Text Similarity in Vector Space Models: A Comparative Study

1st Omid Shahmirzadi
Dept. of Management of Technology EPFL
Lausanne, Switzerland
omid.shahmirzadi@gmail.com

2nd Adam Lugowski
Patent Research Foundation
Seattle, USA
alugowski@gmail.com

3rd Kenneth Younge
Dept. of Management of Technology EPFL
Lausanne, Switzerland
kenneth.younge@epfl.ch

***Abstract*—Automatic measurement of semantic text similarity is an important task in natural language processing. In this paper, we evaluate the performance of different vector space models to perform this task. We address the real-world problem of modeling patent-to-patent similarity and compare TFIDF (and related extensions), topic models (e.g., latent semantic indexing), and neural models (e.g., paragraph vectors). Contrary to expectations, the added computational cost of text embedding methods is justified only when: 1) the target text is condensed; and 2) the similarity comparison is trivial. Otherwise, TFIDF performs surprisingly well in other cases: in particular for longer and more technical texts or for making finer-grained distinctions between nearest neighbors. Unexpectedly, extensions to the TFIDF method, such as adding noun phrases or calculating term weights incrementally, were not helpful in our context.**

***Keywords*-text similarity, vector space model, text embedding, patent, big data**

I. Introduction

AUTOMATIC detection of semantic text similarity between documents plays an important role in many natural language processing applications. Techniques for this task fall into two broad categories: structure based and structure agnostic. In the first category, solutions rely on a logical structure of the text and transform it into an intermediate structure, such as aligning trees, to do the comparison. While useful in many contexts, it often is not clear which structure to use for a particular comparison. In the second category, structure is ignored and the text is represented using a vector space model (VSM). While VSMs often do not capture semantic components of text (e.g., negations), they have been shown to be able to measure text similarity in many applications.

A vector space model converts text into a numeric vector. A key aspect of VSMs is the definition and number of dimensions for each vector. In a common and simple approach, TFIDF defines a space where each term in the vocabulary is represented by a separate and orthogonal dimension. TFIDF measures the term frequency of each term in a text and multiplies it by the logged inverse document frequency of that term across the entire corpus. Despite its simplicity, TFIDF may suffer from an ignorance of n-gram phrases, complications with incremental updates upon addition of new documents, and a large number of dimensions. To deal with such issues, variants of TFIDF have been proposed to incorporate n-grams as new terms, and/or to adjust for the timing of the use of vocabulary across the time line of the corpus.

Other techniques, known as text embedding, attempt to address the high-number of dimensions and the loss of semantic information in TFIDF models, by transforming each text into a low-dimensional vector. Text embedding methods can be grouped into two categories: (i) count based methods based on bag of words (where the order of words are ignored), and (ii) prediction based methods based on sequence of words (where the order of words is taken into account). Topic models are an example of the first approach where each document is represented as a probability distribution of how relevant that document is to a given number of topics (and thus a lower-dimensional space). Each topic is selected as a weighted average of a subset of terms and document vectors are learned from the corpus on the assumption that words with similar meanings will occur in similar documents. Neural models are an example of the second approach where word vectors are learned using a shallow neural network trained from pairs of (target word, context word), where context words are taken as words observed to surround a target word. The assumption behind neural models is that words with similar meanings tend to occur in similar contexts. Document vectors can then be created out of word vectors through an averaging strategy or by considering each document as a special context token, hence obtaining document vectors directly. Prior research suggests that topic models and neural models are fundamentally similar in that they both arrive at a representation of the document in a lower-order space [1].

In this paper, we are interested in similarity measurement between patents. Patent-to-patent similarity can have several applications such as decision making on patent filing, predicting probability of different types of patent rejections and forecasting the innovation space. Previous studies [2], have shown that TFIDF is a powerful technique to detect patent-to-patent similarity, but the performance of other vectorization methods is unknown. We therefore compare the performance of TFIDF to other, newer methods to determine the relative performance of such methods for real world problems.

This paper continues as follows: In section II, we discuss the background material as well as related comparative studies on semantic text similarity. In section III, we introduce our data gathering, pre-processing, vectorizing and performance

978-1-7281-4550-1/19/$31.00 ©2019 IEEE
DOI 10.1109/ICMLA.2019.00120

evaluation pipeline. In section IV, we present our experimental results. In section V, we conclude the paper and propose some avenues for future work.

II. Background

A. Vector Space Models

Vector space models transform text of different lengths (such as a word, sentence, paragraph, or document) into a numeric vector in order to be fed into down-stream applications (such as similarity detection or machine learning algorithms). TFIDF, the most basic text vectorization method, defines a space where each term in the vocabulary is represented by a separate and orthogonal dimension. Despite its popularity and simplicity, basic TFIDF may suffer from (i) ignorance of n-gram phrases, (ii) complications with incremental updates upon addition of new documents, and (iii) a large number of dimensions. To deal with the two former issues, variants of TFIDF have been proposed: (i) to incorporate n-grams as new terms, and/or (ii) to adjust for the timing of the use of vocabulary across the time line of the corpus. To deal with the latter issue, text embedding methods attempt to address the high-number of dimensions by transforming each text into a low-dimensional vector. Text embedding techniques can be categorized into count-based and prediction-based models. Count-based models (a.k.a. topic models) create a document-term matrix where the weight of each cell is based on the number of times a term appears in the focal document. Prediction-based models (a.k.a. neural models) predict the occurrence of a term/document based on surrounding terms to learn a vectorization for each term/document.

In this section, we review mentioned families of the vector space models, namely (i) TFIDF models, (ii) topic models and (iii) neural models. From each family, we also select candidates to be compared for patent-to-patent similarity detection.

1) TFIDF Models.: Term Frequency–Inverse Document Frequency (TFIDF) is one of the most common vectorization techniques for textual data with many possible variations [3]. TFIDF considers two documents as similar if they share rare, but informative, words. In TFIDF, every term is considered as a different dimension orthogonal to all other dimensions. Each term is represented by a weight which is positively correlated with its occurrence in the current document and negatively correlated with its occurrence in all other documents in the corpus. The logic behind TFIDF is to downgrade the importance of terms that are common in many documents, on the view that those terms carrying less information specific to a focal text. Despite the popularity and applicability of TFIDF to many applications, it suffers from the curse of dimensionality in many downstream applications (e.g. computing k nearest neighbors[4]), it ignores n-gram phrases, and all IDF weights might need to be updated upon the addition of new documents. The basic model, however, can be extended in several ways to avoid some of these pitfalls. We consider two recently proposed extensions in this study.

First, we consider adding certain n-grams to the term vocabulary. N-grams allow for the combination of terms into higher-level concepts, which may be particularly important for research in computational social sciences including patent research [5]. Adding n-grams blindly, however, would vastly increase the size of vocabulary, and thus the number of vector dimensions. A more manageable approach, therefore, is to add noun phrases based on synthetic properties of the text. We test the phrase extraction technique from [6] which extracts noun phrases based on a pattern based method.

A second, and separate, extension takes advantage of the timing information of patents to implement incremental IDF [7]. More specifically, whenever a new document is added to the corpus, the corresponding IDF at that point in time is calculated based on the current state of the total corpus.Therefore, a term would have a low IDF when it is first introduced into the vocabulary and high differentiating power; and the IDF would attenuate over time as use of the term became more common. An example would be a niche term for an emerging technology, where the term would have a very high importance at the time of filing the patent, but the term would reduce in importance over time. As a convenient side property, incremental calculation of IDFs also avoids the need to update all TFIDF vectors upon addition of a new document to the corpus.

2) Topic Models.: Topic models transform a text into a fixed size vector, equal to a given number of latent topics. The vector represents the probability distribution that the focal text relates to each of the different topics. In practice, each topic is a weighted average of a subset of terms. Similar to TFIDF, topic models treat the text as a bag of words where order of words is ignored. On the down side, interpretation of each topic can be subjective and determining the right number of topics requires tuning of the model.

Latent Semantic Indexing (LSI) [8], [9] is a commonly used topic model to find low-dimension representation of words or documents. Latent Dirichlet allocation (LDA) is another popular topic model that fits a probabilistic model with a special prior to extract topics and document vectors. We choose LSI as the representative of topic models in this study, as LDA models can be hard to reproduce due to their highly probabilistic nature.

3) Neural Models.: Unlike models that simply count terms, neural models capture information from the context of other words that surround a given word, hence taking ordering into account. A well-known model for predicting word context is W2V (Word to Vector)[10], where the authors propose an algorithm based on a neural network of three layers to learn word vectors. Prior research has shown the W2V model to perform well with analogy and similarity relationships.

To obtain document vectors from word vectors, one can average together all word vectors. Because simple averaging will give the same weight to both important and unimportant words, one may be able to retain more information by assigning weights during averaging based on TFIDF. Alternatively, an extension of W2V, known as D2V (Document to Vector) or paragraph vectors[11], has been proposed to obtain document vectors directly by considering a document as a special context

token that can be added to the training data, such that the model can learn token vectors and consider them as document vectors. D2V can be implemented incrementally and showed a better performance compared to the previous approaches in some similarity detection benchmarks. However, on the negative side, it has numerous hyper-parameters which should be tuned to harvest its power to the full extent. We consider the D2V model as the representative neural model for the experiments in this study.

B. Similarity Metric

Given two vectors, one can measure similarity between them in many ways: Euclidean distance, angular separation, correlation, and others. Although there are differences between each measure, in this study we adopt cosine similarity as the measure of interest as it is frequently-used.

C. Existing Comparative Studies

Several recent studies compare different vector space models with respect to their similarity detection power for texts. Very few of them, however, target similarity detection for longer texts (e.g. documents). In [12] the authors compared D2V variants against an averaging W2V, as well as a probabilistic n-gram model for two similarity tasks. In the first task, the goal is to detect similarity of forum questions; the second task aims to detect similarity between pair of sentences. The authors find that D2V is superior for most cases and that training the model on a large corpora can improve their results. In [13] the authors attempt to detect similarity between sentences and compare several neural models against a baseline that simply averages together word vectors. They find that more complex neural models work best for in-domain scenarios (where both the training data set and the testing data set are from the same domain), while a baseline of averaging word vectors is hard to beat for out-of-domain cases. In [14] the authors proposed a method for sentence embedding through the weighted averaging of word vectors as transformed by a dimensionality reduction technique. They show that their text vectors outperform well-known methods to detect sentence to sentence similarity. In [15] the authors use an unsupervised method to vectorize sentences and show that their method outperforms other state-of-the-art techniques to detect similarity of short sequences of words. In each of these studies, however, the objective was to determine the performance of similarity detection algorithms on relatively short sections of text.

There has been much less research on the performance of similarity comparisons for longer text. In [16] the authors compared D2V to a weighted W2V, LDA, and TFIDF to detect the similarity of documents in Wikipedia and arXiv corpus. They find that D2V can outperform other models on Wikipedia, but that D2V could barely beat a simple TFIDF baseline on arXiv. In [17] the authors compared several algorithms to detect similarity between biomedical papers of PubMed and find that advanced embedding techniques again can hardly beat simpler baselines such as TFIDF. This paper adds to the stream of research comparing text vectorization methods for longer text. In particular, we focus on a real-world problem with an objective standard for determining similarity, whereas prior research has had to rely on broad categorizations from repositories such as Wikipedia and arXiv. To the best of our knowledge, this work is also the first comparative study of semantic similarity methods in the patent space.

III. Data Pipeline

The Patent Research Foundation[1] provided us with a corpus of all publicly available patents from the United States Patent and Trademark Office (USPTO) from January 1976 to January 2018.[2] From the raw data, we create a pipeline to extract the technical description from each patent and vectorize the text. Our code, based on Python 3, is available on GitHub upon email request. For the purpose of this study, we focus on the following fields:

- Number: unique ID for each issued patent
- Title: patent information of high density
- Abstract: patent information of moderate density
- Description: patent information of low density
- Date: date when the patent application was submitted
- Class: one of 491 patent main class classifications[3]
- Subclass: one of 82,520 patent subclass classifications[2]

A. Pre-Processing

For pre-processing of the data, we use the DataProc service of Google Cloud Platfrom[4]. DataProc, a managed Apache Spark[5] service hosted by Google, provides a high-performance infrastructure for rapid implementation of data parallel tasks such as data pre-processing. We apply several pre-processing steps to the textual fields of patent data (title, abstract, description):

- Remove HTML, non-ASCII characters, terms with digits, terms less than 3 characters, internet addresses, and sequences such as DNA
- Stem words and change to lower-case
- Remove stopwords (general NLP and patent-specific)
- Remove rare terms with low total document frequency

B. Vectorizing

We vectorize titles, abstracts and descriptions for each of the following models:

1) Simple TFIDF: We use the machine learning library in the Apache Spark framework for our implementation of TFIDF. There are two flavors of of TFIDF in Spark to consider. The first method is based on CountVectorizer, where vocabulary is generated and term frequencies are explicitly counted before being multiplied with inverse document frequency. The CountVectorizer method, however, creates a highly sparse

[1]Patent Research Foundation: https://www.patrf.org

[2]Data can also be downloaded from: https://bulkdata.uspto.gov

[3]USPTO patent classification: https://www.uspto.gov/web/patents/ classification/selectnumwithtitle.htm

[4]Google Cloud Dataproc Service: https://cloud.google.com/dataproc

[5]Apache Spark: https://spark.apache.org

representation of a document over the vocabulary. The second method takes advantage of HashingTF which transforms a set of terms to fixed-length feature vectors. The hashing trick can be used to directly derive dimensional indices, but it can suffer from collisions. In this work, we use the first method as a slower but more robust technique.

2) Incremental TFIDF: Incremental updating of inverse document frequencies can be implemented in one of two ways: (i) create a new TFIDF model at a fixed time interval for newly arrived patents, or (ii) create a TFIDF model on the whole corpus and then adjust document frequency vectors by subtracting document frequencies of future patents with respect to each focal patent. We implement the second.

3) Phrase-augmented TFIDF: We augment the vocabulary of TFIDF by noun phrases as extracted from the open source NPFST library[6]. The library is based on Python 2, which was ported to Python 3 to be compatible with the rest of the pipeline. NPFST can be configured to limit the number of noun phrases based on their frequency and length.

4) LSI: Despite the advantage of Spark for running data parallel tasks, its support for model parallel tasks such as LSI is limited. Therefore, we use the LSI implementation in Gensim[7], a well-established library for text vectorization. Gensim implements several different vector space models, exploits parallelism of multiple CPU cores, and supports large corpora that cannot be resided in the memory. Gensim also implements LSI in a memory-efficient way to support incremental updates, an important consideration for large corpora. Gensim accepts TFIDF document-word matrix as input and has several hyper-parameters, of which we consider the followings for tuning:

- num-topics: number of latent topics
- chunksize: number of documents in each training chunk
- decay: weight of existing observations relative to new ones (max 1.0)

5) D2V: We also use the implementation of D2V in Gensim, as the Gensim version is memory efficient, allows for incremental updates, and can be parallelized over multiple cores. We fix the random seed and Python hash seed for reproducibility of results. D2V implementation accepts raw pre-processed text in the form of TaggedDocument elements, and has several hyper-parameters, of which we consider the followings for tuning:

- dm: flag to choose *distributed memory* or *distributed bag of words* algorithms
- size: dimension of the feature vectors
- window: maximum distance between target and context word in text
- sample: threshold as to which high-frequency words are randomly ignored
- iter: number of iterations over the corpus
- hs: flag to choose *hierarchical softmax* or *negative sampling* methods

[6]Phrase Machine Library: http://slanglab.cs.umass.edu/phrasemachine
[7]Gensim Library: https://radimrehurek.com/gensim/index.html

C. Evaluation

To evaluate the relative performance for each vectorization method, we construct a classification test on our data, where we use the similarity score between two vectors from a given method to predict whether that pair of patents would be rated as similar (positive label) or not (negative label) according to an independent benchmark. Our evaluation test requires:

1) a benchmark indicator of similarity as the ground truth,
2) a way to calculate a similarity score between vectors,
3) a metric to assess the accuracy of predictions, and
4) a mechanism to tune hyper-parameters.

1) Benchmark: Identifying a ground-truth benchmark for similarity is an important requirement for evaluating the performance of similarity detection from automated text vectorization. While a continuous measure of the ground truth would be ideal, we are not aware of a reliable measure of continuous patent-to-patent similarity that is separate from textual analysis. Instead, we construct a benchmark set of both *more*-similar pairs of patents (positive case) as identified by the USPTO, and *less*-similar pairs of patents (negative case), and evaluate the relative performance of text vectorization techniques in predicting the difference between the two cases.

For the selection of *more*-similar pairs of patents, one might consider patent citations to be a natural choice – for one would expect patent citations to reference *more*-similar patents. Within the full set of patent citations, however, there is a subset of "102 rejections" that one would expect to be even more highly-similar that an average patent citation. Patent examiners issue a 102 rejection when they believe that a cited patent is similar enough to the citing patent to cause them to reject the patent application of the citing patent on the basis that the new invention is not novel enough. Although 102 rejections are not a perfect indicator of similarity, it is reasonable to believe that 102 rejections are considerably *more*-similar to each other than some other comparison sets. We therefore select 102 rejections as our indicator of similarity.[8]

For the selection of the comparison set of *less*-similar patent pairs, one can easily identify patents that are in fact very dissimilar, and therefore very easy to distinguish from positive cases defined above. Alternatively, one can also pick pairs of patents that are, themselves, somewhat similar, and therefore much harder to distinguish from the positive cases. As such, we select and evaluate the performance of vectorization methods across a range of comparison difficulties. Going from harder separation to easier separation, we select negative cases for three testing scenarios: (i) pairs of patents selected at random from the same subclass, (ii) pairs of patents selected at random from the same main class and (iii) pairs of patents simply selected at random.

[8]The Patent Research Foundation provided us with pairs of patents in 102 rejections from public records for use in this study. More recently, the USPTO has also released a data set of 102 rejections[18]. Validation tests on both data sets give similar results.

In summary, we selected a random set of 102 rejection patent pairs for our positive labels, and we selected multiple sets of patent pairs for our negative labels (sets drawn from the same subclass, the same main class, or at random). We choose the size of our test set large enough to avoid selection bias (a set of 50K positive and 50K negative-labeled pairs).

2) Similarity Calculation: We calculate the similarity of each patent pair based on the cosine sim. of their corresponding vectors from a different vector space. To compare pairs of patents from an incremental TFIDF model, a recent study[7] proposes replacing the IDF of the latter patent to that of the former patent so that both vectors have the same time scale to compare. Therefore we pre-compute aggregated DFs by month so that IDF replacement could be done efficiently at run time.

3) Evaluation Metric: The Receiver Operating Characteristic (ROC) curve is a standard method for comparing the accuracy of classifiers in which the class distribution can be skewed and the probability threshold to assign labels is not determined. Given labels and predictions, the Area Under the Curve (AUC) represents the probability that a vectorization method will predict that a positive case (a randomly selected 102 rejection pair of patents) to be more similar than a negative case (a randomly selected pair of patents from a comparison set). The AUC is our preferred metric for comparing models, and we plot ROC curves for visual comparisons.

4) Model Tuning: Models with hyper-parameters (e.g., topic models and neural models) need to be tuned for optimal performance. Studies show that hyper-parameter tuning can be as important as, or even more important than, the choice of the model itself [1]. However, the complexity and cost of tuning can escalate quickly as the number of hyper-parameters increases. Classic approaches to tuning, such as grid or random search, are often a poor choice when the cost of model evaluation is high. For grid search, it is difficult to select grid points for continuous parameters, and every added hyper-parameter will increase the tuning cost geometrically. For random search, one can end up calculating many poor configurations where model evaluation is expensive.

Bayesian optimization is shown to be superior to classic hyper-parameter tuning solutions for expensive models [19]. In particular, it provides a global, derivative-free approach that is suitable for tuning black box models with a high cost of evaluation. The costs of tuning is also linear with respect to the number of hyper-parameters. We used the scikit-optimize library[9] to implement Bayesian optimization due to the library's ability to run in parallel and to integrate with other libraries. We set a tuning budget equal to 10 times the number of hyper-parameters for most experiments, but stopped tuning short for the description field due to the high cost and low improvement expectation.

IV. Experimental Results

A. Model Comparisons

Figure 1 plots the performance of patent-to-patent similarity measurement by model. Each row corresponds to a different vectorization approach and each column corresponds to testing on a different text field. Each pane compares a simple TFIDF model as a baseline (solid lines) with a more complicated vectorization method indicated for that row (dashed lines). The comparison is done for each of the three benchmarks mentioned earlier in Section III-C: a random set of 102 rejection patent pairs with positive labels, and three sets of patent pairs with negative labels (sets drawn from the same subclass, the same main class, or at random). Cosine similarity of each patent pair is calculated using the two vector space models and their corresponding ROC curves are plotted, where different colors correspond to different benchmarks.

In the first row of Fig. 1, TFIDF with noun phrases performs almost identically to the baseline, across all text lengths and benchmarks. This is counter-intuitive, as augmenting a bag-of-words vector with n-grams would seem to add more information. It is possible, however, that adding *every* noun phrases is too granular. We performed additional experiments with including only the top 50, 100, and 200 noun phrases, but found even worse results than the baseline.

In the second row of Fig. 1, incremental TFIDF also performs almost identically to the baseline, across all text lengths and benchmarks. This result is contrary to recent studies [7] where incremental TFIDF was presumed to be beneficial. However, even if incremental TFIDF is *not* better at similarity detection, it is computationally cheaper for a corpus that expands over time, and therefore our results suggest that incremental TFIDF may be a reasonable choice in that context.

In the third row of Fig. 1, a highly-tuned LSI model is able to beat the performance of the baseline, but only for a very easy similarity comparison and only for very short text (titles). Otherwise, LSI performs worse than the baseline.

In the fourth row of Fig. 1, a highly-tuned D2V model is able to beat the baseline, but this gain is considerable only for very short text (title) and only for an easy similarity comparison. D2V gives only a very slight improvement over baseline on all other conditions. It also is important to emphasize that the very minor improvement of D2V over the simple TFIDF baseline were obtained only after very extensive and expensive tuning of the D2V model.

To summarize, Table I reports the AUC and the percentage improvement in AUC of the best vectorization method in all different *text field - test set* evaluation scenarios over the simple TFIDF baseline while Table II depicts the rough wall time estimate required by each method to compute the corresponding vector space representation of the full corpus using our hardware setting (excluding pre-processing time). In each case, the best method[10] performs better than a simple TFIDF model, but the percentage of improvement is negligible

[9]Scikit-Optimize Library: https://scikit-optimize.github.io

[10]Highly tuned D2V performs the best in all testing scenarios.

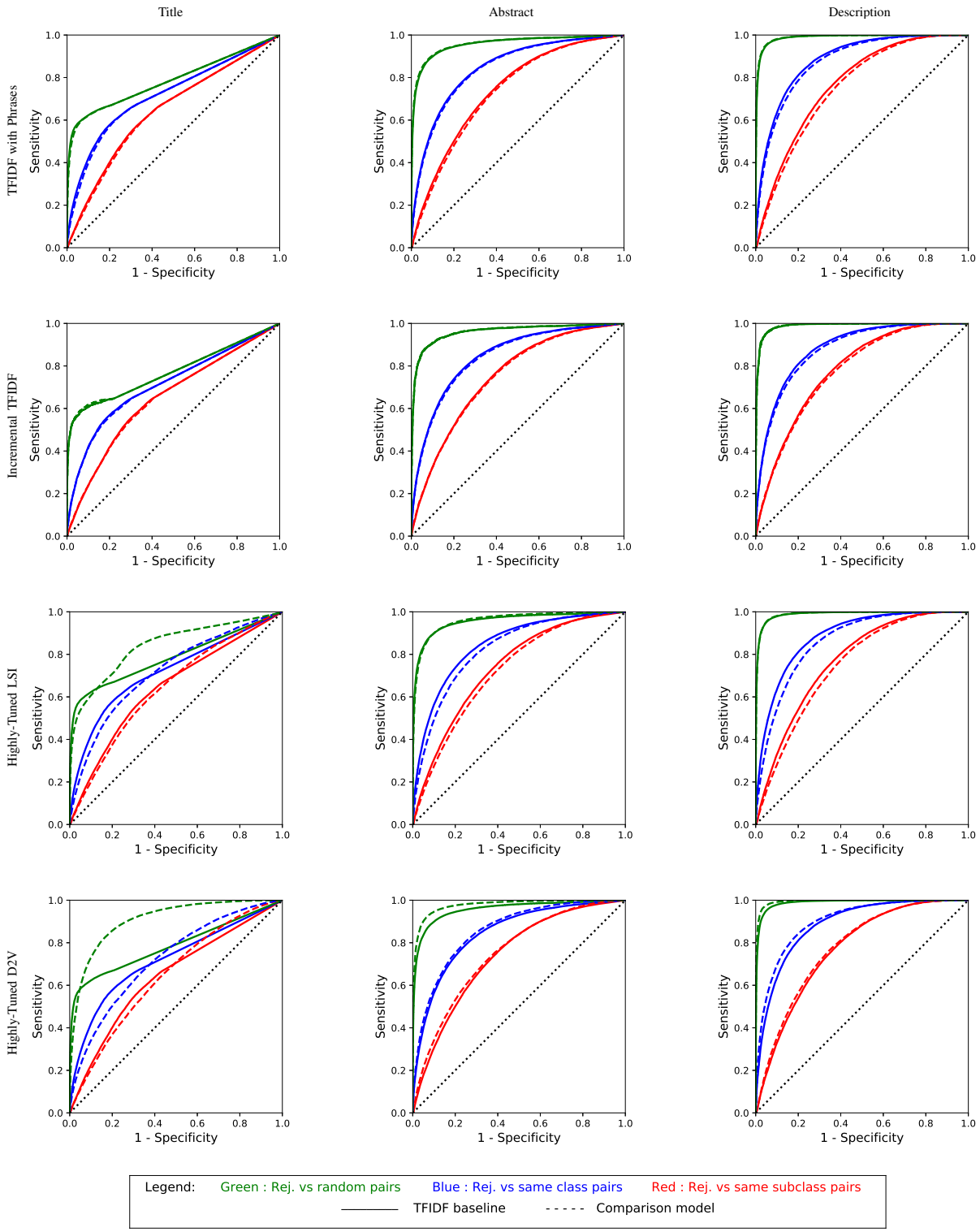

Fig. 1: Inferred accuracy of patent-to-patent similarity by vectorization method: Figures plot ROC curves for the prediction of patent rejection based on similarity score of four vectorization methods: TFIDF with noun phrases; TFIDF with incremental calculation of IDF; highly-tuned LSI; and highly-tuned D2V. In each plot, solid lines represent the simple TFIDF model as a baseline, and dashed lines represent the comparison model. Experiments were run by text length (title, abstract and description) and difficulty of prediction: easy (in green: discrimination between a 102 rejection pair and a random pair); medium (in blue: discrimination between a 102 rejection pair and a pair from the same class); and difficult (in red: discrimination between a 102 rejection pair and a pair from the same subclass).

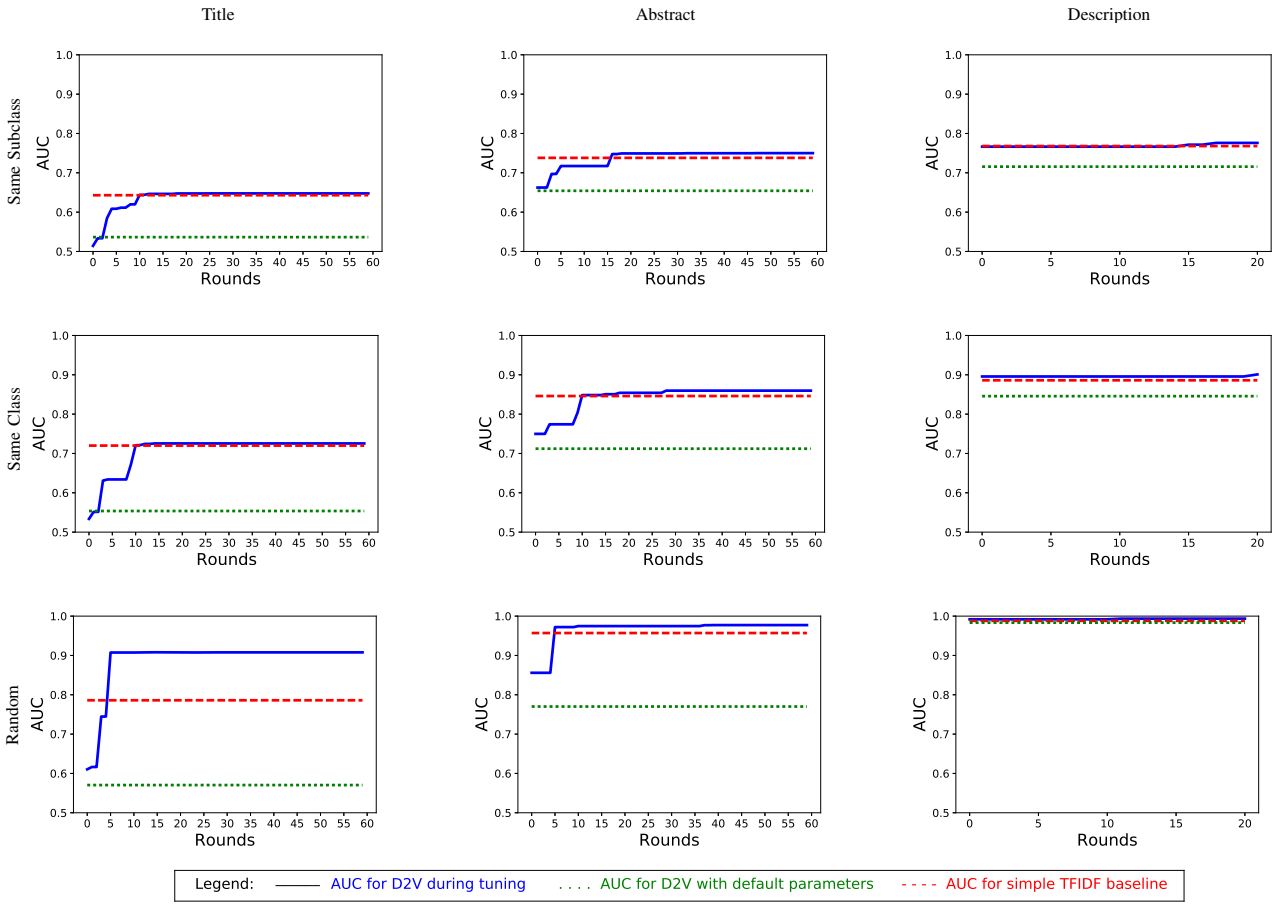

Fig. 2: Hyper-parameter tuning of D2V for each testing scenario is depicted. Rows correspond to testing difficulty: Same subclass (discrimination between a 102 rejection pair and a pair from the same subclass); same class (discrimination between a 102 rejection pair and a pair from the same class); and random (discrimination between a 102 rejection pair and a random pair). Columns correspond to text fields with different lengths.

in most cases other than similarity comparison based on titles with a very easy distinguishability (102 rejections pairs versus random pairs). Moreover the computation wall time of the best method is at least two orders of magnitude highers than the baseline TFIDF in all scenarios.

TABLE I: AUC performance comparison of the best VSM to the simple TFIDF.

	Title	Abstract	Description
Sub-class			
Best VSM	0.646	0.749	0.775
TFIDF	0.643	0.738	0.768
Improvement	0.4%	1.5%	0.9%
Main-class			
Best VSM	0.723	0.858	0.900
TFIDF	0.720	0.846	0.886
Improvement	0.4%	1.4%	1.6%
Random			
Best VSM	0.907	0.977	0.993
TFIDF	0.786	0.957	0.988
Improvement	15.4%	2.0%	0.5%

TABLE II: Computation wall time comparison of the best VSM to the simple TFIDF.

	Title	Abstract	Description
Best VSM	hours	days	weeks
TFIDF	seconds	minutes	hours

B. Hyper-parameter Tuning

To better understand the effect of tuning hyper-parameters using Bayesian Optimization, Figure 2 plots the AUC of D2V while tuning the model. Rows correspond to testing difficulty; columns correspond to text fields. We stopped tuning at 10 times the number of parameters (i.e., 60 rounds), except for Description where there was little improvement.

Several trends in Figure 2 are clear: 1) easier similarity detection makes hyper-parameter tuning more important; 2) longer text makes hyper-parameter tuning less beneficial; and 3) using D2V with default parameters gives very poor performance - substantially worse than a simple TFIDF baseline. Table III shows the values of tuned hyper-parameters of D2V.

TABLE III: Highly-tuned hyper-parameters for D2V.

Field	Benchmark	dm	hs	size	window	sample	iter	AUC
Title	Sub-class	0	1	374	1	1e-3.28	10	0.647
Title	Main-class	0	1	250	10	1e-3	10	0.725
Title	Random	0	0	321	1	1e-3.08	10	0.907
Abstract	Sub-class	1	1	491	1	1e-4.06	10	0.750
Abstract	Main-class	1	0	290	1	1e-4.01	10	0.859
Abstract	Random	1	0	522	1	1e-4.04	9	0.977
Description	Sub-class	1	0	321	7	1e-4.91	10	0.775
Description	Main-class	1	1	592	1	1e-5.52	10	0.900
Description	Random	1	0	501	1	1e-7	10	0.993

V. Conclusion

In this paper, we evaluated the performance of text vectorization methods for the real-world application of automatic measurement of patent-to-patent similarity. We compared a simple TFIDF baseline to more complicated methods, including extensions to the basic TFIDF model, LSI topic model, and D2V neural model. We tested models on shorter to longer text, and for easier to harder problems of similarity detection.

For our application, we find that the simple TFIDF, considering its performance and cost, is a sensible choice. The use of more complex embedding methods which can require extensive tuning, such as LSI and D2V, is only justified if the text is very condensed and the similarity detection is relatively coarse. Moreover extensions to the baseline TFIDF, such as adding n-grams or incremental IDFs, does not seem beneficial. Although our conclusion is based on experiments over patents, we believe that it can be generalized to other corpora due to the minimal patent-specific interventions in our pipeline.

Our results are compatible with previous studies on semantic text similarity detection of embedding methods. The focus of prior research, however, has typically been on short text and simple similarity detection problems. Few studies have evaluated the performance of different vector space models on long text or for more challenging benchmarks.

For the context of this study (patent-to-patent similarity) in practice, discriminating between random pairs of patents and rejection pairs of patents is a rather trivial problem that probably does not require a complicated NLP solution. It is only on such problems, however, that D2V and LSI might outperform the TFIDF model considerably. Instead, for many applications, users are often looking for the automatic detection of differences between relatively similar patents (e.g. same-subclass pairs versus rejection pairs). For such problems, where the differences in similarity are small, simple TFIDF appears to be a good choice. The difference in cost and simplicity is to the extent that the use of a simple TFIDF model, which might do slightly worse under certain conditions then more complex models, may still be justified. An extension of TFIDF with incremental IDF calculation could provide additional benefit of avoiding calculation of all TFIDF vectors upon addition of new patent(s) without scarifying performance for similarity detection task.

This study can be extended by future research in several directions, both in theory and in practice. We observed that incorporating noun phrases and incremental timing information did not lead to better detection of similar patents. For the case of noun phrases, perhaps the low weights of locally-filtered phrases misses the signal, and a more global approach for filtering might improve the performance. For the case of incremental TFIDF, it appears that adjusting IDF vectors based on time is not strong enough to affect the model performance in our context. It remains to be seen, however, how incremental IDFs may affect similarity detection in more rapidly evolving domains. Another dimension for future work would be to study the effect of similarity metrics other than cosine similarity. Finally, investigation of the limitations of text embedding methods is a fertile area for future research.

References

[1] O. Levy, Y. Goldberg, and I. Dagan, "Improving distributional similarity with lessons learned from word embeddings," *TACL*, vol. 3, pp. 211–225, 2015.

[2] K. Younge and J. Kuhn, "Patent-to-patent similarity: A vector space model," *https://ssrn.com/abstract=2709238*, 2016.

[3] J. Ramos, "Using tf-idf to determine word relevance in document queries," 1999.

[4] T. Liu, A. W. Moore, E. Gray, and K. Yang, "An investigation of practical approximate nearest neighbor algorithms," in *NIPS*, 2004. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.110.7942

[5] L. Andersson, M. Lupu, J. a. Palotti, A. Hanbury, and A. Rauber, "When is the time ripe for natural language processing for patent passage retrieval?" in *CIKM*, 2016.

[6] A. Handler, M. Denny, H. M. Wallach, and B. T. O'Connor, "Bag of what? simple noun phrase extraction for text analysis," in *EMNLP*, 2016.

[7] B. Kelly, D. Papanikolaou, A. Seru, and M. Taddy, "Measuring technological innovation over the long run, working paper," 2017. [Online]. Available: https://dimitris-papanikolaou.github.io/website/

[8] S. Deerwester, S. T. Dumais, G. W. Furnas, T. K. Landauer, and R. Harshman, "Indexing by latent semantic analysis," *Journal of American Society for Information Science*, vol. 41, no. 6, pp. 391–407, 1990.

[9] A. Moldovan, R. Boţ, and G. Wanka, "Latent semantic indexing for patent documents," *International Journal of Applied Mathematics and Computer Science*, vol. 15, pp. 551–560, 01 2005.

[10] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," *CoRR*, vol. abs/1301.3781, 2013. [Online]. Available: http://dblp.uni-trier.de/db/journals/corr/corr1301.htmlabs-1301-3781

[11] Q. V. Le and T. Mikolov, "Distributed representations of sentences and documents," *CoRR*, vol. abs/1405.4053, 2014. [Online]. Available: http://arxiv.org/abs/1405.4053

[12] J. Lau and T. Baldwin, "An empirical evaluation of doc2vec with practical insights into document embedding generation," *CoRR*, vol. abs/1607.05368, 2016. [Online]. Available: http://arxiv.org/abs/1607.05368

[13] M. Naili, A. H. Chaibi, and H. H. Ben Ghezala, "Comparative study of word embedding methods in topic segmentation," *Procedia Computer Science*, vol. 112, no. C, pp. 340–349, 2017. [Online]. Available: https://doi.org/10.1016/j.procs.2017.08.009

[14] S. Arora, Y. Liang, and T. Ma, "A simple but tough-to-beat baseline for sentence embeddings," in *ICLR*, 2017.

[15] M. Pagliardini, P. Gupta, and M. Jaggi, "Unsupervised learning of sentence embeddings using compositional n-gram features," *CoRR*, vol. abs/1703.02507, 2017. [Online]. Available: http://arxiv.org/abs/1703.02507

[16] A. M. Dai, C. Olah, and Q. V. Le, "Document embedding with paragraph vectors," *CoRR*, vol. abs/1507.07998, 2015. [Online]. Available: http://arxiv.org/abs/1507.07998

[17] J. E. Alvarez, "A review of word embedding and document similarity algorithms applied to academic text, bachelor thesis, university of freiburg," 2017.

[18] Q. Lu, A. Myers, and S. Beliveau, "USPTO patent prosecution research data: Unlocking office action traits," *https://ssrn.com/abstract=3024621*.

[19] J. Snoek, H. Larochelle, and R. P. Adams, "Practical bayesian optimization of machine learning algorithms," in *NIPS*, 2012. [Online]. Available: http://dl.acm.org/citation.cfm?id=2999325.2999464