Document Embedding for Scientific Articles: Validation of word embeddings

Arjan Meijer, 11425555

June 15, 2018

Abstract

1 Introduction

2 Background

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.

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[1]. 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. [2], 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. [3].

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[4]. 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[4, 5].

Paragraph vectors
Variations on the word2vec model have also been proposed, Le and Mikolov [6] 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 [6].

GloVe
Pennington et al. [5] 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 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.

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. [7] 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.

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[1, 5, 8, 9].

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

Table 1: Word analogies examples

Both this validation technique and the Word Similarity technique use vector distance calculations to validate the embeddings, this can therefore also be written as:

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

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. [3], the MEN test collection by Bruni [10] and the WordSimilarity-353 test collection by Gabrilovich [11]. These sets, among others, have been used in multiple studies of word embeddings[5, 8]. 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

[12] 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 [6] 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. [9] and Hinton and Roweis [13] 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[2, 5, 7, 9, 12] 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 [14] 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.

Motivation

In-domain embeddings and validation
Earlier research, concerning domain specific articles, by Truong [14], 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. [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. 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 [14] 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 [14].

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 .

Related work

Deep Neural Networks for YouTube Recommendations
The paper 'Deep Neural Networks for YouTube Recommendations' by Covington et al.[15] describes the current (16 September 2016) YouTube candidate generation and ranking neural networks. The candidate generation network works with limited data inorder to reduce the needed time for the candidate generation. This process reduces the amount of possible videos for the ranking network, which uses more parameters to increase the accuracy of the ranking. The paper also discusses variables to influence the results of the networks, such as age of the training example and watch-time vs click-through prediction. The paper also discusses the impact that layers of depth have at the results.

Distributed Representations of Sentences and Documents
The paper 'Distributed Representations of Sentences and Documents' by le et al.[6] proposes the Paragraph Vector as a replacement for the popular bag-of-words or bag-of-n-grams. 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. The Paragraph Vector out preforms the state-of-the art techniques on several text classification and sentiment analysis tasks.

Distributed Representations of Words and Phrases and their Compositionality
The paper 'Distributed Representations of Words and Phrases and their Compositionality' by Mikolov et al.[1] presents several extensions that improve the quality of vector training and its speed. This is achieved by introducing Hierarchical Softmax, Negative Sampling and the subsampling of frequent words to the Skip-gram approach. This does not only speed up the learning process, but also improves the quality for rare words.

Document Embedding with Paragraph Vectors
The paper 'Document Embedding with Paragraph Vectors' by Dai et al.[9] describes a larger/more thorough evaluation of the Paragraph Vectors model. The writers compare this technique with the Bag of Words model and the Latend Dirichlet Allocation model. The paper states that the Paragraph Vectors model preforms well for grouping, triplet finding and related object/article finding tasks. The paper states that the Paragraph Vectors model is significantly better then the other models in these tasks on the Wikipedia and arXiv articles.

Stochastic Neighbor Embedding
The paper 'Stochastic Neighbor Embedding' by Hinton et al.[13] describes a probabilistic approach to transform objects form a high-dimensional space to a low-dimensional spaces so that neighbour identities are preserved. To achieve this the objects are transformed to images. A Gaussian is then used to define the (context) probability. This method has the ability to be extended to mixtures in which ambiguous high-dimensional objects can have several widely-separated images in the low-dimensional space.

An empirical evaluation of doc2vec with practical insights into document embedding generation
The paper 'An empirical evaluation of doc2vec with practical insights into document embedding generation' by Lau et al.[12] empirically evaluates the quality of the documents embedding created with the method 'doc2vec' compared to 'word2vec' and the 'n-gram' model. They conclude that doc2vec preforms well and that the 'dbow' approach works better than the 'dmpv' approach for the doc2vec method. They also provide recommendations for the optimal parameters for the doc2vec method.

Efficient estimation of word representations in vector space
The paper 'Efficient estimation of word representations in vector space' by Mikolov et al.[8] shows that, with newly applied techniques, the size of the training sets can increase while the time needed for the training operation does not increase. This new approach greatly reduces the training time per word and allows for bigger sets of texts. This results in a higher accuracy of the created embeddings. The authors state that the newly applied techniques can also be used on other, already existing embedding methods.

GloVe: Global Vectors for Word Representation
The paper 'GloVe: Global Vectors for Word Representation' by Pennington et al.[5] proposes a new model, named GloVe, which is a log-bilinear regression model for unsupervised learning of word representations. This model out preforms other models on three specific tasks, these are: word analogy, word similarity and

named entity recognition.

MapReduce: Simplified data processing on large clusters
The paper 'MapReduce: Simplified data processing on large clusters' by Dean et al.[16] explains the MapReduce programming model. The model is inspired by functional programming languages. It requires programmers to specify a map function which processes key-value pairs to generate intermediate key-value pairs. It requires furthermore a reduce function, that merges all intermediate values associated with the same intermediate key. This method allows for an execution on a large distributed computer network. This allows the processing of large input (and output) sets in reasonable time.

Spark SQL: Relational data processing in spark
The paper 'Spark SQL: Relational data processing in spark' by Armbrust et al.[17] explains Spark SQL. 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[1].

How to Generate a Good Word Embedding?
Paper by Lai et al.[2]

- Word embedding, also known as distributed word representation, can capture both the semantic and syntactic information of words from a large unlabeled corpus and has attracted considerable attention from many researchers. In recent years, several models have been proposed, and they have yielded state-of-the-art results in many natural language processing (NLP) tasks.
- We observe that almost all methods for training word embeddings are based on the same distributional hypothesis: words that occur in similar contexts tend to have similar meanings.
- Training on a large corpus generally improves the quality of word embeddings, and training on an in-domain corpus can significantly improve the quality of word embeddings for a specific task.
- Previous works have shown that models that predict the target word capture the paradigmatic relations between words
- we can conclude that using a larger corpus can yield a better embedding, when the corpora are in the same domain
- In most of the tasks, the influence of the corpus domain is dominant. In different tasks, it impacts performance in the different ways
- The corpus domain is more important than the corpus size. Using an indomain corpus significantly improves the performance for a given task, whereas using a corpus in an unsuitable domain may decrease performance.

Better Word Representations with Recursive Neural Networks for Morphology
Paper by Luong et al.[3]

- The use of word representations or word clusters pretrained in an unsupervised fashion from lots of text has become a key "secret sauce" for the success of many NLP systems in recent years, across tasks including named entity recognition, part-ofspeech tagging, parsing, and semantic role labeling.
- The main advantage of having such a distributed representation over word classes is that it can capture various dimensions of both semantic and syntactic information in a vector where each dimension corresponds to a latent feature of the word. As a result, a distributed representation is compact, less susceptible to data sparsity, and can implicitly represent an exponential number of word clusters.
- The Rare-word dataset introduced by Luong et al.

[1]Based on user feedback

Evaluation methods for unsupervised word embeddings
Paper by Schnabel et al. [7]

- Neural word embeddings represent meaning via geometry. A good embedding provides vector representations of words such that the relationship between two vectors mirrors the linguistic relationship between the two words.
- Existing schemes fall into two major categories: extrinsic and intrinsic evaluation. In extrinsic evaluation, we use word embeddings as input features to a downstream task and measure changes in performance metrics specific to that task. Examples include part-of-speech tagging and named-entity recognition (Pennington et al., 2014). Extrinsic evaluation only provides one way to specify the goodness of an embedding, and it is not clear how it connects to other measures. Intrinsic evaluations directly test for syntactic or semantic relationships between words (Mikolov et al., 2013a; Baroni et al., 2014).
- This is the first paper to conduct a comprehensive study covering a wide range of evaluation criteria and popular embedding techniques. In particular, we study how outcomes from three different evaluation criteria are connected: word relatedness, coherence, downstream performance.
- Embedding methods should be compared in the context of a specific task, e.g., linguistic insight or good downstream performance.
- We observe that word embeddings encode a surprising degree of information about word frequency. We found this was true even in models that explicitly reserve parameters to compensate for frequency effects. This finding may explain some of the variability across embeddings and across evaluation methods. It also casts doubt on the common practice of using the vanilla cosine similarity as a similarity measure in the embedding space.
- Extrinsic evaluations measure the contribution of a word embedding model to a specific task. There is an implicit assumption in the use of such evaluations that there is a consistent, global ranking of word embedding quality, and that higher quality embeddings will necessarily improve results on any downstream task. We find that this assumption does not hold: different tasks favor different embeddings
- Performance on downstream tasks is not consistent across tasks, and may not be consistent with intrinsic evaluations. Comparing performance across tasks may provide insight into the information encoded by an embedding, but we should not expect any specific task to act as a proxy for abstract quality.
- Word frequency information in the embedding space also affects cosine similarity
- Also, the above results mean that the commonly-used cosine similarity in the embedding space for the intrinsic tasks gets polluted by frequency-based effects
- Factors such as word frequency play a significant and previously unacknowledged role. Word frequency also interferes with the commonly-used cosine similarity measure.

Word frequencies: A comparison of Pareto type distributions
Paper by Wiegand et al. [18]. They state that the pareto distribution offers a better fit to the word occurrences in natural language than zipf's law. Their models show that, due to the additional parameters in the pareto-III, the tail of the data fits better to the model. This shows the relation between the word occurrences rank, and the actual occurrences.

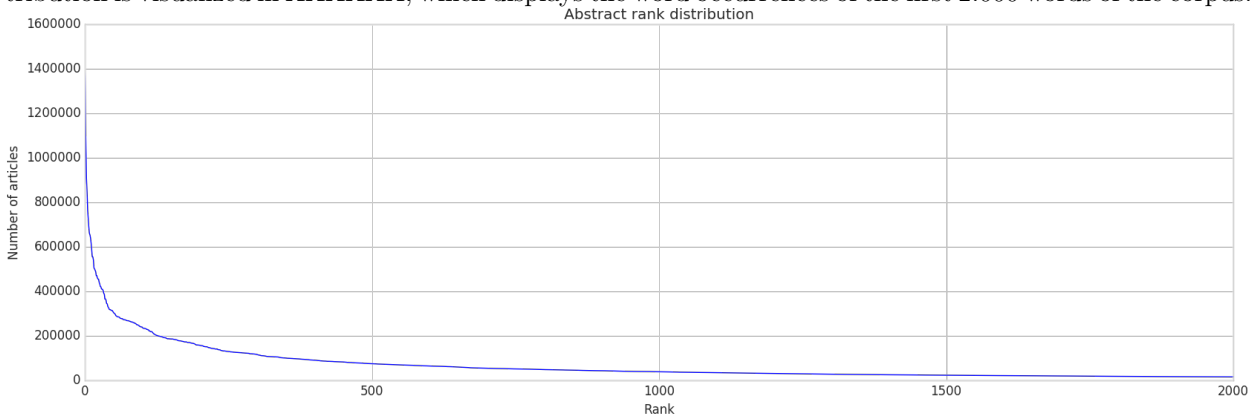
3 Work context

Data
The dataset we used for this research consists of all articles[2] 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. During this research we only worked with the tokenized data,

	Total word count	Total token count	Unique token count
Title	18.822.399	14.742.192	230.805
Abstract	264.653.020	171.474.473	738.961
Total	283.475.419	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. [18]. This distribution is visualized in XXXXXX, which displays the word occurrences of the first 2.000 words of the corpus.



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[19]. 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[19], 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.

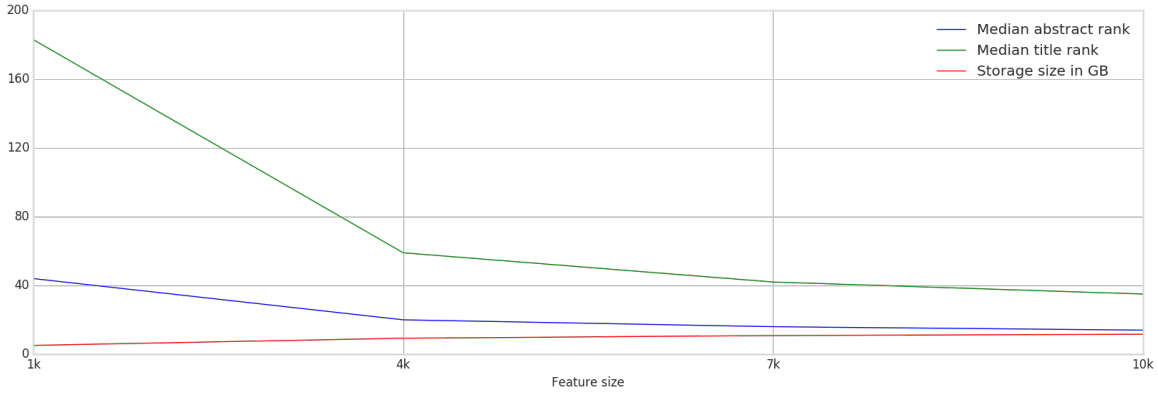
Tokenization

[2]From Elseviers corpus

== TODO ==

TF-IDF selection

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[20]). 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

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:

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

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

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

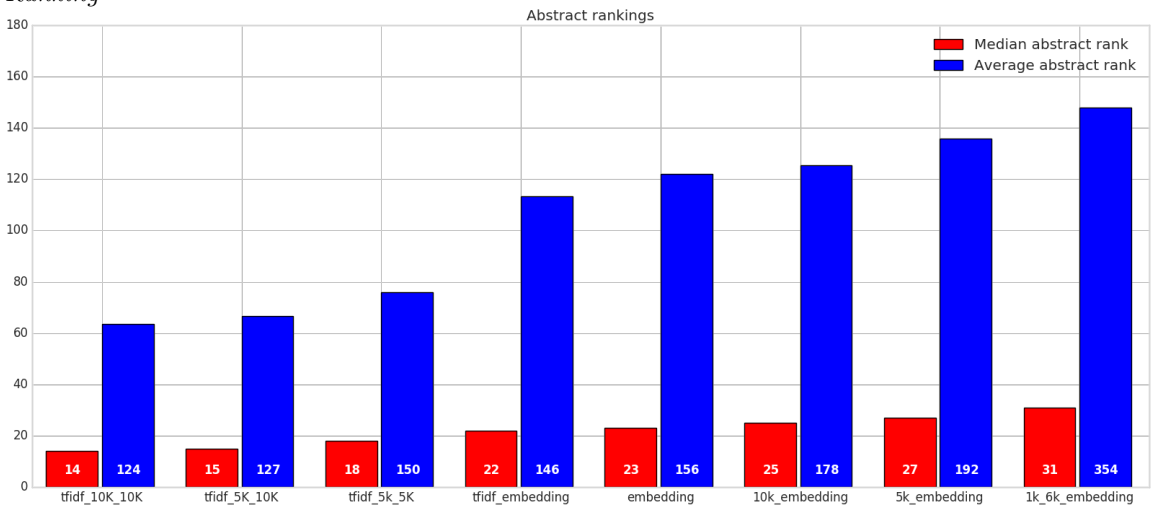
Median & average rank
We use the median rank to indicate around which rank the 'standard' article would be ranked, based on its title or abstract. We do this by taking the median of the respective rank from each article. This gives us an indication of the behaviour of most articles in our validation 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.

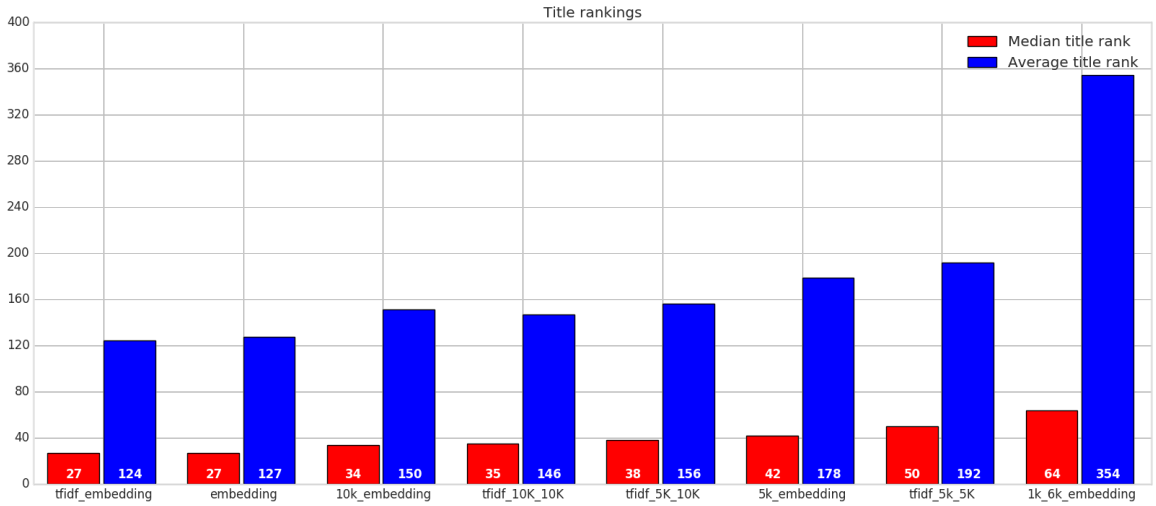
Rank distribution
To further analyse the ranking results, we plot the rank distribution to get an indication of the ranking-landscape. We limit ourselves to the following categories: 1 (absolute hits), top-10, top-20, top-30, top-40, top-50, top-100 and 100+.

4 Results

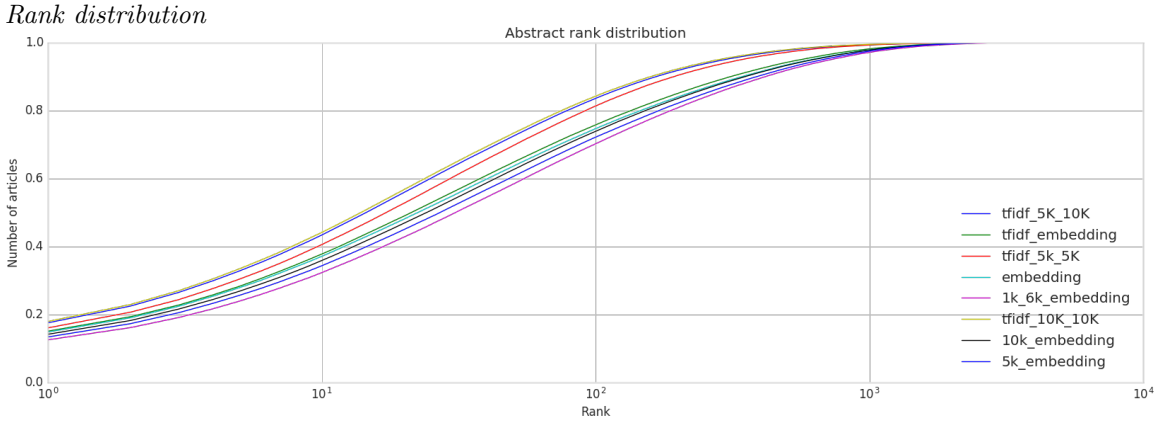
Research results

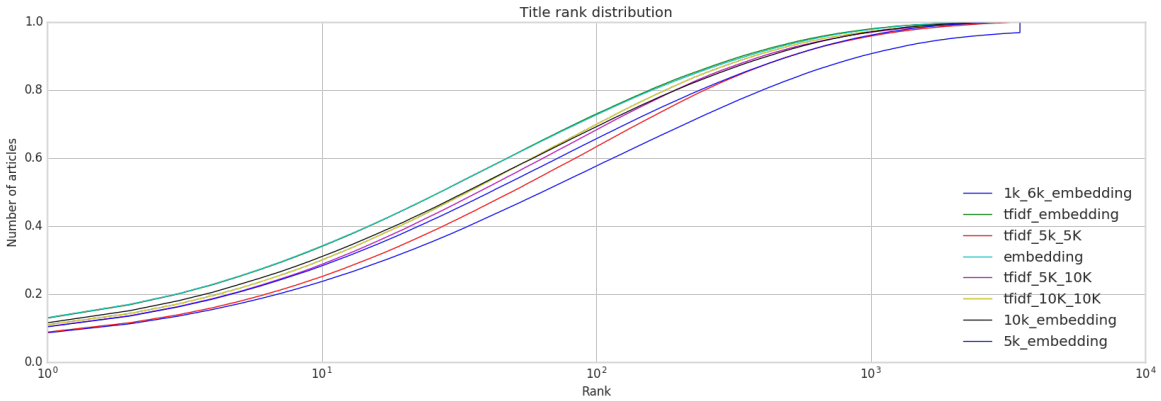
Ranking



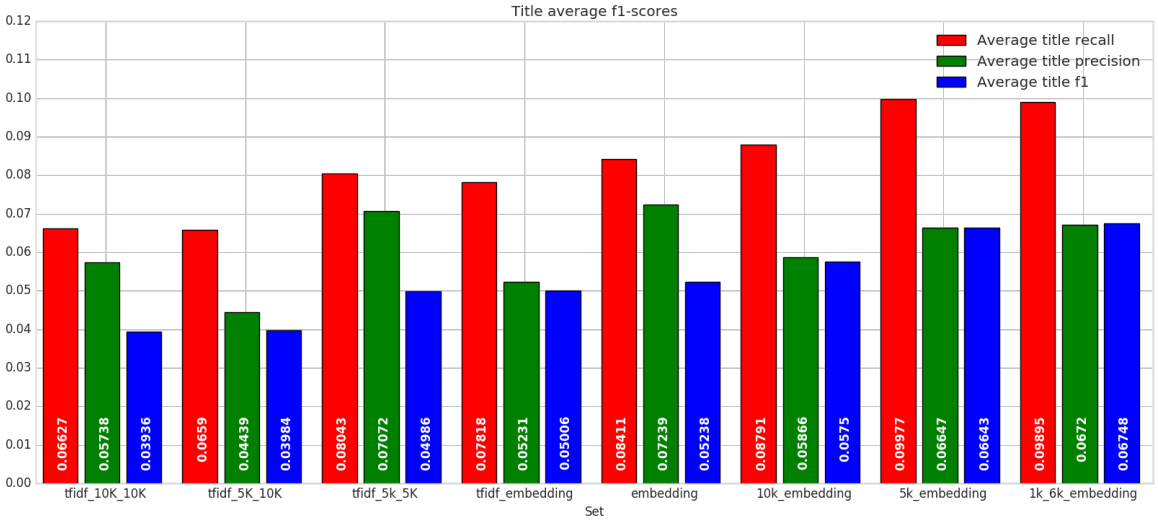


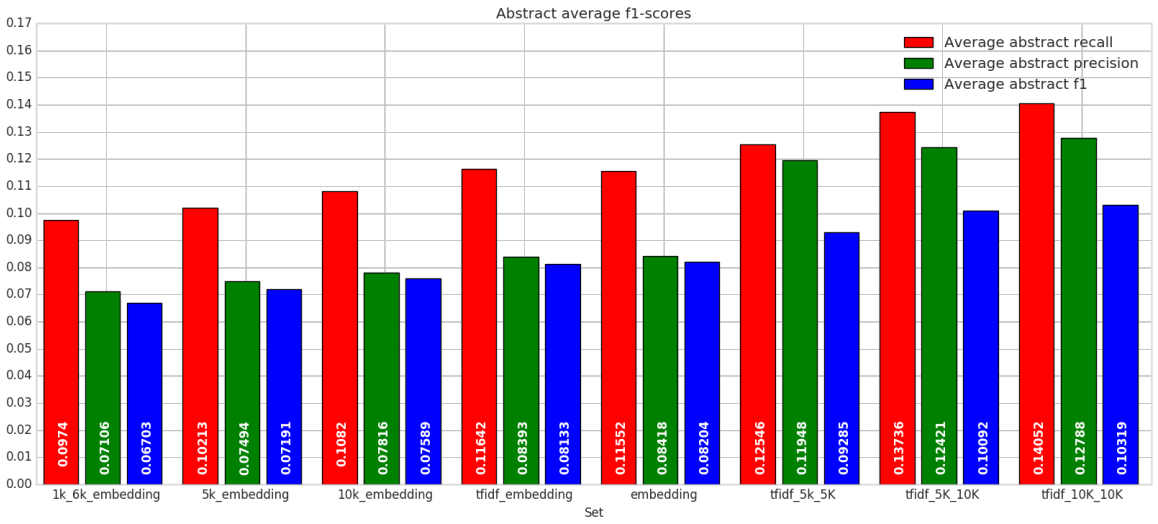
Rank distribution





F1-Score





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

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
minkowski	27	127	23	122
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

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] 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.

[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] 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.

[4] 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.

[5] 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.

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

[7] 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.

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

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

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

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

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

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

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

[15] 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.

[16] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008.

[17] Michael Armbrust, Reynold S Xin, Cheng Lian, Yin Huai, Davies Liu, Joseph K Bradley, Xiangrui Meng, Tomer Kaftan, Michael J Franklin, Ali Ghodsi, et al. Spark sql: Relational data processing in spark. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, pages 1383–1394. ACM, 2015.

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

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

[20] ScyPy. Distance computations. URL `https://docs.scipy.org/doc/scipy-0.14.0/reference/spatial.distance.h`