Document Embedding for Scientific Articles: Validation of word embeddings

H.J. Meijer, 11425555

June 18, 2018

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 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 text, which do not contain the complexity of in-domain knowledge, such as rare domain specific words. This research focusses on the usefulness of word embeddings on domain specific texts, concerning 1.391.543 scientific articles which have been published in 2017. The data shows that the word embeddings only perform better than TF-IDF, which was chosen as a traditional alternative, when the ranking is done on titles. The best ranking result is obtained through a TF-IDF with a vocabulary size of 10.000 applied on the abstracts, resulting in a median rank of 14, while the pure word embeddings result in a median rank of 23. This shows that word embeddings, although useful, are not unchallenged by older techniques.

1 Introduction

Introduction and Background

Information retrieval
Information Retrieval(IR)is the activity of gathering relevant information, given another initial piece of information. The most practical example of this is a search engine. Given one or more search words the search engine will attempt to find relevant information. For example, a Google search on "Information Retrieval" (initial piece of information) will give you a list of results (relevant information). To be able to do this, the computer (search engine) must know which texts are related. To achieve this, multiple techniques can be used, such as TF-IDF, Word2Vec, Paragraph Vectors and GloVe. These techiques can be divided into two categories, ones that use a neural network and ones that do not.

Neural Network
Neural Networks are computational structures, based on vector and/or matrix calculations. They can be applied to many different tasks, but need to be trained for each task. This training is the process of iteratively adjusting multiplication matrices or vectors to achieve the optimal result. Which is the result that is as close as possible to the given output for different input and output sets. In some IR techniques, Neural Networks are used to either predict words that may occur around a given word, or predict a word given words that surround it. For example, given the sentence

"The search engine searches for relevant information"

the Neural Network can be trained to either predict the words around "searches" (the, search, engine, for, relevant, information) or to predict the word "searches", given the surrounding (the, search, engine, for, relevant, information). This trained matrix (vector per word) now indicates "word relatedness" which, as mentioned earlier enables association (thus retrieval) with related(relevant) texts. Neural networks are also used for other tasks such as video recommendation on Google's YouTube, as described by Covington et al. [1].

Embedding
The created vector for a word is referred to as a word embedding. An embedding is a distributed, numerical representation of a text which can capture both the semantic and syntactic information[2]. In the case of a word embedding, the embedding represents the word. The advantage of the machine learning models that create these embeddings is that they do not need human interaction Lai et al. [3], they are so called "unsupervised learning algorithms". Once trained, the embeddings can be used to construct embeddings of larger texts. For example, a word embedding can be used to create a sentence, paragraph, document or corpus embedding. The usage of word embeddings have improved various Natural Language Processing areas such as named entity recognition, part-of-speech tagging, parsing, and semantic role labelling Luong et al. [4].

Text analysis techniques
To enable a computer to process text, for embedding creation or for tasks, the text has to be processed by an algorithm. In this research we used embeddings created by Word2Vec and TF-IDF feature vectors.

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[5]. 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[5, 6]. Mikolov et al. [2][7] presented several extensions to the word2vec model that improve the quality of vector training and its speed, such as introducing Hierarchical Softmax, Negative Sampling and the subsampling of frequent words to the Skip-gram approach. Variations to the word2vec model have also been proposed, such as the doc2vec model described by Lau and Baldwin [8] which creates document embeddings instead of word embeddings.

Paragraph vectors

Variations on the word2vec model have also been proposed, Le and Mikolov [9] introduced the paragraph vector based in their paper "Distributed representations of sentences and documents". The Paragraph Vector framework is based on the word vectors framework. The difference between the frameworks is the calculation of the probability, the Paragraph Vector framework uses an matrix D, which consists of every paragraph. This matrix is used to replace a concatenation or average of word vectors. An advantage of the paragraph vector model is that it takes the word order into consideration, atleast in a small context [9]. Dai et al. [10] state in their paper "Document embedding with paragraph vectors" that the Paragraph Vectors model performs significantly better than other models on grouping, triplet finding and related object/article finding tasks on the Wikipeida and arXiv texts.

GloVe

Pennington et al. [6] 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. This model out performs other models on three specific tasks, word analogy, word similarity and entity recognition.

TF-IDF

TF-IDF is an abbreviation for Term Frequency times Inverted Document Frequency. This method does not rely on a neural network and therefore does not require training. The TF-IDF score is the product of the term frequency in a text and the inverted document frequency of the same term in a corpus of texts. Both of which can be calculated in a variety of ways. The feature vectors produced by TF-IDF do not capture syntactic or semantic information about words, but capture information about word occurrences.

As Lai et al. [3] state in their paper "How to generate a good word embedding?", all embedding methods rely on the same hypothesis, words that occur in similar contexts have similar meanings. They furthermore found that larger corpus' lead to better quality embeddings, but that the domain in which the embeddings are trained is dominant over the corpus size.

Validation methods

The results produced by the techniques have to be validated to determine their quality (in usage). The quality of the results can be validated through, among others the F1 score and, for classification tasks, the rank of the correct category. The embeddings can furthermore be validated through their performance on tasks such as word analogies, word similarities, categorization and position visualization. These tasks can be designed to produce a score that gives an indication of the performance on a specific task. Schnabel et al. [11] found that a single validation metric cannot produce a representative result for other tasks. Embeddings that perform well on one task do not have to perform well on another task. As a result, the findings about performance of an embedding method are limited to the task on which they are tested, their results cannot be generalized to state that the embedding are overall "performing well". Validation tasks 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, in contrast to unlabelled data. The validation tasks can also be diveded into two groups, extrinsic evaluation, ones that use word embeddings as input for a downstream task. And Intrinsic evaluation, which directly test the relationships of the word embedding themselves. Schnabel et al. [11] note that the extrinsic evaluation may not be consistent with intrinsic evaluations, since the performance on downstream tasks is not consistent across tasks.

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 1 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[2, 6, 7, 10].

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

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

Table 1: Word analogies examples

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. [4], the MEN test collection by Bruni [12] and the WordSimilarity-353 test collection by Gabrilovich [13]. These sets, among others, have been used in multiple studies of word embeddings[6, 7]. This validation metric also relies on labelled data.

Classification
A classification validation method is a simple task which assigns a label to a text. Lau and Baldwin [8] used data from StackExchange and tried to determine if a pair was a duplicate. In their setup, the text was a pair of texts, and their categories were duplicate and non-duplicate. 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
Dai et al. [10] and Hinton and Roweis [14] mapped their word embeddings from a high dimensional vector to a two dimensional vector to be able to display them in a scatter plot and applied colors to various categories. These categories can be created with labelled or unlabelled data. The advantage of this is that a human can directly see the word distributions, and see if it is distributed in a way that seems logical. It gives furthermore insight in the overall spectrum of the words. However, this representation does not give a score, since it is not a evaluation of the data, but an alternative representation.

General and Domain specific
Since the word embeddings are created form a given text, these embeddings are bound to the text. All meaning embedded in the word embedding is derived from the original text. Because of this, embeddings can be "Domain specific" meaning that it only knows words (or a specific word) in a certain context. This becomes most clear when faced with words that can have different meanings in different contexts. For this research we categorize the embedding in two categories, general embeddings and domain specific embeddings. The general embeddings are trained on a collection of texts that use common English and contains a wide variety of topics. The domain specific embeddings are trained on a collection of texts that uses uncommon English (i.e. domain specific terms) and/or is limited to a small amount of topics. Given these terms, we regard the embeddings trained on the Wikipedia corpus[3, 6, 8, 10, 11] as general, since Wikipedia uses common English and spans a wide range of topics. On the other hand we regard the embeddings created by Truong [15] as Domain specific, these embeddings were created on academic articles, which use domain-specific terms and notations and only consists of academic texts, which contains less general/generic words compared to the Wikipedia corpus.

Research environment
The research will be done in a python-databricks environment, which uses spark, a library that offers tools to work with big-data. **?**] state that Spark SQL lest programmers leverage the benefits of relational processing and lets SQL users call complex analytix libraries in Spark. This allows for much tighter intergration between relational and procedural processing. The paper further states that Spark SQL makes it significanly simpler and more efficient to write data piplines that mix relational and procedural processing, while offering substantial speedups over previous SQL-on-Spark engines.

Text properties
Wiegand et al. [16] state that the pareto distribution offers a good fit to the word occurrences in natural language. Their models show that, due to the additional parameters in the pareto-III, the tail of the data fits better to the model than the Zipf model. This shows the relation between the word occurrences rank,

and the actual occurrences. This kind of word-occurrences distribution holds for many texts, including the writings of William Shakespeare and scientific texts and novels[17].

Motivation

In-domain embeddings and validation
Earlier research, concerning domain specific articles, by Truong [15], 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. Lai et al. [3] 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. Truong et al. encountered a problem in the validation of these in-domain embeddings. The word embedding produced correct document clustering results, leading to the conclusion that these embeddings are of good quality, since they capture the document relatedness needed to create correct clusterings. However unpublished results by Truong et al. state that the embeddings show high error rates on the validation scores. This seems to indicate that the word-vectors are of good quality, but that the available validation metrics cannot confirm this. Truong [15] used multiple word similarity validations to asses the quality of the word embeddings. However, these set are created to validate the generic embeddings, they fail to asses the quality of the domain specific embeddings.

Research
To asses the problem of the limited availability of pre-labelled validation sets for domain specific articles, we compare the embeddings to TF-IDF on a categorization task. This A) gives an indication of the embedding quality for categorization tasks and B) contrasts the performance of embeddings to the performance of the more traditional TF-IDF approach. To ensure the quality of the embeddings for our research, we reuse the embeddings created in the research of Truong [15].

RQ.1 Have word-embeddings a higher accuracy for academic texts than TF-IDF for article classification?

RQ.2 Which metric can be used to measure the accuracy of word embeddings for scientific articles?

Embedding and TF-IDF
Research question one focusses on the classification results of both embedding-based techniques and TF-IDF. To measure classification task we use the rank of the class to which the item belongs. Transforming the classification task from a binary metric to a ranking metric. For this task, we will use different versions of embeddings, and different TF-IDF versions to not only compare the two techniques, but also look for the optimal results of both techniques. For this part of the research, we will use the following hypothesis:

Embedding based techniques give lower rankings than the TF-IDF based techniques

By validating our invalidating this hypothesis we get an indication of the performance of the embeddings compare to older techniques, and get insight into possible performance and resource trade-off's and if the computational power and AI expertise needed for the creation of embeddings is worth the investment .

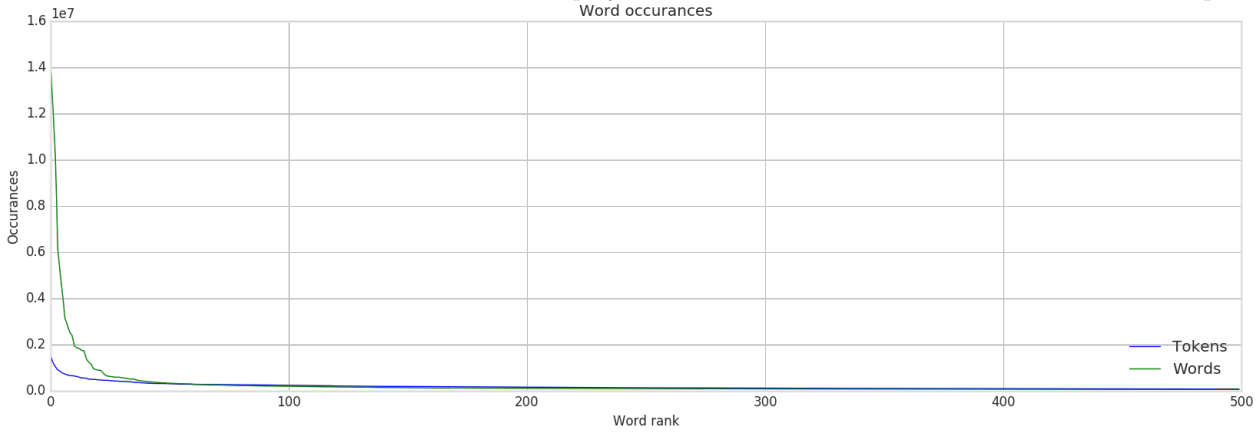
2 Data

Corpus
The dataset we used for this research consists of all articles[1] published in 2017 which have been published in a journal that has, in 2017, atleast 150 publications. This results in a total dataset of 1.391.543 articles from 3.759 journals.

	Total word count	Unique word count	Total token count	Unique token count
Title	18.822.399	939.665	14.742.192	230.805
Abstract	264.653.020	5.853.077	171.474.473	738.961
Total	283.475.419	6.209.769	186.962.354	763.475

Table 2: Corpus size

The word occurrences follow the pattern of a pareto distribution as described by Wiegand et al. [16]. This distribution is visualized in XXXXXX, which displays the occurrences of the first 500 tokens of the corpus.



Datasets
For this researched we used a (pre-made) tokenized dataset, which reduces the total amount of words by 34%, from this tokenized set, we created the embeddings and the TF-IDF feature vectors.

Tokenization
The following steps have been applied to the words to create a tokenized set:

1. Removed punctuation
2. Removed all non-ascii characters
3. Transformed all characters to lower-case
4. Removed stop-words, as provided by the NLTK library
5. Removed numbers
6. Stemmed all words, using the stemmer provided by the NLTK library

These transformations reduced our dataset by 34%, resulting in a tokenized set of 186.962.354 tokens.

Embedding
For this research we reused the word embeddings created by Truong [15]. These embeddings have a vector size of 300, which is an industry default. They have been trained on the entire Elsevier corpus, not limited to the subset we used for this research. To construct article embeddings we take the average of all normalized word embeddings for that article. Journal embeddings are constructed in the same way, we averaged all the normalized article embeddings to create the journal embeddings. We have used multiple embedding

[1] From Elseviers corpus

configurations for this research

Default embedding
The default embedding is created form the pre-trained word embeddings, no modifications have been applied to this set.

TF-IDF embedding
The TF-IDF weighted embedding set, referred to as TF-IDF embedding, are the default word embeddings weighted with a TF-IDF score per word.
The TF-IDF is calculated with a raw token count, and a smoothed inverted document frequency, calculated as follows:

$$IDF = \log_{10}(\frac{|A|}{|A_t|}) \tag{1}$$

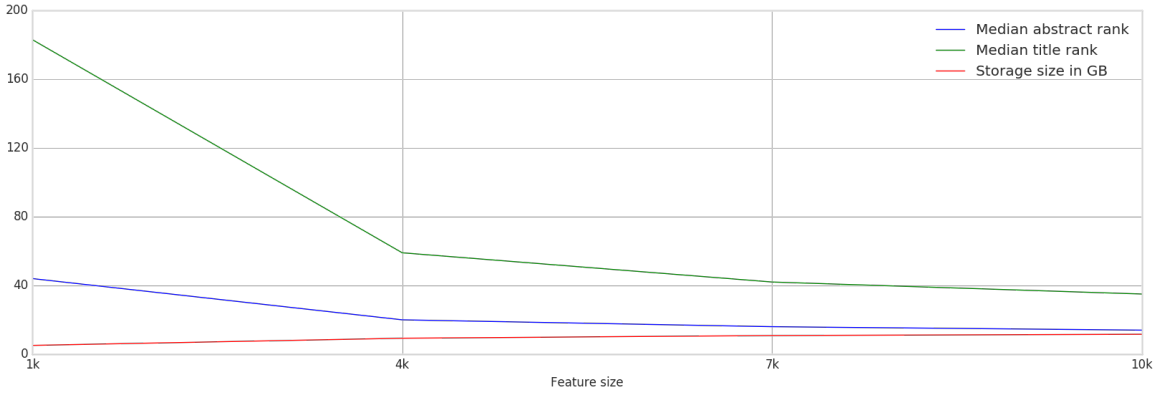
Where $|A|$ is the total count of articles and $|A_t|$ is the count of articles containing term t. The articles embeddings are a normalized summation of each word vector multiplied by it TF-IDF value. Since we take a sum of all words, the Term Frequency is embedded as the raw count of each word.

10K TF-IDF embedding
The 10K embedding set is generated similarity to the TF-IDF embedding, this version only uses the 10.000 most common tokens, reducing the amount of tokens it uses. This set was created to see if the limitation to 10.000 tokens reduces the amount of noise, and with that increasing the performance.

5K TF-IDF embedding
The 5K TF-IDF embedding is the TF-IDF embedding set, limited to the 5.000 most common words. This set was created to more aggressively limit the amount of tokens, and with that, cancel out more noise.

1K-6K TF-IDF embedding
The 1K-6K TF-IDF embedding is the TF-IDF embedding limited to the top 6.000 most common words, without the top 1.000 most common words. The rationale for this is that common words will occur in many articles, creating noise, by cutting of the top 1.000 and cutting of everything below 6.000 we tried to reduce the noise by filtering common words. This cut results in a set of 5.000 tokens, which allows us to compare it to the 5K TF-IDF set.

TF-IDF
To create the TF-IDF feature vectors, we used the TF-IDF model and a hasher from pysparks the MlLib library. 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 which vector dimensions equals 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 denote 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.



The graph XXX shows the performance and storage size[2] of the 1k/1K, 4K/4K, 7K/7K and 10K/10K TF-IDF configurations. This plot shows that, while the required storage size keeps rising, the performance on title quickly stagnates, and the performance on abstract follows too. Given this information, we have chosen to use the 10K/10K, 10K/5K and 5K/5K configurations to compare our embedding results to. The TF-IDF features are created on article level. We average the set of article embeddings to create a journal embedding.

[2]in gigabyte, 1024 based

Pipeline
We procesessed the TF-IDF sets an the embedding sets via the same pipeline, based on vector calculations. This ensures comparable results. The pipeline is setup as follows:

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

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. This split is based on a random number given to article each record, ensuring that all set 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[3] by averaging the article embeddings or feature vectors.

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, 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

Distance metrics
To calculate the distance between to vectors, cosine similarity is commonly used CITATIONS NEEDED. We validated the quality of cosine similarity as a distance metrics by comparing it to all other similarity matrices available in the SciPy library, which we used to calculate the distances. We calculate the similarities based on the normalized embeddings, and compared the distance metrics based on the default embedding set. XXXX shows the results of this validation.

[3] see XXXXX

Metric[a]	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**
Dice	1995	1929	1995	1929
Euclidean	*27*	*127*	*23*	*122*
Hamming	1995	1929	1995	1929
Jaccard	1995	1929	1995	1929
Kulsinski	1995	1929	1995	1929
Mahalanobis	136	544	75	449
Matching	1995	1929	1995	1929
Rogerstanimoto	1995	1929	1995	1929
Russellrao	1995	1929	1995	1929
Seuclidean	*27*	*124*	*22*	*115*
Sokalmichener	1995	1929	1995	1929
Sokalsneath	1995	1929	1995	1929
Sqeuclidean	*27*	*127*	*23*	*122*
Yule	1995	1929	1995	1929

[a]As defined and provided by the SciPy library

These results 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 should yield the same results on normalized sets. The results show that some enhancement on the euclidean algorithm result in slightly improved results, although not significant XXX EXPLAIN? XXX. Also the Cityblock metric yields results close to the Cosine metric, it has a slightly worse performance, which is also not significant. Because of this, we will use the cosine-similarity as the distance matrix, which will make our results better comparable with other work.

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:

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

F1-score

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.

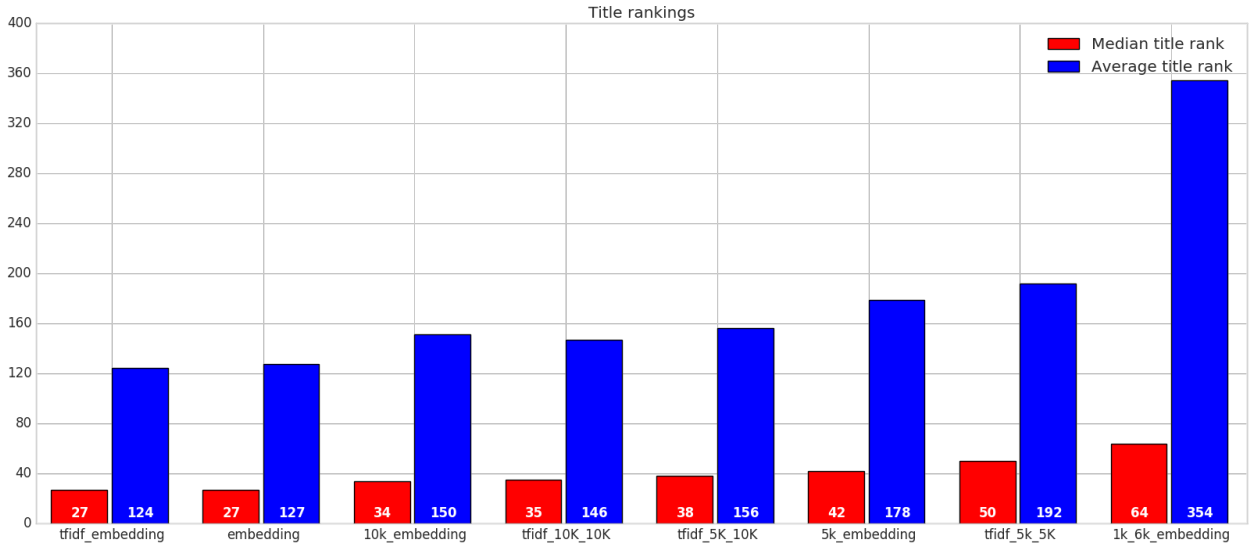
Rank distribution

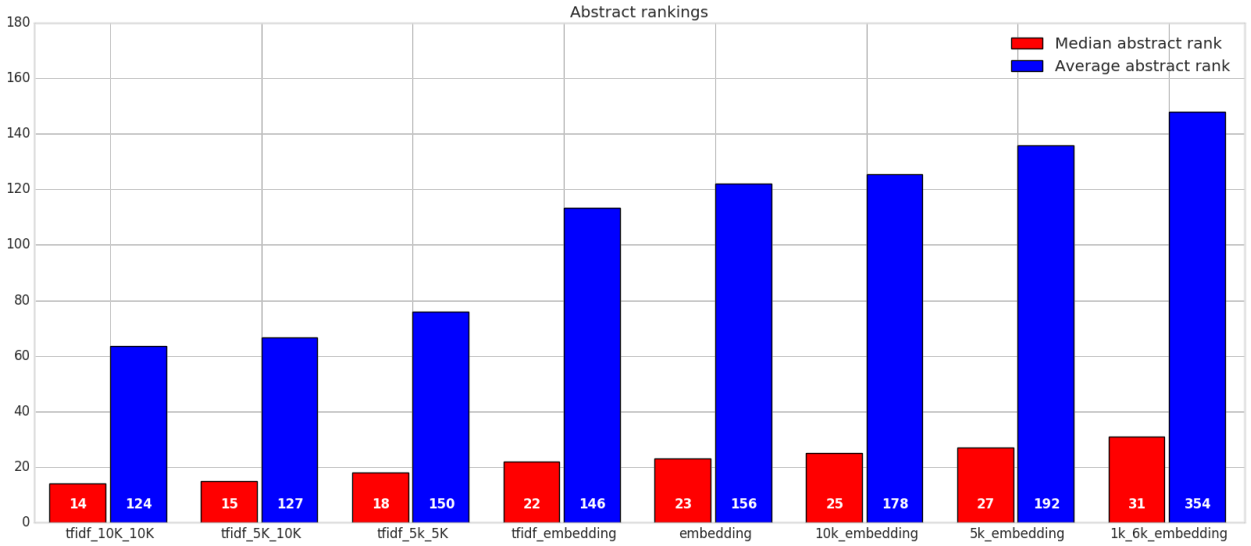
To further analyse the ranking results, we plot the rank distribution to get an indication of the ranking-landscape.

3 Results

Research results

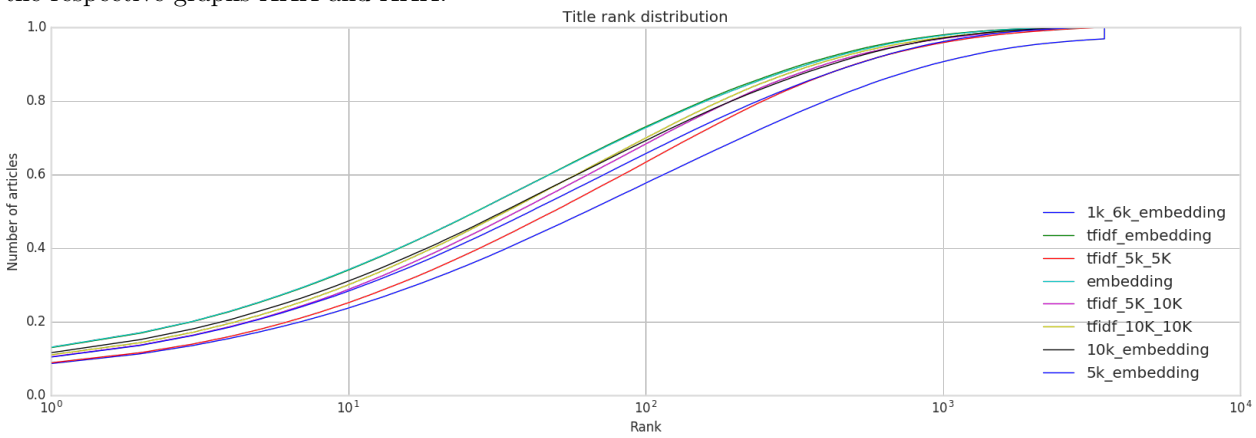
Ranking
The graph XXX shows the ranking results for the different sets based on the title. Graph XXX displays the ranking results based on the abstract. Both graphs show both average and median ranks, based on the cosine-similarity between the article and journal embeddings or feature vectors.

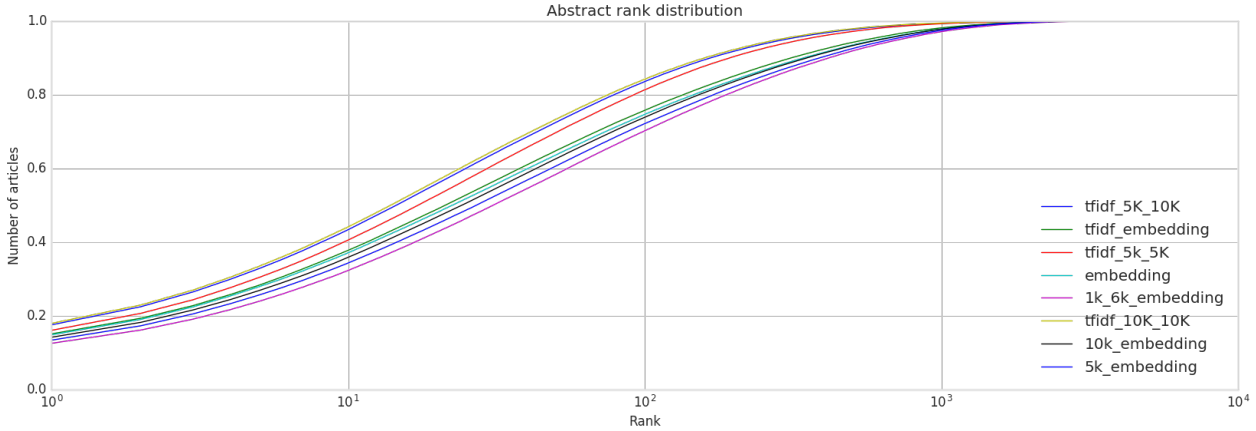




Rank distribution

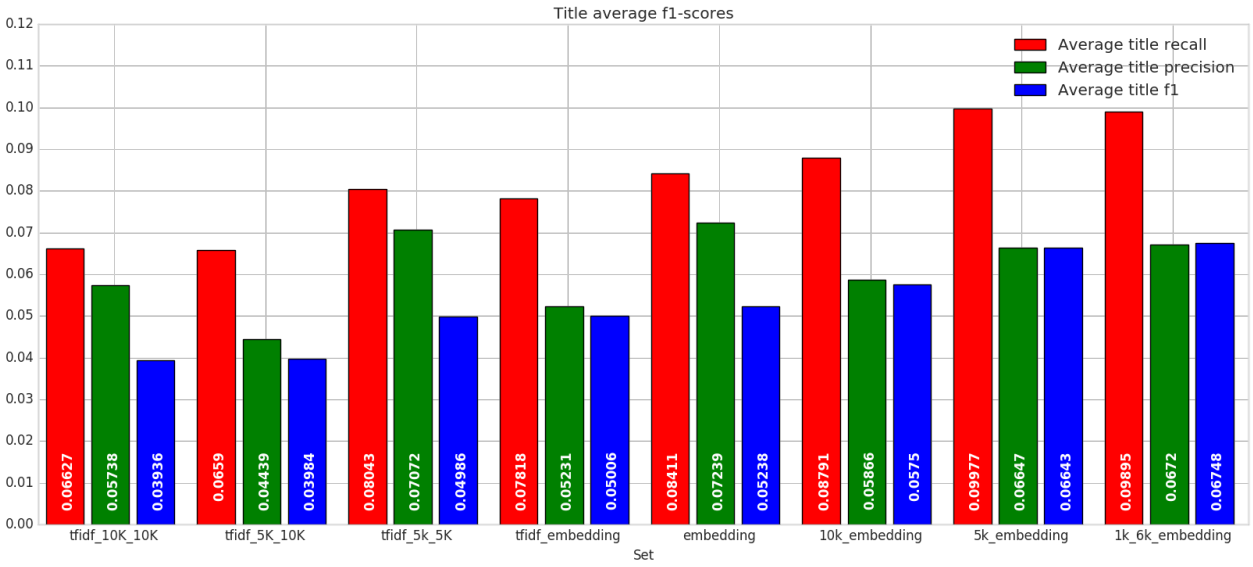
Graph XXX shows the rank distribution for the titles on, graph XXX shows this for the ranks based on the abstract. Both graphs use a logarithmic scale. These graphs give a detailled view of the ranks presented in the respective graphs XXX and XXX.

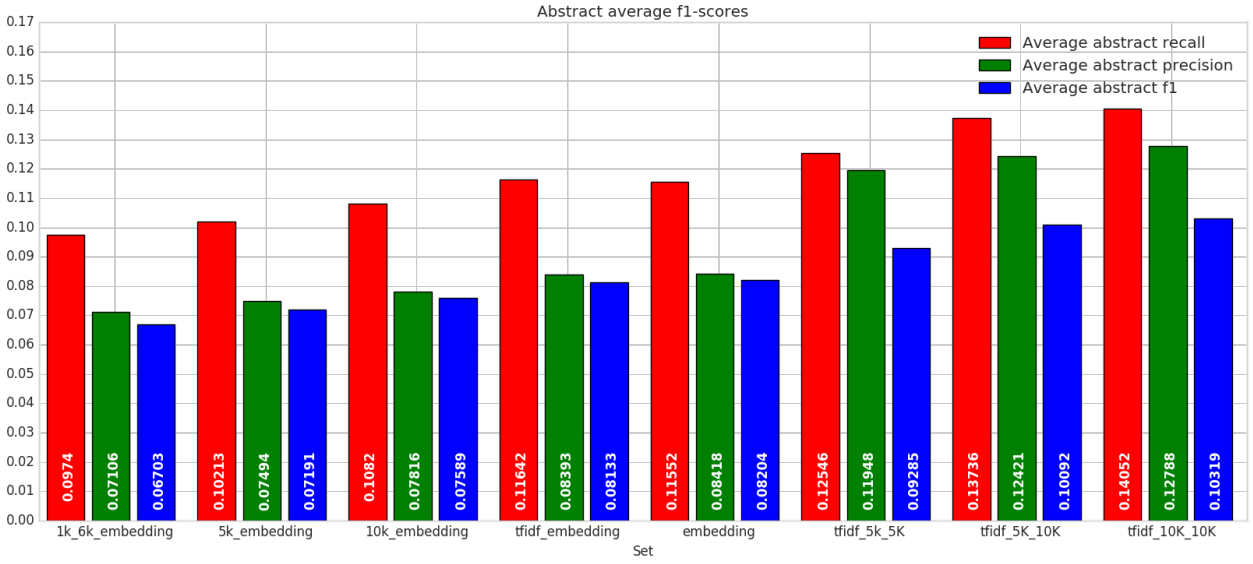




F1-Score

The graphs XXX and XXX show the F1 score for the title and abstract respectively. These scores are the average journals scores for each set, indicating the quality of the sets for absolute hits (top-1).





Memory usage
XXX shows the total memory usage of each *Validation set*, indicating their costs in memory storage & RAM.

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

Sets

Best performers
The data 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 (7.14%) on abstract and 3 median ranks (8.57%) on title. For the embeddings the TF-IDF weighted embedding works better than the others, although it is has is not a significant improvement compared to the default embeddings, which 1 median rank higher on abstract, and equal on title.

TF-IDF
The TF-IDF feature vectors outperform the embeddings on the abstract, while the embeddings ourperform the TF-IDF feature vectors on the title. The main difference between the abstract & title is that the title contains less unique tokens (see XXXXX), this means that the titles contain less unique information, making them more generic. Due to this, the TF-IDF method, which is purely based on word-occurrences & counts cannot differentiate between the highly similar titles. The TF-IDF method works better on the abstract, which contain more unique words, which help in differentiating the different texts. Furthermore, increasing the vocabulary size increases the performance of the TF-IDF, since the vocabulary size is cut-off at word occurrences, which reduces the amount of unique words.

Embedding

- Limited tfidf embeddings
The limited TF-IDF embeddings all under perform, compared to the non-limited TF-IDF embedding, on the median and average ranking. Indicating that the noise reduction is too much, and it removes meaningful words. If the noise reduction would be too low, we would only see a slight increase or none at all. However, the rank lowers, indicating the reduction in embedding quality due to missing words. However, graphs XXX and XXX show that their rank distribution is different from the other embeddings. Their pattern show a decent performance indicates the following pattern: a high/average performance on the top-rankings, an under performance 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 (Graph XXX), on which the limited TF-IDF embeddings are the top performance. Indicating a better performance on the top-1 articles compared to the other sets. This leads us to believe that the cut-off was effective, but that is did not suit our purpose. The cut-off moved the "middle-ranked" articles to either the high end of the rankings, or the lower end. Resulting in low median and average scores, but in (relatively) high accuracy scores. The reduction in vocabulary size did not reduce the storage size for the embeddings, except for the 1K-6K embedding. This indicates that only the 1K-6K cut actually removes entire titles and/or abstracts, since all vectors are stored as dense-vectors[4]. This results in a lower memory requirement.

TF-IDF & 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.

Memory usage

[4]Dense vectors are bigger in memory, since they store all their values, including zeros. However they can be processed more efficiently during calculations

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, 10K/10K uses 11.61 GB, which is 270.93% of the storage size needed for the embedding. The closest TF-IDF configuration we used was 1K/1K, 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 titles, small texts, which contain limited information

2. TF-IDF performs better than article based embeddings on abstracts, larger texts, which contain more information

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

Future work
- How do unique token count, token count and vocabulary size relate for TF-IDF? - Can high quality word similarity sets be created from embedding for future benchmarking? - Can embeddings give a visual representation of the evolution of scientific research over time? - Does TF-IDF improve when top-occurring words are cut-off?

References

[1] Paul Covington, Jay Adams, and Emre Sargin. Deep neural networks for youtube recommendations. In *Proceedings of the 10th ACM Conference on Recommender Systems*, pages 191–198. ACM, 2016.

[2] 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.

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

[4] 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.

[5] 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.

[6] 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.

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

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

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

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

[11] 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.

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

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

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

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

[16] Martin Wiegand, Saralees Nadarajah, and Yuancheng Si. Word frequencies: A comparison of pareto type distributions. *Physics Letters A*, 2018.

[17] Stefan Thurner, Rudolf Hanel, Bo Liu, and Bernat Corominas-Murtra. Understanding zipf's law of word frequencies through sample-space collapse in sentence formation. *Journal of the Royal Society Interface*, 12(108):20150330, 2015.