Document Embedding for Scientific Articles: Validation of word embeddings

Arjan Meijer, 11425555

June 12, 2018

Abstract

	WordSim	Men	RareWords
Best results from the research by Truong:	0.49	0.61	0.32
Average results from the research by Truong	0.45	0.59	0.32

Table 1: Results for the different validation sets of word simularity validations on domain specific texts from the study by Truong [1]

1 Introduction

2 Motivation

Domain specific

Earlier work on this topic, concerning domain specific articles, by (Truong [1]), found that in-domain training of the word embeddings can improve the process of document clustering. This effect is even stronger than the number of training examples and the model architecture.[1]. Lai et al. [2] found that the corpus domain is more important than the corpus size. Using an in-domain corpus significantly improves the performance for a given task, whereas using a corpus in an unsuitable domain may decrease performance. These findings both indicate that an in-domain corpus improves the performance of word embeddings for the specific domains.

Problems in validation

To assess the quality (or usability) of the embeddings, validation methods are used. This are tasks designed to produce a metric that gives an indication of the usability of the provided embeddings. Schnabel et al. [3] found that the validation method indicates only the quality of an embedding for a specific task. There is (yet) no methtod that can asses the usability of an embedding on all possible tasks, since each task may require other information to be embedded into the embedding. Validation methods use either labelled or unlabelled data. Labelled data is data that is in some way marked, so that the correct answer can be derived from it. Unlabelled is the opposite, this data is not marked.

The usage of labelled data is common practice for validation methods, since the results produced by this data can be easily checked checked. Unpublished results of the study by Truong encounter this problem, they show high error rates on the validation scores, presented in Table 1. However, the word embeddings created correct document clusterings[1], this seems to indicate that the word-vectors are able to represent the words correctly but that the available validation sets cannot confirm this.

Furthermore, a study by Schnabel et al. [3] found that the quality of embeddings are tasks specific, *different tasks favour different embeddings.* They also found that the embeddings encode information about word frequency, even in models that are created to prevent this. *This casts doubt on the common practice of using vanilla cosine similarity as a similarity measure.*

Therefore, we propose the validation of domain specific word-embeddings through a classification tasks, using multiple vector-distance calculations. This eliminates the need for labelled data in the validation of these domain specific word embeddings, will validate the quality of word embeddings for domain specific texts, and will validate the impact of different vector-distance measures on a categorization task.

3 Background

Embedding
Machine learning (ML) tasks rely on a numerical (vectorial) representation of text which we refer to as an embedding. These can be calculated for texts of different lengths such as a title, sentence, paragraph or an entire document[4]. Word embeddings are these numerical representations of a word, these vectors are an distributed representation of the word over the multiple (vector) dimensions(Mikolov et al. [5]). The word embeddings can be used to construct embedding of larger texts. Word embeddings can capture both the semantic and syntactic information of words. The advantage of the machine learning models is that it can be done without human-interaction(Lai et al. [2]). Word embeddings have improved various Natural Language Processing (NLP) areas such as named entity recognition, part-of-speech tagging, parsing, and semantic role labelling (Luong et al. [6]).

Word2Vec
Word2vec word embeddings are created using neural network, Word2vec learns word embeddings via maximizing the log conditional probability of the word given the context word(s) occurring within a fixed-sized window. Therefore the learnt embeddings contain useful knowledge about word co-occurrence[7]. There are multiple input/output possibilities for the neural network, best known are Skip-gram and the Continuous Bag-of-Words model (CBOW). The Skip-gram model takes a target word as input and outputs the predicted output words, while CBOW takes the context words as input and outputs the predicted target word[7, 8].

Paragraph vectors
Variations on the word2vec model have also been proposed, Le and Mikolov [9] introduced the paragraph vector, based on the word2vec model. The paragraph vector model uses additional variables to improve the accuracy of the word-embeddings. An advantage of the paragraph vector model is that it takes the word order into consideration, atleast in a small context [9].

GloVe
Pennington et al. [8] introduced the GloVe (Global Vectors) model. This model captures the global corpus statistics. The model transforms the word co-occurrences of all words in the corpus to chances, it excludes all the zero values and uses that as initial input for the neural network.

TF-IDF
TF-IDF is an abbreviation for Term Frequency - Inversed Document Frequency. This method does not rely on a neural network, and does not require training. According to a paper by Beel et al. [10] from 2016, "TF-IDF was the most popular weighting scheme (70%) among those approaches for which the scheme was specified" (in the recommendation class 'Content-based filtering'). The TFIDF score is the product of the term frequency in a text and the inversed document frequency of the same term in a corpus of texts. Both of which can be calculated in a variety of ways.

Man	Women	King	Queen
Athens	Greece	Oslo	Norway
great	greater	tough	tougher

Table 2: Word analogies used in word embedding validation

Embedding validation
Embedding validation techniques are methods that are used to validate the quality[1] of an embedding for a specific task(Schnabel et al. [3]). Multiple validations of word embeddings have been used, including: Word analogy, text similarity, categorization and positional visualization.

Word Analogy
Word analogy validation is based on a labelled validation set, containing, commonly, word pairs of four, that can be logically divided into two parts. As Table 2 shows, each last word can be derived from the three words before. The score is the fraction of correctly given fourth words, given the first three words. This validation metric is used in multiple studies[5, 8, 11, 12]. Both this validation technique and the Word Similarity technique use vector distance calculations to validate the embeddings, this can therefore also be written as:

$$X_{\text{Man}} - X_{\text{King}} \approx X_{\text{Women}} - X_{\text{Queen}}$$

This means that the resulting vector of embedding of "Man" minus the embedding of "King" is approximately the embedding of "Woman" minus the embedding of "Queen". This resulting vector may be close to a vector "monarch" for example.

Word Similarity
A method to test the quality of word embeddings is the word similarity test. For these test, the distance between the word embeddings (vectors) is measured and compared to similarity scores defined by humans. Multiple non domain specific validation sets are publicly available including: the Rare-word dataset introduced in the paper "Better Word Representations with Recursive Neural Networks for Morphology" by Luong et al. [6], the MEN test collection by Bruni [13] and the WordSimilarity-353 test collection by Gabrilovich [14]. These sets, among others, have been used in multiple studies of word embeddings[8, 11]. This validation metric also relies on labelled data.

Classification
The classification validation method is a simple task which compares multiple texts. Lau and Baldwin [15] used data from StackExchange and tried to determine if a pair was a duplicate. Even though the validation method is simple, it too used labelled data to validate the acquired results.

Categorization
Le and Mikolov [9] used for their research a dataset of IMDB with 100,000 movie review. They validated their proposed paragraph vector model by determining whether a review was positive or negative.

Position Visualization

[1] With quality we mean the extend to which the task is completed correctly

(Unlabelled, Needs human validation)Dai et al. [12] and Hinton and Roweis [16] mapped their word embeddings to a two dimensional vector to be able to display them in a graph and applied colors to various categories. The advantage of this is that a human can directly see that the embeddings make sense, however this approach is not applicable by a computer.

Even though these validation methods are not limited to domains, the labelled data they use are, since they do not consist of words of specific domains. At this moment, there are no sets for every domain which make it difficult to compare the accuracy of domain specific word embeddings to non domain specific word embeddings.

Related work

4 Work context

Data

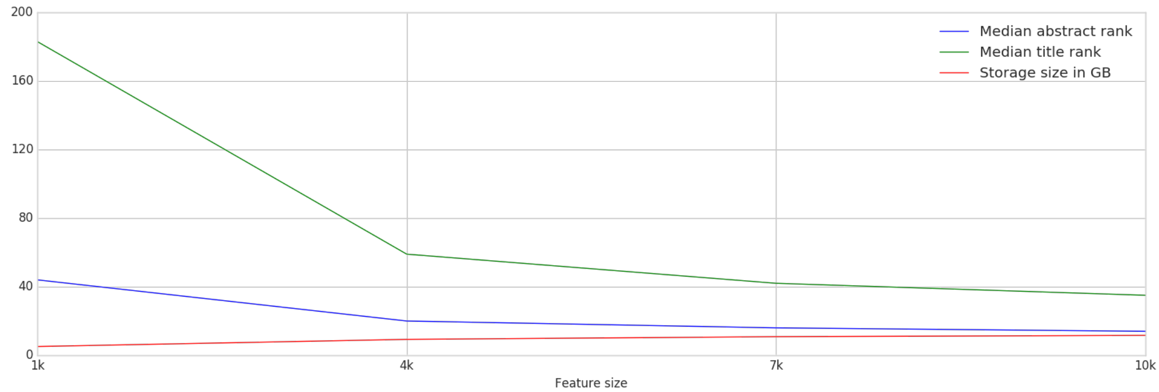
Experiment setup
We used the following pipeline to collect the performance metrics for the categorization task:

1. Create embeddings
2. Filter articles
3. Create training and validation set
4. Create journal embeddings
5. Categorize validation articles
6. Calculate performance metrics

Create embeddings
For this research, we used word-embeddings (created using the word2vec model) and TF-IDF embeddings. The word-embeddings were created with the word2vec model from PySparks MlLib library[17]. The embeddings were pre-trained and have a vector size of 300, which is an industry default. The TF-IDF embeddings were created with the TF-IDF model from pyspark's MlLib library[17], in combination with a token hasher (HashingTF) from the same library. We used multiple hashing dimensions and multiple vocabulary sizes. Since the TF-IDF uses the output of the term hasher, the TF-IDF model produces the same dimensions. We will denote the tfidf sets as vocabulary size/hashing size. All of our sets, both embedding and TF-IDF, use tokenized texts.

<u>Tokenization</u>
== TODO ==

<u>TF-IDF selection</u>
To limit computational expenses, we used only the top performing which can reasonably be kept in RAM. The plot displays the ranking & size for the 1000/1000, 4000/4000, 7000/7000 and 10.000/10.000 sets respectively.



The plot show the median abstract & title rank and the storage size in Gigabytes (1024 based) plotted over feature size, which is equal to the hashing size for these sets. The plot shows that both title and abstract are stagnating, while the memory usage is, slowly, going up. At a feature size of 10.000, we have a storage size of 11.6GB. Given this size and the stagnation of the rankings, we chose to use the 10.000/10.000 feature size tfidf vectors, we furthermore used the 10.000/5000 set and the 5000/5000 set for comparison.

Embeddings

We have made use of multiple embedding sets for this research. All sets share the same (default) embedding and have been (uniquely) modified. For this research we have used the following embedding sets:

1. Default embedding

   The embedding as generated by word2vec, without further enhancements.

2. TF-IDF embedding

   The embedding as generated by the word2vec, multiplied by their tfidf weights to embed word priorities. We use this enhancement to give the embeddings more information about the corpus.

3. 10K TF-IDF Embedding

   The embedding as generated by the word2vec, filtered on the top 10.000 most common words, multiplied by their tfidf weights.
   We use this enhancement to try to filter out possible noise created by lots of words with few occurrences.

4. 5K TF-IDF Embedding

   The embedding as generated by the word2vec, filtered on the top 5.000 most common words, multiplied by their tfidf weights.
   We use this enhancement to cancel out the noise caused be rare words more aggressively

5. 1K 6K TF-IDF Embedding

   The embeddings as generated by the word2vec, filtered on the top top 1.000 till top 5.000 most common words, multiplied by their tfidf weights.
   We use this enhancement to ignore the top 1.000 most common words, which are most likely

generic words, since they occur often. And to cancel out the rare-words, by cutting off all words after 6K. This gives us a set of 5K words.

Filter articles
We create our initial set of articles by collecting all articles from the journals that were published in the year 2017, and have at least 200 publications in 2017. This reduces the journal set to $3,759$ thousand journals, resulting in a set of $1,391,543$ million articles ().

Create training and validation set
We split our initial set 80% - 20%,. We use the 80% set as the training set for the journal representations, and the 20% set as the validation set for the journal representations.

Create journal embeddings
From our training set we create the journal embeddings by averaging all title embeddings as the journal title embedding, and by averaging all abstract embeddings as the journal abstract embedding. We also normalized both embeddings.

Categorize validation articles
To categorize the articles, we calculate the distance between the title- and abstract embedding of each article, from the validation set, to the title- and abstract embedding of each journal. To calculate the distance, we use cosine similarity (as provided by the SciPy library[18]). During this process we keep track of:

- Title-based-rank of the actual journal
- Abstract-based-rank of the actual journal
- Best scored journal on the abstract similarity
- Best scored journal on the title similarity
- Abstract similarity between the actual journal and the article
- Title similarity between the actual journal and the article

Performance measurement
We use multiple metrics to indicate the performance of the embeddings on a categorization task. These metrics are:

1. F1-score
2. Median & average rank
3. Rank distribution

<u>F1-score</u>
We define the positive & negative metrics as follows:

$TruePositive$ = Articles that are correctly matched to the current journal
$FalsePositive$ = Articles that are incorrectly matched to other journals
$FalseNegative$ = Articles that are incorrectly matched to the current journal

We used these metrics to calculate the Recall, Precision & F1 as follows:

$$Recall = \frac{TruePositive}{TruePositive + FalseNegative}$$

$$Precision = \frac{TruePositive}{TruePositive + FalsePositive}$$

$$F1 = \frac{2 * Precision * Recall}{Precision + Recall}$$

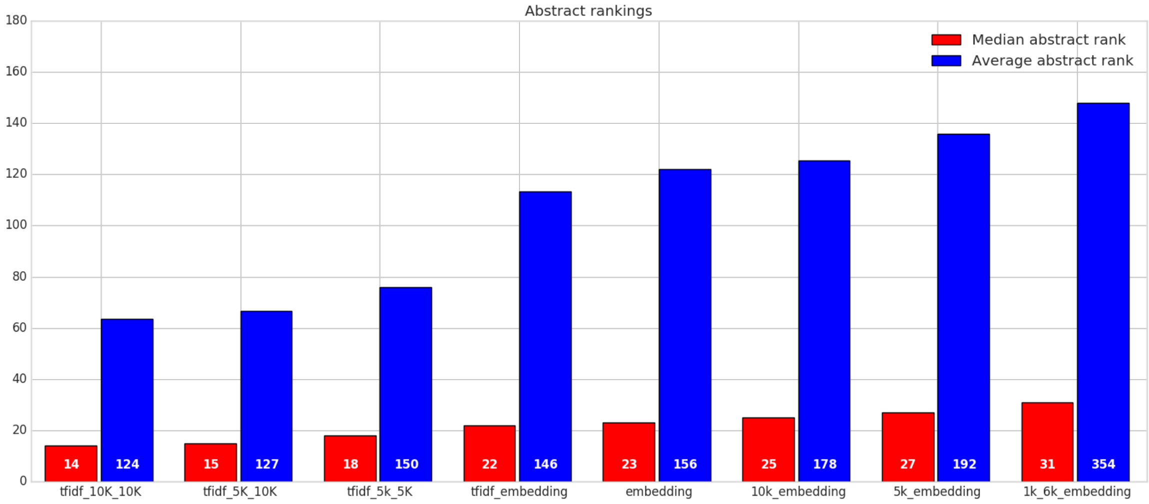
Median & average rank
We use the median rank to indicate around which rank the 'standard' article would be ranked, based on its title or abstract. We do this by taking the median of the respective rank from each article. This gives us an indication of the behaviour of most articles in our validation set. This median rank (mostly) ignores the outliers, we therefore also use the average rank, which gives a more global indication, although this rank may be over-influenced by some outliers.

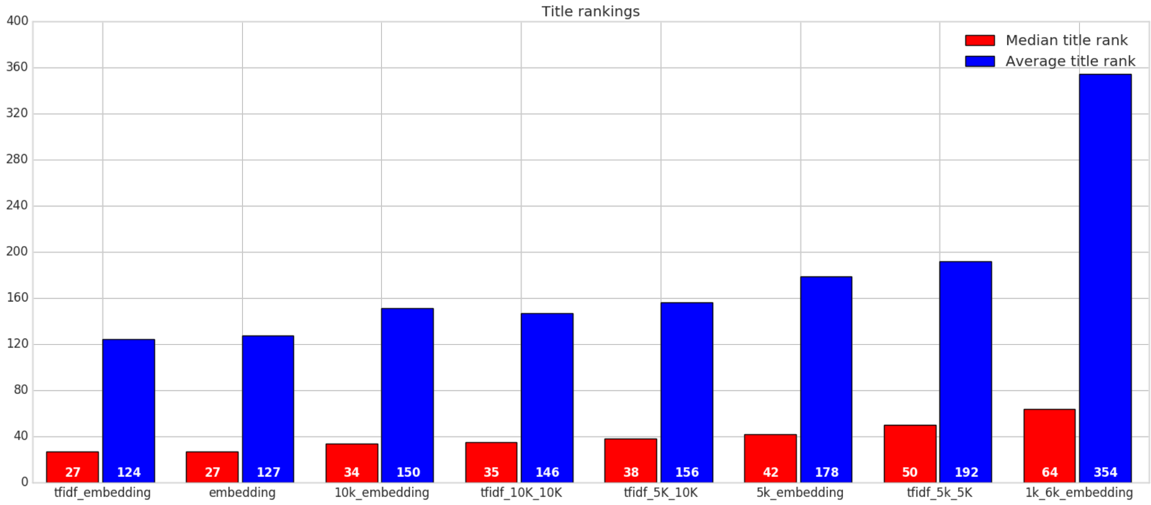
Rank distribution
To further analyse the ranking results, we plot the rank distribution to get an indication of the ranking-landscape. We limit ourselves to the following categories: 1 (absolute hits), top-10, top-20, top-30, top-40, top-50, top-100 and 100+.

5 Results

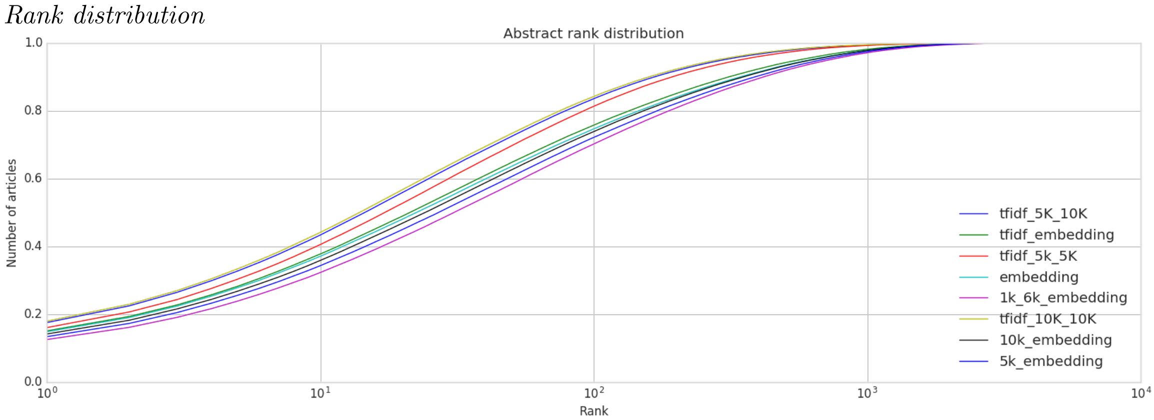
Research results

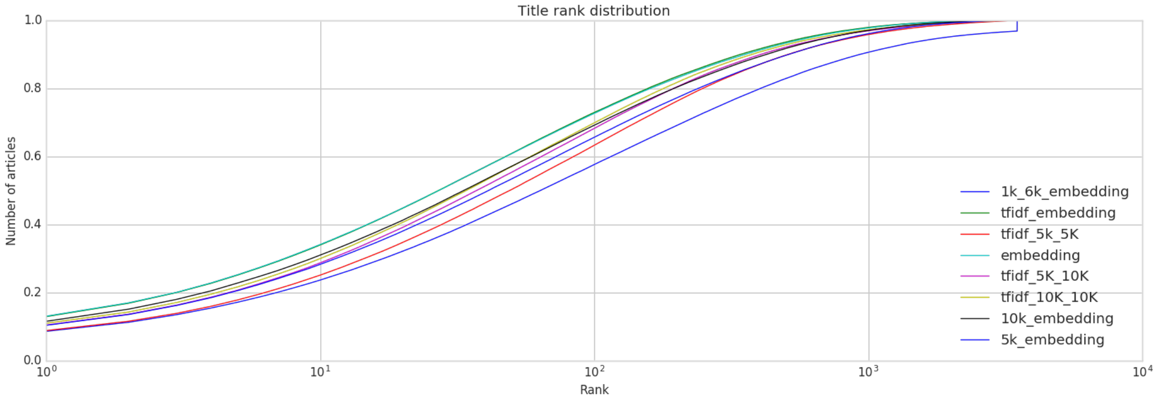
Ranking





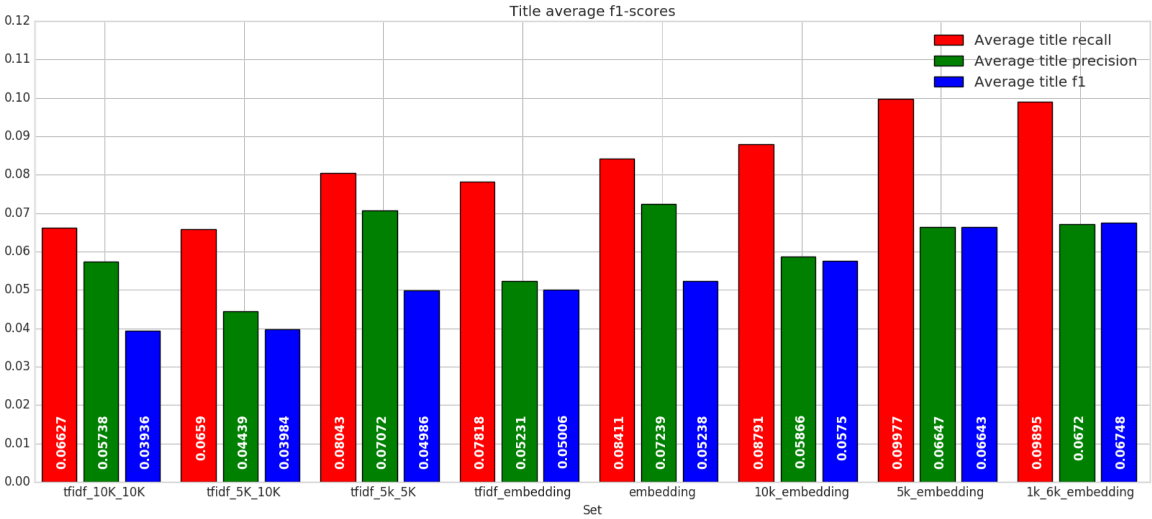
Rank distribution



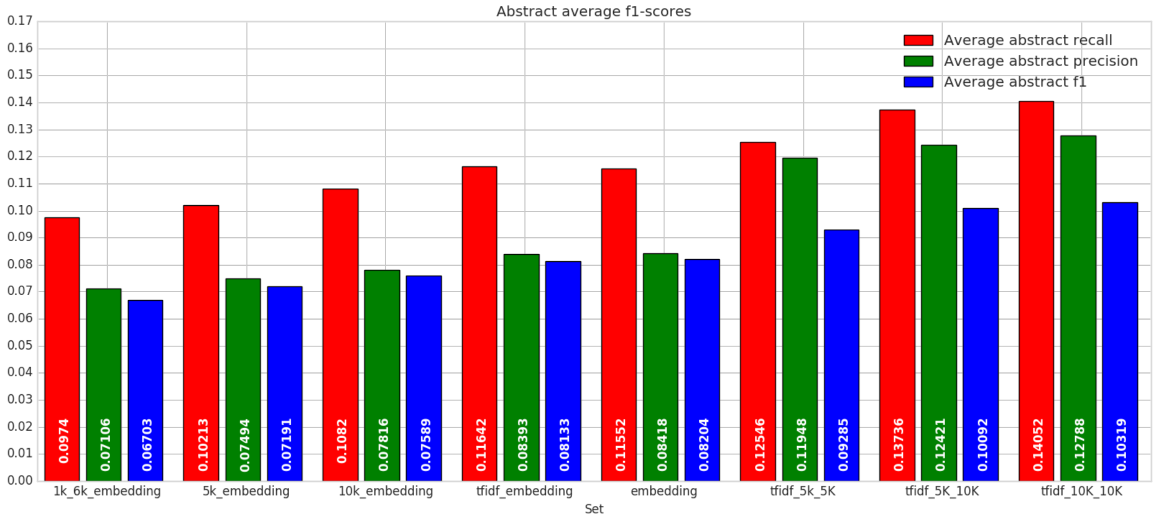


Title rank distribution
Number of articles
Rank
1k_6k_embedding
tfidf_embedding
tfidf_5k_5K
embedding
tfidf_5K_10K
tfidf_10K_10K
10k_embedding
5k_embedding

F1-Score



Title average f1-scores
Average title recall
Average title precision
Average title f1
Set
tfidf_10K_10K
tfidf_5K_10K
tfidf_5k_5K
tfidf_embedding
embedding
10k_embedding
5k_embedding
1k_6k_embedding
0.06627
0.05738
0.03936
0.0659
0.04439
0.03984
0.08043
0.07072
0.04986
0.07818
0.05231
0.05006
0.08411
0.07239
0.05238
0.08791
0.05866
0.0575
0.09977
0.06647
0.06643
0.09895
0.0672
0.06748



Memory usage

Set	Size in GB
tfidf 5k 5K	9.82
tfidf 5K 10K	11.47
tfidf 10K 10K	**11.61**
embedding	3.13
5k embedding	3.13
10k embedding	3.13
tfidf embedding	**3.13**
1k 6k embedding	3.06

Discussion

1k 6k behaviour

1k 6k embedding has lower storage than the other embeddings, this is most likely due to it's word filtering. Cutting off most-used words results probably in empty titles/abstracts which reduce the space that is needed for this embedding variant. All other embeddings have the same size, which indicates that this does not occur in the other situations, even in the 5k set, where the same amount of words are cut off.

The, relatively, much higher average rank compared to the median rank of the 1k 6k embedding can be explained with the data of table (RANKING DISTRIBUTION TABLE). This shows that the embedding goes up at the start, indicating that there are articles in the low ranks, flats out in the center more than the others do, indicating a relatively low amount of articles in the middle ranks, and a strong increase at the end. Indicating that there are many titles on the high end of the distribution. This explains the relatively low median, which is dawn towards the lower end of the distribution, and the high average, which is drawn to the higher end of the distribution.

Best performers
The data shows that the 10.000/10.000 set preforms better than all other TF-IDF sets, although the difference with the 5.000/10.000 is low, 1 (7.14%) on abstract and 3 (8.57%) on title. For the embeddings the TF-IDF weighted embedding works better than the others, although it is near equal to the default embeddings, which 1 rank higher on abstract, and equal on title.

TF-IDF weighting on embeddings
The difference between the TFIDF weighted embedding and the default embedding can be explained as follows: The embeddings seem to outperform the TF-IDF in situation when there is little information available, the titles in our case. This indicates that the embeddings store some kind of 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, since the TF-IDF lacks the information to perform well. This can be seen in our data, only the average rank increased by 3, indicating that there is a difference between the two embeddings, but not a major one. We could 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(4.35%) median rank and 10(6.41%) average ranks.

Raw, readable results
The TF-IDF outperform the embeddings on the abstract with a difference of 8 ranks for the best of each. On the title however, the best embedding outperforms the best TF-IDF by 8 ranks.

Memory usage
Although the TF-IDF outperforms the embeddings on the abstracts, the memory usage of the TF-IDF 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, 10.000/10.000 uses 11.61 GB, which is 270.93% more. The closest TF-IDF configuration we used was 1.000/1.000, which uses 5.13 GB (SEE GRAPH XXX). This TF-IDF set has a median title rank of 183 and a median abstract rank of 44. Which is worse than the embedding, which also uses less memory.

Conclusion
This research shows that the article embeddings, created with word embeddings, perform better than the reasonable TF-IDF alternatives for our categorization 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 weighting them with the TF-IDF values on word level. This improved embedding results in a median rank decrease of 8 on title and an median rank increase of 8 on title, compared to the best performing TF-IDF alternative. The embedding also results in a memory decrease of 73.04% making it more viable to keep it in memory. We have furthermore shown that limiting vocabulary size to exclude rare words or common words decreases the performance of the embeddings. We thus come to the following conclusions:

1. Article based embeddings perform better than TF-IDF on small texts
2. TF-IDF performs better than article based embeddings on larger texts

3. Embeddings give a significant decrease in memory usage compared to TF-IDF

References

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

[2] Siwei Lai, Kang Liu, Shizhu He, and Jun Zhao. How to generate a good word embedding. *IEEE Intelligent Systems*, 31(6):5–14, 2016.

[3] Tobias Schnabel, Igor Labutov, David Mimno, and Thorsten Joachims. Evaluation methods for unsupervised word embeddings. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 298–307, 2015.

[4] R. Karimi. Go deep or go out. Technical report, Elsevier, 2017.

[5] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119, 2013.

[6] Thang Luong, Richard Socher, and Christopher Manning. Better word representations with recursive neural networks for morphology. In *Proceedings of the Seventeenth Conference on Computational Natural Language Learning*, pages 104–113, 2013.

[7] Eric Nalisnick, Bhaskar Mitra, Nick Craswell, and Rich Caruana. Improving document ranking with dual word embeddings. In *Proceedings of the 25th International Conference Companion on World Wide Web*, pages 83–84. International World Wide Web Conferences Steering Committee, 2016.

[8] Jeffrey Pennington, Richard Socher, and Christopher Manning. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543, 2014.

[9] Quoc Le and Tomas Mikolov. Distributed representations of sentences and documents. In *International Conference on Machine Learning*, pages 1188–1196, 2014.

[10] Joeran Beel, Bela Gipp, Stefan Langer, and Corinna Breitinger. paper recommender systems: a literature survey. *International Journal on Digital Libraries*, 17(4):305–338, 2016.

[11] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

[12] Andrew M Dai, Christopher Olah, and Quoc V Le. Document embedding with paragraph vectors. *arXiv preprint arXiv:1507.07998*, 2015.

[13] Elia Bruni. Men test collection, 2012. URL `https://staff.fnwi.uva.nl/e.bruni/MEN`.

[14] Evgeniy Gabrilovich. Wordsimilarity-353 test collection, 2002. URL `http://www.cs.technion.ac.il/ gabr/resources/data/wordsim353/`.

[15] Jey Han Lau and Timothy Baldwin. An empirical evaluation of doc2vec with practical insights into document embedding generation. *arXiv preprint arXiv:1607.05368*, 2016.

[16] Geoffrey E Hinton and Sam T Roweis. Stochastic neighbor embedding. In *Advances in neural information processing systems*, pages 857–864, 2003.

[17] Apache Spark. Pyspark mllib package. URL `http://spark.apache.org/docs/2.0.0/api/python/pyspark.mlli`

[18] ScyPy. Distance computations. URL `https://docs.scipy.org/doc/scipy-0.14.0/reference/spatial.distan`