GRAGGLE: A Graph-based Approach to Document Clustering

Isaiah J. King and H. Howie Huang
GraphLab
The George Washington University
Washington DC, USA
{iking5,howie}@gwu.edu

2022 IEEE International Conference on Big Data (Big Data) | 978-1-6654-8045-1/22/$31.00 ©2022 IEEE | DOI: 10.1109/BigData55660.2022.10020645

***Abstract*—Document recommendation systems have traditionally relied upon high-dimensional vector representations that scale poorly in corpora with diverse vocabularies. Existing graph-based approaches focus on the metadata of documents and, unfortunately, ignore the content of the papers. In this work, we have designed and implemented a new system we call GRAGGLE, which builds a graph to model a corpus. Nodes are papers, and edges represent significant words shared between them. We then leverage modern graph learning techniques to turn this graph into a highly efficient tool for dimensionality reduction. Documents are represented as low-dimensional vector embeddings generated with a graph autoencoder. Our experiments show that this approach outperforms traditional document vector-based and text autoencoding approaches on labeled data. Additionally, we have applied this technique to a repository of unlabeled research documents about the novel coronavirus to demonstrate its effectiveness as a real-world tool.**

***Index Terms*—Data mining, graph analytics, recommender systems, text mining**

I. INTRODUCTION

Since the early days of machine learning, classifying text information has been a major area of study. It is now almost trivial to assign multidimensional data points to accurate classes based on their proximity. But turning documents into points in Euclidian space is a challenge. Feature extraction is defined here as finding a function $f : C \rightarrow \mathbb{R}^d$ where C is a set of documents, also called a *corpus*, and d is an arbitrary constant. The goal is to find an f such that the resulting vectors contain as much information about the original texts as possible while keeping d low. If this is done correctly, any labels the original documents had can be inferred from the feature vectors using standard clustering techniques.

The most prominent way to represent documents in the literature is through TF-IDF encoding [1], or some variant thereof. In this method, features are some function of word count, and the dimension of vectors is necessarily proportional to diversity of the corpus' vocabulary. As a result, many state-of-the-art document clustering approaches simply accept high dimensionality as a given and strive to build effective algorithms despite it [2]–[4].

In our view, it is more apropos to represent a corpus as a whole, as a network of interrelated parts, where any individual document is better understood in relation to its neighbors. By representing corpora as graphs, this view of documents as a collective rather than discrete units is realized. However, existing graph-based techniques are often based on relationships in the metadata of the papers such as citation networks. They are more often used for supervised learning [5]–[7]. Often these techniques use computationally expensive means of graph clustering such as geodesic distance [8]. Few exist which have graph structures derived from document text. Those that do, do not use a graph autoencoding technique [9]–[11] and lack the generality and expressiveness it offers.

To address these perceived shortcomings in the literature, we propose GRAGGLE[1], a novel approach to feature extraction from text that captures complex contextual relationships between documents. Rather than focusing on metadata, or using vectors generated just from the text, GRAGGLE generates a graph structure that captures textual similarities between documents. By representing a collection of texts as a graph of significant words shared between documents we can use graph autoencoding techniques [12] to extract features from text. Our experiments show that even though our vectors are often 10 times smaller than those used by traditional techniques, the clusters we generate have greater purity than those consisting of TF-IDF vectors.

This work makes the following contributions:

- **Tunable Dimensionality.** Node embeddings from GRAGGLE represent the relationships between documents rather than the words in a single document. As such, the dimension of these vectors is a tunable parameter in the autoencoding algorithm. In practice, the degree of vectors built by GRAGGLE are far smaller than those required by TF-IDF vectors for the same data.
- **Generalizability**. GRAGGLE can analyze any corpus. Unlike prior work, it is not reliant on the presence of citations in the metadata, the topics covered by the documents, or any preprocessing which the corpus may have undergone. Additionally, it adapts well to bag-of-words representations of documents, so it is memory efficient.
- **Graph representation.** In addition to its ultility for generating document embeddings, the graph data structure itself holds value. As we will show, the resulting graphs

[1] Full code available at https://github.com/iHeartGraph/Graggle

978-1-6654-8045-1/22/$31.00 ©2022 IEEE

are useful research tools and recommender systems simply through the relations they make explicit.

The rest of the paper is organized as follows: in § II we outline related work and techniques for document clustering. § III describes how the graph is constructed, and how document-vectors are built from the weighted graph. § IV compares our method to traditional document clustering methods. § V contains an analysis of optimal hyperparameters for GRAGGLE. § VI presents a case study of GRAGGLE as a paper recommender on CORD-19, a text mining dataset for papers about COVID-19 [13]. Finally, we conclude with some suggestions for future work on this topic.

II. Related Work

Our method draws from many disparate corners of graph and machine learning. We are most motivated by traditional text classification methods, autoencoding techniques to reduce dimensionality, and graph-based methods generally. To contextualize our technique, we present a brief overview of those three fields.

A. Traditional Methods

Very advanced algorithms exist to quickly find points close together in Euclidean space and assign them to clusters, but the feature vectors they analyze have changed very little in the last twenty years [14]. For text data, the default embedding choice is TF-IDF vectors [1], [15], [16]. These vectors are generated by calculating the score of each word w, in a document D according to the following formula

$$\text{TF-IDF}(w, D) = \log\left(\frac{|C|}{n_w}\right) \times f_{w,D} \tag{1}$$

where n_w is the number of documents in the corpus, C that contain w, and $f_{w,D}$ is the number of occurrences of w in D. Consequently, the vectors representing each document have dimensionality equal to the total number of unique words in the corpus. Vectors represent which bag of words each document uses, weighted in such a way that low-information words that appear throughout the corpus, are given less significance than more rare words.

This approach produces obvious problems which this work aims to address. For example, corpora with diverse vocabulary require TF-IDF vectors of enormous degree. To combat this, certain words are omitted from the corpus either through stemming or according to some heuristic [17]. The information lost in this process is negligible, and it reduces the overall noise in the data. However, even with these minor improvements, more recent studies have shown that TF-IDF vector representations are not as effective for cluster analysis as other text preprocessing techniques [18]. Despite this, many modern papers still use this representation, and aim instead to find clustering algorithms optimized for high-dimensional data [2]–[4]. With more informative vectors to cluster, these algorithms could perform even more effectively, yet TF-IDF remains the popular choice.

B. Autoencoders

More recent research in clustering, and unsupervised learning in general has focused on autoencoders. Autoencoders aim to learn low-dimensional embeddings of input data that can be reconstructed with high fidelity. These embeddings should carry the same amount of information as their inputs, or at least a good enough approximation, to be used for clustering or other unsupervised learning tasks [19].

More recently, transformer-based [20] autoencoding approaches such as BERT have become the popular choice for natural language processing [21]. These models can outperform traditional machine learning techniques, especially in fields such as sentiment analysis and machine translation [22]. But for simpler tasks like text clustering and recommendation, BERT and simple K-Means clustering of TF-IDF vectors are roughly equal, and in fact K-Means can out-perform BERT in this domain [23]. We feel that to justify the number of parameters required for these models, there ought to be major performance improvements, and this is just not the case in this domain.

However, deep autoencoders do not necessarily need to use self-attention, or have millions of parameters. A simple deep neural network can perform quite well on many NLP tasks [19], [24], [25]. Even shallow neural network, skip-gram approaches such as Word2Vec [26] and Doc2Vec [27] can be highly effective. These approaches have been shown to be far more effective on sentiment analysis than TF-IDF encodings for the same clustering algorithms [27]. However, for text classification, the bag of words model still out-performs traditional autoencoders [28]. Nonetheless, the dimensionality reduction afforded by autoencoding techniques was our major motivator to use a graph autoencoder for this task.

C. Graph-based approaches

Using graphs to represent a body of text is not a new idea, however very few prior works build graphs from the actual text of their corpora. Most of the work in graph-based document clustering algorithms has been in citation network analysis [5]–[7]. Citation networks are directed graphs of which academic papers have been cited by others. This tradition is so ingrained in graph neural network research that citation network classification has become a standard benchmark for major algorithms [5], [29], [30]. While this is certainly a well-respected area of study, citation networks are constantly evolving, and the newest papers–which are ultimately the ones most interesting to classify–will inevitably be root nodes in these networks. No one has cited them yet making them harder to accurately classify without many older papers in the corpus. Additionally, these techniques only work on academic writing; for analyzing general-purpose natural language data sets, a different approach is needed.

A few graph-based methods that learn from document text have been proposed. We have been most influenced by the following two. In [10] edges are only created between documents if they are dissimilar according to some distance measure of their TF-IDF scores. They prove that by assigning

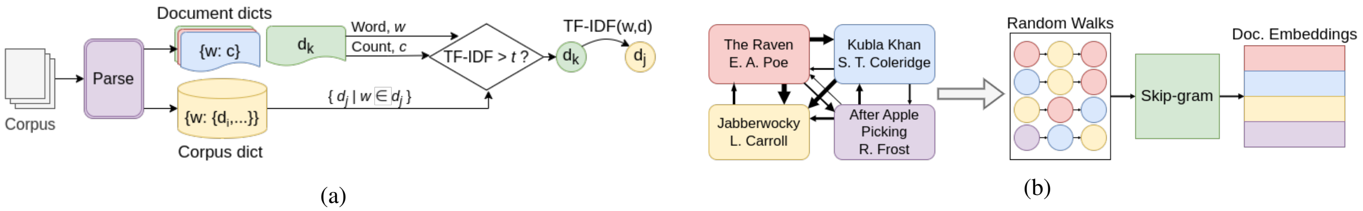


Fig. 1: The GRAGGLE pipeline: (a) First, build internal dictionaries for each document in the corpus, and calculate TF-IDF scores. If a word's score is above some tunable threshold, it is added into the edge weight from that paper to any other paper containing that word. (b) Next, generate random walks biased by edge weight. These walks are fed into a skip-gram and processed into vector embeddings for each node.

each node a color such that no two colors are adjacent, sets of nodes with the same color approximate clusters of TF-IDF vectors. Unfortunately, this method is difficult to adapt to larger graphs. The method used by [31] assigns topics to papers using latent Dirichlet allocation before constructing the graph. Documents are represented as nodes, connected by edges if the correlation coefficient between nodes' topics in is above some threshold. The Louvain algorithm is used to detect communities within the graph to identify seminal papers. Both [10] and [31] use data structures similar to ours, but they rely on inefficient graph analytical methods to generate clusters. By using a more modern, but still low-parameter autoencoder approach to represent these graphs, we can discover the same or more information in a more efficient way.

III. Method

GRAGGLE consists of two main components: graph construction, and generating node embeddings. In this section, we describe these processes in detail.

A. Graph Construction

To generate node embeddings and cluster the documents, we first extract all words from the corpus, and build the graph representing it. The preprocessing stage consists of two steps: building document and corpus dictionaries, and building the graph[2]. This process is illustrated in Subfigure 1a.

To build the dictionaries, each document in the corpus is parsed to generate a hash map that keeps track of how many times each word in the document was used. At the same time, a corpus-wide dictionary is constructed which maps each word to the set of documents that use that word. By the end of this process, each document in the corpus is converted to a bag of words representation that maps words to the number of their occurrence in that document, and the corpus-wide dictionary holds a map from each word in the corpus, to the set of documents containing them.

After the dictionaries have been generated, the directed, weighted graph can be constructed. This process is shown in Algorithm 1. The graph, $\mathcal{G}(\mathcal{V}, \mathcal{E})$ is defined as a set of nodes, $\mathcal{V}$ which represent individual documents, and a set $\mathcal{E}$ of weighed edges between them. The set $\mathcal{E}$ is defined as $\mathcal{E} = \{(u, v) \mid u, v \in \mathcal{V}\}$ where each edge in $\mathcal{E}$ has a weight $W : \mathcal{E} \to \mathbb{R}^+$. To build it, GRAGGLE iterates over each unique word w in each document d and calculates the TF-IDF scores as defined by Equation 1 (lines 3–6). If this value is above a tunable threshold t, an edge from d to any document containing w is created with the TF-IDF score as its weight. If an edge already exists between d and another document, the TF-IDF score is added to the edge weight (lines 7–15). We use a compressed sparse row (CSR) matrix to store the graph for its efficiency in sampling node v's one-hop neighborhood, $\mathcal{N}(v)$ (lines 17–19).

Algorithm 1 Build a weighted, directed graph from the dictionaries

```
 1: procedure BUILDGRAPH($corpus$, $docDicts$)
 2:     Initialize
            $idxPtr \leftarrow [0]$
            $column \leftarrow$ empty list
            $values \leftarrow$ empty list
            $threshold \leftarrow$ tunable parameter $\in \mathbb{R} \geq 0$
 3:     for $doc \in docDicts$ do
 4:         $edges \leftarrow$ empty hashmap of $keys$ to $values$
 5:         for $word, cnt \in doc$ do
 6:             $weight \leftarrow$ TF-IDF($cnt$, len($corpus[word]$))
 7:             if $weight \geq threshold$ then
 8:                 for $docID \in corpus[word]$ do
 9:                     if $docID \in edges.keys$ then
10:                         $edges[docID] \mathrel{+}= weight$
11:                     else
12:                         $edges[docID] = weight$
13:                     end if
14:                 end for
15:             end if
16:         end for
17:         append $idxPtr[-1]$+len($edges$) to $idxPtr$
18:         append $edges.keys$ to $column$
19:         append $edges.values$ to $values$
20:     end for
21:     return $idxPtr$, $column$, $values$
22: end procedure
```

For smaller data sets, the tunable threshold can be quite low; even setting t to zero produces usable results. However, to limit the amount of noise in the data, and to make the graph smaller and easier to process, we recommend setting it to at least 1, and setting it higher for larger data sets.

[2] We omit the text sanitizing process as we only work with ARFF files which have undergone this already.

Algorithm 2 Cluster documents

```
 1: Initialize
        $g \leftarrow$ a CSR matrix representing the graph
        $walkLen, numWalks \leftarrow$ hyperparameters
        $walks \leftarrow []$
        $numClusters \leftarrow$ user specified
 2: for $node \in g$ do  in parallel
 3:     $nWalks \leftarrow []$
 4:     while $i < numWalks$ do
 5:         $walk \leftarrow [node]$
 6:         while $\ell < walkLen$ do
 7:             $nextNode \leftarrow g[node]$.column[
                    $\mathcal{M}(\pi = g[node]$.values)
 8:             ]
 9:             append $nextNode$ to $walk$
10:             $node \leftarrow nextNode$
11:             $\ell + +$
12:         end while
13:         append $walk$ to $nWalks$
14:         $i + +$
15:     end while
16:     concatenate $nWalks$ and $walks$
17: end for
18: $X \leftarrow$ word2vec($walks$)
19: $y \leftarrow$ KMeans($X, numClusters$)
```

Note: $\mathcal{M}(\pi)$ denotes a selection from the multinomial distribution with probability mass function π.

B. Embedding

In the second step, node embeddings are generated using random walks. These embeddings can be used for clustering or any other form of vector analysis. This process is illustrated in Subfigure 1b.

After the graph has been created, standard graph autoencoding techniques may be used to generate document vectors. Formally, the problem can be defined as finding a function

$$f : \mathcal{V} \rightarrow \mathbb{R}^d \tag{2}$$

where $d \ll |\mathcal{V}|$. We accomplish this task by using a variant of the Node2Vec algorithm [12]. This process is formally outlined in Algorithm 2. We generate i random walks of length ℓ concurrently from each node, where i and ℓ are tunable parameters (lines 4–6). These walks are biased by the edge weights of potential neighbors. Selecting the next neighbor in a walk can then be described by the discrete-time Markov chain

$$P(N_{t+1} = v | N_t = u) = \frac{W(u, v)}{\sum_{n \in \mathcal{N}(u)} W(u, n)} \tag{3}$$

A single step of the random walk is abstracted to a random variable from a multinomial distribution, where the probability mass function is the normalized edge weights, and its realization is the index of the neighbor to walk to (line 7). This is done for each node in parallel, as the order that the skip-gram reads the completed walks in is unimportant, and the only operation on the shared graph data structure is `read`.

This results in a sequence of walks $\{S_0, S_1, \ldots, S_i\}$ which can then be analyzed by Word2Vec [26] to generate node embeddings (line 18). This algorithm optimizes its embeddings such that nodes which appear together in a sequence no more than ω hops apart have a higher cosine similarity than nodes which do not. Here, ω denotes the window size, and is a tunable hyperparameter. Tests showed that uniformly random walks yielded very poor results, and the weighting is an effective method to provide better information about the relationships in the graph.

IV. Results

In this section, we investigate the effectiveness of our proposed embedding method for document clustering. We evaluate its effectiveness against benchmarks set by prior works [32] and [33].

A. Evaluation Measures

To test the effectiveness of GRAGGLE, we compare our output to the output of two benchmarks set for document clustering. Like the prior works, we use purity and entropy to measure the effectiveness of our embedding technique. These are defined by [34] as follows.

For a given set of k clusters, C, and a set of labels L which both partition n documents, for a particular cluster $C_r \in C$, purity is defined as

$$P(C_r) = \frac{1}{n_r} \underset{i}{max}(n_r^i) \tag{4}$$

where $n_r = |C_r|$ and n_r^i denotes the number of elements in C_r with true label i. The purity of the entire cluster solution is then

$$Purity = \sum_{r=1}^{k} \frac{n_r}{n} P(C_r) \tag{5}$$

Entropy is defined as the weighted sum of individual cluster entropy balanced for the size of the cluster.

$$\begin{aligned} E(C_r) &= -\frac{1}{\log |L|} \sum_{i=1}^{|L|} \frac{n_r^i}{n_r} \log \frac{n_r^i}{n_r} \\ Entropy &= \sum_{r=1}^{k} \frac{n_r}{n} E(C_r) \end{aligned} \tag{6}$$

A perfect purity score is 1, and a perfect entropy score is 0. However, by setting the cluster size to just 1 point this can be trivially achieved. For this reason, unless otherwise specified, the measurements reported use data that have been clustered into an equal number of partitions as class labels.

B. Data Set Information

To verify the effectiveness of GRAGGLE, for our experiments, we use four labeled data sets. The CSTR corpus[3], the Reuters-21578 corpus[4], the WebKB corpus[5], and the K data set[6], all of which have been widely used in prior works.

CSTR is a collection of abstracts from technical reports from the University of Rochester between the years 1991 and

[3] http://sites.labic.icmc.usp.br/text_collections/CSTR.arff.zip

[4] http://sites.labic.icmc.usp.br/text_collections/Reuters-21578.arff.zip

[5] http://sites.labic.icmc.usp.br/text_collections/webkb.arff.zip

[6] http://glaros.dtc.umn.edu/gkhome/fetch/sw/cluto/s.tar.gz

TABLE I: Data Set Metadata

Data Set	Documents	Unique Words	Classes
CSTR	299	1,725	4
Reuters-10	10,345	13,276	10
WebKB	8,282	22,891	7
WebKB-4	4,199	19,028	4
K Data Set	2,340	21,839	20

2002. **Reuters-10** is a collection of categorized articles from the Reuters newswire in 1987. As was done by [32] we only use documents from the top 10 most represented classes from the Reuters-21578 data set. **WebKB** is a collection of web pages from various university computer science departments. As was done by [32], [33], we do two experiments on this data set: one on documents from all classes, denoted WebKB, and one on the top 4 classes excluding the class "other", denoted WebKB-4. **K Data Set** contains documents published online in October 1997 by Reuters. Table I contains information about the metadata of each data set.

TABLE II: Cluster Purity

Data set	Vector	K-Means	SLINK	CLINK	UPGMA
CSTR	TF-IDF	0.744	0.532	0.423	0.643
	AE	0.742	0.434	0.732	0.639
	VAE	0.445	0.441	0.443	0.441
	Graggle	**0.813**	0.434	0.598	0.780
Reuters-10	TF-IDF	0.643	0.393	0.531	0.498
	AE	0.611	0.436	0.542	0.491
	VAE	0.583	0.428	0.495	0.435
	Graggle	**0.664**	0.382	0.417	0.402
WebKB	TF-IDF	0.423	-	-	-
	AE	0.455	0.455	0.455	0.455
	VAE	0.505	0.505	0.505	0.505
	Graggle	**0.627**	0.455	0.456	0.455
WebKB-4	TF-IDF	0.534	0.392	0.446	0.395
	AE	0.506	0.505	0.505	0.505
	VAE	0.505	0.505	0.505	0.505
	Graggle	**0.732**	0.391	0.451	0.391
K Data Set	TF-IDF	0.627	0.220	0.514	0.551
	AE	0.426	0.273	0.404	0.360
	VAE	0.406	0.291	0.378	0.343
	Graggle	**0.716**	0.220	0.639	0.593

C. Comparison to Vectors

To analyze the effectiveness of our method, we compare it to the results of two benchmarks generated with standard techniques.

The first benchmark is reported by [32]. This work clusters TF-IDF vectors using several different algorithms including an experimental one. As we are proposing a novel method of document embedding, not clustering, we only compare our results to those generated through widely used clustering algorithms.

We cluster the same embeddings using four different algorithms: K-Means, Single Linkage (SLINK), Complete Linkage (CLINK), and Unweighted Pair Group Method with Arithmetic mean (UPGMA). In addition to the results from [32] we also compare Graggle to vectors generated with an autoencoder (AE) and a variational autoencoder (VAE). Both autoencoders follow similar architecture to that used by [35].

TABLE III: Change in cluster purity and entropy with respect to the number of clusters, K

K	Word Count		TF-IDF		Graggle	
	Pur.	Ent.	Pur.	Ent.	Pur.	Ent.
4	0.6211	0.6652	0.6322	0.6264	**0.7164**	**0.4858**
8	0.6234	0.6462	0.6535	0.6031	**0.8171**	**0.3364**
12	0.6500	0.6105	0.6413	0.6162	**0.8023**	**0.3463**
16	0.6448	0.6073	0.6435	0.6170	**0.8388**	**0.2775**

The benchmarks for TF-IDF vectors used the 1,000 most significant words of each data set as calculated with mutual information to class labels to form their TF-IDF vectors, meaning they are all 1,000 dimensional. Conversely, We use at most 512 dimensions for both the WebKB data sets and the K data set, 256 dimensions for the Reuters data set, and 128 dimensions for the CSTR data set. For the autoencoding methods, we use the same embedding dimensions as we use for Graggle.

The average results of 5 independent runs are reported in Table II. We observe that regardless of the algorithm, embeddings generated by Graggle perform equally, or better than simple TF-IDF vectors, with the exceptions of the agglomerative methods on the Reuters-10 data set, and UPGMA on the WebKB-4 data set by a very small margin. However, we note that with the best performing clustering algorithm, K-Means, Graggle embeddings always outperform both standard TF-IDF vectors, and those generated with autoencoders.

When testing against [32], we use as many clusters as there are classes, as this is how the scores we compare to were generated. However, with any clustering algorithm, allowing for more clusters can produce better results. To measure this empirically, we compare to the benchmarks set by [33]. This benchmark set measures the effect of increasing the number of clusters on purity and entropy. The prior work uses only clusters generated by the K-Means algorithm on the WebKB-4 data set. These benchmarks include scores for both TF-IDF vectors, and word count vectors.

Table III shows Graggle embeddings have better purity and entropy scores with the minimum number of clusters than the traditional methods and that adding more clusters improves both metrics at a far greater rate. When the number of clusters is quadrupled, purity increases by 0.1224 for Graggle embeddings, compared to a maximum change of 0.0289 in the word count vectors. Likewise, entropy decreases by 0.2083 over the same period for Graggle embeddings, compared to a decrease of only 0.0579 in the word count vector clusters. Additionally, for both word count and TF-IDF vectors, there are very swift diminishing returns when increasing the number of clusters; Graggle however, continues to improve as the number of clusters increases.

V. Hyperparameter Tuning

There are three important hyperparameters associated with Graggle: the threshold for what is considered an important word (t), the number of random walks per node (i), and

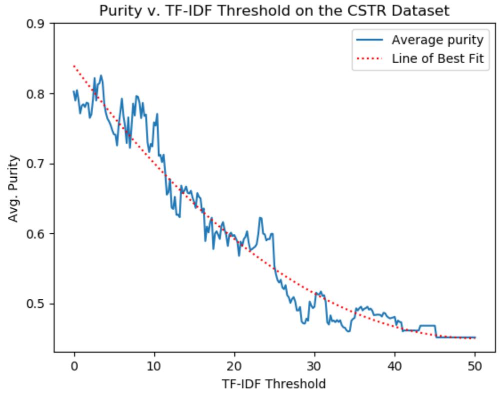


Fig. 2: The average cluster purity after 5 independent runs of GRAGGLE on the CSTR data set with variable hyperparameter TF-IDF threshold. The highest average purity occurs at threshold 3.4.

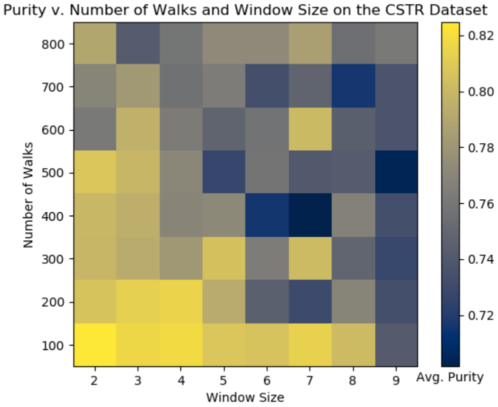


Fig. 3: The average cluster purity after 5 independent runs of GRAGGLE on the CSTR data set with variable hyperparameters number of walks and window size. The highest average purity occurrs when window size is 2 with 100 random walks.

the window size (ω). In this section we discuss methods of estimating these hyperparameters, as well as explore why some of them appear so invariant. Unless otherwise specified, the threshold value is 3.4, the number of walks is 200 and the window size used by Node2Vec is 3 for all experiments in this section.

Possibly the most important hyperparameter to tune is the TF-IDF minimum threshold. This parameter regulates the minimum impact a word must have in a document before it may be added into its edge weights. Leaving this parameter too low can cause an explosion of low-weight edges in the underlying graph data structure. This causes the output data to be very noisy and have a high degree of variance. It also directly influences memory and time complexity, both of which grow in proportion to the number of edges. On the other hand, setting this parameter too high comes at the risk of information loss. If the threshold value is too high, one risks filtering out too many edges, leaving some nodes orphaned. While this may sometimes be a desirable effect, and act to sort out low information papers from a corpus, for document classification, it is detrimental.

As Figure 2 shows, there is an negative exponential relationship between increasing the threshold value and cluster purity. We note however, that a threshold of zero, even for the smallest corpus, is still not a local maximum. Rather, the best threshold is a value greater than zero, but not too large. Because it takes longer to compute the smaller the threshold is, and the optimal threshold value is greater than zero, we recommend a backward parameter search to find this value.

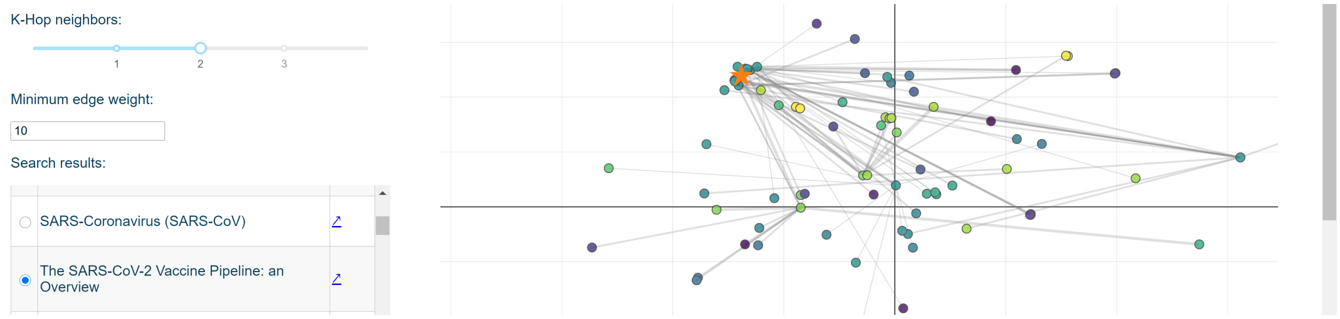
Figure 3 shows the effect on cluster purity of changing the number of walks per node and the window size in the random walk stage of GRAGGLE. As is evident from the figure, the best values for both the walk length, and the number of random walks are quite low. This is interesting, because the original Node2Vec paper found no diminishing returns for either parameter [12]. However, their algorithm was not running on graphs with weighted edges.

Intuitively, if one node is weakly connected to another, then their co-occurrence in a given set of positive samples should be exceedingly rare, if it occurs at all. For example, suppose two papers share one word with enough impact not to be filtered out, but no other words; their similarity is coincidental at best. But despite their listless linkage, if the number of walks from that first node is high enough, there is non-zero probability that they will be placed together in the encoder's positive sample set at least once per epoch. Thus, the weak connection is overlooked, and the learned embeddings are more similar. For this reason, when edge weight is important, a few biased walks are better than many long ones.

Large window sizes introduce an entirely new problem. By placing neighbors from too far away into the same set of positive samples, papers with no relation whatsoever to the starting paper may mistakenly be associated with it. The underlying graph data structure is extremely dense, so each node has an enormous degree. For this reason, it is critical not to stray far from the starting node's neighborhood. Rather than implementing an expensive algorithm to control this aspect of the walk, we find it is more efficient to clamp down the window size so there is no possibility of nodes outside of a starting node's immediate neighborhood appearing in a given positive sample set.

VI. Case Study: the CORD-19 Data Set

In the midst of the coronavirus pandemic, access to pertinent information is more important than ever. The rapid rate of new discoveries about this virus has created a wealth of research papers that should not be understood simply on their own, but as a gestalt of knowledge that builds upon itself. This rapid rate of publication means traditional citation network analysis is less effective as the most recent, and potentially most urgent papers have yet to be cited; this requires a text-based approach.



K-Hop neighbors: 2

Minimum edge weight: 10

Search results:

○ SARS-Coronavirus (SARS-CoV)

● The SARS-CoV-2 Vaccine Pipeline: an Overview

Fig. 4: The GRAGGLE search engine displaying t-SNE projected embeddings of the 2-hop neighbors of a search result. Different colored nodes represent different cluster assignments.

But the specific and sesquipedalian vocabulary used in medical journals makes TF-IDF approaches intractable.

To demonstrate the efficacy of GRAGGLE, we have implemented it on the CORD-19 corpus as a case study.[7] This data set consists of over 107,000 research papers about the coronavirus [13]. As the experiments show, using an autoencoder to generate vectors from a graph built with our method produces vectors which cluster better than those generated from just TF-IDF vectors. We then infer that the graph data structure itself holds valuable information about the papers it describes and is worthy of exploration on its own.

To do this, we generate a graph, and vector embeddings as described in § III and use K-Means clustering to label the 512-dimensional embeddings we generate. To accommodate for the great diversity of subcategories of papers contained in the CORD-19 data set, we set the number of clusters to 100 by default. By manually exploring the labels assigned to the documents, we can subjectively confirm that papers clustered together are highly related.

We then visualize the papers' embeddings using t-SNE decomposition [36] to generate two-dimensional coordinates. Finally, using a simple search engine based on the Euclidean distance between the TF-IDF encoding of the search term, and the titles in the corpus, we display papers, as well as neighbors from their highest weighted edges plotted according to their embeddings. Figure 4 shows the output of this simple search. We note that many of the well clustered neighbors do not appear in the TF-IDF search results; the only means of accessing them is through exploration of the graph. These are highly related papers that would have otherwise been overlooked.

Once one paper is identified using traditional methods, the entirety of the graph is available for exploration. With GRAGGLE, papers are shown in their full context. We have shown that this technique is empirically more effective for paper clustering, and we believe that our visual graph exploration method is not only more aesthetically pleasing than traditional search engines, but also more informative.

In the results shown in Figure 5, we observe that only a small subset of the possible 100 classes appear in this neighborhood, and that each neighborhood adjacent to the node in focus is highly homogeneous. In this way, we see the paper is most strongly related to this handful of classes represented by its neighbors. It is computationally expensive to express that a paper is similar to multiple different classes using just TF-IDF vectors; here it is displayed visually, and even encoded into the vector representation.

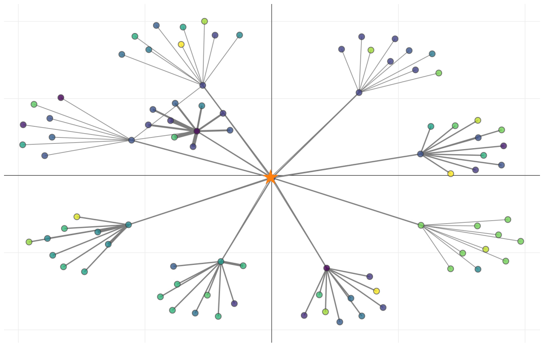


Fig. 5: A subgraph of GRAGGLE centered on the 2-hop neighborhood of the first result of a search viewed without t-SNE coordinates.

VII. CONCLUSION

In this work, we proposed GRAGGLE, a novel document embedding method. We build a graph that models shared words between papers and use a skip-gram autoencoder to generate low-dimensional node embeddings to represent documents. Clusters generated from these node embeddings consistently out-perform benchmarks set by prior works that used TF-IDF and word-count vectors, as well as non-graph-based autoencoding methods. We conclude that the vectors derived from GRAGGLE contain more information despite their lower-dimensional representation. Finally, we demonstrated the utility of this method by implementing it as a visual search engine on a real-world corpus.

Future work may include finding a method of building the graph that allows new nodes to be added *ad hoc*. And though the autoencoding stage of the method is quite efficient due to its high capability for parallelism, new breakthroughs in graph autoencoders, especially ones capable of inductive node

[7] https://graphlab.seas.gwu.edu/graggle

representation, would improve this process. We hope to see this process used with more advanced clustering algorithms. Such great reduction in document vectors' dimensionality could yield fast and efficient text classification algorithms in the future.

VIII. Acknowledgment

This work was supported in part by National Science Foundation grants 1618706 and 1717774.

References

[1] M. Lan, C. L. Tan, J. Su, and Y. Lu, "Supervised and traditional term weighting methods for automatic text categorization," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 31, no. 4, pp. 721–735, 2009.

[2] L. M. Abualigah, A. T. Khader, and E. S. Hanandeh, "A new feature selection method to improve the document clustering using particle swarm optimization algorithm," *Journal of Computational Science*, vol. 25, pp. 456–466, 2018.

[3] ——, "A combination of objective functions and hybrid krill herd algorithm for text document clustering analysis," *Engineering Applications of Artificial Intelligence*, vol. 73, pp. 111–125, 2018.

[4] D. R. Cutting, D. R. Karger, J. O. Pedersen, and J. W. Tukey, "Scatter/gather: A cluster-based approach to browsing large document collections," in *ACM SIGIR Forum*, vol. 51, no. 2. ACM New York, NY, USA, 2017, pp. 148–159.

[5] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," *CoRR*, vol. abs/1609.02907, 2016. [Online]. Available: http://arxiv.org/abs/1609.02907

[6] J. Atwood and D. Towsley, "Diffusion-convolutional neural networks," in *Advances in Neural Information Processing Systems 29*, D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, Eds. Curran Associates, Inc., 2016, pp. 1993–2001. [Online]. Available: http://papers.nips.cc/paper/6212-diffusion-convolutional-neural-networks.pdf

[7] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio, "Graph attention networks," 2017.

[8] D. Combe, C. Largeron, E. Egyed-Zsigmond, and M. Géry, "Combining relations and text in scientific network clustering," in *2012 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining*. IEEE, 2012, pp. 1248–1253.

[9] M. S. Hossain and R. A. Angryk, "Gdclust: A graph-based document clustering technique," in *Seventh IEEE International Conference on Data Mining Workshops (ICDMW 2007)*. IEEE, 2007, pp. 417–422.

[10] J. Dörpinghaus, S. Schaaf, and M. Jacobs, "Soft document clustering using a novel graph covering approach," *Biodata Mining*, vol. 11, 2018. [Online]. Available: http://proxygw.wrlc.org/login?url=https://search.proquest.com/docview/2056619535?accountid=11243

[11] D. Boley, M. Gini, R. Gross, E.-H. S. Han, K. Hastings, G. Karypis, V. Kumar, B. Mobasher, and J. Moore, "Document categorization and query generation on the world wide web using webace," *AI Review*, vol. 13, pp. 365–391, 1999.

[12] A. Grover and J. Leskovec, "Node2vec: Scalable feature learning for networks," in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '16. New York, NY, USA: Association for Computing Machinery, 2016, p. 855–864. [Online]. Available: https://doi.org/10.1145/2939672.2939754

[13] L. L. Wang, K. Lo, Y. Chandrasekhar, R. Reas, J. Yang, D. Eide, K. Funk, R. Kinney, Z. Liu, W. Merrill, P. Mooney, D. Murdick, D. Rishi, J. Sheehan, Z. Shen, B. Stilson, A. D. Wade, K. Wang, C. Wilhelm, B. Xie, D. Raymond, D. S. Weld, O. Etzioni, and S. Kohlmeier, "Cord-19: The covid-19 open research dataset," 2020.

[14] B. Larsen and C. Aone, "Fast and effective text mining using linear-time document clustering," in *Proceedings of the fifth ACM SIGKDD international conference on Knowledge discovery and data mining*, 1999, pp. 16–22.

[15] L.-P. Jing, H.-K. Huang, and H.-B. Shi, "Improved feature selection approach tfidf in text mining," in *Proceedings. International Conference on Machine Learning and Cybernetics*, vol. 2. IEEE, 2002, pp. 944–946.

[16] P. Bafna, D. Pramod, and A. Vaidya, "Document clustering: Tf-idf approach," in *2016 International Conference on Electrical, Electronics, and Optimization Techniques (ICEEOT)*. IEEE, 2016, pp. 61–66.

[17] S. Vijayarani, M. J. Ilamathi, and M. Nithya, "Preprocessing techniques for text mining-an overview," *International Journal of Computer Science & Communication Networks*, vol. 5, no. 1, pp. 7–16, 2015.

[18] W. Zhang, T. Yoshida, and X. Tang, "A comparative study of tf* idf, lsi and multi-words for text classification," *Expert Systems with Applications*, vol. 38, no. 3, pp. 2758–2765, 2011.

[19] P. Baldi, "Autoencoders, unsupervised learning, and deep architectures," in *Proceedings of ICML workshop on unsupervised and transfer learning*, 2012, pp. 37–49.

[20] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.

[21] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.04805*, 2018.

[22] S. González-Carvajal and E. C. Garrido-Merchán, "Comparing bert against traditional machine learning text classification," *arXiv preprint arXiv:2005.13012*, 2020.

[23] A. Subakti, H. Murfi, and N. Hariadi, "The performance of bert as data representation of text clustering," *Journal of big Data*, vol. 9, no. 1, pp. 1–21, 2022.

[24] S. R. Bowman, L. Vilnis, O. Vinyals, A. M. Dai, R. Jozefowicz, and S. Bengio, "Generating sentences from a continuous space," *arXiv preprint arXiv:1511.06349*, 2015.

[25] Z. Yang, Z. Hu, R. Salakhutdinov, and T. Berg-Kirkpatrick, "Improved variational autoencoders for text modeling using dilated convolutions," *arXiv preprint arXiv:1702.08139*, 2017.

[26] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," pp. 3111–3119, 2013.

[27] Q. Le and T. Mikolov, "Distributed representations of sentences and documents," in *Proceedings of the 31st International Conference on International Conference on Machine Learning - Volume 32*, ser. ICML'14. JMLR.org, 2014, p. II–1188–II–1196.

[28] Y. Shao, S. Taylor, N. Marshall, C. Morioka, and Q. Zeng-Treitler, "Clinical text classification with word embedding features vs. bag-of-words features," in *2018 IEEE International Conference on Big Data (Big Data)*, 2018, pp. 2874–2878.

[29] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio, "Graph attention networks," *arXiv preprint arXiv:1710.10903*, 2017.

[30] T. N. Kipf and M. Welling, "Variational graph auto-encoders," *arXiv preprint arXiv:1611.07308*, 2016.

[31] A. Gupta, S. Sikdar, P. Mohapatra, and N. Ganguly, "Topic influence graph based analysis of seminal papers," in *Proceedings of the 7th ACM IKDD CoDS and 25th COMAD*, ser. CoDS COMAD 2020. New York, NY, USA: Association for Computing Machinery, 2020, p. 224–228. [Online]. Available: https://doi.org/10.1145/3371158.3371191

[32] T. Li, S. Ma, and M. Ogihara, "Document clustering via adaptive subspace iteration," in *Annual ACM Conference on Research and Development in Information Retrieval: Proceedings of the 27th annual international ACM SIGIR conference on Research and development in information retrieval; 25-29 July 2004*, 2004, pp. 218–225. [Online]. Available: http://search.proquest.com/docview/31135460/

[33] R. Lakshmi and S. Baskar, "Dic-doc-k-means: Dissimilarity-based initial centroid selection for document clustering using k-means for improving the effectiveness of text document clustering," *Journal of Information Science*, vol. 45, no. 6, pp. 818–832, 2019.

[34] Y. Zhao and G. Karypis, "Criterion functions for document clustering," Tech. Rep.

[35] X. Peng, H. Zhu, J. Feng, C. Shen, H. Zhang, and J. T. Zhou, "Deep clustering with sample-assignment invariance prior," *IEEE Transactions on Neural Networks and Learning Systems*, pp. 1–12, 2019.

[36] L. v. d. Maaten and G. Hinton, "Visualizing data using t-sne," *Journal of machine learning research*, vol. 9, no. Nov, pp. 2579–2605, 2008.