the method for creating domain-oriented term embeddings for IT-related topics, using Stack Overflow questions tags. In our opinion, the LSA algorithm proved to be very useful in

MOST WIEDZY Downloaded from mostwiedzy.pl

Table 3: mAP calculated for *nice_to_have_factor*=1.0.

	Stack Overflow				FastText			
	1	2	3	4	1	2	3	4
0.0	0.27	0.27	0.27	0.27	0.27	0.27	0.27	0.27
0.2	0.28	0.28	0.28	0.26	0.25	0.26	0.27	0.24
0.4	0.28	0.28	0.28	0.27	0.25	0.24	0.26	0.23
0.6	0.28	0.28	0.29	0.27	0.25	0.26	0.26	0.23
0.8	0.28	0.29	0.32	0.28	0.25	0.26	0.26	0.24
1.0	0.28	0.28	**0.33**	0.30	0.25	0.26	0.27	0.24

Table 4: mAP calculated for *nice_to_have_factor*=0.8.

	Stack Overflow				FastText			
	1	2	3	4	1	2	3	4
0.0	0.24	0.24	0.24	0.24	0.24	0.24	0.24	0.24
0.2	0.24	0.24	0.24	0.26	0.24	0.25	0.26	0.24
0.4	0.24	0.25	0.25	0.27	0.24	0.24	0.25	0.24
0.6	0.24	0.25	0.25	0.27	0.24	0.25	0.26	0.24
0.8	0.24	0.24	0.28	0.28	0.25	0.24	0.26	0.24
1.0	0.25	0.25	**0.30**	**0.30**	0.25	0.25	0.25	0.24

Table 5: mAP calculated for *nice_to_have_factor*=0.6.

	Stack Overflow				FastText			
	1	2	3	4	1	2	3	4
0.0	0.24	0.24	0.24	0.24	0.24	0.24	0.24	0.24
0.2	0.24	0.24	0.24	0.26	0.24	0.24	0.24	0.24
0.4	0.24	0.24	0.25	0.26	0.24	0.24	0.25	0.24
0.6	0.24	0.24	0.25	0.27	0.24	0.25	0.25	0.24
0.8	0.24	0.24	**0.30**	0.28	0.24	0.25	0.25	0.24
1.0	0.24	0.24	0.29	0.28	0.24	0.25	0.25	0.24

Table 6: mAP calculated for *nice_to_have_factor*=0.4.

	Stack Overflow				FastText			
	1	2	3	4	1	2	3	4
0.0	0.24	0.24	0.24	0.24	0.24	0.24	0.24	0.24
0.2	0.24	0.24	0.24	0.26	0.24	0.24	0.24	0.24
0.4	0.24	0.24	0.25	0.27	0.24	0.24	0.25	0.24
0.6	0.24	0.24	0.26	0.28	0.25	0.25	0.25	0.24
0.8	0.24	0.24	**0.30**	**0.30**	0.25	0.25	0.25	0.24
1.0	0.25	0.24	**0.30**	0.29	0.25	0.25	0.25	0.24

Table 7: mAP calculated for *nice_to_have_factor*=0.2.

	Stack Overflow				FastText			
	1	2	3	4	1	2	3	4
0.0	0.24	0.24	0.24	0.24	0.24	0.24	0.24	0.24
0.2	0.24	0.24	0.24	0.24	0.24	0.24	0.24	0.24
0.4	0.24	0.24	0.25	0.27	0.24	0.24	0.25	0.24
0.6	0.24	0.24	0.27	0.28	0.25	0.24	0.25	0.24
0.8	0.24	0.24	**0.30**	**0.30**	0.25	0.25	0.25	0.24
1.0	0.24	0.24	**0.30**	**0.30**	0.24	0.25	0.25	0.24

SCITEPRESS SCIENCE AND TECHNOLOGY PUBLICATIONS

MOST WIEDZY Downloaded from mostwiedzy.pl

catching the *topical* sense of similarity in IT-related terms. LSA is definitely not dead and can work very well despite begin relatively old solution.

The presented embeddings have proven to be useful for the Teamy.ai candidates retrieval pipeline in the Query Expansion module. However, there are still many places where the method can be improved.

Firstly, we are planning to try some alternative algorithms for building latent representations of our terms such as PLSA (Hofmann, 1999), LDA (Blei et al., 2003) or Paragraph2vec (Le and Mikolov, 2014). Particularly Paragraph2vec seems to be very promising since it is designed to better reflect the topical notion of similarity than the traditional word2vec.

Another idea that comes to our minds is using more information from Stack Overflow than only the question tags. Eventually, we want to develop a method for automatically building a Knowledge Graph of IT-related concepts. The KG may be then used to better capture dependencies between various entities which can improve the results of Query Extension and candidates retrieval in general.

Finally, we understand that our gold standard dataset is not very big but since the Teamy.ai system is still in its development stage, it is the best that we can have for now. In the future, we want to find a way to extend it. On the one hand, this will give us more reliable accuracy measurements, on the other may allow us to try some learning-to-rank techniques that require training data.

ACKNOWLEDGEMENTS

The research has been founded by The National Centre for Research and Development within the project POIR.01.01.01-00-076120 ”System sztucznej inteligencji korelujacy zespoły pracownikow z projektami IT”

REFERENCES

Blei, D. M., Ng, A. Y., and Jordan, M. I. (2003). Latent dirichlet allocation. *Journal of machine Learning research*, 3(Jan):993–1022.

Bojanowski, P., Grave, E., Joulin, A., and Mikolov, T. (2016). Enriching word vectors with subword information. *arXiv preprint arXiv:1607.04606*.

Chiche, A. and Yitagesu, B. (2022). Part of speech tagging: a systematic review of deep learning and machine learning approaches. *Journal of Big Data*, 9(1):1–25.

Deerwester, S., Dumais, S. T., Furnas, G. W., Landauer, T. K., and Harshman, R. (1990). Indexing by latent semantic analysis. *Journal of the American society for information science*, 41(6):391–407.

Deng, L. and Liu, Y. (2018). *Deep learning in natural language processing*. Springer.

Guo, J., Fan, Y., Ai, Q., and Croft, W. B. (2016). A deep relevance matching model for ad-hoc retrieval. In *Proceedings of the 25th ACM international on conference on information and knowledge management*, pages 55–64.

Hofmann, T. (1999). Probabilistic latent semantic indexing. In *Proceedings of the 22nd annual international ACM SIGIR conference on Research and development in information retrieval*, pages 50–57.

Huang, P.-S., He, X., Gao, J., Deng, L., Acero, A., and Heck, L. (2013). Learning deep structured semantic models for web search using clickthrough data. In *Proceedings of the 22nd ACM international conference on Information & Knowledge Management*, pages 2333–2338.

Le, Q. and Mikolov, T. (2014). Distributed representations of sentences and documents. In *International conference on machine learning*, pages 1188–1196. PMLR.

Li, J., Sun, A., Han, J., and Li, C. (2020). A survey on deep learning for named entity recognition. *IEEE Transactions on Knowledge and Data Engineering*, 34(1):50–70.

Liu, T.-Y. et al. (2009). Learning to rank for information retrieval. *Foundations and Trends® in Information Retrieval*, 3(3):225–331.

Mikolov, T., Chen, K., Corrado, G., and Dean, J. (2013). Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*.

Mitra, B., Craswell, N., et al. (2018). *An introduction to neural information retrieval*. Now Foundations and Trends Boston, MA.

Mitra, B., Diaz, F., and Craswell, N. (2017). Learning to match using local and distributed representations of text for web search. In *Proceedings of the 26th International Conference on World Wide Web*, pages 1291–1299.

Mitra, B., Nalisnick, E., Craswell, N., and Caruana, R. (2016). A dual embedding space model for document ranking. *arXiv preprint arXiv:1602.01137*.

Pennington, J., Socher, R., and Manning, C. D. (2014). Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543.

Roy, D., Paul, D., Mitra, M., and Garain, U. (2016). Using word embeddings for automatic query expansion. *arXiv preprint arXiv:1606.07608*.

Sahlgren, M. (2006). *The Word-Space Model: Using distributional analysis to represent syntagmatic and paradigmatic relations between words in high-dimensional vector spaces*. PhD thesis, Institutionen för lingvistik.

Řehůřek, R. (2011). Fast and faster: A comparison of two streamed matrix decomposition algorithms.

MOST WIEDZY Downloaded from mostwiedzy.pl