Document Embedding for Scientific Articles: Validation of word embeddings

H.J. Meijer,
11425555

NOTE: This is not a final version, the contents of this thesis may still change.
== This thesis is not suitable for submission ==

June 29, 2018

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

Contents

1 Introduction and Background

1.1 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, an online search for "Information Retrieval" (initial piece of information) will give you a list of results (relevant information). To be able to do this, the search engine must know which texts are related.

This can be achieved with, and without neural networks, a traditional technique which does not use a neural network is TF-IDF, shot for Term Frequency - Inverted Document Frequency. Techniques that use a neural network are newer, these are (among others) Word2Vec, Paragraph Vectors and GloVe.

1.2 Neural Network

A complete in-depth background into neural networks is beyond the scope of this thesis, therefore we will only describe the simplified working of a neural network and its basic application in IR. Neural Networks are multi-layered computational software, based on matrix transformations, which tries to map input values to output values. This mapping uses a pre-defined amount of layers that modify the values through matrix transformations. Neural networks rely on training to create optimal values for these layers. The values in the layers are only changed at training time; high-quality training data is therefore essential for neural networks. Neural networks can be applied to many different tasks, as long as there is sufficient training data available. The 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 all input and output sets, without under performing on other sets[1].
Neural Networks are used, in the creation of word embeddings, 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 quick brown fox jumps over the lazy dog"

the Neural Network can be trained to either predict the words around "jumps", which we refer to as context words (the, quick, brown, fox, over, the, lazy, dog). It can also predict the word "jumps", given the context words (the, quick, brown, fox, over, the, lazy, dog). This result in a matrix, a vector per word which is referred to as a word embedding. This embedding indicates "word relatedness" which, as mentioned earlier enables association (thus retrieval) with related(relevant) texts. It is to be noted that the word embeddings are only able to represent word relatedness to the other words trained in the same run. Due to the random initialization of the initial values of the layers in the neural networks, neural networks do not produce every run the same results. Therefore, word embeddings from different runs of the same neural network cannot be compared.

1.3 Embedding

An embedding is a distributed, numerical representation of text in a multi-dimensional vector space[2] which can capture both the semantic and syntactic information[1]. In the case of a word embeddings, the embeddings represent the words. These embeddings are created using machine learning models, which do not have the need for human interaction [2], they are so-called *unsupervised learning algorithms*. Once trained, the embeddings can also be used to construct embeddings for collections of texts. For example, a word embedding can be used to create a sentence, paragraph, document or corpus embeddings. The usage of word embeddings has improved various Natural Language Processing areas such as named entity recognition, part-of-speech tagging, parsing, and semantic role labelling Luong et al. [3].

[1] Having only good results for the training/validation set is know as overfitting. This means that the layer-values are fine-tuned to only perform (extremely) well on one set, while under performing on others

[2] The vector spaces of separately trained word embeddings differ, since each run the initial values of the neural network are randomly initialized. This means that the same word trained in two separate runs do not have to have the same embedding. However, they will have the same relationship to other words trained in their respective runs.

1.4 Text analysis techniques

To enable a computer to process text, for embedding creation or other 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 a 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 learned 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]. This is illustrated in figure 1, in this figure, $v(w)$ represents the target word, and $v(w...)$ represent the context words. Mikolov et al. [1][6] 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 [7] which creates document embeddings instead of word embeddings.

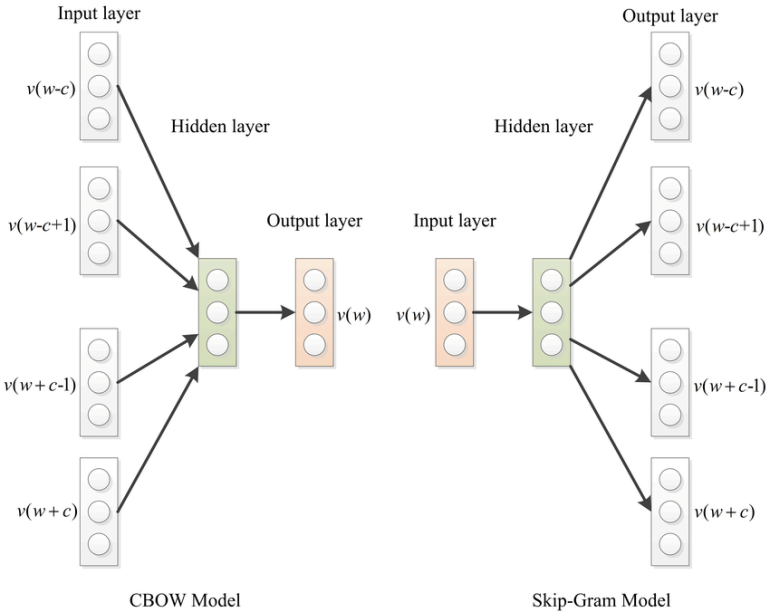

Figure 1: Illustration of the CBOW and Skip-Gram models, as presented by Chen et al. [8]

Paragraph vectors

Variations on the word2vec model have also been proposed, Le and Mikolov [9] introduced the paragraph vector 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 a matrix, 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, at least in a small context [9]. Dai et al. [10] state that the Paragraph Vectors model performs significantly better than other models on grouping, triplet finding and related object/article finding tasks on Wikipedia and arXiv texts.

GloVe

Pennington et al. [5] introduced the Global Vectors (GloVe) 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 outperforms other models on word analogy, word similarity and entity recognition according to the findings

Term	Frequency (TF)	Document Frequency (DF)	Inverted Document Frequency (IDF)	TF-IDF score
Exponential	4	15	0.824	3.296
Occurrence	1	20	0.699	0.699
Multitude	1	40	0.398	0.398
Abstract	100	100	0	0

Table 1: Example of TF-IDF score calculation.

by Pennington et al. [5].

TF-IDF

Term Frequency * Inverted Document Frequency (TF-IDF) is a method that 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. The text, provided to the algorithm, is analyzed on word occurrences on corpus and document level, this results in a score per word. This score can be converted to a feature vector, in which the indexes of this vector represent the words and the value is the TF-IDF score. The size of this feature vector can optionally be controlled by hashing the words in the text, which can then be reduced to a given size. If this is not applied, the size of the feature vector is equal to the number of unique words in the corpus, also known as the (corpus) vocabulary. Some applications of this technique limit the number of unique words in the text supplied to the TF-IDF algorithm, they do this by taking only a certain amount of top words, ordered on their occurrence. This reduces the amount of storage needed when hashing is not applied. It furthermore reduces the number of words which occur rarely. Due to the nature of TF-IDF, these rare words have a high score, cancelling out other more frequent words, while rarely occurring in the corpus.

As Lai et al. [2] state in their paper "How to generate a good word embedding?", that 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 has more influence on this than the corpus size.

1.5 Validation methods

The results produced by the previously mentioned 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[3] and, for classification tasks, the rank of the correct class, which is in this research a journal. The embeddings can furthermore be validated through their performance on tasks such as word analogy, word similarity, categorization and embedding visualization. These tasks can be designed to produce a score that indicates 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, findings about the performance of an embedding method are limited to the task on which they are tested. Their results cannot be generalized to state that the embeddings are overall "performing well". Validation tasks use either labeled or unlabeled data. Labeled data is data that is in some way marked so that the correct answer can be derived from it, while this is not possible with unlabelled data. Labeled data enables the use of unsupervised learning models since *a system or model which is not involved in the actual learning* can automatically validate the given results and use it as feedback for the next learning iteration. Unlabelled data, on the other hand, requires human interaction to give a score for the produced result. The validation tasks can also be divided into two groups, extrinsic evaluation, ones that use word embeddings as input for a downstream task and Intrinsic evaluation, which directly tests 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 word pairs of four that can

[3]The F1 score combines the precision score and the recall score in a single metric

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[1, 5, 6, 10].

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

Table 2: 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_{King} - $X_{Man} \approx X_{Queen}$ - X_{Women}

This means that the resulting vector of embedding of "King" minus the embedding of "Men" is approximately the embedding of "Queen" minus the embedding of "Women". This resulting vector may, for example, be close to the vector representing "Monarch".

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 [12] and the WordSimilarity-353 test collection by Gabrilovich [13]. These sets, among others, have been used in multiple studies of word embeddings[5, 6]. This validation method is limited by the availability of word similarity sets that share the same domain as the trained embeddings.

Classification
A classification validation method is a simple task which assigns a label to a text. Lau and Baldwin [7] 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 reviews. 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. The advantage of this visualization is that a human can directly see the embedding distribution, and see if it is distributed in a way that seems logical. It gives furthermore insight in the overall spectrum of the embedding. However, this representation does not give an empirical score, since it is not an evaluation of the data, but an alternative representation.

1.6 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 embeddings into two categories, generic embeddings and domain-specific embeddings. The generic 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) or is limited to a small number of topics. Given these terms, we regard the embeddings trained on the Wikipedia corpus[2, 5, 7, 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.

1.7 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. Armbrust et al. [16] state that Spark SQL lets programmers leverage the benefits of relational processing and lets SQL users call complex analytic libraries in Spark. This allows for much tighter integration between relational and procedural processing. The paper further states that Spark SQL makes it significantly simpler and more efficient to write data pipelines that mix relational and procedural processing while offering substantial speedups over previous SQL-on-Spark engines.

1.8 Text properties

Wiegand et al. [17] 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 with 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[18].

2 Motivation

2.1 In-domain embeddings and validation

In earlier research concerning domain specific articles, Truong [15] found that in-domain training of the word embeddings can improve the process of document clustering. The usage of in-domain data is more important 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 fail to confirm this. Truong [15] used multiple word similarity validations to asses the quality of the word embeddings. However, these sets are created to validate the generic embeddings; they fail to asses the quality of the domain-specific embeddings.

2.2 Research

To asses the problem of the limited availability of pre-labeled validation sets for domain-specific articles, we compare the embeddings to TF-IDF on a categorization task. This A) indicates 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. 1.1 Does TF-IDF weighting increase the performance of word embedding on academic texts for article classification?

RQ. 1.2 Can the usage of alternative distance metrics improve the performance of word embedding on academic articles for article classification?

RQ. 2 Can word embeddings, combined with pca[4]-based TSNE, create a two-dimensional plot that preserves the journal relatedness?

Embedding and TF-IDF

RQ. 1 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. This transforms the classification task from a binary metric to a ranking metric. For this task, we will use different versions of embeddings, to answer RQ 1.1, and different TF-IDF versions to not only compare the two techniques but also look for the optimal results of both techniques. To achieve the optimal results, we compare 20 distance metrics from the SciPy library on ranking performance on the standard embeddings. For this part of the research, we will use the following hypothesis:

H. 1 *Embedding based techniques give lower rankings than the TF-IDF based techniques.*

H. 1.1 *TF-IDF weighted document embeddings outperform standard embeddings on the classification of academic articles.*

H. 1.2 *Cosine similarity based ranking results in the best performance for the classification of academic articles.*

H. 1.3 *Word embeddings use less memory while giving better ranking results than TF-IDF on the classification of academic articles.*

By validating or invalidating these hypotheses, we get an indication of the performance of the embeddings compared to TF-IDF, get insight into possible performance and resource trade-off's and get insight into the performance of different distance calculation metrics. RQ. 2 concerns the visualization of word embeddings and the accuracy of this visualization. To answer this research question, we will use the following hypothesis:

H. 2 *Word embeddings, combined with PCA-based TSNE can preserve the journal relatedness on a two-dimensional plot.*

[4]principal component analysis

The validation of this hypothesis will rely on visual confirmation. We expect to see clustering of journals in certain areas, which indicates a research subject. We also expect that articles which are visually close together are closely related by subject.

3 Corpus

The dataset we used for this research consists of articles 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. Details about the corpus can be found in table 3.

	Total count	Unique count	Average length
Title words	18.822.399	939.665	13,53
Title tokens	14.742.192	230.805	10,64
Abstract words	264.653.020	5.853.077	190,19
Abstract tokens	171.474.473	738.961	123,71
Total words	283.475.419	6.209.769	203,71
Total tokens	186.962.354	763.475	134,36

Table 3: Corpus size

The word occurrences follow the pattern of a pareto distribution as described by Wiegand et al. [17]. This distribution is visualized in figure 2, which displays the occurrences of the first 500 tokens of the corpus.

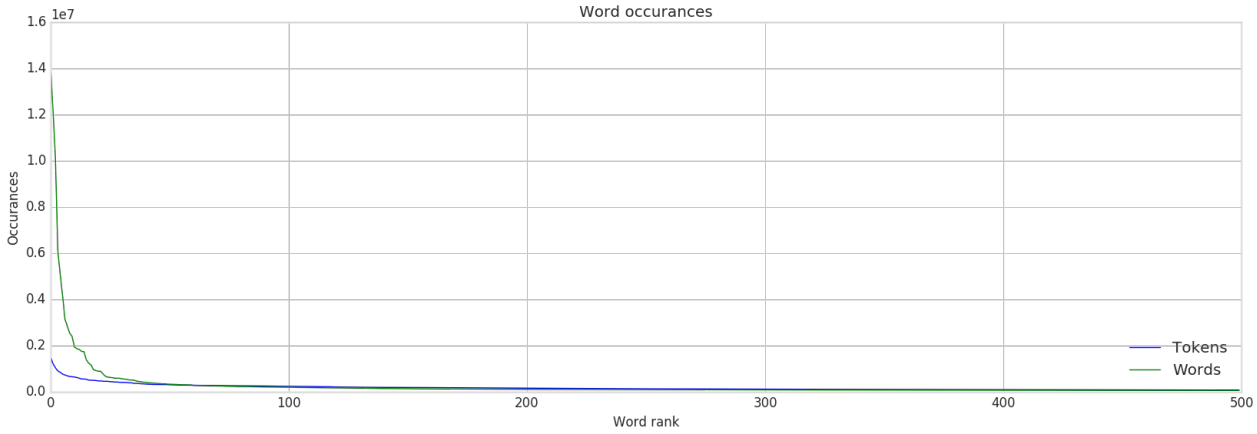

Figure 2: Word and token occurrences

4 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.

4.1 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.

4.2 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 create article embeddings, we take the average of all normalized word embeddings for that article. Journal embeddings are created in the same way. We averaged all the normalized article embeddings to create the journal embeddings. We have used multiple embedding 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 number of tokens it uses. This set was created to see if the limitation to 10.000 tokens reduces the amount of noise, 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 limit the number of tokens more aggressively, 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 off the top 1.000 and cutting off 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.

4.3 TF-IDF

To create the TF-IDF feature vectors, we used the TF-IDF model and a hasher from PySparks 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 equal the number 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: $vocabularysize/hashbucketnumber$. Furthermore, we denote 1.000 as 1K, since we deal with chosen values which can be exactly noted given this notation.

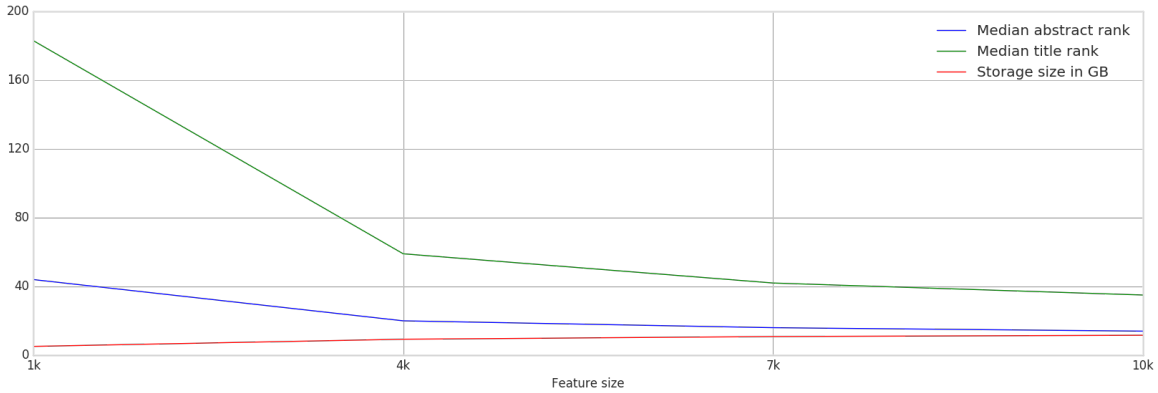

Figure 3: TF-IDF performance and memory usage

Figure 3 shows the performance, as median rank, and storage size, in gigabyte, 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.

5 Pipeline

We procesessed the TF-IDF sets and the embedding sets via the same pipeline, using their common vector properties. This ensures comparable results, the pipeline is set-up as follows:

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

5.1 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.

5.2 Create journal embeddings

From our training set we create the journal embeddings, which are created for most sets[5] by averaging the article embeddings or feature vectors.

5.3 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

[5] see paragraph datasets

- Title similarity between the actual journal and the article

Distance metrics
To calculate the distance between to vectors, cosine similarity is commonly used. 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. Table 4 shows the results of this validation.
These results show high similarity between cosine-based metrics (Cosine & Correlation) and euclidean based

Metric[6]	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

Table 4: Distance metric performance for word embeddings on the categorization of academic texts

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. 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.

5.4 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 at 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 the 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.

6 Two-dimensional plot

To create a plot, we transformed the 300-dimensional journal vectors into 2-dimensional vectors using TSNE based on pca[7]. These 2-dimensional vectors, representing the x & y coordinate, can then be drawn in a plot. To visualize the preservation of journal-relatedness while converting the 300 dimensions to 2 dimensions, we create groupings using k-means. These groupings are created on the 300-dimensional vectors and are visualized in the plot using colors. We use the k-means groupings due to the lack of subject-based grouping values in our dataset.

[7]Principal component analysis

7 Research results

7.1 Ranking

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

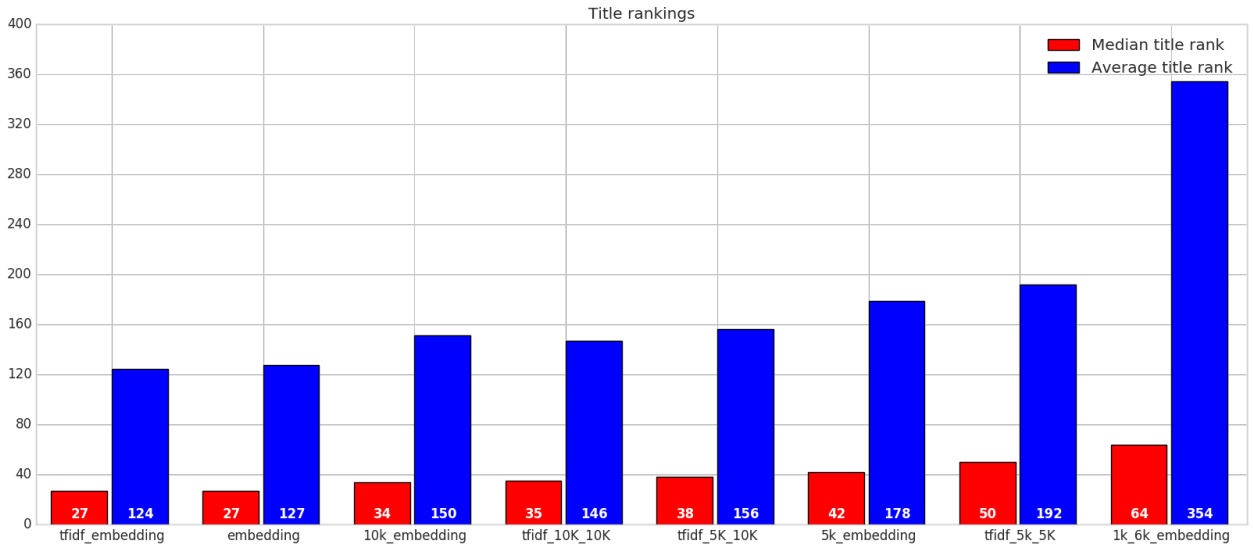

Figure 4: Median and average title rankings

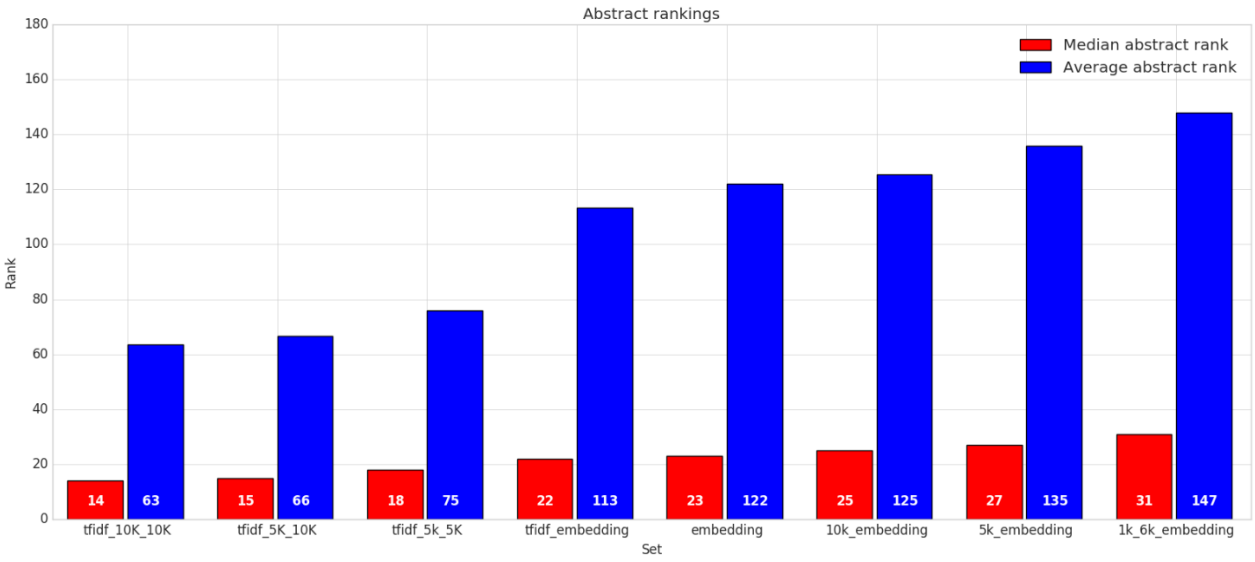

Figure 5: Median and average abstract rankings

7.2 Rank distribution

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

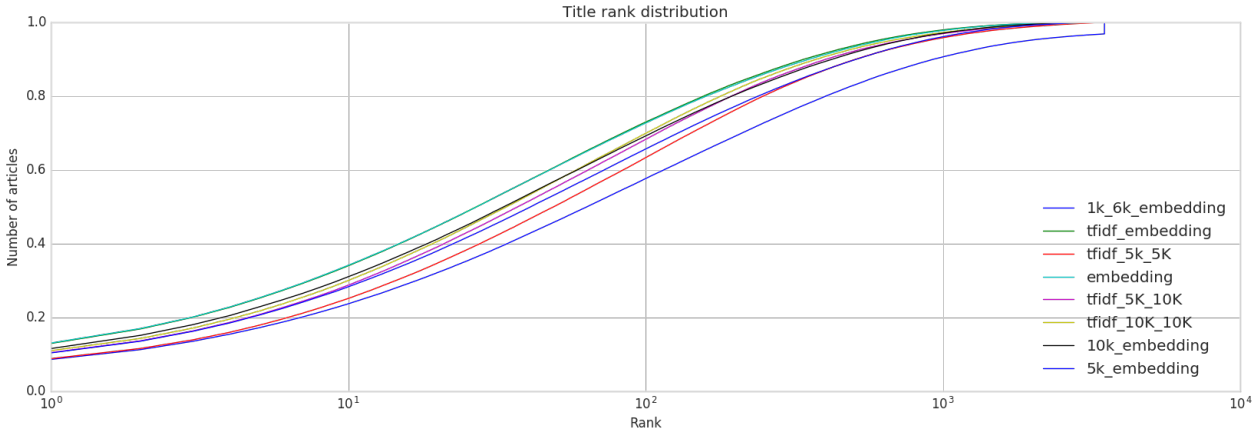

Figure 6: Title rank distribution per set

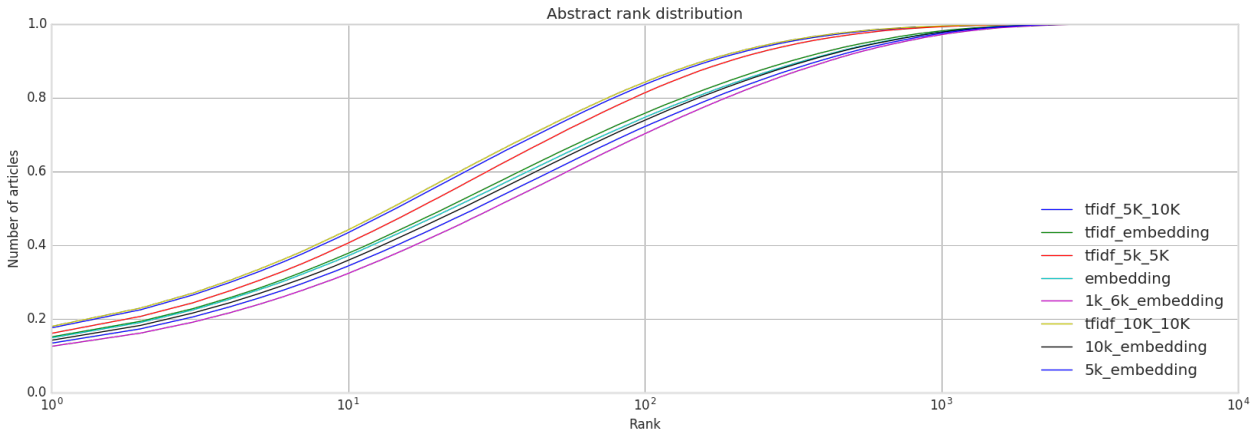

Figure 7: Abstract rank distribution per set

7.3 F1-Score

Figures 8 & 9 show the precision, recall and f1 scores. These scores are calculated on journal level, and are averaged per set. Figure 8 shows the F1 score for the title and figure 9 shows the scores for the abstract. These scores indicate the performance of the sets on absolute hits/top-1.

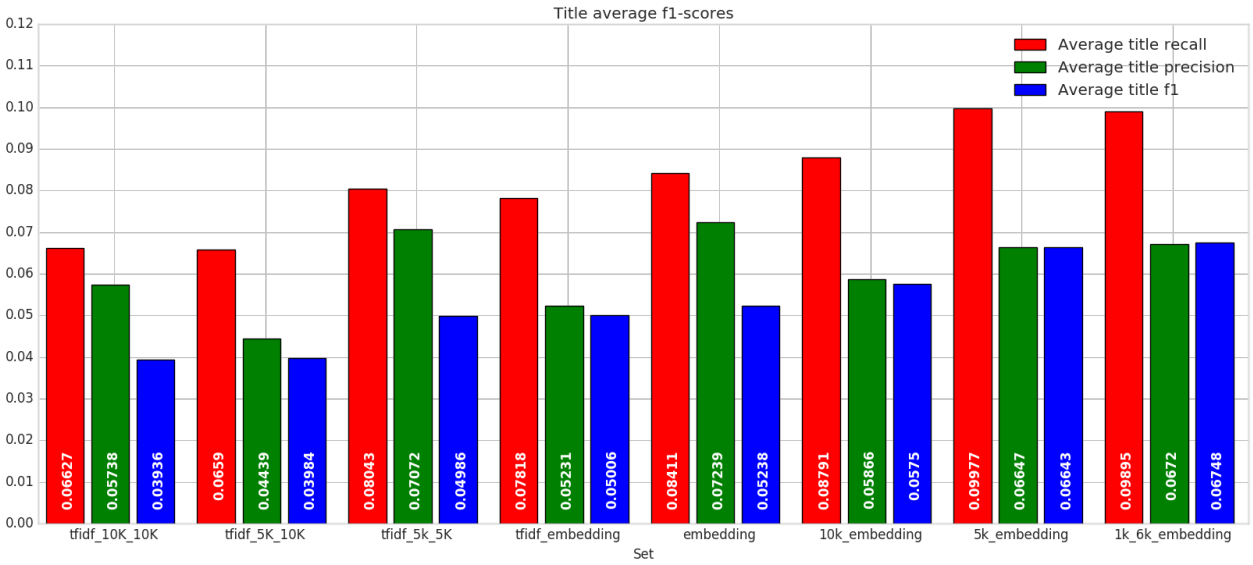

Figure 8: Precision, recall and F1 scores based on title

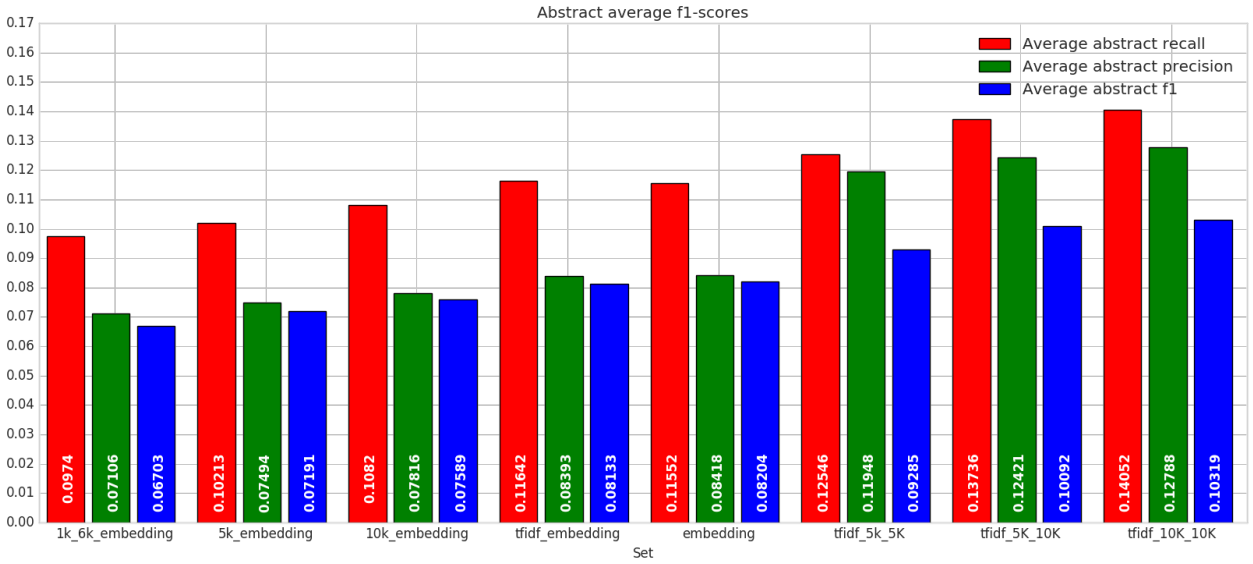

Figure 9: Precision, recall and F1 scores based on abstract

7.4 Memory usage

Table 5 shows the total memory usage of each set for the *Validation set*, indicating their storage costs in gigabytes[8].

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

Table 5: Memory usage and performance for each set

7.5 Journal relatedness plot

We created two seperate journal plot, figures 10 & 12 show the title based embeddings, and figures 11 & 13 show the abstract based embeddings. Figures 10 & 11 show the two dimensional plots of the journal embeddings. The journals are color-marked by publisher. Red is Wiley, lime is Elsevier and blue is Springer Nature. The grey points are other or not-specified publishers. Figures 12 & 13 show the journal embeddings grouped by a k-means algorithm, creating 8 groups. The bottom right shows the names of the journals closest to the center of the group. While most names state the topic of the research field, some do not. These are *RSC Advances*: chemical sciences, and *Symmetry*: research on symmetry phenomena wherever they occur in mathematical or scientific studies. The k-means algorithm ran on the 300-dimensional vectors, while the plot shows the 2-dimensional vectors. In the title-based figures (10 & 12) the journals *PNAS* and *Cell* overlap each others labels on the bottom right side of the graph. In the abstract-based figures (11 & 13) the journals *PNAS* and *Nature* overlap each others label on the bottom right side of the graph. The digital versions of these plots can be found at: `https://goo.gl/nCGMcu` (title-based) and `https://goo.gl/LwYP6k` (abstract-based). The usage of this digital version is advised, since this interactive plot enables zooming and adjusting plot settings, which can give a broader insight into the results than the presented figures. Although the presented figures give sufficient insight for the purpose of this thesis.

[8] 1024 based

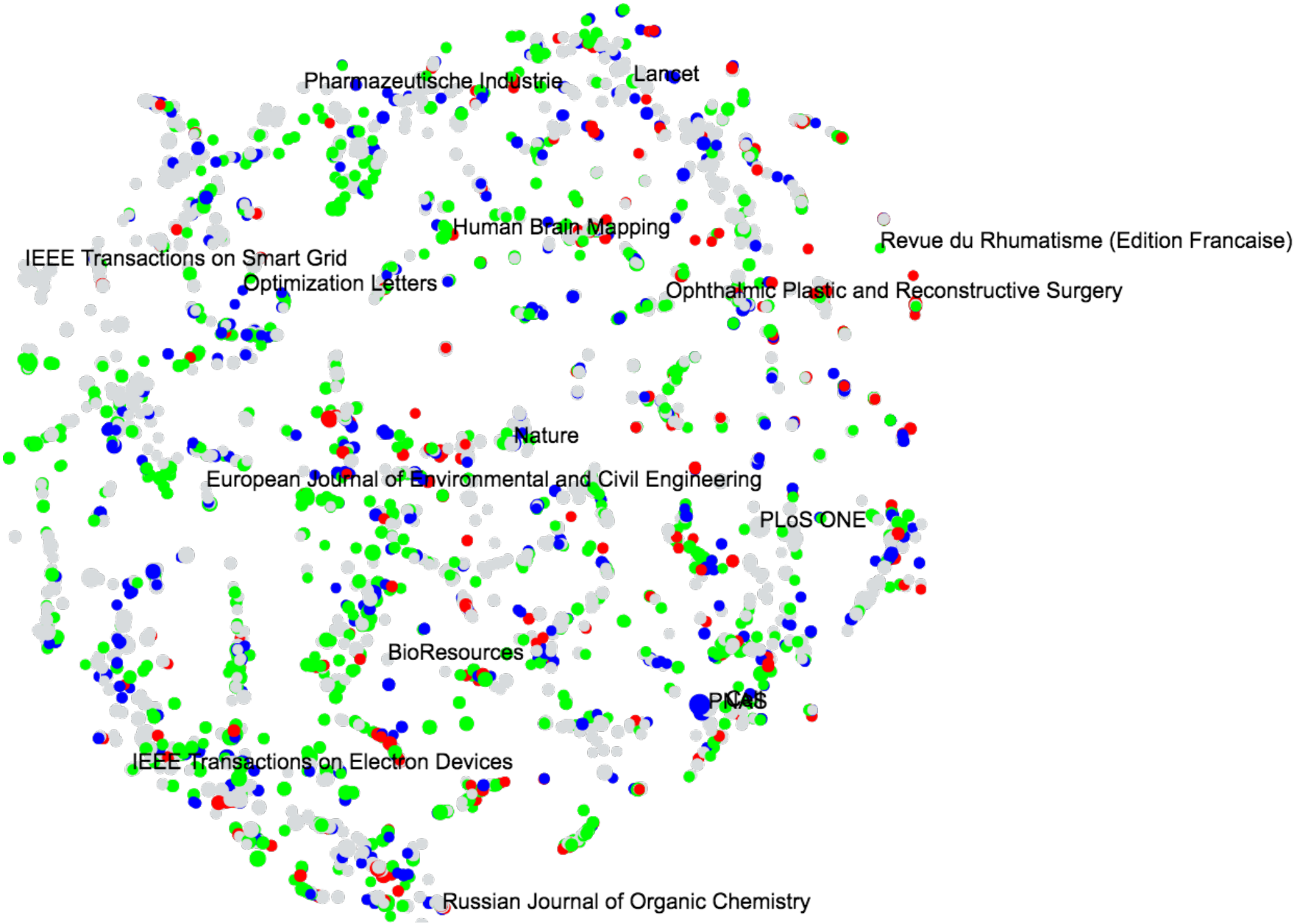

Figure 10: Journal plot of title embeddings, grouped by publisher

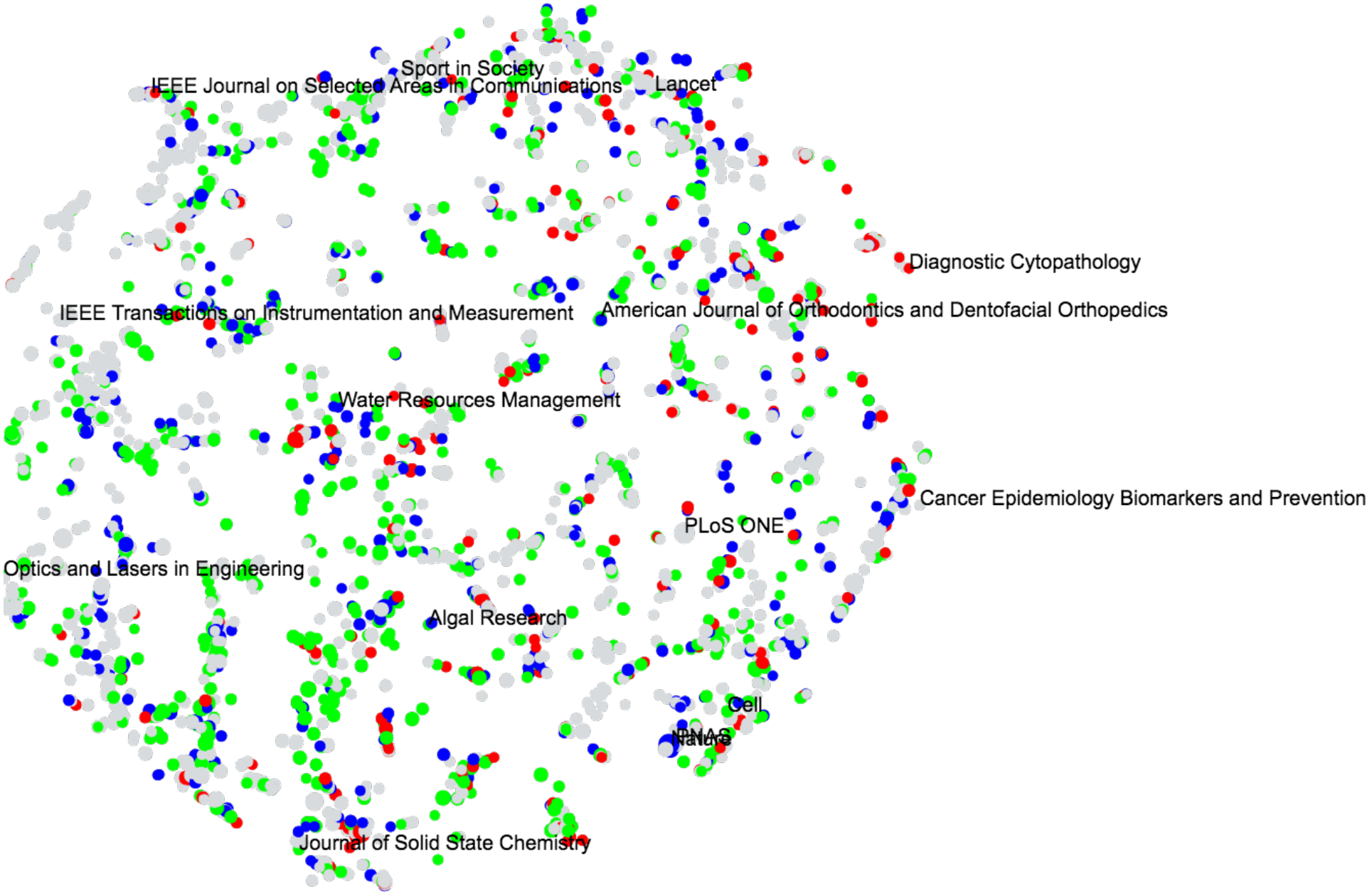

Figure 11: Journal plot of abstract embeddings, grouped by publisher

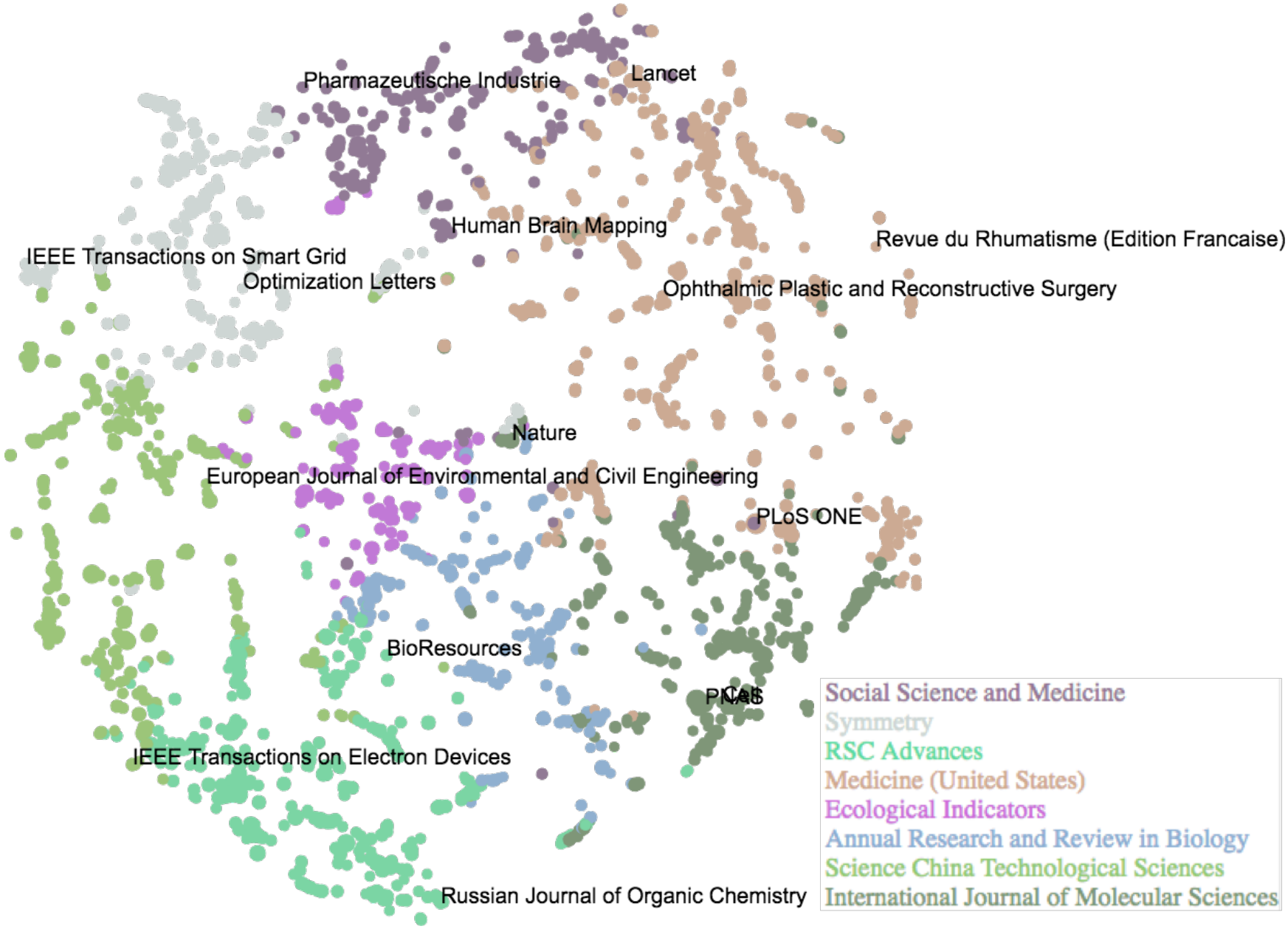

Figure 12: Journal plot of title embeddings, k-means grouped

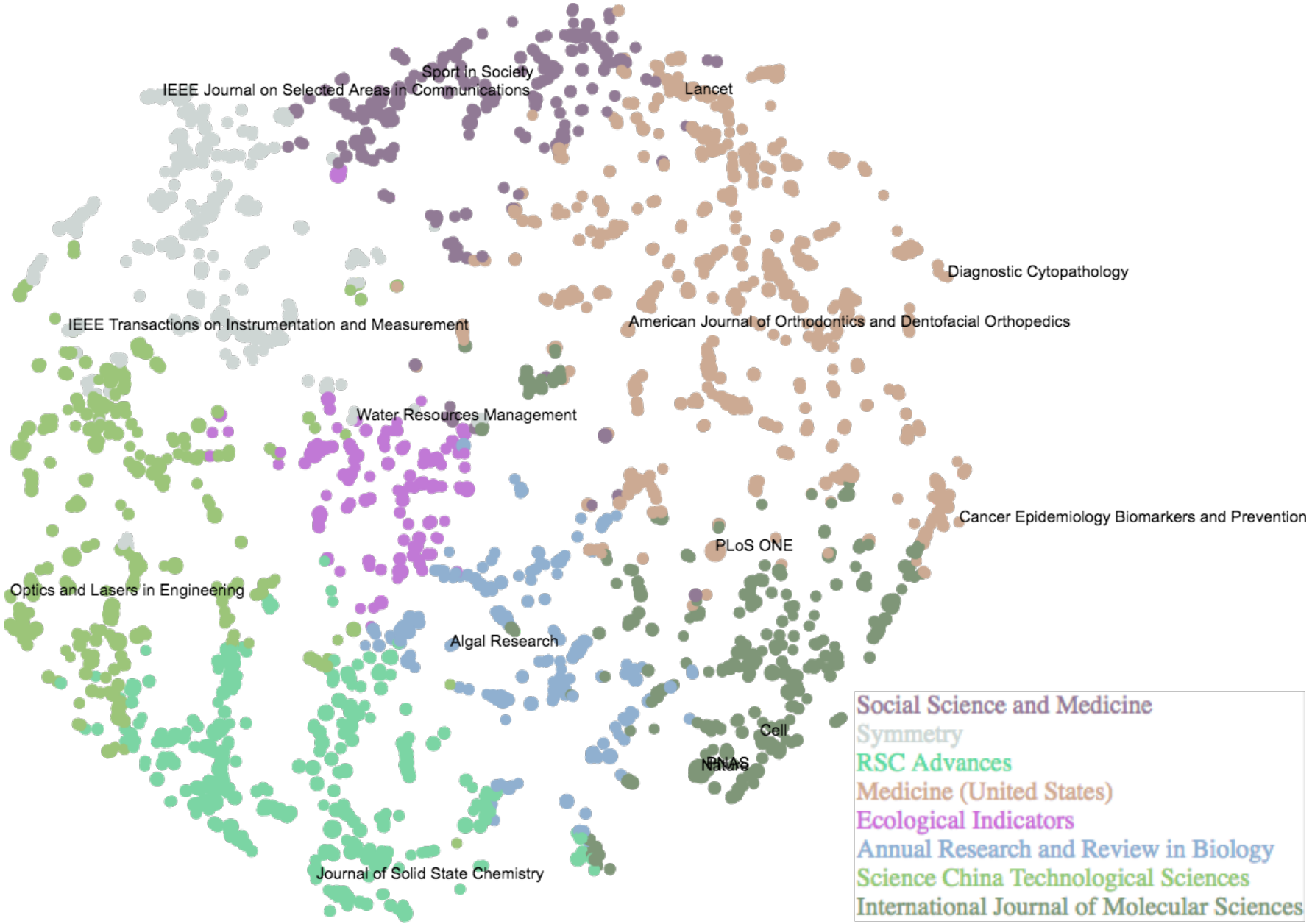

Figure 13: Journal plot of abstract embeddings, k-means grouped

8 Discussion

8.1 Result analysis

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

TF-IDF
The TF-IDF feature vectors outperform the embeddings on the abstract, while the embeddings outperform the TF-IDF feature vectors on the title. The main difference between the abstract & title is that the title contains fewer tokens compared to the abstract (see table 3). This means that the titles contain less information than the abstracts. Due to this, the TF-IDF method, which is purely based on word-occurrences & counts has less information on the titles. The TF-IDF method works better on the abstract, which contain more tokens, which improves the differentiating the different abstract. The data furthermore shows that increasing the vocabulary size increases the performance of the TF-IDF, this means that none of the created cut-off's resulted in cutting off noise, the increasing size of the vocabulary only improved the performance in this case. It could be possible that at higher vocabulary sizes the cut-off would result in a sharper signal, we did not look into this further due to our findings that the TF-IDF performance stagnates (presented in figure 3).

Limited TF-IDF embeddings
The limited TF-IDF embeddings all underperform, 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. This is in line with what we found with the TF-IDF results; higher vocabulary sizes give better performance. However, figures 6 & 7 show that their rank distribution is different from the other embeddings. Their pattern shows a decent performance indicates the following pattern: a high/average performance on the top-rankings, an underperformance on the middle rankings and a resulting stack-up of articles with a high-ranking. This is further supported by the TF-IDF score on titles (figure 4), on which the limited TF-IDF embeddings are the top 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 it did not suit our purpose. The cut-off moved the "middle-ranked" articles to either the higher end or the lower end of the rankings. Resulting in high median and average ranks, and 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 abstracts, since all vectors are stored as dense-vectors[9]. This results in a lower memory requirement.

TF-IDF & embeddings
Our hypothesis on the difference between the TF-IDF and the standard embedding is as follows:
The embeddings seem to outperform the TF-IDF in situations when there is little information available, the titles in our case. This indicates that the embeddings store some word meaning that enables them to perform relatively well on the titles. The abstracts, on the other hand, contain much more information. Our data seems to indicate that the amount of information available in the abstracts enable the TF-IDF to cope with the lack of embedded information. If this is the case, we could expect that there would little performance increase on the title when we compare the embeddings to the weighted TF-IDF embeddings, 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 would also expect on the abstract an increase in performance since the TF-IDF has more information in this context. We would expect that the weighting applied by the TF-IDF

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

improves the performance of the embedding by indicating word importance. Our data shows a minor improvement in performance of 1 median rank and 10 average ranks while these improvements cannot be seen as significant, our data at least indicates that weighting the embeddings with TF-IDF values has a positive effect on the embeddings.

Memory usage
Although the TF-IDF outperforms the embeddings on the abstracts, 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 (as displayed in figure 3). This TF-IDF set has a median title rank of 183 and a median abstract rank of 44. Which is significantly worse than the embedding, which also uses less memory.

Journal plots
Figures 10 & 11 show the journals in the, what we will refer to as "subject spectrum". We do this because the journal embeddings capture journal-relatedness, which leads to the clustering of related articles, which share, in varying degree, the common subject. In this figure, we can see that the publisher Wiley is more active in the right part of the spectrum than the left. We can further see that Nature is far to the center, as we would expect a generic journal to be, but it is on the biology/medicine side of the center (indicated by the position of Cell and Pharmaceutical Research).
Figures 12 & 13 show the journals, grouped with k-means on the original 300-dimensional embedding vector. In the plot we can see that the original embedding-clustering, as provided by the k-means algorithm, is relatively well preserved, most groups stay clustered together. The interactive version shows that the clusterings, as seen on all four figures, are subject based. Journals concerning the same subjects are correctly clustered together, and the journals in between topic clusters are also positioned at logical positions. It furthermore shows that the k-means groupings give insight in research fields, although it is limited by the number of groups the k-means algorithm creates. Our findings on the 2-dimensional representation of embeddings are similar to those of Dai et al. [10], who plotted 4.4 million Wikipedia articles. The major difference between our plot and their plot is that they focus on just a view subjects in an entire plot, and have access to (human-made) partitioning data. They have therefore a more reliable grouping metric.

8.2 Improvements

This research shows that even tough the embeddings can capture and preserve relatedness, TF-IDF is able to outperform the embeddings. Earlier research already proposed improvements to the word embeddings. Dai et al. [10] show that the usage of paragraph vectors improve the accuracy of word embeddings with 4.4% on triplet creation with the Wikipedia corpus and a 3.9% improvement on the same task based on the arXiv articles.
Furthermore, Le and Mikolov [9] show that the usage of paragraph vectors decrease the error rate (positive/negative) with 7.7% compared to averaging the word embeddings on categorizing text as either positive or negative. While this looks promising, we have to keep in mind that our task differs from earlier tasks. We do not categorize on two categories but more than 3k. Still, we would expect an improvement by using paragraph vectors since the classification task if fundamentally the same, only on a much larger scale., which complicates the task due to the "grey areas" between categories, which increases given more categories.
Pennington et al. [5] showed that the GloVe model outperforms the CBOW model, which is used in this research, on a word analogy task. Wang et al. [19] introduced the Linked Document Embedding method (LDE) method, which makes use of additional information about a document, such as citations. Their research specifically focused on categorizing documents, showed a 5.89% increase of the micro-F1 score on LDE compared to CBOW, and a 9.11% increase of the macro-F1 score. We would expect that applying this technique to our dataset would improve our scores, given earlier results on comparable tasks.
Even though much research has been done, we have not been able to find published results which are directly comparable to our results. This is likely due to our high amount of categorization groups, which enabled us to handle our results as a ranking problem, instead of an absolute hit, which has been used in earlier researches[19]. Even though we have an F1 score, which indicates performance on the absolute hits, our results are not comparable to other works due to the number of categories, and their overlapping subjects[10].

[10] As visualized in the journal embedding plots and discussed earlier

9 Conclusion

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

1. Article based embeddings perform better than TF-IDF on titles, 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
4. Visualization of the journal embeddings show that the embeddings capture and preserve subject relatedness when they are combined to create embeddings for larger texts

10 Future work

This research focussed on the quality of word embeddings on academic texts. To do this, we used both a comparison to TF-IDF and a visualization of the word embeddings. Future works may seek to improve the quality of the embeddings further or determine the limit of the capabilities of the embedding technique.

10.1 Method differences

Levy et al. [20] observed that there were no significant differences between the various embedding creation methods. They state that the global/hyperparameters mainly cause the difference in performance. In this research, we have not looked at the comparison between various models and (hyper) parameters. We instead used a standard configuration, to focus more on the actual performance of the embeddings, instead of the best possible performance. Future work could seek to validate the results of Levy et al. [20] by expanding our research with multiple metrics. It should be noted however that Levy et al. base their conclusions on word similarity and word analogy tasks, and not on categorization. This is the reason why we did not take this research into account in our discussion since their experiments are not comparable to ours.

10.2 Intelligent cutting

An interesting improvement to enhance the word embeddings with could be a smarter way to remove noise, based on word embeddings. This might be achieved by analyzing the word embedding spectrum before normalization, and to then cut the center of the vector space out. This must be applied before normalization since normalization causes all embeddings to have a distance of 1 to the center point. All words which are generic are in the center of the spectrum. Removing these words prevents the larger texts to be pulled towards the middle, where they lose the parts of their meaning which sets them apart from the other journals. We expect that this way of cutting, instead of word-occurrence cutting, will improve the quality of the word embeddings.

10.3 Text combination

To cope with the problem of articles that do have an abstract but no title, or vice versa, it would be interesting to see what the quality of the embeddings would be if both texts were combined into one text. This should be possible since title and abstract per article share a common topic. We would expect that the common text would have the quality of the abstract, which is according to our findings the part that is best used for the embeddings and TF-IDF.

10.4 TF-IDFs performance point

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

10.5 Reversed word pairs

At this point, there are no domain-specific word pair sets available. However, as we demonstrated, we can still test the quality of word embeddings. Once we established that we have word vectors of high quality, could we create word pairs from the embeddings? If this is the case, we could reverse-engineer domain specific word pair sets for future use. These word pairs should most likely still be validated by humans, but the automatic generation of word pairs should already reduce the effort needed for this process.

10.6 Historical overview

We have shown that the word embeddings can be used to create a subject spectrum, in which we plotted all the articles for one year. This gives insight into the currently popular research field, indicated by dense areas on the plot. If we would create this plot over multiple years, and then show these years in chronological order, we should be able to see the evaluation of research fields in time, giving more insight in shifting interests and the development of new research areas.

10.7 TF-IDF top-cutoff

In our research, we used a dataset from which we removed the top 1k words, together with everything beyond 6k. This dataset did not perform well in our research; future work could look into this by validating if the top-words cut-off decreases the TF-IDF performance. Our findings on this topic are minimal, although we can say that we would not expect that the top-cutoff improves the performance since a set with the top 5k words performed better than the 1k till 6k words set.

10.8 Collecting a set of terms

Our results show that, for larger texts, TF-IDF outperforms the embeddings. Given this, it would be logical to use TF-IDF for search tasks on sets with many tokens. However, the search term itself will contain a small number of tokens, as our research showed, embeddings perform better in this situation. Future work could try to combine these findings, by collecting a large number of words, resembling the meaning (captured by the embeddings), and transforming this into a (large) collection of words (mimicking word occurrence, captured by the TF-IDF). If this could be done effectively, the power of the TF-IDF method could be applied to smaller texts. This process could be seen as a translation from word-embedding space into TF-IDF space. Furthermore, it would be interesting to see which words would be selected and if these words represent the given sentence as an "extracted version" which would still be interpretable by humans.

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

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

[8] Yuanyuan Chen, Yisheng Lv, Xiao Wang, and Fei-Yue Wang. A convolutional neural network for traffic information sensing from social media text. In *Intelligent Transportation Systems (ITSC), 2017 IEEE 20th International Conference on*, pages 1–6. IEEE, 2017.

[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] 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.

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

[18] 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.

[19] Suhang Wang, Jiliang Tang, Charu Aggarwal, and Huan Liu. Linked document embedding for classification. In *Proceedings of the 25th ACM International on Conference on Information and Knowledge Management*, pages 115–124. ACM, 2016.

[20] Omer Levy, Yoav Goldberg, and Ido Dagan. Improving distributional similarity with lessons learned from word embeddings. *Transactions of the Association for Computational Linguistics*, 3:211–225, 2015.