Document Embedding for Scientific Articles: A validation of word embeddings

H.J. Meijer[1,2][0000−1111−2222−3333] and R. Karimi[2][1111−2222−3333−4444]

[1] University of Amsterdam, Science park 904, 1012WX Amsterdam, The Netherlands
[2] Elsevier, Radarweg 29, 1043 NX Amsterdam, The Neterlands
meijerarjan@live.nl, r.karimi@elsevier.com

Abstract. Over the last few years, word embeddings have taken a dominant position in the Information Retrieval domain. Many studies have been done concerning the quality and application of word embeddings on general texts, such as the Wikipedia corpus and comments on review websites. Giving promising results, the word embeddings have been studied and improved over recent years. However, these studies have been focused on generic texts, which are not limited to the characteristics of in-domain texts such as rare domain-specific words or have been focussed on small sets of academic texts. This research focusses on the quality and application of word embeddings on domain-specific texts, concerning a large corpus of 1.391.543 scientific articles which have been published in 2017. We validate the word embeddings using a categorization task to match articles to journals. We furthermore create a 2-dimensional visualization of the word embeddings on journal level, to visualize journal relatedness.

Keywords: Word Embedding · Document Embedding · Article Embedding · Journal Embedding · Embedding Visualization · Embedding Validation.

1 Research Data

1.1 Corpus

The dataset we used for this research consists of 1.391.543 articles from 3.759 journals, all articles have been published in journals in 2017, that have (in 2017) at least 150 publications. Details about the corpus can be found in table 1. The distribution of the text in our dataset follows a Pareto-like distribution, which is a common property of texts (**(author?)** [2]). Our word and token distribution is visualized in Figure 1, which displays the occurrences of the first 500 tokens and words of the corpus, ordered on occurrences.

	Total count	Unique count	Average length[3]
Title words	18.822.399	939.665	14
Title tokens	14.742.192	230.805	11
Abstract words	264.653.020	5.853.077	190
Abstract tokens	171.474.473	738.961	124
Total words	283.475.419	6.209.769	204
Total tokens	186.962.354	763.475	135

Table 1. Corpus size

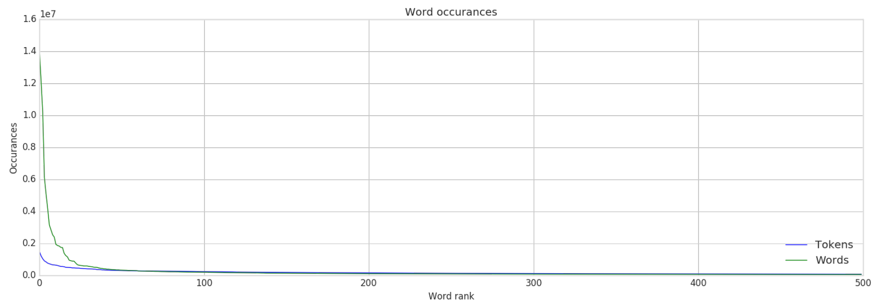

Fig. 1. Word and token occurrences. The y-axis shows the number of occurances for each word/token, the x-axis shows the occurence rank of the word/token; the position of the word/token in a list sorted on occurance.

1.2 Datasets

For this research we used a tokenized dataset, created from the earlier described corpus (see table 1). From this tokenized set, we created the embeddings and the TF-IDF feature vectors. The following steps have been applied to convert the words from the text to tokens: removing punctuation, non-ascii characters, number & stop-words, transforming all characters to lower-case and by stemming all remaining tokens using the NLTK stemmer[4]. These steps reduced our dataset by 34% (words to tokens), resulting in a set of 186.962.354 tokens.

Embedding For this research, we reused the word embeddings created by **(author?)** [1]. These embeddings have a vector dimension of 300, and have been trained on the entire Elsevier corpus (titles and abstracts respectively), not limited to the subset we used for this research. To create higher-level embeddings, we average all component embeddings. Thus, to create article embeddings (higher

[3] Rounded
[4] Natural Language ToolKit library, `https://www.nltk.org/`.

level), we take the average of all normalized word embeddings (component) for that article. Journal embeddings are created by taking the average of all normalized article embeddings. We use the average to combine multiple embeddings into one because we determine the meaning of a larger text for this research as the "average meaning" of all components. We have used multiple embedding variations for this research. **Default embedding**; the embeddings obtained from the word embeddings are referred to as the default embedding, *embedding* in figures. A variation on this embedding is the **TF-IDF weighted embedding**. The word embeddings in this set have been weighted with the TF-IDF value for each word, we refer to this set as TF-IDF embedding, }.tfidf_embedding in figures. The TF-IDF score is calculated using the raw token count and the smoothed inverted document frequency. The **10K TF-IDF embedding** uses the TF-IDF embedding as basis, The 10K TF-IDF embedding set has been weighted with the same TF-IDF, but is limited to the top 10.000 most occurring tokens. This set is referred to as 10K TF-IDF embedding, *10K_embedding* in the figures. Another variation on the TF-IDF set is the **5K TF-IDF embedding**. The 5K TF-IDF embedding has also been weighted with the TF-IDF values, but has been limited to the top 5.000 most occurring tokens. This set is referred to as 5K TF-IDF embedding, *5K_embedding* in the figures. The **1K-6K TF-IDF embedding** set weights the word embeddings with the TF-IDF values and is limited to the tokens between the top 1.000 and top 6.000 based on their occurrence. Creating a set of 5.000 tokens, without using the top 1.000 tokens. We refer to this set as *1k_6k_embedding* in the figures.

TF-IDF The TF-IDF feature vectors are created using the TF-IDF model and a hasher from the PySpark MlLib[5].The TF-IDF feature vectors are created by hashing the tokens with the hasher, which has a set hash bucket size. These hashed values are passed on to the TF-IDF model, resulting in a feature vector with a dimension equal to the amount of hash buckets. To limit the computational and storage expenses and to reduce noise by rare words, we limit our vocabulary size. We label the TF-IDF configurations as follows: $vocabularysize/hashbucketsize$. Furthermore, we denote 1.000 as 1K, since we deal with chosen values which can be exactly noted given this notation. This means that the set with a 10.000 vocabulary size and a 10.000 hash bucket size will be denoted as "10K/10K". We will refer to the TF-IDF configurations in our figures as "tfidf_vocabulary size_hash bucket size". Figure 2 shows the performance of TF-IDF on title and abstract as median rank, and Figure 3 shows the storage size in gigabyte, of the 1k/1K, 4K/4K, 7K/7K and 10K/10K TF-IDF configurations. These figures show that, while the required storage size stagnates, the performance on title also quickly stagnates; the performance of the abstracts show similar behaviour. The stagnation of the storage size is likely due to the fact that the terms which are added to the larger vocabulary sets occur less often than the other words (the cut-off is based on word occurrence, descending). Given this information, we have chosen to use the 10K/10K, 10K/5K and

[5] `http://spark.apache.org/docs/2.0.0/api/python/pyspark.mllib.html`.

5K/5K configurations to compare our embedding results to because these configurations can be kept in memory (storage size) while it is not likely that larger sets will give much improvement. The TF-IDF feature vectors are created on article level and we average the set of article feature vectors to create a journal feature vector, just as we did with the embeddings.

2 Pipeline

Data processing; The research has been done in a python-databricks environment, which used Spark SQL, a library that offers tools to work with big-data. We processed the TF-IDF sets and the embedding sets [6] via the same pipeline, using their common vector properties. The pipeline contains the following steps: creating a training and validation set, creating the journal embeddings from the training set, categorizing the validation set articles using the journal embeddings and calculating the performance metrics.

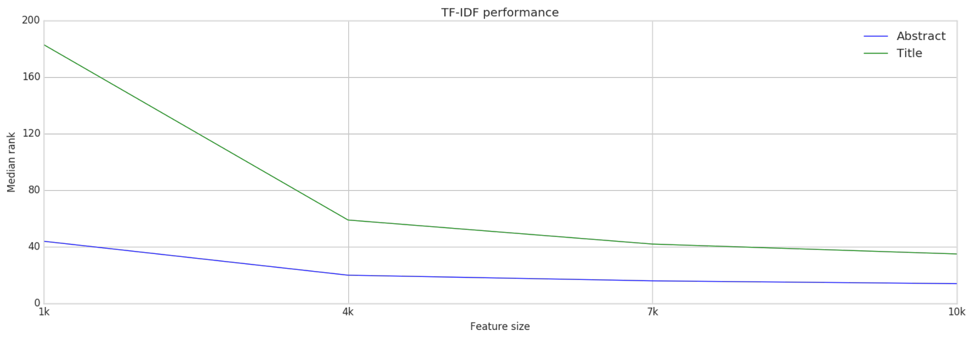

Fig. 2. TF-IDF performance on title and abstract.

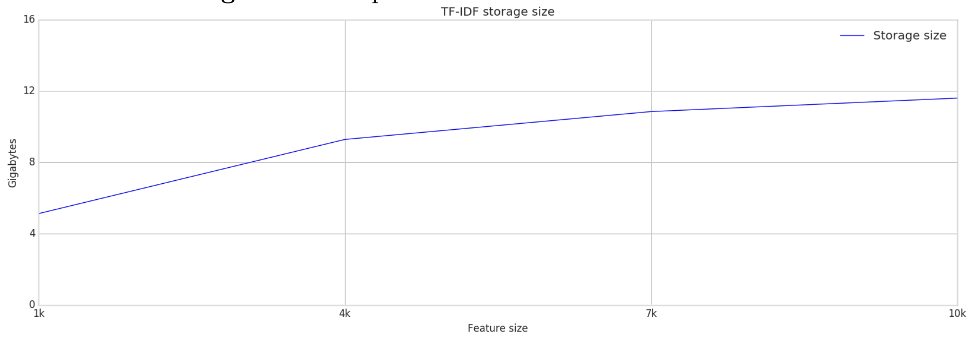

Fig. 3. TF-IDF memory usage for title and abstract combined.

7

[6] See Chapter 3.2 Datasets.

- Create training and validation set; We split our initial set in a training set (80%) and a validation set (20%). This split of the data is based on a random number given to each article, ensuring that all datasets (i.e. tfidf_10k_10k, embedding) have the same (random) training and validation set.
- Create journal embeddings; From our training set we create the journal embeddings, which are created for most sets[8] by averaging the article embeddings or feature vectors.
- Categorize validation articles; To categorize the articles, we calculate the distance between the title of the article (validation set), and the title of each journal (training set). We also do this for the abstract of the article and the abstract of each journal. During this process we record of the title-based-rank and abstract-based-rank of the source journal, best scored journal on abstract and title and the similarity between the source journal and article for both abstract and title.
- Distance metrics; To calculate the distance between two vectors, cosine similarity is commonly used. We validated the quality of cosine similarity as a distance metric by comparing it to other similarity metrics available in the SciPy library[9]. We calculate the similarities based on the normalized embeddings, and compared the distance metrics based on the default embedding set. Table 2 shows the results of this validation.

Metric[10]	Median title rank	Average title rank	Median abstract rank	Average abstract rank
Braycurtis	28	130	23	124
Canberra	33	148	26	133
Chebyshev	57	256	41	191
Cityblock	*28*	*130*	*23*	*124*
Correlation	*27*	*127*	*23*	*122*
Cosine	**27**	**127**	**23**	**122**
Euclidean	*27*	*127*	*23*	*122*
Mahalanobis	136	544	75	449
Seuclidean	*27*	*124*	*22*	*115*
Sqeuclidean	*27*	*127*	*23*	*122*

Table 2. Distance metric performance for word embeddings on the categorization of academic texts, only showing relevant results.

The results in Table 2 show high similarity between cosine-based metrics (Cosine & Correlation) and euclidean based metrics (Euclidean, Seuclidean & Sqeuclidean). This similarity is expected, since the cosine and euclidean distances

[8] See chapter 3.2 Datasets.

[9] `https://www.scipy.org/`

[10] As defined and provided by the SciPy library

should yield the same results on normalized sets[11]. The results show that some enhancement on the euclidean algorithm (Seuclidean) result in slightly improved results, although not significant. Also the Cityblock metric yields results close to the Cosine metric, only having a slightly worse performance, which is also not significant. Because of this, we will use the cosine-similarity as the distance metric, which will make our results easier to compare with other work, since the cosine-similarity is a commonly used similarity measure.

2.1 Performance measurement

We use multiple metrics to validate the performance of the embedding sets and TF-IDF sets on the categorization task. These metrics are: the F1-score, the Median & average rank and the Rank distribution.

- F1-score; For our matching algorithm, we classify the results as follows:
  $TruePositive =$ Articles correctly matched to the current journal
  $FalsePositive =$ Articles incorrectly matched to other journals
  $FalseNegative =$ Articles incorrectly matched to the current journal
  With these definitions, we calculate the Recall, Precision & F1 using the standard formulas.
- Median & average rank; We use both average and median ranks as rank-indication for a journal. The median rank indicates at which point 50% of all values is larger and 50% of all values is smaller, indicating the ”typical article”. While the average indicates the exact average of all articles in the set. We used both measures for our research.
- Rank distribution; To further analyse the ranking results, we plot the article rank distribution to get an indication of the ranking-landscape.

3 Research results

In this section the results of our research are presented; the detailed discussion on the meaning and implications of these results is presented in section 4: Discussion.

Ranking The Figures 4 & 5 show the result of the categorization task as ranking results. The rank indicates the position of the correct journal in the sorted list of matched journals. Figure 4 shows the ranking results for the different sets based on the title. Figure 5 displays the ranking results based on the abstract. Both graphs show both average and median ranks, based on the cosine-similarity between the article and journal embeddings or feature vectors.

[11] See: `https://en.wikipedia.org/wiki/Cosine_similarity#Properties`.

Rank distribution Figures 6 and 7 show the distributions of the ranks for each set. The Figures plot the summed amount of articles against the ranks on a logarithmic scale. Figure 6 shows the rank distribution for the titles, Figure 7 shows this for the ranks based on the abstract. These graphs give a detailed view of the ranks presented in their respective figures (Figures 4 and 5).

F1-Score Figures 8 and 9 show the precision, recall and F1 scores. These scores are calculated on journal level, and are averaged per set. Figure 8 shows the F1 score for the title and Figure 9 shows the scores for the abstract. These scores indicate the performance of the sets on absolute hits/top-1.

Memory usage Table 3 shows the total memory usage of each set for the *Validation set*, indicating their storage costs in gigabytes[12]. It furthermore shows the absolute hit percentage of the title and the abstracts, i.e. the percentage of articles that have their source[13] journal as the first result in the ranking. The table furthermore shows the median rank and the median abstract rank, as visualized in Figures 4 and 5. Thus, this table gives an overview of the memory usage of the sets, combined with their performance on the ranking task.

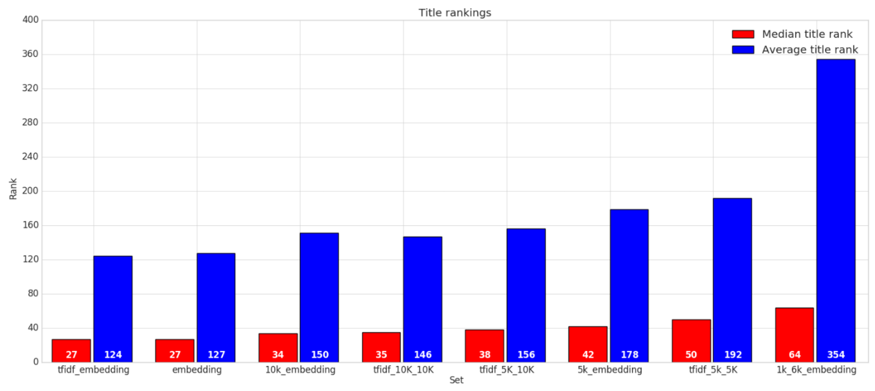

Fig. 4. Median and average title rankings

[12] 1024 based

[13] Journal from which the article was taken

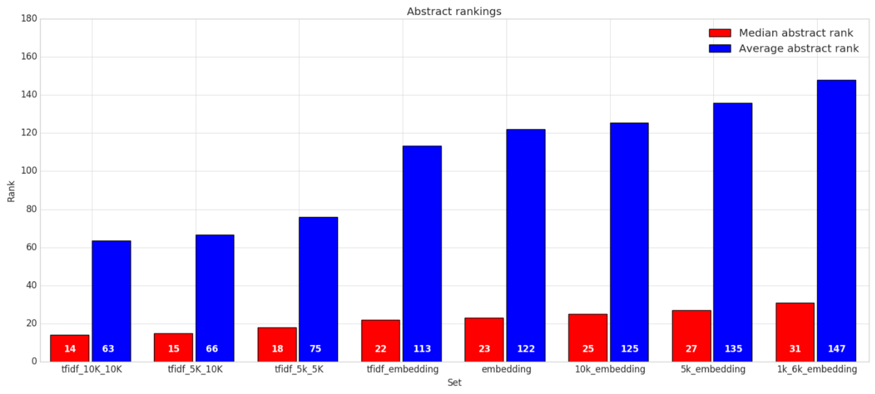

Fig. 5. Median and average abstract rankings

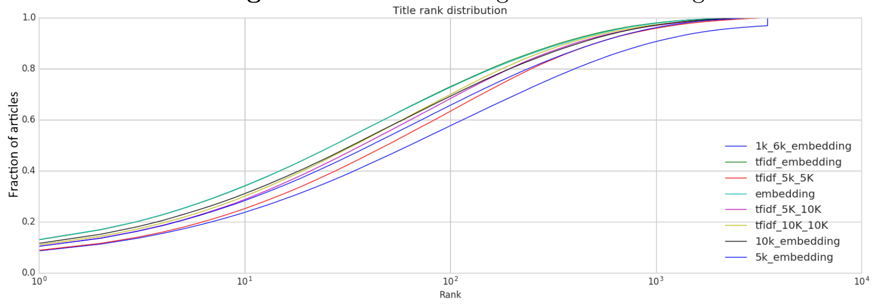

Fig. 6. Title rank distribution per set, Y-axis shows the fraction of articles.

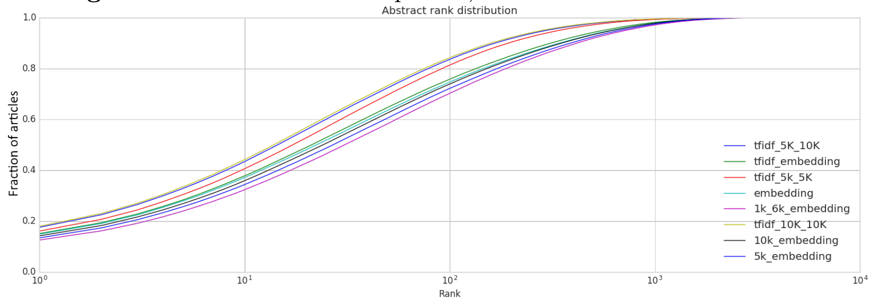

Fig. 7. Abstract rank distribution per set, Y-axis shows the fraction of acticles.

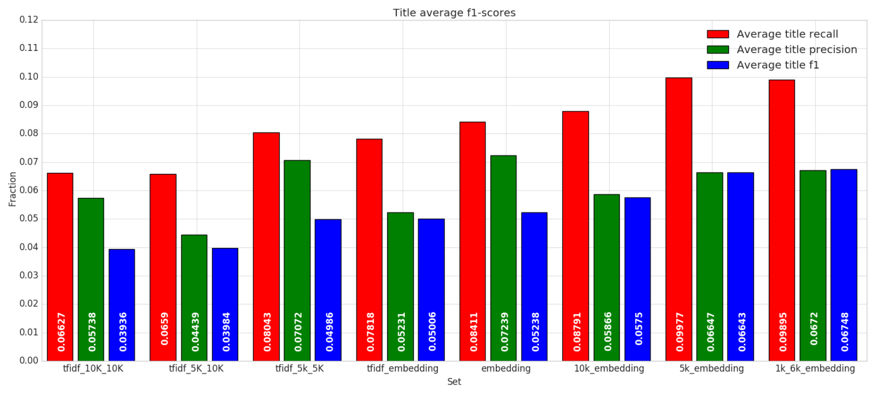

Fig. 8. Precision, recall and F1 scores based on title

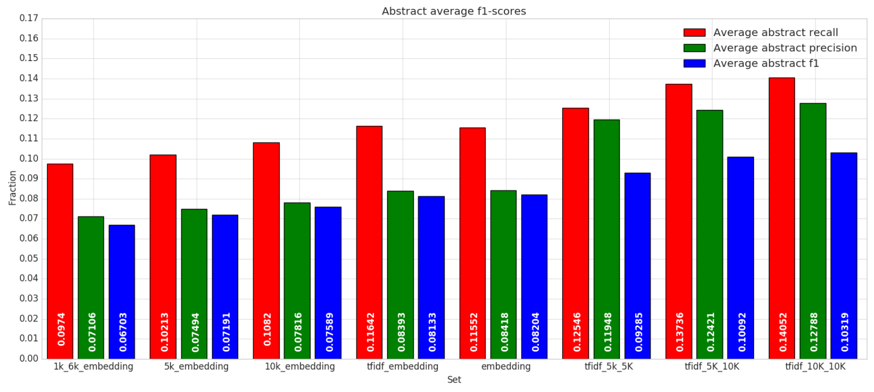

Fig. 9. Precision, recall and F1 scores based on abstract

Set	Size in GB	Absolute hit percentage		Median title rank	Median abstract rank
		Title	Abstract		
tfidf 5k 5K	9.82	5.42%	10.18%	50	27
tfidf 5K 10K	11.47	6.49%	11.08%	38	15
tfidf 10K 10K	**11.61**	**6.79%**	**11.32%**	**35**	**14**
embedding	3.13	7.92%	9.24%	27	23
5k embedding	3.13	6.34%	8.36%	42	27
10k embedding	3.13	7.03%	8.76%	34	25
tfidf embedding	**3.13**	**7.89%**	**9.33%**	**27**	**22**
1k 6k embedding	3.06	5.16%	7.86%	64	31

Table 3. Memory usage and performance for each set

4 Discussion

4.1 Results analysis

Best performers The data, as presented in Figures 4 and 5 shows that the 10k/10k set performs better than all other TF-IDF sets, although the difference with the 5k/10k is low (1 median rank on abstract and 3 median ranks on the titles). For the embeddings, the TF-IDF weighted embedding works better than the others, although it is not a significant improvement (1 rank on a set of more than 3.000) compared to the default embeddings.

TF-IDF The TF-IDF feature vectors outperform the embeddings on the abstracts, while the embeddings outperform the TF-IDF feature vectors on the titles. The main difference between the abstract and title is that the title contains fewer tokens compared to the abstract (see Table 1). This means that the titles contain less information than the abstracts. Due to this, the TF-IDF method, which is purely based on word-occurrences & counts has less information on the titles. The TF-IDF method works better on the abstract, containing more tokens than the tile, thus improving the differentiating the different abstract. The data furthermore shows that increasing the vocabulary size increases the performance of the TF-IDF, meaning that none of the created cut-off's resulted in cutting off noise.

Limited TF-IDF embeddings The limited TF-IDF embeddings all underperform, compared to the TF-IDF embedding on median and average ranking. This result indicates that the noise reduction is too much, it removes meaningful words. If the noise reduction would be too low, we would only see a slight increase of performance compared to the TF-IDF embedding, or none at all. However, the rank increases, indicating the reduction in embedding quality due to missing words. This is in line with what we found with the TF-IDF results: higher vocabulary sizes give better performance.

Rank distribution. However, Figures 6 and 7 show that their rank distribution is different from the other embeddings. The rank distribution of the limited TF-IDF embeddings show the following pattern: a high/average performance on the top-rankings, an underperformance on the middle rankings and a resulting stack-up of articles with a high-ranking. This is further supported by the TF-IDF score on titles (Figure 4), on which the limited TF-IDF embeddings are the top performers, indicating a better performance on the top-1 articles compared to the other sets.

The rank distribution seems to indicate that the cut-off was effective, but not for our purpose. The cut-off moved the "middle-ranked" articles to either the higher end or the lower end of the rankings, resulting in high median and average ranks. The reduction in vocabulary size did not reduce the storage size for the embeddings, except for the 1K-6K embedding[14].

TF-IDF & embeddings Our hypothesis on the difference between the TF-IDF and the standard embedding is as follows: The embeddings seem to outperform the TF-IDF feature vectors in situations where there is little information available (titles). This indicates that the embeddings store some word meaning that enables them to perform relatively well on the titles. The abstracts, on the other hand, contain much more information. Our data seems to indicate that the amount of information available in the abstracts enable the TF-IDF to cope with the lack of embedded information.

If this is the case, we could expect that there would little performance increase on the title when we compare the embeddings to the weighted TF-IDF embeddings, because the TF-IDF lacks the information to perform well, this can be seen in our data. We would also expect on the abstract an increase in performance since the TF-IDF has more information in this context. We would expect that the weighting applied by the TF-IDF improves the performance of the embedding by indicating word importance. Our data shows a minor improvement in performance of 1 median rank and 10 average ranks while these improvements cannot be seen as significant, our data at least indicates that weighting the embeddings with TF-IDF values has a positive effect on the embeddings.

Memory usage Although the TF-IDF outperforms the embeddings on the abstracts, its memory usage is higher than the memory usage of the embeddings. The top-performing embedding, TF-IDF weighted embedding, uses 3.13 GB, the top performing TF-IDF, 10K/10K uses 11.61 GB, which is 270.93% of the storage size needed for the embedding.

5 Conclusion

This research shows that the article embeddings, created with word embeddings, perform better than the reasonable TF-IDF alternatives on our categorization

[14] This indicates that he 1K-6K embedding removed some articles entirely from the dataset.

task, based on article titles. The TF-IDF alternatives give better results than the embeddings based on abstracts. The performance of the embeddings have been improved by weighing them with the TF-IDF values on the word level, although this improvement cannot be seen as significant on our dataset. This improved embedding set results in a median rank decrease of 8 on the titles and a median rank increase of 8 on the abstract, compared to the best performing TF-IDF alternative. The embedding also results in a memory decrease of 73.04% compared to the best performing TF-IDF alternative, making it more viable to keep it in memory. The visualization of the journal embedding shows that similar journals are grouped together, indicating a preservation of relatedness between the journal embeddings. We thus come to the following conclusions:

1. Article based embeddings perform better than TF-IDF on titles, i.e. small texts which contain limited information
2. TF-IDF performs better than article based embeddings on abstracts, i.e. larger texts which contain more information
3. Embeddings give a significant decrease in memory usage compared to TF-IDF
4. Visualization of the journal embeddings show that the embeddings capture and preserve subject relatedness when they are combined to create embeddings for larger texts

6 Future work

6.1 Intelligent cutting

A better way of cutting could improve the quality of the embeddings. This improvement might be achieved by cutting the center of the vector space out before normalization. All words which are generic are in the center of the spectrum, removing these words prevents the larger texts to be pulled towards the middle of the vector space, where they lose the parts of their meaning which set them apart from the other texts. We expect that this way of cutting, instead of word-occurrence cutting, will improve the quality of the word embeddings.

6.2 TF-IDFs performance point

In our research, TF-IDF performed better on the abstracts than on the titles, which, according to us, is caused by the text size of the two texts. This leads to the question, how do token count and unique token count relate for TF-IDF? Is there a point at which the TF-IDF outperforms the embeddings, and will continue to outperform? If this relation is found, one could skip the TF-IDF calculations in certain situations, and skip the embedding training in other scenario's, reducing costs.

6.3 Reversed word pairs

At this point, there are no domain-specific word pair sets available. However, as we demonstrated, we can still test the quality of word embeddings. Once one has established that the word vectors are of high quality, could one create word pairs from the embeddings? If this is the case, we could create word pair sets using the embeddings, reverse-engineering the domain specific word pair sets for future use.

Bibliography

[1] Truong, J.: An evaluation of the word mover's distance and the centroid method in the problem of document clustering (2017)

[2] Wiegand, M., Nadarajah, S., Si, Y.: Word frequencies: A comparison of pareto type distributions. Physics Letters A (2018)